IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 10 March 2024, accepted 3 June 2024, date of publication 11 June 2024, date of current version 18 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3412656

RESEARCH ARTICLE

An Ontological Behavioral Modeling Approach With SHACL, SPARQL, and RDF Applied to Smart Grids

MOHAMED LARHRIB[1], MIGUEL ESCRIBANO[2], CARLOS CERRADA[1], AND JUAN JOSE ESCRIBANO[1]

[1]Systems and Software Engineering Department, Universidad Nacional de Educación a Distancia, 28040 Madrid, Spain
[2]Department of Forecast and Load Scheduling, Red Eléctrica de España, 28109 Alcobendas, Spain

Corresponding author: Juan Jose Escribano (jjescri@issi.uned.es)

This work was supported by the Department of Computer Systems and Software Engineering at the Universidad Nacional de Educación a Distancia, Spain.

ABSTRACT Every engineering process, especially software, involves two complementary aspects: structural and behavioral. Behavior is, in essence, transforming the structure associated with the system. As a language for the object-oriented paradigm, Unified Modeling Language (UML) offers constructs for both aspects, for example, class diagrams for the structural aspect and activity diagrams for the behavioral aspect. However, without obtaining directly executable models, in glass-box terms, or reasoning support, on the other hand, when software engineering is approached with ontologies, only constructs for structural aspects are provided to develop a directly executable model, thanks to their reasoning capability. However, there are no constructs or approaches for this paradigm's specification or definition of behavior. This lack appears mainly in the early stages of the software engineering process, where there are no constructs similar to, e.g., the activity diagram in the object-oriented domain. Object Management Group (OMG) already addressed the transformation between the two paradigms in structural terms throughout Ontology Definition Metamodel (ODM) from UML to Resource Description Framework (RDF) and Web Ontology Language (OWL). However, there is no transformation of the object-oriented behavioral constructs into ontologies because they are not defined in the ontological paradigm. This paper addresses the definition of behavior in the ontology paradigm and the transformation of behavioral constructs between the two paradigms. The foundation of behavior specification is the flow concept, and the basis of this is the transformation of the structural model in an evaluative sense. Therefore, once the behavior has been defined in the ontology domain, the artifacts obtained throughout the life cycle are directly executable, and their validation and testing are automatic. With this approach, the life cycle is reduced to a modeling process. Thus, the resulting software engineering process improves features such as agility, simplicity, productivity, and formalism. The target audience for this work is the software engineering community, especially in the Model-Driven Engineering (MDE) paradigm approached from object-oriented and ontology perspectives. The evaluation of the proposed approach has been performed in the electric utilities, solving the problem of the validation flow for the interoperability process specified by the Common Grid Model Exchange Standard (CGMES) standard.

INDEX TERMS Behavioral modeling, CIM for ENTSO-E (CGMES), directly executable, ontology RDF/RDFS/OWL/SHACL.

LIST OF ACRONYMS

BPMN	Business Process Model and Notation.
CGMES	Common Grid Model Exchange Standard.
CIM	Common Information Model.
ENTSO-E	European Network of Transmission System Operators for Electricity.
EQ	Equipment.
IEC	International Electrotechnical Commission.

The associate editor coordinating the review of this manuscript and approving it for publication was Manoj Datta.

MDE	Model-Driven Engineering.
OCL	Object Constraints Language.
ODM	Ontology Definition Metamodel.
OMG	Object Management Group.
OWL	Web Ontology Language.
QOCDC	Quality of CGMES Datasets and Calculations.
RDF	Resource Description Framework.
RDFS	Resource Description Framework Schema.
SHACL	Shapes Constraint Language.
SPARQL	SPARQL Protocol and RDF Query Language.
SSH	Steady State Hypothesis.
SV	State Variables.
TP	Topology.
TS	Technical Specification.
TSO	Transmission System Operator.
UML	Unified Modeling Language.
W3C	World Wide Web Consortium.
XML	Extensible Markup Language.
XSD	XML Schema Definition.

I. INTRODUCTION

Any engineering process to obtain a specific artifact involves building its structural and behavioral components. The two aspects are dependent since one is built on the other. In software engineering for the object-oriented paradigm, the UML is used in the early stages of the life cycle, especially in the requirements specification and design stage, where constructs are provided for the structural aspects, such as class and object diagrams, along with OCL rules. Other behavioral constructs, such as state machines and activity diagrams, are provided for the behavioral aspect. However, the resulting artifacts are neither directly executable nor have reasoning capacity.

On the other hand, in the ontological paradigm, the semantic web technologies provided by W3C, such as RDF, OWL, SPARQL, and SHACL, are used for the structural aspect, thus obtaining directly executable (without any model transformation) artifacts given the reasoning capacity they have implicitly. However, no language is provided for specifying behavior, similar to what the UML offers, such as the activity diagram or state machine. Therefore, no language with an ontological approach is available for the software engineering process, especially in the early phases of the life cycle, for the specification and definition of behavior.

The authors have focused and implemented their approach in the ontology provided by W3C since semantic web technologies RDF, OWL, SHACL, and SPARQL are widely used.

To the best of the authors' knowledge in the state-of-the-art, no language is available for specifying the behavior in the development phase. The requirements specification in the ontological paradigm system's behavior is realized by wrapping ontological models into object-oriented models and then performing functions on them (PySHACL [1], SHACL Jena [2], TOPBRAID [3], and RDFLIB [4]) or directly with SPARQL CRUD operations.

The provided approach to address this gap is based on a language founded on the following constructs: task, precondition, model, and postcondition. The semantics associated with the language is that a task is executed when a certain precondition is fulfilled on a specific model, and an output model can be made available after its execution. The postcondition has to be validated against the models. The orchestration of the graph/flow execution that represents the behavior is achieved through the flow processor, a core component of the approach.

Therefore, the behavior will be represented by a flow whose nodes are the tasks represented in Fig. 1., where the core of the approach and its visual representation as language are represented.

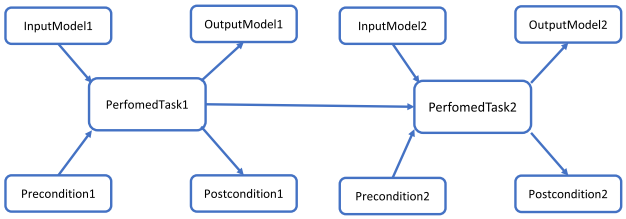


FIGURE 1. Basic visual elements for defining a flow.

A transformation of the behavioral constructs in the object-oriented paradigm provided by UML to the proposed approach constructs has been realized; this enables heterogeneous systems and serves as part of the evaluation of the approach.

The evaluation has been performed on CGMES, a standard consisting of a UML model representing the elements of the electric utilities and a set of rules. The rules are classified into eight levels and specified in natural language and in OCL, whose transformation into SHACL has been addressed in a previous work by the authors.

A data model generated by a specific entity, such as a TSO, is subject to eight validation processes corresponding to the eight validation levels specified in Quality of CGMES Datasets and Calculations (QOCDC), a set of rules provided by CGMES.

The software engineering community, ontology modeling engineers, the community dedicated to defining standards, and stakeholders involved in CIM CGMES for electric utilities will benefit from this behavior modeling approach.

In Section II, related work is presented in the state-of-the-art related to the proposed approach. Section III presents the behavioral modeling approach in the ontological domain and a didactic example of how this approach is applied. In section IV, the transformation of the basic constructs of the UML into the proposed approach constructs is performed. Section V develops the modeling of the validation flow of the CGMES standard. Finally, Section VI contains conclusions and future work.

II. RELATED WORK

At the state-of-the-art, no work has been found to make the UML constructs directly executable through a processor, with reasoning capacity and a glass-box approach.

For the early phases of software engineering, especially the specification of requirements with ontologies, there are frameworks and tools for the structural aspect. However, the community is limited to using the structural modeling of the ontologies (Protégé [5], SHACL Jena [2], AllegroGraph [6], StarDog [7], and RDF4J [8]), and the behavior is implemented by wrapping the ontologies in object-oriented technology (Jena [2]) or through SPARQL constructs for CRUD operations [9]. Hence, a high-level visual language that stakeholders from different technical and non-technical backgrounds can consume is needed.

Behavioral modeling in object-oriented with UML [10] covers the requirements specification phase, such as interaction diagrams, activity diagrams, and state machines. However, no artifacts that are directly executable with reasoning capacity are provided; even if code can be produced automatically, a later refinement phase is needed.

To the authors' best knowledge, no visual language like the UML or an adaptation of the software process with ontologies for the early phases is available in the state-of-the-art.

Works found in literature review research related to ontologies generally consist of developing an ontology for a given domain. Nevertheless, to the best of our knowledge, no academic research effort addresses the software development process with ontologies in both the behavioral and structural aspects. Moreover, the ones that exist are ad-hoc through a non-reusable process. The following research works are related to the concepts of the proposed approach, although they are all developed from an object-oriented perspective.

The criteria for classifying the related works that are valuable for the work being developed by the authors are as follows: i) Direct executability [ED] ii) The use of BPMN to address flows and orchestration in a given domain [FBPMN], iii) The application of the ontological approach to both a structural and behavioral domain [OADD], iv) The transformation between different paradigms and languages [TR], v) The validation of models at different levels of abstraction [VAL], vi) Ontology modeling [MOD-ONT], vii) Object-oriented modeling [MOD-OO], viii) Frameworks that address these criteria [FRMK], ix) Literature reviews in the areas corresponding to these criteria [LR].

Table 1 illustrates the classification of the literature review according to the set of criteria described above.

TABLE 1. Classification of the literature review.

Work ref.	ED	FBPMN	OADD	TR	VAL	FRMK	LR	MOD-ONT	MOD-OO
[11]								X	
[12]			X					X	
[13]			X					X	
[14]			X						
[15]			X					X	
[16]			X					X	
[17]			X					X	
[18]			X					X	
[19]			X					X	
[20]			X					X	
[21]			X					X	
[22]			X					X	
[23]							X		
[24]								X	
[25]						X	X		
[26]							X		X
[27]							X		X
[28]		X		X					
[29]		X							
[30]		X		X					
[31]		X		X					
[32]		X							
[33]				X	X	X			
[34]				X	X	X			
[35]				X				X	
[36]									X
[37]									X
[38]			X					X	
[39]			X					X	
[40]			X			X		X	
[41]			X					X	
[42]			X		X			X	
[43]			X					X	
[44]			X					X	
[45]			X					X	
[46]			X					X	
[47]	X								X

The following are efforts that apply ontologies to a specific domain.

In this work, given the complexity of selecting a web service that meets the requirements of a user, the authors have proposed a Semantic-based Business Process Execution Engine to execute business processes based on selecting web services and their orchestration at runtime. It uses an ontology designed to fit the users' requirements with the specification of the web services to optimize the selection from the service repository [11].

