Received April 5, 2020, accepted April 24, 2020, date of publication April 28, 2020, date of current version May 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991010

Application Ontology for Multi-Agent and Web-Services' Co-Simulation in Power and Energy Systems

BRÍGIDA TEIXEIRA[1], GABRIEL SANTOS[1], TIAGO PINTO[1], (Member, IEEE), ZITA VALE[2], (Senior Member, IEEE), AND JUAN M. CORCHADO[3], (Member, IEEE)

[1]Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD), Institute of Engineering, Polytechnic Institute of Porto, 4249-015 Porto, Portugal
[2]Polytechnic Institute of Porto, 4249-015 Porto, Portugal
[3]Bioinformatics, Intelligent Systems, and Educational Technology (BISITE) Research Group, University of Salamanca, 37007 Salamanca, Spain

Corresponding author: Gabriel Santos (gajls@isep.ipp.pt)

This work was supported by the MAS-Society Project co-funded by Portugal 2020 Fundo Europeu de Desenvolvimento Regional (FEDER) through PO CI, and under Grant UIDB/00760/2020. Gabriel Santos was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) PhD studentship with reference SFRH/BD/118487/2016.

ABSTRACT Power and energy systems are very complex, and several tools are available to assist operators in their planning and operation. However, these tools do not allow a sensitive analysis of the impact of the interaction between the different sub-domains and, consequently, in obtaining more realistic and reliable results. One of the key challenges in this area is the development of decision support tools to address the problem as a whole. Tools Control Center – TOOCC – proposed and developed by the authors, enables the co-simulation of heterogeneous systems to study the electricity markets, the operation of the smart grids, and the energy management of the final consumer, among others. To this end, it uses an application ontology that supports the definition of scenarios and results comparison, while easing the interoperability among the several systems. This paper presents the application ontology developed. The paper addresses the methodology used for its development, its purpose and requirements, and its concepts, relations, facets and instances. The ontology application is illustrated through a case study, where different requirements are tested and demonstrated. It is concluded that the proposed application ontology accomplishes its goals, as it is suitable to represent the required knowledge to support the interoperability among the different considered systems.

INDEX TERMS Application ontology, co-simulation, multi-agent systems, power and energy systems, semantic interoperability, web-services.

I. INTRODUCTION

Considering the climatic urgency that society is facing in recent years, the European Commission (EC) has defined a set of targets to be achieved by 2020, known as 20-20-20 targets [1], [2]. These targets are: i) a 20% reduction in CO_2 emissions compared to 1990 levels; ii) a 20% increase in energy efficiency; and iii) increase the use of Renewable Energy Sources (RES) to represent 20% of energy production in the European Union. As a result, the EC intends to achieve a significant change in the energy sector, by implementing new legislation to increase the inclusion of RES and make their use more intelligent and sustainable. The evolution of Power and Energy Systems (PES) to support the intermittent nature of RES raises new challenges [3], [4]. It is crucial to reduce the inherent risk in the intermittency and unpredictability of the use of RES, to lower prices for production and installation of renewable generation technology, to adapt the existing physical infrastructure, and to adopt new regulatory measures, among others. Electricity Markets (EM) have also to be adapted to the different segments of PES (e.g. generation, transmission, distribution, and commercialization),

The associate editor coordinating the review of this manuscript and approving it for publication was Chaitanya U. Kshirsagar.

to the new policies and needs of RES penetration, by conceiving and implementing new market models, changing the market operation rules, and creating new legislation [5], [6].

In this context, the use of simulation tools developed to analyze and study the PES domain is indispensable, since they allow the participating entities to deal with its unpredictability and complexity [7], [8]. Simulators based on multi-agent technology have particularities that make them suitable tools for the study of PES, mainly due to their distributed nature [9], [10]. These tools make it easier to model the various systems and entities, as well as their constraints, making the model closer to reality, while decomposing the problem into less complex modules. Although there are several tools in the literature for the study of PES, most of them only solve problems of a specific PES sub-domain. Therefore, by using those tools individually, it is not possible to simulate and study the energy sector with realism and precision, as the sub-domains have a great interdependence that strongly impacts the results [11].