Adaptive mobile phone interfaces are modeled and implemented in this work through an ontological approach with semantic web technologies such as SPARQL and OWL, but SHACL is not used [12].

In this paper, the authors have developed an ontology for the UML sequence diagram without providing specific constructs for behavior in development with ontologies [13].

It is another effort that applies an ontology to a specific domain, in this case, Natural Environment Crisis Management. It addresses the IoT to prevent disasters with semantic web technologies. The workflow is addressed with Business Process Model and Notation (BPMN); the flow engine is powered by Activiti [https://www.activiti.org/]. The rules are designed and implemented separately from the flow definition, which generates a scalability problem. In contrast to our approach, rules are an implicit part of the flow [14].

This work is another application of an ontology for a specific domain, the chemical domain in this case. Despite being a recent work, it is not addressed with semantic web

technologies. Instead, automatic code generation is produced using the graphs for the ontologies [15].

It is a work that integrates an ontology into the domain of scientific workflows [16].

It is a work where an ontology has been developed for specific areas corresponding to smart grids, the multi-agent world, and web services [17].

This work has developed an ontology for the river flow and flood mitigation domains. The tool used in the ontology building process is Protegé, which provides only the model's structural definition [18].

The work described in this paper involves ontologies, requirements, and web services [19].

In this work, an ontology has been used to model events in the electric utilities domain of the Indian National Grid. Nevertheless, our approach considers that the event is not an elemental concept but the execution of a performed task when the precondition is fulfilled on a given model [20].

In this work, the automatic generation of a decentralized cyber-physical system has been performed using ontology transformation as part of the MDE approach [21].

This paper presents a platform for the authorization of linked data using a language for linked data protection policies. The policy language has been implemented on top of SPARQL [22].

The following works are literature reviews and surveys on the criteria related to the approach of this work.

This survey deals with Network Service Orchestration (NSO), reviewing the historical background, relevant research projects, enabling technologies, and standardization efforts [23].

This work is a compilation and presentation of the research results and the potential benefits of applying ontologies to address three challenges in software engineering: i) difficulty in communicating and sharing information; ii) effective management of software development phases; and iii) development techniques and environments to support the production of semantic software through an interdisciplinary approach [24].

This work is a survey on business process management suites [25].

It is a valuable job that has performed a systematic literature review of modeling approaches and tools for embedded systems [26].

A literature review on the application of ontologies to systems engineering is presented. Since ontologies allow a common and unambiguously interpretable vocabulary, both by humans and machines, in the structural and behavioral areas of the systems, it is also a survey that presents the transformation between UML and ontology constructs in the state-of-the-art. The work criticizes that mapping all the constructs of both paradigms is impossible. We consider that the mapping for all constructs is not needed; otherwise, they would be equivalent paradigms. Therefore, the usefulness of each one lies in the semantic differences between their constructs [27].

The following works are related to the use of BPMN together with criteria such as modeling and transformation.

This work improves the productivity related to the implementation of services through M2M and M2T, but without the following criteria: i) direct executability, ii) glass-box process, iii) reasoning capacity, iii) using semantic web technologies [28].

In this work, the microservices composition and orchestration have been specified in BPMN, and event-based choreography is used to perform their execution. The relationship between this work and our approach is that the abstraction level has been raised through BPMN constructs [29].

In this work, abstraction has been performed to encapsulate event stream processing as a set of business functions, using Event-driven Process Chains (EPCs) and BPMN 2.0, mapping transformations to executable process representation with an execution engine, and integrating it with third-party software [30].

This work models the behavior of IoT within the context of BPMN for business processes, performing a transformation to Callas code, a programming language for sensors executed through a virtual machine for IoT devices [31].

It is an effort to integrate BPMN with the process-driven approach (PDA) paradigm. The processes defined with BPMN are made executable thanks to a process engine in charge of orchestrating the flow defined by BPMN [32].

The foundation for the following works is transformation.

In this valuable work, a framework for formal verification in embedded systems design has been developed using computation tree logic (CTL), OCL for system Verilog, and a transformation engine to obtain the low-level code from the highest level defined with Systems Modeling Language (SysML) and UML [33].

This remarkable work presents a framework to specify the design and verification of requirements by developing a profile based on UML and SysML to represent the semantics of System Verilog from the low abstraction level of RTL to the high abstraction level provided by UML. As part of the effort, a transformation engine has been developed to generate RTL (Register Transfer Level (RTL)) code along with System Verilog assertion (SVA) code [34].

This paper proposes a transformation process from Computation Independent Model (CIM) to platform-independent model (PIM) and modeling validation in an ontology-based approach [35].

The following works address modeling, behavior criteria, and evaluation of modeling languages.

This paper presents a framework for the behavior-driven development (BDD) approach by introducing UML profiles [36].

In this work, to evaluate software process modeling languages, a quality model has been proposed that allows them to be characterized and compared [37].

This paper introduces a system framework grounded in linked data and ontologies, aiming to facilitate the efficient sharing and access of safety educational content for construction workers from diverse sources in a standardized fashion. To realize this objective, RDF and SPARQL have been used [38].

This work introduces the CURIOCITY ontology (Cultural Heritage for Urban Tourism in Indoor/Outdoor Environments of the CITY) designed to depict cultural heritage knowledge under UNESCO's definitions. The ontology incorporates an automated population process facilitating the conversion of a museum data repository (in CSV format) into RDF triples, thereby automating the population of the CURIOCITY repository [39].

This research introduces a method for modeling and analyzing knowledge in the context of health management for IT equipment, utilizing a knowledge graph. It entails the development of an ontological framework tailored to the entity-relationship model for creating a maintenance knowledge map. Subsequently, various technologies, including the knowledge graph, text classification, and Bayesian network, among others, are systematically integrated and employed to manage knowledge related to faults in IT equipment, facilitate fault localization, and enhance fault diagnosis reasoning [40].

This paper presents a classification of knowledge domains and a method for classifying datasets and ontologies of Linked Open Data (LOD) based on it. A significant portion of LOD is classified, and research is conducted to determine if a set of observed phenomena exhibits domain-specific characteristics [41].

This paper proposes a novel ontological model to implement interoperability in distributed Electronic Health Records (EHR) environments. The proposed semantic ontological model can unify different formats of EHR data. In this approach, RDF and SPARQL have been used [42].

In this work, to semantically enrich mobility data, a methodology is presented where a knowledge graph-based RDF representation is utilized to store datasets related to trajectories, thereby enabling uniform querying and analysis of enriched movement data [43].

In this work, a query language for ontologies for JOWL called SQWRL is proposed, designed as a temporal extension of the ontology query language [44].

This work proposes an approach that allows building a temporal ontology and managing its incremental maintenance to accommodate the evolution of this data in a temporal and multi-schema environment [45].

This work proposes an approach for creating a Knowledge Representation system on Blockchain involving RDF graph databases, Linked Data access methods, and Blockchain functionalities. These elements ensure the immutability, non-repudiation, and decentralization of the Knowledge Representation realized by standard RDF graph databases [46].

In this paper, the authors described a workflow runtime model concept for scientific workflows while maintaining a direct connection of individual tasks to their required infrastructure. However, this work has not involved ontology modeling, and the workflow does not support reasoning; furthermore, no flow modeling paradigm is observed [47].

The works addressing direct executability are based on automatic code generation without reasoning capability. Although the executability is direct, it is performed in black-box terms. BPMN is used in some works to model behavior flow in specific domains, such as the orchestration of web services. A BPMN engine is needed for the BMPN models to be executable but is implemented as a black-box artifact without reasoning capability. BPMN is intended for human interoperability, although it is forced to be executable on the machine through a BPMN engine. Many works apply the ontological approach to a particular domain to obtain models, but only in structural terms. However, once the ontology is obtained, models acquire reasoning capability. There are works dealing with the transformation between different languages of different levels of abstraction, and paradigms have also been included. The works consisting of frameworks that include criteria of interest to the proposed approach have been considered; this is useful for presenting the proposed approach as a framework and a tool. Given the criteria of interest in our approach, the authors have considered surveys and literature reviews.

The authors' research found no work dealing with behavioral modeling with ontology or a visual language that allows a flow's direct execution.

III. ONTOLOGY BEHAVIORAL APPROACH

Therefore, since no high-level language is available for the requirements specification phases, which is helpful for both technical and non-technical stakeholders for behavioral modeling for the software engineering process with an ontological approach, this work aims to provide a methodological approach supported by a visual modeling language to address the process of behavioral specification in the ontological field.

Behavior can be represented as a flow. A flow can be described as a graph. The flow's basic constructs are task, transition, event, decision, parallelism, and synchronization, as represented in Fig. 2. BPMN and the activity diagram, among others, are specified with this approach for the early phases of the software engineering life cycle, such as the specification of requirements.

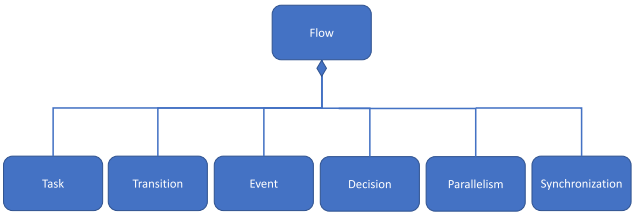


FIGURE 2. Basic constructs of a flow.

The following are definitions of the basic concepts involved in behavior.

The task is a transformation performed on the models.

The transition indicates the path from the execution of one task to the execution of another one.

event = (previous model, conditions, transformation, current model, such that previous model ≠ current model).

The decision is what will be executed once a condition has been validated against a model.

The main idea of the approach in this work stems from the observation that certain conditions have to be met on a specific set of resources to execute a specific thing. Even though it may seem trivial, it is a rigorous definition that offers enough concepts to develop a visual language directly.

Behavior, in its essence, is the transformation of a model. What is going to be executed is the task, modeled with the directly executable ontological paradigm. First, structural and behavioral information will be defined using an ontological model. Then, the conditions will be modeled and implemented with ontological rules to validate the information model. The decision concept in our approach is simply whether the information model is compliant with the rules that define the conditions.

Definition: A transition is a path from one task to another once the first has finished, which is essentially a transformation of a model.

Transition = <task1,task2,finished(task1),started(task2)>

The approach for this work consists of the following basic constructs: task, model, precondition, postcondition, and transition between tasks (arrows), as illustrated in Fig. 3.

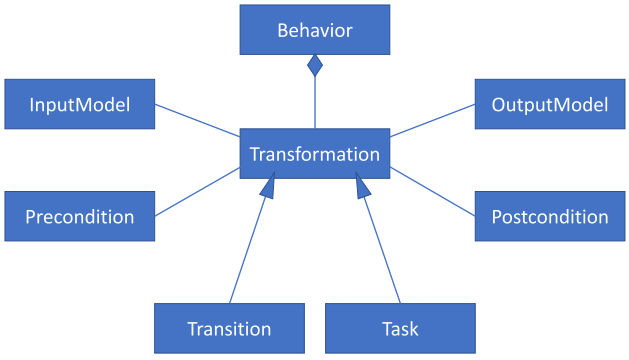


FIGURE 3. Basic constructs of behavioral specification.

The basic idea consists of executing a specific task when a particular condition is fulfilled on a specific data set (model).

The development of the approach is realized with the technologies of the semantic web is as follows:

Models are defined using the RDF or OWL languages. Models can be simple or complex; in this case, they will be constructed by a group of references to graphs representing models. The precondition is built using a set of SHACL rules.

The task can be modeled using the SPARQL language to perform the transformations behind the CRUD operations.

The semantics of the approach in the base case is shown in Fig. 4.

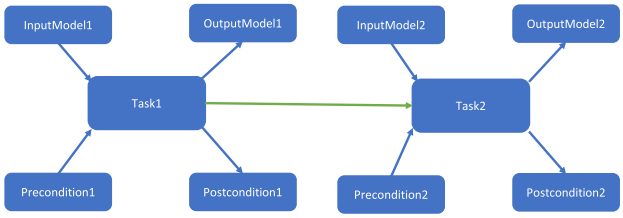


FIGURE 4. Base case for the proposed approach.

To execute task 1, model 1 must be valid against precondition 1, and postcondition 1 must be fulfilled on the models of interest for the requirements. Once task 1 is executed, task 2 is executed if model 2 is valid against precondition 2. According to our approach, the flow processor is in charge of orchestrating and executing the flow model that represents the behavior. The semantics representing the behavior flow will be executed by the flow processor that the authors have implemented through an algorithm, in this case, in Java.

The transition (green arrow) has an input model, an output model, a precondition, a postcondition, and a task T, as shown in Fig. 5. After Task1 is executed, Task2 is executed only if the input model meets the precondition of the transition, and the transition task T has been executed.

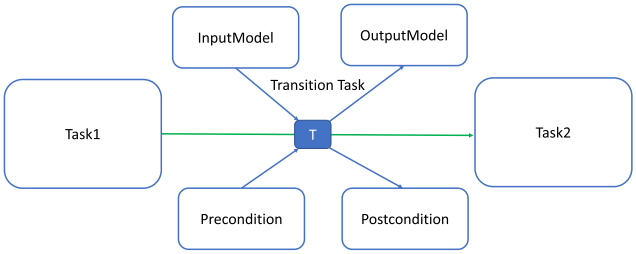


FIGURE 5. Transition model.

The flow processor architecture is shown in Fig. 6.

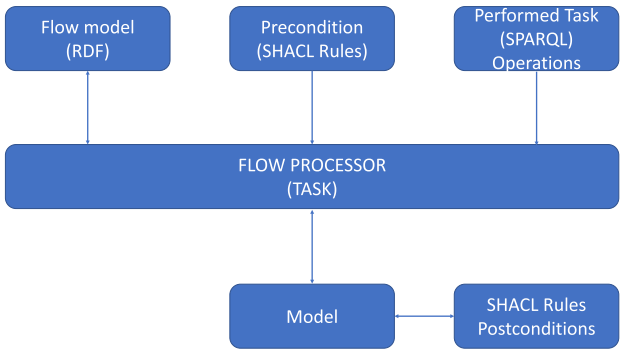


FIGURE 6. Flow processor architecture.

The flow of the flow processor shown in Figs. 7 and 8 illustrates the basic concepts and actions to execute the flow.

The process is executed at each node of the graph representing the flow. The red arrow indicates the iteration.

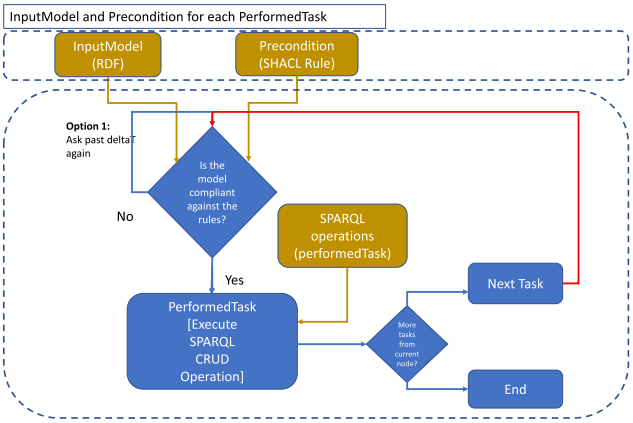


FIGURE 7. Flow of the flow processor for option 1.

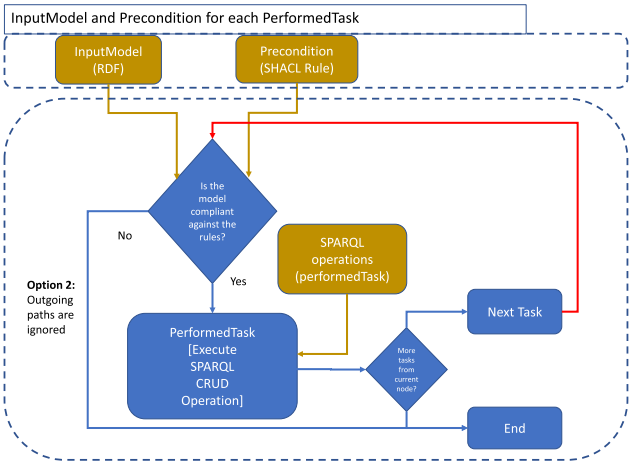


FIGURE 8. Flow of the flow processor for option 2.

When the validation result of the input model against the precondition has *false* as value, the following options for configuring the behavior of the flow processor can be adopted. (i) It remains permanently pending until the validation result is true to move on to the next task, as presented in Fig. 7. (ii) It ignores the outgoing tasks from the current node and never reruns the current task, as presented in Fig. 8.

The algorithm that corresponds to the flow processor is shown in Listing 1.

Fig. 9 shows the node class for implementing the proposed approach.

The graph behind the flow is executed by visiting each node. The node type can be: (i) *StartNode*; in this case, its outgoing edges and the corresponding destination node are obtained and executed; (ii) *GatewayNode*; In this case, the input model is validated against the *GatewayNode* precondition, and the *EdgeYes*, *EdgeNo*, *NodeYes*, and *NodeNo* are obtained. If the validation is correct, the *NodeYes* is executed; otherwise, the *NodeNo* is executed; finally, all nodes of type *SyncNode* that are reachable from the path that will not be executed are notified not to consider them. (iii)

```
# Execute the graph  behind the flow
function executeGraph() {
      sn = getStartNode();
      execute(sn);
}
function execute(node) {
    # If the current Node is equals to start node
    if(nodeType==StartNode){
        # The outgoing edges are obtained from the current node
        # which represent concurrent paths.
        oe = getOutgoingEdges();
        # Each node is executed concurrently
        foreach edge in oe {
           node = getNextNode(edge)
           execute(node);
        }
    }
    # If the current node is a gateway node
    else if(nodeType==GateWayNode){
        # The outgoing edges are obtained from the current node
        oe = getOutgoingEdges();
        # The input model is validated against the gateway node
        validationResult = validate(getPrecondition(),getInputGraph())
        # Edge yes, edge no, node yes and node no are obtained
        nodeYes = getNodeYes();
        nodeNo = getNodeNo();
        edgeYes = getEdgeYes();
        edgeNo = getEdgeNo();
        # If the validation  confoms.
        if(validationResult){
        # All reachable sync nodes from the path that is not traversed are notified not to consider the
            # resulting incoming edges.
            desableSyncPaths(edgeNo);
            # Node Yes is executed.
            execute(nodeYes);
        }
        # Else the inverse is executed
        else{
            desableSyncPaths(edgeYes);
            execute(nodeNo);
        }
    }
    # If the current node is a performed task
    else if(nodeType==PerformedTask){
        # The task corresponding to this node is executed
        executeTask(getTask());
        # The outgoing edeges are obatained from the current node
        oe = getOutgoingEdges();
        foreach edge in oe {
            # The node connected to the edege is obtained
            node = getNextNode(edge)
            # if the obtained node is a SyncNode
            if(getType(node)==SyncNode){
                # This Edge is registered as arrived
                node.hasArrived(edge,true);
                # If all edges that should arrive have arrived the syncNode is executed
                # (go to next nodes in the graph)
                if(allEdgesHaveArrived(node)){
                        execute(node);
                }
            }
            else{
                  execute(node);
            }
        }
    }
    # If the current node is a sync node
    else if(nodeType==SyncNode){
        # The outgoing edeges are obatained from the current node
        oe = getOutgoingEdges();
        foreach edge in oe {
            # The node connected to the edege is obtained
            node = getNextNode(edge)
            if(getType(node)==SyncNode){
                node.hasArrived(edge,true);
                # If all edges that should arrive have arrived the syncNode is executed
                if(allEdgesHaveArrived(node)){
                    execute(node);
                }
            }
            else{
                execute(node);
            }
        }
    }
    # If the current node is end node nothing is done.
    else if(nodeType==EndNode){
    }
}
```

LISTING 1. The flow processor algorithm.

PerformedTask; this scenario executes the task corresponding to this node, and the outgoing edges and their corresponding destination nodes are obtained. If the destination node is

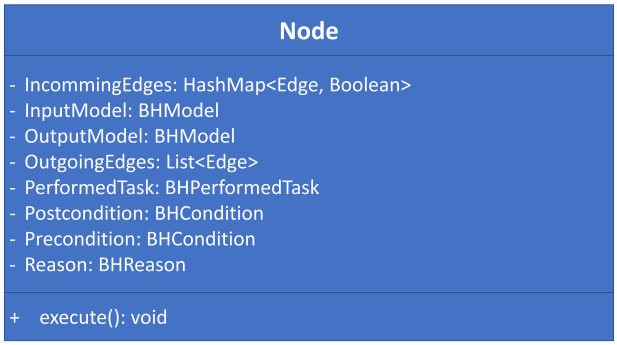


FIGURE 9. Node class.

not a *SyncNode*, it is executed; otherwise, the *SyncNode* is notified that this edge has already reached this node. If all incoming edges have arrived, then the node is executed. (iv) *SyncNode*; the outgoing edges and their corresponding destination nodes are obtained. For each node: if it is not of type *SyncNode*, it is executed; otherwise (if it is of type *SyncNode*), if all incoming edges have arrived, the node is executed.(v) *EndNode*; Nothing is done since it must not have outgoing edges.

The flow processor model with the proposed approach is illustrated in Fig. 10. Where: (1) represents the flow processor, enabling direct execution. (2) denotes the flow model representing the process behind the behavior to be executed, which is an RDF graph. (3) are the SHACL rules that establish the syntax conditions for the visual grammar of the language, defining the flow. (4) is the RDF model containing the required information about the flow execution status in structural terms. Finally, (5) are the conditions in SHACL that must be satisfied once the flow is executed.