One possible solution to take advantage of existing and well-established PES simulation tools, is to make them interoperable through middleware that enables the co-simulation of heterogeneous tools [12]–[16]. A set of interoperable tools provides results with higher reliability and realism, besides of a better understanding of wider implications, restrictions and influences [17]. It is possible to find in the literature a few solutions for the cooperation of simulation tools in PES, namely the Electric Power and Communication Synchronizing Simulator (EPOCHS) [12], the Global Event-driven Co-simulation (GECO) [13], Mosaik [14], and Tools Control Center (TOOCC) [15], [16], conceived and developed by the authors of the current paper.

These tools use different approaches to achieve interoperability between heterogeneous systems. From these, TOOCC is the only tool that takes advantage of semantic web technologies for the interoperability with external systems. Ontologies give semantic meaning to the data exchanged between heterogeneous parties, promoting their interoperability [18]. The motivation for the development of ontologies as a means to provide interoperability between heterogeneous Multi-Agent Systems (MAS) in the scope of PES is addressed in [19], where the inclusion of external systems that may arise in the future is also considered.

This paper presents TOOCC's application ontology for MAS and web-services co-simulation in PES. TOOCC's semantic model describes the scenarios' configuration while easing the interoperability between the different simulation tools and enables the subsequent comparison of results, thereby overcoming the identified limitations in the field.

The following section gives a background on the different co-simulation tools found in literature, describing their limitations, and explaining how TOOCC overcomes those limitations. Section III presents an overview of TOOCC, detailing its architecture, the multi-agent model, and explaining why semantic interoperability has been chosen. Section IV introduces TOOCC's ontology, describing its concepts, properties, and purposes. Section V demonstrates the usefulness of TOOCC's semantic model through a case study where the ontology evaluation is also carried out. Finally, section VI presents the conclusions of this work.

II. BACKGROUND

Few relevant tools have emerged in the literature to provide interoperability among already existing PES well-established simulators. Examples of such tools are EPOCHS [12], GECO [13], Mosaik [14], and TOOCC [15], [16].

EPOCHS [12] is a multi-agent platform created to simulate PES components together with the communication network, to study the grid with the aim to prevent blackouts. It essentially combines three simulators: i) the Power Systems Computer-Aided Design/Electromagnetic Transients including Direct Current (PSCAD/EMTDC) [20]; ii) the Positive Sequence Load Flow (PSLF) [21]; and iii) the Network Simulator 2 (NS2) [22]. An entity called Runtime Infrastructure (RTI) performs the interface between all components, ensuring the synchronization in the messages' exchange. To interconnect these simulators, EPOCHS uses an application programming interface (API) encapsulated by the RTI.

GECO [13] has similar characteristics to EPOCHS, as it also integrates NS2 and PSLF simulators. Its purpose is to validate monitoring schemes, control, and grid protection. The communication between GECO and the simulation tools NS2 and PSLF is made through both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The messages exchanged include information about power data and control commands.

Mosaik [14] provides interoperability among heterogeneous applications through two components: the simulation interface (SIM API) and the Master Control Program (MCP). The SIM API enables the communication between Mosaik and external simulation tools, while the MCP manages the scenarios' composition and the tools' execution. To integrate a new tool with Mosaik, it is necessary to proceed with the implementation of a default interface to guarantee the proper model configuration and the scheduling of the tasks' execution.

Although these three solutions allow interoperability with external tools, only Mosaik provides a way to integrate any tool, while EPOCHS and GECO are restricted to the simulators they use. Furthermore, whenever it is intended to run a different scenario, their configuration, data preparation, and execution's schedule definition, are complex tasks since there is a need to write code. Ideally, the execution of alternative scenarios should be possible with a simple system reconfiguration, without the need of reprogramming it. It is possible to realize that these tools were not designed having in mind a smooth scenarios' definition, nor a simplified results analysis.