The Listing 2 shows an excerpt of the ontological model of the flow.

```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
#
#
bh:Node a rdfs:Class.
bh:Condition a rdfs:Class.
bh:StartNode rdfs:SubClass bh:Node.
bh:EndNode rdfs:SubClass bh:Node.
bh:PerformedTask rdfs:SubClass bh:Node.
bh:Gateway rdfs:SubClass bh:Node.
bh:SyncNode rdfs:SubClass bh:Node.
bh:Transition a rdfs:Class.
bh:inputModel a rdf:Property;
    rdfs:domain bh:Model;
    rdfs:range bh:Node.
bh:precondition a rdf:Property;
    rdfs:domain bh:Condition;
    rdfs:range bh:Node.
bh:ouputModel a rdf:Property;
    rdfs:domain bh:Node;
    rdfs:range bh:bh:Model.
bh:postCondition a rdf:Property;
    rdfs:domain bh:Node;
   rdfs:range bh:Condition.
```

LISTING 2. Ontological model of the flow.

'@prefix bh: <http://www.bh.org/bh#>.' is the prefix that represents the namespace of behavioral modeling context.

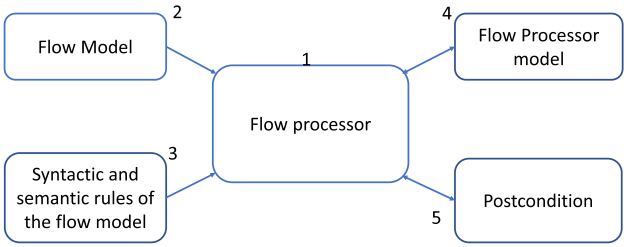


FIGURE 10. The flow processor model with the proposed approach.

```
# A bh:transition represented by an arrow must have one and only one bh:from node.
bh:br1 a sh:NodeShape;
   sh:targetClass bh:Transition;
   sh:property[
       sh:path bh:from;
       sh:minCount 1;
       sh:maxCount 1
   ].
# A bh:transition represented by an arrow must have one and only one bh:to node.
bh:br2 a sh:NodeShape;
    sh:targetClass bh:Transition;
    sh:property[
        sh:path bh:to;
        sh:minCount 1;
        sh:maxCount 1
    ].
```

LISTING 3. Syntactical rules defined in SHACL for the proposed approach.

An instance of the ontological model of the flow is shown in Fig. 11, representing all constructs involved in the approach.

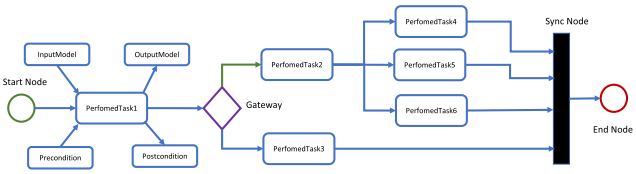


FIGURE 11. An instance of the ontological model of the flow.

An excerpt of the preconditions for the syntax and semantics rules of the flow model defined in SHACL is shown in Listing 3.

Fig. 12 shows a simple example of two tasks running sequentially.

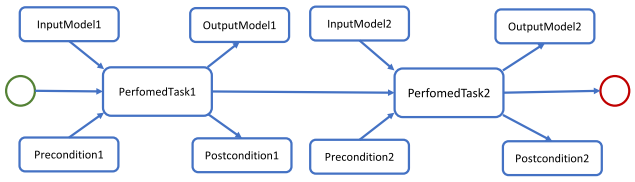
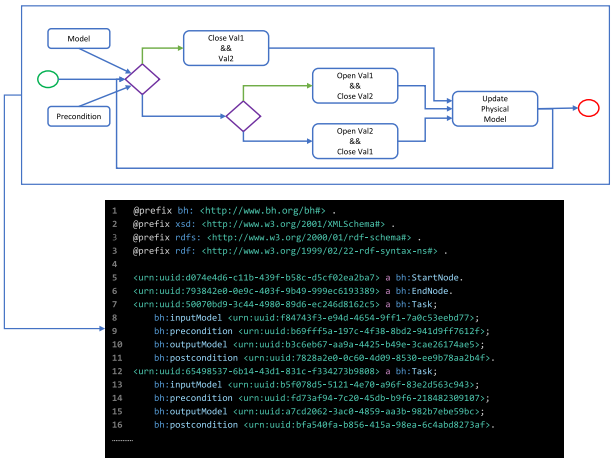


FIGURE 12. A simple flow example.

The RDF corresponding to the previous flow is illustrated in Listing 4. The identifiers of the RDF objects that correspond to visual elements are the values of the attributes of the SVG elements of each visual concept in the diagram.

```
<urn:uuid:bfbeadd9-4d2e-4106-860a-8aa78f7ebf05> rdf:type bh:StartNode.
<urn:uuid:d4b3c780-9943-4f58-81c8-806a3eb15dfb> rdf:type bh:EndNode.
<urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c> rdf:type bh:Task;
    bh:inputModel <urn:uuid:0a7d2786-13bc-4e5b-b3ae-7302b7686845>;
    bh:precondition <urn:uuid:beee3615-d927-46a6-88b4-2d9c26e26351>;
    bh:outputModel <urn:uuid:d2cd035d-cc32-41b1-8805-4b65a8791ac5>;
    bh:postcondition <urn:uuid:c900f09b-d9ea-42bd-bf4d-d04d603cf86f>.
<urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b> rdf:type bh:Task;
    bh:inputModel <urn:uuid:cec40576-2609-415d-8204-c8ccb5d16e37>;
    bh:precondition <urn:uuid:6f82e849-3882-443d-be8d-888144692326>;
    bh:outputModel <urn:uuid:9cb6b3c6-2074-43c3-abaa-c2dd2983f128>;
    bh:postcondition <urn:uuid:532b5424-b67e-4d1e-8462-0747ec15f99d>.
<urn:uuid:d5499124-71f9-4eec-b505-4db1eb6af660> rdf:type bh:Transition;
    bh:from <urn:uuid:bfbeadd9-4d2e-4106-860a-8aa78f7ebf05>;
    bh:to <urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c>.
<urn:uuid:6e2ad94d-db61-421b-8211-a5fbd710b3f4> rdf:type bh:Transition;
    bh:from <urn:uuid:f3b62a31-404c-4f13-889c-0dc65a7c262c>;
    bh:to <urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b>.
<urn:uuid:ba621958-22b6-4f86-9a0c-615452dd3295> rdf:type bh:Transition;
    bh:from <urn:uuid:ef54264b-57ef-4194-ab26-3cc2fe944e9b>;
    bh:to <urn:uuid:d4b3c780-9943-4f58-81c8-806a3eb15dfb>.
```

LISTING 4. An excerpt from the RDF that represents the model for Fig. 12.




```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<urn:uuid:d074e4d6-c11b-439f-b58c-d5cf02ea2ba7> a bh:StartNode.
<urn:uuid:793842e0-0e9c-403f-9b49-999ec6193389> a bh:EndNode.
<urn:uuid:50070bd9-3c44-4980-89d6-ec246d8162c5> a bh:Task;
    bh:inputModel <urn:uuid:f84743f3-e94d-4654-9ff1-7a0c53eebd77>;
    bh:precondition <urn:uuid:b69fff5a-197c-4f38-8bd2-941d9ff7612f>;
    bh:outputModel <urn:uuid:b3c6eb67-aa9a-4425-b49e-3cae26174ae5>;
    bh:postcondition <urn:uuid:7828a2e0-0c60-4d09-8530-ee9b78aa2b4f>.
<urn:uuid:65498537-6b14-43d1-831c-f334273b9808> a bh:Task;
    bh:inputModel <urn:uuid:b5f078d5-5121-4e70-a96f-83e2d563c943>;
    bh:precondition <urn:uuid:fd73af94-7c20-45db-b9f6-218482309107>;
    bh:outputModel <urn:uuid:a7cd2062-3ac0-4859-aa3b-982b7eb59bc>;
    bh:postcondition <urn:uuid:bfa540fa-b856-415a-98ea-6c4abd8273af>.
```

LISTING 5. The RDF behavioral model for the water tank.

IV. AN EXAMPLE OF THE APPLICATION OF THE APPROACH TO THE WATER RESERVOIR MODEL

This example illustrates how to address a specific problem from a particular domain with the ontological paradigm using our approach: model, precondition, postcondition, and performed task. The physical model that represents the water tank is shown in Fig. 13.

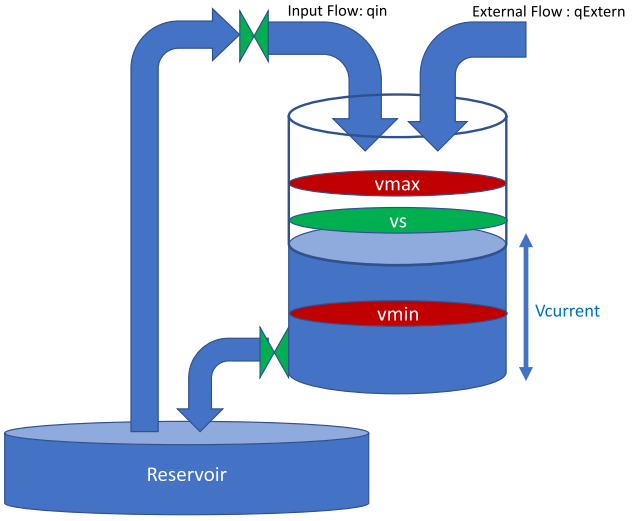


FIGURE 13. Reservoir physical model.

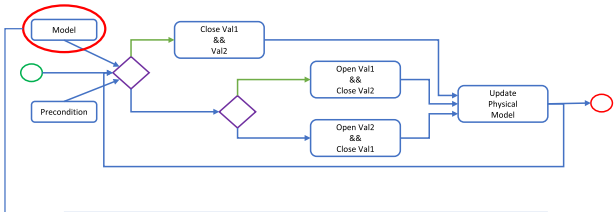
The goal is to control the volume of water ''vcurrent,'' which is regulated by ''vs'', taking into account that the minimum volume ''vmin'' and maximum volume ''vmax'' are fixed. ''valve1'' controls the inlet flow rate to the tank, and ''valve2'' controls the outlet flow rate. The user decides which set point to assign, and the system reacts to maintain the ''vcurrent'' at the value of ''vs'' by acting on the two valves.

The scenarios that have been considered are as follows:

Scenario 1: Open valve2 and close valve1 if vc > vs.

Scenario 2: Close valve2 and open valve1 if vc < vs.

Scenario 3: Close valve 2 and close valve 1 if vc == vs.




```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix bh: <http://www.bh.org/bh#> .
#
#
[] rdf:type bh:Model .
#
bh:vc bh:hasValue 0.
bh:vs bh:hasValue 10.
bh:val1 bh:hasValue 1.
bh:val2 bh:hasValue 1.
```

LISTING 6. RDF for valves model.