On the other hand, semantic-based approaches are particularly suitable for solving interoperability issues [23]. Semantic models establish a common vocabulary so that applications can interact and communicate, regardless of the communication mechanisms [24]. Furthermore, it is possible to compose more sophisticated solutions by reusing already existing robust applications and merging different domains, without interfering with their capabilities [25]. Besides, these models combined with semantic reasoners allow to perform more intelligent tasks such as complex queries, the application of rules, and to infer new knowledge.

TOOCC [15], [16] has been conceived to overcome the previously identified limitations in the co-simulation domain. TOOCC has been proposed by the authors as a multi-agent tool capable of creating, simulating, and analyzing scenarios covering different PES domains through the interoperability between heterogeneous simulation tools developed in the GECAD research center. TOOCC takes advantage of ontologies to be able to interoperate with different systems. On one side, domain ontologies are used to describe the knowledge exchanged between the external systems. On the other, the formalization of the scenarios and respective simulations' configuration is achieved by TOOCC's application ontology that enables the definition of the external tools to be used, their input and output models, their input data, order of execution, and how the results shall be analyzed. The following section overviews the TOOCC tool, describes its architecture and multi-agent model, as well as the options made on semantic interoperability.

III. TOOCC OVERVIEW

Tools Control Center (TOOCC) [15], [16] is a co-simulation solution that acts as a facilitator between heterogeneous tools, enabling them to share vocabularies and concepts, and thus collaborate in the simulation of PES scenarios. It allows us to simulate complex scenarios that result from merging the individual capabilities of each embedded tool, considering different PES domains in the same scenario, making the simulation more realistic and precise.

TOOCC can be seen as a decision support system, as it provides the user with the means to analyze different problems with different particularities. It can be adopted to study PES from the perspective of various entities, such as system operators, market operators, grid operators, aggregators, prosumers, producers, consumers, among others. There are several domains where TOOCC is being used, namely: electricity markets, to study the impact of the inclusion of RES, buildings energy management, demand response (DR), tariffs application, among others. The simulation for specific time horizons (e.g.: real-time, hour-ahead, day-ahead) is also considered, as well as the analysis and results comparison of alternative scenarios. TOOCC has been designed having in mind the reduction of the complexity in the definition of simulation scenarios.

A. ARCHITECTURE

The need to establish interoperability between heterogeneous systems is one of TOOCC's key goals. TOOCC's architecture enables the communication with external systems regardless of the programming languages they have been developed. It also supports scalability and distribution of agent-based systems, considering the processing capacity of the machines available for the simulation. Figure 1 illustrates the core modules of TOOCC's architecture.

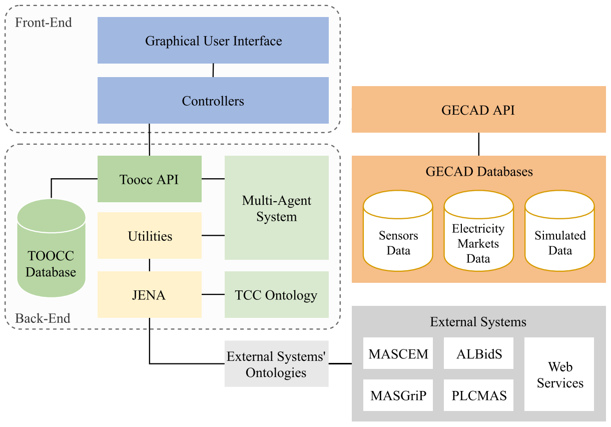

FIGURE 1. **TOOCC's architecture.**

Analyzing Figure 1, it is visible that TOOCC's architecture is based on two main modules: the *Front-End*, for the user's interaction, and the *Back-End*, where the processing occurs. The interaction between the *Front-End* and the *Back-End* is established through the communication between the *Controllers* and the *Toocc API* sub-modules. The *Multi-Agent System* sub-module is responsible for managing the interoperability with the external systems, as well as for scheduling simulation scenarios defined by the user. Additionally, TOOCC uses auxiliary libraries to communicate, interpret, and transform the knowledge exchanged among the various integrated systems.