According to our approach for a simple water tank model such as a dam, the behavioral modeling is presented in Fig. 14.

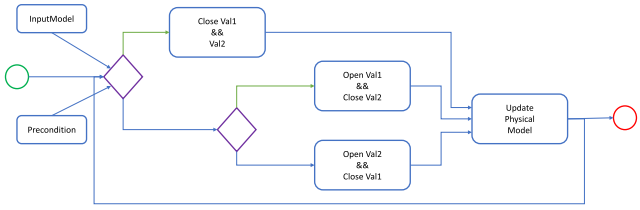


FIGURE 14. Behavioral modeling for a simple water tank.

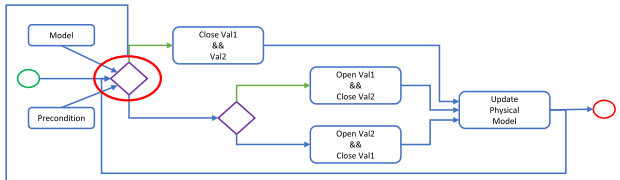
The RDF model associated with this flow is shown in the Listing 5.

For simplicity's sake, just an excerpt is shown for the three scenarios.

The RDF for the inputModel representing the two valves is depicted in Listing 6.

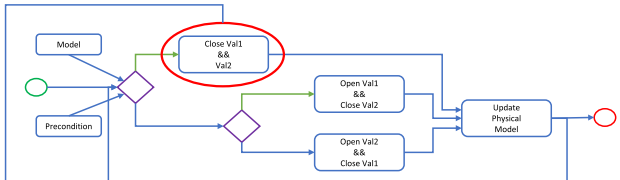
An excerpt of the conditions representing the different scenarios is defined in SHACL and depicted in Listing 7.

The task behind the scenario of closing the two valves is modeled in the SPARQL code represented in Listing 8.



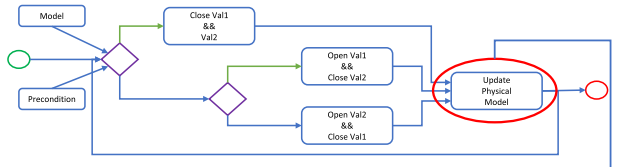

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix bh: <http://www.bh.org/bh#> .
#
#
#
#This rule checks if the current volume vc is the same as the set point volume vs.
bh:r1 a sh:NodeShape;
    sh:targetNode bh:vc ;
    sh:sparql [
        sh:select """
            ask
                Where{
                  $this <http://www.bh.org/bh#hasValue> ?x .
                  <http://www.bh.org/bh#vs> <http://www.bh.org/bh#hasValue> ?y.
                  FILTER (?x=?y)
                }
          """ ;
     ].
```

LISTING 7. An excerpt of SHACL shapes for the first gateway precondition.




```
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bh: <http://www.bh.org/bh#>
#
#
WITH <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>
DELETE {
    bh:val1 bh:hasValue 1.
    bh:val2 bh:hasValue 1.
}
insert {
    bh:val1 bh:hasValue> 0.
    bh:val2 bh:hasValue 0.
}
WHERE {}
```

LISTING 8. SPARQL code for the task of closing the two valves.




```
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bh: <http://www.bh.org/bh#>
#
with <urn:uuid:8ae8c965-c90c-4000-8e24-93aed2b8a175>
delete {bh:vc bh:hasValue ?cv.}
insert {bh:vc bh:hasValue ?nv.}
WHERE {
    bh:vc bh:hasValue ?cv.
    bh:val1 bh:hasValue ?vv1.
    bh:val2 bh:hasValue ?vv2.
    bind (<http://www.w3.org/2001/XMLSchema#int>(?vv1) as ?vvd1)
    bind (<http://www.w3.org/2001/XMLSchema#int>(?vv2) as ?vvd2)
    bind (<http://www.w3.org/2001/XMLSchema#int>(?cv) as ?cvd)
    bind( (?cvd+?vvd1-?vvd2) as ?nv)
    # vc(t) = vc(t-1)+k1*status_val1-k2*status_val2
}
```

LISTING 9. SPARQL for updating the physical system model.

The task of updating the current volume vc takes into account the valves' status and the equation $vc(t) = vc(t-1) + k1 * status\_val1 - k2 * status\_val2$, is depicted in Listing 9.

TABLE 2. The essential UML diagrams.

Modeling	UML diagrams
Use	Use case
Conceptual domain	Class
Process	Activity
Architecture	Components, Package, Deployment
dynamic	Communication, Structure, Interaction, Sequence, State machine, Timing
Structural	Class, Object

For simplicity k1 and k2, which represent the flow rates of each valve, take the value 1. The variation of the volume of the tank per unit of time is one unit.

In charge of updating the physical system, the performed task models a simple equation: $vc(t) = vc(t-1) + k1 * status\_val1 - k2 * status\_val2$, where time is a discrete variable. $vc(t)$ is the current volume at the instant t. $vc(t-1)$ the volume at the previous instant (one unit before instant t) k1 and k2 are the amounts of water supplied or withdrawn by valves val1 and val2, respectively, in a unit of time (flow rate of val1 and val2), and $status\_val1$ and $status\_val2$ represent whether the valves are open or closed. In this example, a basic linear model describes the system's dynamics. However, the model can be replaced by a more complex one in which the differential equation is of a higher order, keeping data from previous instants in the RDF model and dealing with the non-linearity of the valves.

V. TRANSFORMATION OF BEHAVIORAL UML CONSTRUCTS TO THE PROPOSED APPROACH

The proposed approach for behavioral modeling provides a set of visual or graphical constructs with their corresponding syntax in the semantic web, RDF SHACL, SPARQL, or OWL.

A. DEFINITION

A semantic processing entity is an entity with the capability to receive and send messages and reasoning about them, which can be represented as graphs.

The UML provides a set of constructs as a language for the requirements specification, analysis, and design phases. However, they are not directly executable and are implemented with general-purpose, object-oriented languages. The essential diagrams are shown in Table 2.

B. ACTIVITY DIAGRAM

The activity diagram is a flowchart comprising the constructs represented in Table 3.

In order to illustrate our approach, exclusive gateways with only two yes-or-no outputs are addressed for simplicity's sake. Although the most fundamental constructs for flow elaboration are addressed, the remaining constructs can be

TABLE 3. Activity diagram basic constructs with concept mapping of the proposed approach.

Construct	UML Symbol	Semantics	The mapped construct in the approach
Action	Action	It represents a step in a given activity.	Model, Precondition, Action, Model, postcondition
Decision Node		It represents conditional branches.	Model, Precondition, Yes, No
Control Flow		Connector between steps where the execution is directed in a flow.	
Initial Node		The starting point for executing a flow.	
Final Node		It represents a node where there is no execution beyond it.	
Fork Node		It represents parallel execution.	
Join Node		It synchronizes multiple flows.	

modeled with our approach. Activity nodes also include control flow constructs such as synchronization, decision, and concurrency control.

The visual elements of behavioral UML have been reused for our approach, although the semantics may vary to adapt it to the ontological domain.

OMG provides, among others, the BPMN, the activity diagram, and the state machine, which are mainly graphs but ignore the graph conceptualization, and are just treated as diagrams.

Our approach assumes that everything is a graph. Thus, structures, processes, and behaviors are graphs.

Suppose a new construct is needed to cover a specific use case in the future. In that case, it only is necessary to give a name as an identifier or a symbol to the construct along with its semantics and whose implementation will be added to the behavior graph processor.

C. STATE MACHINE

The basic constructs of a state machine are the states, transitions, inputs, and outputs, as shown in Fig. 15.

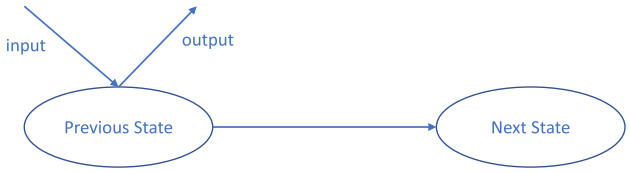


FIGURE 15. State machine basic constructs.

The concept mapping is illustrated in Fig. 16.

Fig. 17 can represent the case when the focus is on the state node, where an input is expected, a task is performed, an output is produced, and the transition to the next state is realized.

Fig. 18 can represent the case when the focus is on the transition that starts from a state, an input model is expected, a task is performed, an output is produced, and the transition to the next state is accomplished.

As in our approach, the transition has its input model, precondition, output model, and postcondition explicitly;

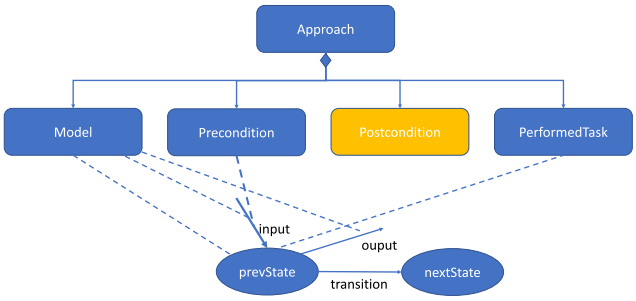


FIGURE 16. State machine basic constructs with concept mapping of the proposed approachs.

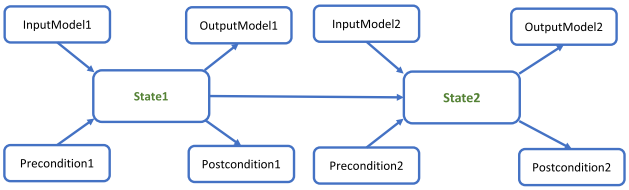


FIGURE 17. The corresponding flow for a state machine where the focus is on the node.

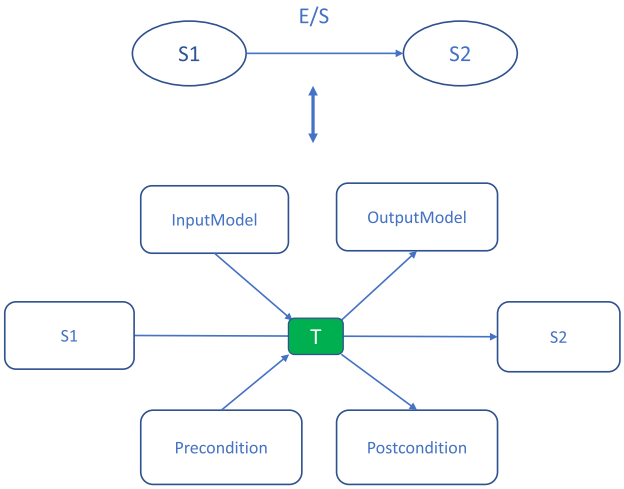


FIGURE 18. The corresponding flow for a state machine where the focus is on the transition.

thus, the behavior of the state machine can be implemented when the input and output are described in the transition.

If the input and output are described in the state, then the performed task is used as the state; the state machine input will be represented by (input model, precondition) and the output by (output model, postcondition).

D. INTERACTION DIAGRAM

In UML, interaction is mainly represented by sending and receiving messages, which can carry parameters between two components or objects.

In our approach, messages are modeled based on a specific metamodel of instances and the capability to reason about them. The basic structure for the interaction is represented in Fig. 19.

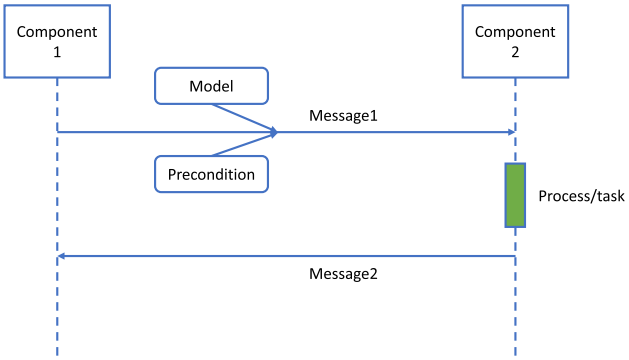
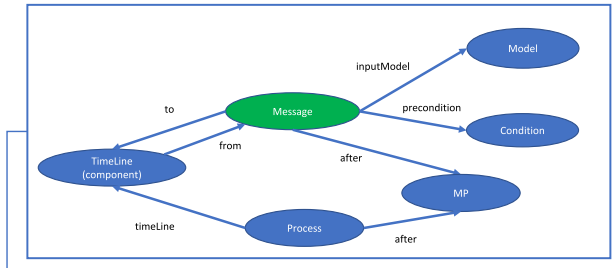


FIGURE 19. The interaction model according to our approach.