B. MULTI-AGENT MODEL

TOOCC is developed in Java using the Java Agent DEvelopment framework (JADE) [26], [27], which is compliant with the Foundation for Intelligent Physical Agents (FIPA) [28] standards.

FIPA promotes the *Agent Communication Language* (ACL) as the standard for communication between agent-based systems. It is also possible to add meaning to the messages exchanged through the use of ontologies, which allow the definition of syntax and semantics to their content. Consequently, agents are able to communicate meaningfully since they can interpret each message correctly.

Figure 2 presents TOOCC's multi-agent model.

This model considers six types of agents that provide a conceptual perspective on the execution of a scenario, namely:

- *TOOCC API Agent* (ApiA): is the agent responsible for bridging the *Toocc API* with the *Multi-Agent System* sub-module. It asks the main agent to run the simulation and waits for the results.
- *TOOCC Main Agent* (TMA): is the agent responsible for initiating and coordinating the entire simulation at a high level. When the simulation starts, it triggers the creation

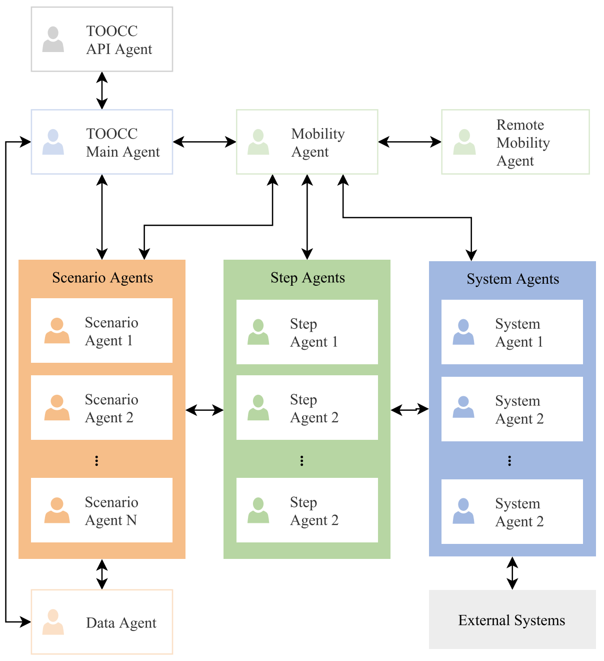

FIGURE 2. **TOOCC's multi-agent model.**

of a *Scenario Agent* for each configured scenario. It also concludes the simulation after the execution of all *Scenario Agents*.

- *Scenario Agent* (ScenA): is responsible for coordinating the execution of a specific scenario. The execution phases are managed by creating *Step Agents* and ensuring that the scenario runs entirely.
- *Step Agent* (StepA): its function is to manage communications with external systems in parallel. That is, it will create *System Agents* that will simultaneously communicate with their respective external systems, within the same execution phase, allowing agility in obtaining results.
- *Service Agent* (ServA): establishes direct communication with external systems, being aware of the semantics used in the communication. As soon as it receives the results, it notifies its creator (*Step Agent*).
- *Mobility Agent* (MobiA): has the responsibility to distribute the several agents to the available machines. The decision is made according to the operating system and software installed on each machine.
- *Remote Mobility Agent* (RMobA): it is hosted on the available domain machines to which the agents can be moved and communicates with the *Mobility Agent* sending the information it needs to decide where to move the simulation agents.
- *Data Agent* (DataA): performs centralized management of simulation data, ensuring that the results are stored correctly, and that the status of each simulation point is updated, ensuring complete system's execution.

In addition to the agents presented, there are also others native to JADE. These are the *Directory Facilitator* agent as a yellow page directory for service delivery, the *Agent Management System* agent for the management and control of agents and platform, and the *Remote Agent Management* which provides a graphical interface for managing and viewing agent status and communications.

C. SEMANTIC INTEROPERABILITY

Establishing interoperability between heterogeneous systems is a complex task. It is not just the exchange of data messages between systems, but the exchange of knowledge. This knowledge can be about the domain, about data, or about features that can be made available and shared.

In computer science, ontologies describe vocabularies that can model domains shared between heterogeneous entities [29]. The inclusion of semantics in the messages exchanged allows an unambiguous conceptualization about the knowledge shared by both parties, making the communication more effective by removing misunderstandings. Additionally, other advantages result from the use of semantic models, such as computational inference and knowledge reuse [30]. They can be used to develop systems decoupled from the data model, with a high level of abstraction and flexibility that eases the evolution of the system, such as in [31]; as well as to validate the system's knowledge or apply rules by using, for example, the Semantic Web Rule Language (SWRL) [32].

Using these techniques, it is possible to strategically use the individual capabilities of each external tool integrated with TOOCC, allowing the study of scenarios addressing several PES sub-domains. At the same time, it allowed to develop TOOCC with the ability of integrating any external tool without the need to be reprogrammed, nor to extend any of TOOCC's class, nor to implement any interface in the system to be integrated.

TOOCC's semantic model is described in Section IV, where the engineering process is detailed, namely its purpose, goals, requirements, implementation, evaluation, and evolution.

IV. TOOCC's APPLICATION ONTOLOGY

TOOCC makes use of semantics to ensure the co-simulation between heterogeneous simulation tools. These simulation tools may be MAS or web services available in the authors' research center laboratory. Each system that interoperates with TOOCC has its knowledge model, which can be semantic or syntactic, and must be considered to ensure the systems' co-simulation. This section presents TOOCC's application ontology (TCC), the methodology used in the engineering process, and the options made to fulfill our requirements. There are several methodologies for the development of ontologies that specify the methods, principles, and rules to follow during the engineering process [33]. These processes support the specification, conceptualization, formalization, implementation, and maintenance of an ontology, which result in its life cycle. TCC was developed based on the 101 development methodology [34]. This approach is characterized by its simplified view regarding the ontology development. It is based on the premise that the development of an ontology is an iterative process, where the ontology is continuously refined to the needs of its users. This method considers that there are several possible approaches for the representation of a domain, where the concepts and their relations must be clearly stated through the specification of subjects and predicates.

The 101 methodology is based on the following steps:

- Scope and domain identification;
- Reuse of ontologies;
- Enumeration of important terms;
- Classes definition;
- Properties definition;
- Properties facets definition;
- Creation of instances.

TCC was written using the open-source application Protégé [35]. It has Ontology Web Language with Logic Description (OWL-DL) [36] syntax, being represented in the Resource Description Framework (RDF) Turtle language [37]. Being an application ontology, TCC is embedded in the TOOCC's application as a resource file. The following subsections specify TCC's development process.

A. DOMAIN AND SCOPE

TCC's purpose is to configure TOOCC's simulations, facilitate the process of interoperability between external tools ran by TOOCC, and enable the results' comparison, when applicable, at the end of the simulation. To this end, TCC must be able to: i) describe TOOCC's configuration model (i.e., the model that describes the simulations); ii) include the input and output data models of external systems (semantic or syntactic); iii) reuse the output model of a system (or part of it) to get the necessary data for the input of the next system to execute; and iv) to take advantage of the output models to perform the automatic results' comparison, whenever it makes sense.

Since the input and output models of the systems to be integrated can be semantic or syntactic, TCC must be agnostic regarding other data models. TCC must be able to use other ontologies whenever required, but it does not need to import them. However, these models must be publicly available or shared by the external tools.