```
@prefix bh: <http://www.bh.org/bh#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

bh:TimeLine a rdfs:Class.
bh:Process a rdfs:Class.
bh:Message a rdfs:Class.
bh:Model a rdfs:Class.
bh:Condition a rdfs:Class.
bh:Flow a rdfs:Class.
bh:from a rdf:Property;
rdfs:domain bh:TimeLine;
rdfs:range bh:Message.
bh:to a rdf:Property;
   rdfs:domain bh:Message;
   rdfs:range bh:TimeLine.
```

LISTING 10. RDF Schema of the interaction diagram constructs.

The RDF graph that corresponds to the interaction diagram of Fig. 19 is shown in Fig. 20. The MP class is the superclass that represents the two Message and Process classes.

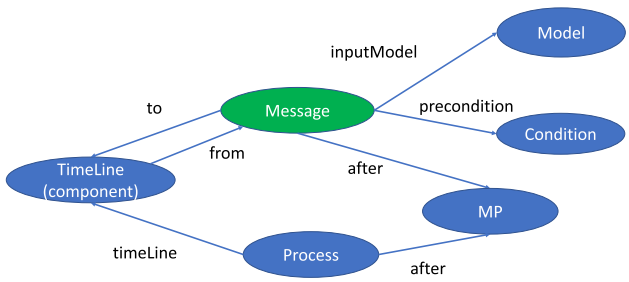
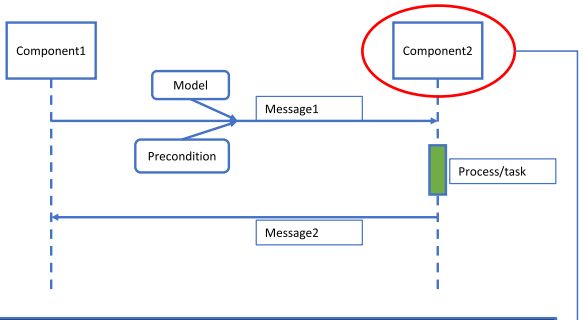


FIGURE 20. The RDF graph corresponding to the interaction diagram of Fig. 19.

The RDFS corresponding to the constructs on which the interaction diagram is based is shown in Listing 10.

The RDF corresponding to component2 of Fig. 19 is shown in Listing 11.




```
<urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> rdf:type bh:TimeLine .
<urn:uuid:6499c27a-fcc7-48d9-9080-2d75926d8149> rdf:type bh:Message ;
    bh:from <urn:uuid:182d3393-17f1-46bc-9f10-62cbdd3f6c41> ;
    bh:to <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
    bh:name "Message1" ;
    bh:inputModel <urn:uuid:d6887daf-5180-4e25-ad1f-c332b9fc37d0> ;
    bh:precondition <urn:uuid:f35267d2-96b7-4fd7-8beb-0c57eb6bd363> .
<urn:uuid:605cb512-357c-42db-a036-2482a3313d72> rdf:type bh:Process ;
    bh:timeLine <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
    bh:hasFlow <urn:uuid:573451bb-bca5-41ee-a1e3-ace6af0f1583> ;
    bh:after <urn:uuid:6499c27a-fcc7-48d9-9080-2d75926d8149> .
<urn:uuid:10fe2a7d-3657-4343-803e-cb5ae8f37376> rdf:type bh:Message ;
    bh:from <urn:uuid:b02c2e12-96ed-4499-9769-6960bf1b92ed> ;
    bh:to <urn:uuid:182d3393-17f1-46bc-9f10-62cbdd3f6c41> ;
    bh:name "Message2" ;
    bh:inputModel <urn:uuid:b3b28bbd-a72f-4fe7-8ea0-578cd0873cf4> ;
    bh:precondition <urn:uuid:f1072829-7429-488a-8310-fd6cf3300faa> ;
    bh:after <urn:uuid:605cb512-357c-42db-a036-2482a3313d72> .
```

LISTING 11. RDF corresponding to component2 of Fig. 19.

The implementation of the proposed approach concerning the interaction diagram from the ontological paradigm has been performed as follows:

First, the endpoints (interaction processors) are deployed, already implemented for the proposed approach's semantics, which will attend to messages, execute flows and send messages to other message attendants.

Second, in the modeling/development/deployment phase, each time-line is loaded in the corresponding endpoint. Once loaded, the messages are ready to be processed; Thus, the direct executability.

The modeler's effort is reduced to defining the models and conditions together with the organization of the temporal semantics based on the semantics of the interaction processor (flow execution metamodel)

Direct execution is performed by the interaction graph processor corresponding to each time-line. The interaction processor algorithm is illustrated in Listing 12. In addition, the authors have implemented the algorithm for the web domain using the SPRING BOOT framework and STOMP for messages.

VI. EVALUATING THE PROPOSED APPROACH

"Common Grid Model Exchange Specification (CGMES) is an IEC Technical Specification (TS) based on the IEC CIM family of standards. It was developed to meet necessary requirements for Transmission System Operator (TSO) data exchanges in the areas of system development and system operation" [48]. In the context of SmartGrids, and according to European regulation, TSOS must provide different network models (IGMs) as a forecast to build (merging process) a Common Grid Model (CGM). CGMs are the basic building blocks for different services: coordinated security assess-

```
function attendMessage(BHMessage msg) {
    # The elements following the currently received message until
    # the following message to be received are executed.
    executeNextElements(msg);
}
function executeNextElements(BHMessage msg){
    # The RDF of the message, defined in the interaction structure, is obtained.
    String msgRdf = msg.getMessageContent();
    # From the RDF of the message, it is obtained to which timeline it is addressed, which is a graph.
    String tlineUri = getTimeLineUri(msgRdf);
    # The next element to the current one is obtained, which can be either a message or a process.
    BHNext next = getNextEltIdFromMessg(msg.getMessageId(),tlineUri);
    # While there exist items to deal with, following the current one, it stays in the loop.
    while(next!=null){
        # The type of the next element is obtained.
        String type = next.getType();
        # If it is a process
        if(type == "Process"){
            #The process identifier to be executed is obtained.
            String flowId = next.getEltId();
            # The graph of the flow representing the process is obtained.
            Model m = BhUtils.getGraphModel(flowId);
            # A flow is created.
            Flow flw = new Flow();
            # The object m is assigned to the model attribute of the flw object.
            flw.setModel(m);
            # The process flow graph implementing the concepts of the proposed approach is created.
            BHGraph bhgrf = flw.createBhGraph();
            # The flow is executed.
            bhgrf.executeFlow();
            # The next element to be executed is found.
            next = getNextEltIdFromMessg(next.getMsgId(),next.getTo());
        }