Thus, the following functional requirements have been defined to fulfill all the above-mentioned objectives:

- The model must allow the configuration of a set of simulations to be run simultaneously;
- The model should allow configuring different scenarios in each simulation (to be executed simultaneously too, where, after execution, the results can be compared);
- A scenario consists of one or more steps;
- A step can include one or more external systems that provide services (the step ensures that all services in it are performed before proceeding to the next step. Services in the next step are waiting for the results of the previous step);
- Each service (provided by a given system) includes input and output data models, as well as the input data source;
- The input data source can be a local file, a web resource, or a database;
- The model must allow the automatic results' comparison at different levels, namely: simulation level, scenario level, or service level.

Regarding the non-functional requirements, the following have been determined:

- Accuracy - determines if the knowledge asserted in the ontology is according with the domain expert's knowledge;
- Clarity - validates if the ontology communicates effectively the intended meaning of the defined terms;
- Cohesion - refers to the ontology's relatedness of elements, i.e., if the defined classes are strongly related;
- Completeness - checks if the ontology can answer all the questions;
- Computational efficiency - relates to how fast tools (like reasoners) can work with the ontology;
- Conciseness - reflects if the ontology defines irrelevant or redundant elements regarding the domain;
- Consistency - ensures that the ontology does not include nor allow contradictions;
- Coverage - how well the ontology represents the domain model.

The functional and non-functional requirements help to efficiently identify the knowledge that the ontology must define. Additionally, they also offer a baseline for the validation and verification of the developed model.

B. REUSING ONTOLOGIES

One of the first phases to be considered in the ontology development process concerns the reuse of other existing ontologies. Although there are several ontologies publicly available in [38] specifying concepts that could be useful to the configuration of TOOCC's simulations, these semantic models were designed for different purposes in different domains and contexts. Thus, they include decontextualized vocabulary regarding TOOCC's configuration purpose. For this reason, TCC does not import any ontology. On the other hand, importing those inappropriate models could lead to inconsistencies or ambiguity, since a concept may have distinct meanings depending on the context or domain.

However, it is important to keep in mind that TCC must be able to work with the data models (semantic or syntactic) of the external systems with which it operates. Regarding the semantic models of external systems, one option could be to import them into TCC. However, it implies that whenever a system is included in TOOCC, TCC ontology also has to be redesigned to ensure there are no inconsistencies nor ambiguity, which is very time costly.

Instead, it is intended that TCC establishes a relationship with the input/output models of external services, being

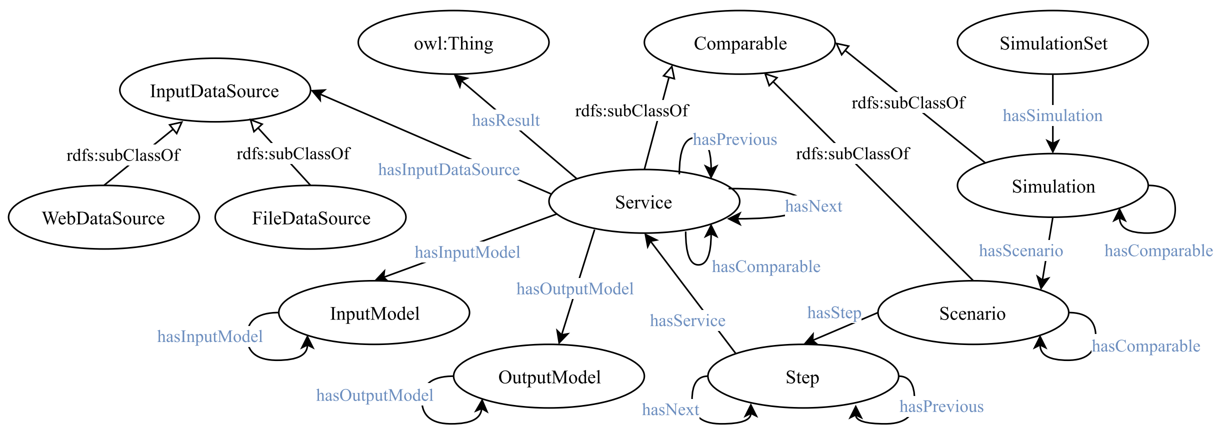

FIGURE 3. TCC's classes and relations.

desegregated from the domain concepts of each system, unnecessary to the configuration of simulations in TOOCC. As a result of this approach, TOOCC's semantic model will only know the necessary vocabulary to perform its tasks. In this way, well established PES ontologies such as SEAS [39], SAREF [40], EMO [41], DABGEO [42], among others, can be used by tools that interoperate within TOOCC, without the need to be imported by TCC.