        # If it is a message.
        if(type=="Message"){
            # A message object is created whose identifier is that of the next object
            # and whose content mainly is the message destination.
            BHMessage msgn = new BHMessage();
            msgn.setMessageId(next.getMsgId());
            String msgContent = createRDFMsg(next.getMsgId(),next.getTo());
            msgn.setMessageContent(msgContent);
            # The message is sent through the corresponding transport layer.
            sendMessage(msgn);
            # The next element to the current one is obtained.
            next = getNextEltIdFromMessg(next.getMsgId(),next.getTo());
        }
    }
}
```

LISTING 12. Algorithm for attending the messages.

ment, coordinated capacity calculation, outage planning coordination, and short- and medium-term adequacy. The Individual Grid Model (IGM) consists of all data instances necessary to specify a scenario as input and output for a power flow tool. A Common Grid Model (CGM) is the steady state of a pan-European system for a given point in time and a collection of IGMs building an utterly balanced system with a solved load flow. Merging consists of combining information from multiple IGMs and external constraints into a coherent network model with operating assumptions for a given time. European Network of Transmission System (ENTSO-E) provides CGMES standard specification, which consists of a set of documents. The key documents of interest for this work are: the XSD file for defining the rules model; XML files defining QoCDC Rules for each level which specify how validation reports are produced; Resource Description Framework Schema (RDFS), and OCL files for models corresponding to the different profiles which are described as follows:

- EQ - Network equipment and EQ_BD - Boundary network equipment. EQ is subdivided into Core, Short Circuit, and Operation
- TP - Topology and TP_BD - Boundary topology (power flow buses)
- SV - State Variables (power flow solution)
- SSH - Steady State Hypothesis (power flow input)
- DL - Display Layout
- GL - Geographical Location
- DY - Dynamics data

  An IGM is described mainly by six profiles: EQ, TP, SV, SSH, EQ_BD, and TP_BD.

Basic technical specifications that describe CGMES are:

- IEC 61968-100:2013, Application integration at electric utilities - System interfaces for distribution management - Part 100: Implementation profiles.
- IEC 61970-301:2016 RLV (Red Line Version), Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base.
- IEC 61970-452:2017 (Edition 3.0), Energy management system application program interface (EMS-API) - Part 452: CIM static transmission network model profiles.
- IEC 61970-453:2014 (Edition 2.0) and AMD1:2018 CSV, Energy management system application program interface (EMS-API) - Part 453: Diagram layout profile.
- IEC 61970-501:2006 (Edition 1.0), Energy management system application program interface (EMS-API) - Part 501: Common Information Model Resource Description Framework (CIM RDF) schema.
- IEC 61970-552: 2013 (Edition 1.0), Energy management system application program interface (EMS-API) - Part 552: CIMXML Model exchange format.
- IEC TS 61970-600-1:2017, Energy management system application program interface (EMS-API) - Part 600-1: Common Grid Model Exchange Specification (CGMES) - Structure and rules.
- IEC TS 61970-600-2:2017, Energy management system application program interface (EMS-API) - Part 600-2: Common Grid Model Exchange Specification (CGMES) - Exchange profiles specification.
- IEC 62325-451-1, Framework for energy market communications - Part 451-1: Acknowledgement business process and contextual model for CIM European market.
- IEC 62325-451-5, Framework for energy market communications - Part 451-5: Status request business process and contextual model for CIM European market.

The authors consider that the elements provided by the CIM standard (CGMES) can be used to evaluate the proposed approach. CIM and especially CGMES have created an ontology in the form of RDFS graphs for the structural part of the standard together with a set of rules classified in 8 levels and specified in natural language and OCL. This standard enables TSOs to create models that conform to the RDF schemas and rules. Furthermore, the CGMES standard specifies that the validation process is done in sequence from level 1 to level 8.

Each of the validation levels is described as follows [49]: Level 1 defines metadata in file names and packaging of CIMXML files,

Level 2 defines the structure and syntax of the individual CIM/XML files as well as the metadata header,

Level 3 includes constraints that can be evaluated within the scope of the CIMXML files,

Level 4 describes issues that can be detected during model assembly,

Level 5 includes cross-profile consistency of data,

Level 6 covers diagnostic information that may help solve convergence issues by identifying modeling issues that seem troublesome,

Level 7 describes the coordination of IGMs regarding neighboring TSOs and reference values.

Level 8 covers the convergence behavior of IGMs and CGMs and the plausibility of the CGM.

If a model does not conform to the rules of a certain level, a violation report is returned, and the next validation level is not performed. Therefore, the validation process specified by the CGMES is a flow of validation tasks against the rules of the standard on data models, which are mainly ontologies. Moreover, this approach has provided a solution developed using the W3C standards for validating CIM (CGMES) models, both for the structural and behavioral aspects.

All standards are a set of concepts, relationships, rules, and flows of transformations that can be represented with graphs or ontologies to perform the reasoning task. Thus, any standard from another domain could have been chosen and applied the same approach for validation against the standard's rules.

Any business process can be considered as a flow. Hence, having a flow processor along with a flow modeling language that supports reasoning and, therefore, direct executability. Therefore, the effort is saved in the different phases of the development process (specification of requirements, implementation, testing, and postdelivery tasks). As a result, improvement of transparency (glass box), and scalability, among others, are achieved.

Fig. 21 shows the flow modeling of the validation process of the eight levels of the CGMES standard. Each level's precondition corresponds to the standard's rules for that level and is modeled in SHACL. The input model is the needed model for a given level of validation. The reason contains the validation report for each level.

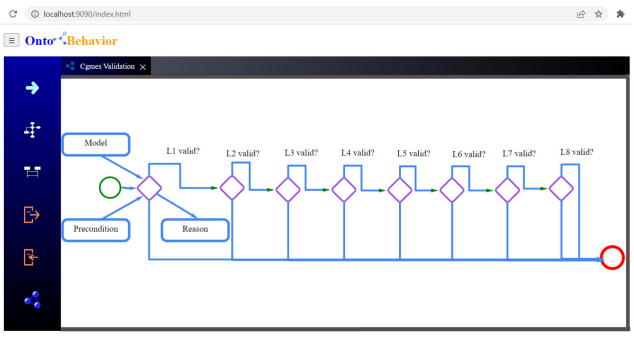


FIGURE 21. Flow for the process of the eight validation levels of the CGMES standard.

The authors have developed a platform whose architecture is shown in Fig. 22. The platform has been developed in Java, specifically as a Spring Boot project using the JENA library for managing RDF graphs and the Fuseki endpoint for storing the RDF graphs that represent the models.

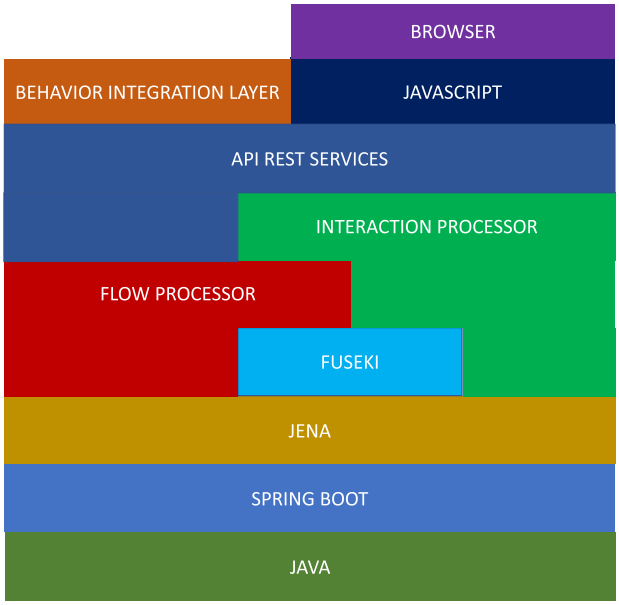


FIGURE 22. The architecture of the flow and interaction processors platform.

The application mainly consists of a flow processor and an interaction processor. The former is a processor of process flows, initially defined by the modeler as an RDF graph. The latter is a processor for communication and interaction between different components that can be hosted on different machines. For using these two processors, API REST services have been implemented for loading and executing flow and interaction graphs. In addition, Javascript libraries have been developed to create, execute, and monitor the execution of these diagrams from a web browser. The platform integration layer can be built on top of the API REST services that expose the flow and interaction processors.

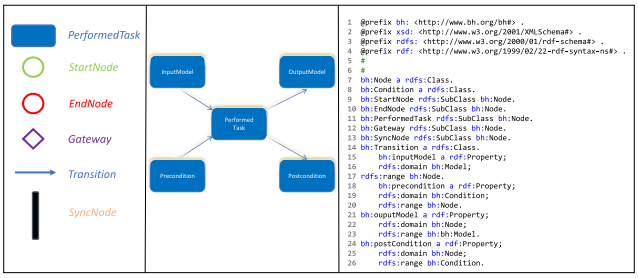


FIGURE 23. Visual constructs, core concept and RDF language model.

Fig. 23 illustrates the proposed visual language within this framework. The foundational elements for defining a flow are positioned on the left side of the diagram. These include a blue rectangle representing PerformedTask, a green circle denoting startNode, a red circle indicating endNode, a diamond symbolizing the decision-making Gateway, an arrow indicating the transition, and a vertical bar signifying syncNode. In the central portion, the fundamental concept of the framework is elucidated. The RDF SCHEMA

definition of the proposed language for specifying a flow is presented on the right side of the diagram. This approach enables direct executability of the model once it has been established using this language.

VII. LIMITATIONS

The authors have considered the following limitations: (i) The CGMES version on which the approach was evaluated is 2.4.15. (ii) Only semantic web technologies for ontological modeling have been used. (iii) The modeling language of the rules for the preconditions and postconditions has been defined in SHACL only. (iv) Other constructs can be added to the approach, such as the concept of a loop in the interaction diagram. (v) It has only been limited to mapping the Object-Oriented paradigm (UML) for mapping fundamental behavioral concepts to the proposed approach, such as activity diagram, state machine, and interaction diagram.

VIII. CONCLUSION AND FUTURE WORKS

In the software engineering process with ontologies for behavior specification, an approach has been provided with its corresponding visual language for requirements specification, design, implementation, and testing-our approach considers a process defined as an evolutionary modeling process. The testing phase can be treated as a validation process, viewed as a model against the population of test instances. The behavioral modeling paradigm consists of tasks performed when their conditions are accomplished against their input model, considering a model as a set of concepts and relationships related to the task. The two aspects of behavior that have been addressed and implemented are flow and interaction. A transformation has been realized from the fundamental UML behavioral diagrams to the proposed approach, including the activity diagram, the interaction diagram, and the state machine diagram. In this work, only the most common constructs for behavior have been addressed; For the temporal logic of behavior, no constructs have been provided with their corresponding visual syntax in RDF and its semantics. However, even in the absence of these constructs, modeling the behavior's temporal logic can be addressed with our approach using the input model and the precondition. In case a new construct is needed to cover a specific use case in the future, only it is necessary giving a name (identifier or symbol) to the construct and its semantics whose implementation will be added to the behavior graph processor. Temporal diagram, process mining with semantics, process improvement, and its application in the security domain are proposed as future works. The evaluation of the approach has been conducted in the domain of smart grids; for modeling and achieving direct execution of the CGMES validation process.

REFERENCES

[1] A. Sommer. (2017). *A Python Validator for SHACL*. Accessed: Mar. 10, 2024. [Online]. Available: https://github.com/RDFLib/pySHACL

[2] T. A. S. Foundation. (2017). *Apache Jena SHACL*. Accessed: Mar. 10, 2024. [Online]. Available: https://jena.apache.org/documentation/shacl/

[3] H. Knublauch. (2017). *SHACL API in Java Based on Apache Jena*. Accessed: Mar. 10, 2024. [Online]. Available: https://github.com/TopQuadrant/shacl

[4] (2023). *Pure Python Package for Working With RDF*. Accessed: Mar. 10, 2024. [Online]. Available: https://github.com/RDFLib/rdflib

[5] (2016). *A Free, Open-Source Ontology Editor and Framework for Building Intelligent Systems*. Accessed: Mar. 10, 2024. [Online]. Available: https://protege.stanford.edu/

[6] (2009). *Allegrograph Rdfstore Web 3.0's Database*. Accessed: Mar. 10, 2024. [Online]. Available: http://www.franz.com/agraph/allegrograph

[7] (2009). *Enterprise Knowledge Graph Platform*. Accessed: Mar. 10, 2024. [Online]. Available: https://www.stardog.com/

[8] E. Foundation. (2015). *Open-Source Framework for Storing, Querying, and Analysing RDF Data*. Accessed: Mar. 10, 2024. [Online]. Available: https://rdf4j.org/

[9] (2008). *Sparql Protocol and RDF Query Language*. Accessed: Mar. 10, 2024. [Online]. Available: https://www.w3.org/TR/sparql11-query/

[10] (2017). *Unified Modeling Language*. Accessed: Mar. 10, 2024. [Online]. Available: https://www.omg.org/spec/UML/

[11] M. Fahad, N. Moalla, and Y. Ourzout, "Dynamic execution of a Bus. Process via web service selection and orchestration," *Proc. Comput. Sci.*, vol. 51, pp. 1655–1664, Aug. 2015.

[12] M. W. Iqbal, N. A. Ch, S. K. Shahzad, M. R. Naqvi, B. A. Khan, and Z. Ali, "User context ontology for adaptive mobile-phone interfaces," *IEEE Access*, vol. 9, pp. 96751–96762, 2021.

[13] M. Elsayed, N. Elkashef, and Y. F. Hassan, "Mapping UML sequence diagram into the web ontology language OWL," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 5, pp. 1–17, 2020.

[14] S. Poslad, S. E. Middleton, F. Chaves, R. Tao, O. Necmioglu, and U. Bügel, "A semantic IoT early warning system for natural environment crisis management," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 2, pp. 246–257, Jun. 2015.

[15] A. T. Elve and H. A. Preisig, "From ontology to executable program code," *Comput. Chem. Eng.*, vol. 122, pp. 383–394, Mar. 2019.

[16] J. Fabra, M. J. Ibáñez, P. álvarez, and J. Ezpeleta, "Behavioral analysis of scientific workflows with semantic information," *IEEE Access*, vol. 6, pp. 66030–66046, 2018.

[17] B. Teixeira, G. Santos, T. Pinto, Z. Vale, and J. M. Corchado, "Application ontology for multi-agent and web-services' co-simulation in power and energy systems," *IEEE Access*, vol. 8, pp. 81129–81141, 2020.

[18] M. H. Mughal, Z. A. Shaikh, A. I. Wagan, Z. H. Khand, and S. Hassan, "ORFFM: An ontology-based semantic model of river flow and flood mitigation," *IEEE Access*, vol. 9, pp. 44003–44031, 2021.

[19] M. Driss, A. Aljehani, W. Boulila, H. Ghandorh, and M. Al-Sarem, "Servicing your requirements: An FCA and RCA-driven approach for semantic web services composition," *IEEE Access*, vol. 8, pp. 59326–59339, 2020.

[20] Y. Pradeep, S. A. Khaparde, and R. K. Joshi, "High level event ontology for multiarea power system," *IEEE Trans. Smart Grid*, vol. 3, no. 1, pp. 193–202, Mar. 2012.

[21] C.-W. Yang, V. Dubinin, and V. Vyatkin, "Ontology driven approach to generate distributed automation control from substation automation design," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 668–679, Apr. 2017.

[22] R. Stojanov, S. Gramatikov, I. Mishkovski, and D. Trajanov, "Linked data authorization platform," *IEEE Access*, vol. 6, pp. 1189–1213, 2018.

[23] N. F. Saraiva De Sousa, D. A. Lachos Perez, R. V. Rosa, M. A. S. Santos, and C. Esteve Rothenberg, "Network service orchestration: A survey," *Comput. Commun.*, vols. 142–143, pp. 69–94, Jun. 2019.

[24] S. Isotani, I. Ibert Bittencourt, E. Francine Barbosa, D. Dermeval, and R. Oscar Araujo Paiva, "Ontology driven software engineering: A review of challenges and opportunities," *IEEE Latin Amer. Trans.*, vol. 13, no. 3, pp. 863–869, Mar. 2015.

[25] A. Meidan, J. A. García-García, M. J. Escalona, and I. Ramos, "A survey on business processes management suites," *Comput. Standards Interface*, vol. 51, pp. 71–86, Mar. 2017.

[26] M. Rashid, M. W. Anwar, and A. M. Khan, "Toward the tools selection in model based system engineering for embedded systems—A systematic literature review," *J. Syst. Softw.*, vol. 106, pp. 150–163, Aug. 2015.

[27] M. Mejhed Mkhinini, O. Labbani-Narsis, and C. Nicolle, "Combining UML and ontology: An exploratory survey," *Comput. Sci. Rev.*, vol. 35, Feb. 2020, Art. no. 100223.
[28] I. Zafar, F. Azam, M. W. Anwar, B. Maqbool, W. H. Butt, and A. Nazir, "A novel framework to automatically generate executable web services from BPMN models," *IEEE Access*, vol. 7, pp. 93653–93677, 2019.
[29] P. Valderas, V. Torres, and V. Pelechano, "A microservice composition approach based on the choreography of BPMN fragments," *Inf. Softw. Technol.*, vol. 127, Nov. 2020, Art. no. 106370.
[30] S. Appel, P. Kleber, S. Frischbier, T. Freudenreich, and A. Buchmann, "Modeling and execution of event stream processing in business processes," *Inf. Syst.*, vol. 46, pp. 140–156, Dec. 2014.
[31] F. Martins and D. Domingos, "Modelling IoT behaviour within BPMN Bus. Processes," *Proc. Comput. Sci.*, vol. 121, pp. 1014–1022, May 2017.
[32] E. Schäffer, V. Stiehl, P. K. Schwab, A. Mayr, J. Lierhammer, and J. Franke, "Process-driven approach within the engineering domain by combining Bus. Process model and notation (BPMN) with process engines," *Proc. CIRP*, vol. 96, pp. 207–212, Apr. 2021.
[33] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," *IEEE Access*, vol. 8, pp. 104407–104431, 2020.
[34] M. W. Anwar, M. Rashid, F. Azam, M. Kashif, and W. H. Butt, "A model-driven framework for design and verification of embedded systems through SystemVerilog," *Des. Autom. Embedded Syst.*, vol. 23, nos. 3–4, pp. 179–223, Nov. 2019.
[35] N. Silega, M. Noguera, and D. Macias, "Ontology-based transformation from CIM to PIM," *IEEE Latin Amer. Trans.*, vol. 14, no. 9, pp. 4156–4165, Sep. 2016.
[36] I. Lazăr, S. Motogna, and B. Pârv, "Behaviour-driven development of foundational UML components," *Electron. Notes Theor. Comput. Sci.*, vol. 264, no. 1, pp. 91–105, Aug. 2010.
[37] J. A. García-García, J. G. Enríquez, and F. J. Domínguez-Mayo, "Characterizing and evaluating the quality of software process modeling language: Comparison of ten representative model-based languages," *Comput. Standards Interface*, vol. 63, pp. 52–66, Mar. 2019.
[38] A. Pedro, S. Baik, J. Jo, D. Lee, R. Hussain, and C. Park, "A linked data and ontology-based framework for enhanced sharing of safety training materials in the construction industry," *IEEE Access*, vol. 11, pp. 105410–105426, 2023.
[39] A. Pinto, Y. Cardinale, I. Dongo, and R. Ticona-Herrera, "An ontology for modeling cultural heritage knowledge in urban tourism," *IEEE Access*, vol. 10, pp. 61820–61842, 2022.
[40] W. Lin, S. Yuchen, F. Haixiang, B. Huihui, and S. Liang, "Design and implementation of IT health diagnosis system based on knowledge graph and artificial intelligence technology," in *Proc. IEEE 3rd Int. Conf. Electron. Technol., Commun. Inf. (ICETCI)*, May 2023, pp. 1232–1236.
[41] L. Asprino and V. Presutti, "Observing LOD: Its knowledge domains and the varying behavior of ontologies across them," *IEEE Access*, vol. 11, pp. 21127–21143, 2023.
[42] E. Adel, S. El-Sappagh, S. Barakat, K. S. Kwak, and M. Elmogy, "Semantic architecture for interoperability in distributed healthcare systems," *IEEE Access*, vol. 10, pp. 126161–126179, 2022.
[43] F. Lettich, C. Pugliese, C. Renso, and F. Pinelli, "Semantic enrichment of mobility data: A comprehensive methodology and the MAT-builder system," *IEEE Access*, vol. 11, pp. 90857–90875, 2023.
[44] Z. Brahmia, F. Grandi, and R. Bouaziz, "TSQWRL: A TSQL2-like query language for temporal ontologies generated from JSON big data," *Big Data Mining Analytics*, vol. 6, no. 3, pp. 288–300, Sep. 2023.
[45] Z. Brahmia, F. Grandi, and R. Bouaziz, "τJOWL: A systematic approach to build and evolve a temporal OWL 2 ontology based on temporal JSON big data," *Big Data Mining Analytics*, vol. 5, no. 4, pp. 271–281, Dec. 2022.
[46] M. Sopek, D. Tomaszuk, S. Glab, F. Turobos, I. Zielinski, D. Kuzinski, R. Olejnik, P. Luniewski, and P. Gradzki, "Technological foundations of ontological ecosystems on the 3rd generation blockchains," *IEEE Access*, vol. 10, pp. 12487–12502, 2022.
[47] J. Erbel and J. Grabowski, "Scientific workflow execution in the cloud using a dynamic runtime model," *Softw. Syst. Model.*, vol. 23, no. 1, pp. 163–193, Jun. 2023.
[48] ENTSO-E. (2019). *Common Information Model*. Accessed: Mar. 10, 2024. [Online]. Available: https://www.entsoe.eu/digital/common-information-model/
[49] ENTSO-E. (2022). *Quality of Cgmes Datasets and Calculations for System Operation. Version 3.3. Approved by Opde Tf*. Accessed: Mar. 10, 2024. [Online]. Available: https://eepublicdownloads.azureedge.net/clean-documents/digital/QualityOfCGMESdatasetsAndCalculations_v3_3.pdf

MOHAMED LARHRIB received the M.S. degree in computer engineering and the Ph.D. degree from Universidad Nacional de Educación a Distancia, Spain, in 2015 and 2023, respectively. His research interests include software engineering, power systems data exchange formats, semantic web, common information model, and model driven engineering.

MIGUEL ESCRIBANO received the M.S. degree in industrial engineering and the Ph.D. degree from the Polytechnic University of Madrid, Spain, in 1999 and 2005, respectively. Since 2005, he has been an Assistant Professor with the Financial Economics, Statistics and Operations Research and Actuarial Science, Universidad Complutense de Madrid, Spain. Since 2008, he has been a Power Engineer with Red Eléctrica de España. His research interests include software engineering, power systems data exchange formats, semantic web, and common information model.

CARLOS CERRADA received the M.S. degree in industrial engineering and the Ph.D. degree from the Polytechnic University of Madrid, Spain, in 1983 and 1987, respectively. He is currently a Full Professor with the Systems and Software Engineering Department, Universidad Nacional de Educación a Distancia, Spain. He was a Fulbright Scholar with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, from 1989 to 1990. His research interests include software engineering, robotics, pattern recognition, 3D object representation, and ubiquitous computing. He is a member of the IFAC.

JUAN JOSE ESCRIBANO received the M.S. degree in industrial engineering from the Polytechnic University of Madrid, Spain, in 1995, and the Ph.D. degree from Universidad Nacional de Educación a Distancia, Spain, in 2003. Since 2001, he has been an Assistant Professor with the Systems and Software Engineering Department, Universidad Nacional de Educación a Distancia. His research interests include software engineering, semantic web, and common information model.