C. ENUMERATION OF IMPORTANT TERMS

Another important step in the development of an ontology is the enumeration of important terms that must be represented as concepts. These concepts are introduced in the ontology as a class hierarchy.

Considering TCC's domain and scope, these are the terms that describe the sufficient and necessary conditions to meet the above-mentioned requirements:

- **Simulation Set** - it is the root element of the configuration, where several simulations can be configured to run simultaneously;
- **Simulation** - this term describes a user defined simulation;
- **Scenario** - to define a simulation scenario, since a user may want to run simultaneously different scenarios and compare their results at the end;
- **Step** - identifies the execution phase of a scenario, helping the system to realize which services can run concurrently;
- **Service** - describes an external service (agent-based or web service) used at a phase of a scenario;
- **Input Model** - defines the input model (semantic or syntactic) of a service;
- **Output Model** - defines the output model (semantic or syntactic) of a service;
- **Input Data Source** - identifies the service's data source, which can be a local file or web-based;
- **File Data Source** - describes a service's file data source;
- **Web Data Source** - describes a service's web-based data source;
- **Comparable** - identifies classes that can be compared between instances of themselves.

The following subsection details the definitions of classes and their sub-classes, as well as their properties, facets, and instances.

D. CLASSES, PROPERTIES, FACETS AND INSTANCES

TCC's concepts were created using the middle-out approach, which starts from the most fundamental terms in the domain before moving on to more abstract and more specific terms. According to [43], it makes it easier to relate terms more precisely while it is also likely to reduce rework.

In the previous subsection, a list of terms and respective descriptions were defined to assist in the development of the classes and properties of the ontology. Figure 3 introduces TCC's class hierarchy and the relations between them. Each class (in italic) is then described, including the relationships (in blue) between them.

The *Comparable* class allows the abstraction of classes that are intended to be comparable between instances of themselves, namely the *Simulation*, the *Scenario*, and the *Service* classes. It is an abstract class that is not supposed to be instantiated by itself. Instead, one must instantiate its sub-classes, ensuring that they are only comparable to other instances of the same class.

The *SimulationSet* class is the root concept in TCC's model. It gathers the various simulations configured by the user, resulting in a set of *Simulation* instances. Each *Simulation* instance is related to this class by using the *hasSimulation* object property.

The *Simulation* class describes a user-defined simulation. It is a subclass of *Comparable* and collects a set of *Scenario*s, by using the *hasScenario* object property, as well as a set of comparable *Simulation*s settle by the *hasComparable* object property, ensuring that only instances of itself are accepted.

In turn, the *Scenario* class describes the user-defined scenario. It is also a subclass of *Comparable*, meaning that a set of comparable *Scenario*s can be included by using the *hasComparable* object property, allowing only instances of the *Scenario* class. This class reunites a set of *Step*s established through the object property *hasStep*.

The *Step* class describes the execution phase of a scenario. The execution phase allows the system to understand which services will run concurrently. A *Step* has configured one or more *Service*s through the *hasService* relationship. Each *Step* is related to the next by the *hasNext* object property, and to the previous by the *hasPrevious* object property. It must be stressed that the first *Step* of each *Scenario* only has the *hasNext* object property, while the last *Step* only considers the *hasPrevious* object property.

The *Service* class defines a service provided by a MAS or a web-service. It is a subclass of *Comparable*, allowing only to add *Service*s instances with the object property *hasComparable*. Similarly to the *Step* class, a *Service* is also related to the previous or next service to be executed by using the *hasPrevious* and *hasNext* object properties. However, both its precedents and the following belong to different implementation phases, i.e., *Step*s. These properties serve to create a precedence in which the result of a service will serve as input to the next one. In addition, a *Service* is characterized by an *InputModel* and respective *InputDataSource*, an *OutputModel*, and the actual result. The object properties *hasInputModel*, *hasInputDataSource*, and *hasOutputModel* relate the class *Service* to the classes *InputModel*, *InputDataSource*, and *OutputModel* respectively. In turn, the *hasResult* object property relates to the superclass *owl : Thing*. This way, the result of a *Service* can be related to instances of the service's output model.

The *InputModel* and *OutputModel* classes characterize the *Service*'s input and output models, respectively. These are abstract classes that enable a recursive definition of *InputModel*s and *OutputModel*s, by using the *hasInputModel* and *hasOutputModel* object properties respectively. This way, a complex model can be composed of simpler ones recursively. On the other hand, it is also allowed to define an existing input model (from other ontology) as a subclass of *InputModel*, as well as an existing output model as a subclass of *OutputModel*. Thus, the reuse of semantic data models from external systems is assured. Regarding syntactic data models, their use is guaranteed through their (XML[1] or JSON[2]) schemes, made available by the tools integrated in TOOCC. These are identified by Uniform Resource Identifiers (URIs) that can be included in the ontology as sub-classes of *InputModel* and *OutputModel*, since classes are identified as URIs in RDF languages [37]. TOOCC is then able to interpret those schemes to operate with the integrated systems.

Finally, the *InputDataSource* class defines the data source responsible for providing data to the input model. It is an abstract class that is not supposed to be instantiated. Instead, one must use its sub-classes, namely the *FileDataSource* and the *WebDataSource* classes. The former describes files as input data sources, while the latter declares APIs end-points as data sources.

So far, we have seen the class hierarchy and relations (object properties) among them. The next step is to present the classes' attributes (datatype properties), their value types, and the properties' cardinality. The value type is the most relevant facet in the development of an ontology, as it defines the type of each property used in the classification process [44].

Table 1 presents TCC's classes, their properties, and respective facets, where TCC's object properties are written in blue, and the datatype properties are in green.

Observing Table 1, it can be seen that both the *Comparable* and *InputDataSource* classes have no properties defined. As already explained, these are abstract classes that are not supposed to be instantiated.

In the *SimulationSet* class, the *hasSimulation* object property must have at least one *Simulation* class associated. In turn, the object properties *hasInputModel* and *hasOutputModel*, of *InputModel* and *OutputModel* respectively, have no cardinality restrictions.

The *Simulation* class is defined by an unsigned integer *id*, a string *name*, a string *description*, and a *created* and a *modified* date-time. The *hasComparable* and the *hasScenario* object properties must have at least one *Simulation* or *Scenario* classes respectively.

The *Scenario* class is also defined by an unsigned integer *id*, a string *name*, a string *description*, and a *created* and a *modified* date-time. Additionally, it is also described by an unsigned integer *numberOfSteps*, with the number of *Step* classes set with the object property *hasStep*. The *hasStep* object properties must have at least one *Step* class, while the *hasComparable* object property can have one or more *Scenario* class.

The *Step* class, in turn, is defined by one unsigned integer *step* describing the number of the execution phase (i.e., the *Step* number), and one boolean flag (*isCompleted*) to indicate if the *Step* has finished its execution. The *hasService* object property must have at least one *Service* set, while the *hasPrevious* and *hasNext* object properties can only have at most one *Step* each.

The *Service* class is not defined by any datatype property. Similarly to *Simulation* and *Scenario*, the *hasComparable* object property, if set, must have at least one *Service* class defined. In the same way, the *hasPrevious* and *hasNext* object properties can only relate to one *Service* at most. Finally, the *hasInputModel*, the *hasInputDataModel*, the *hasOutputModel*, and the *hasResult* object properties, can only be related to one *InputModel*, one *InputDataModel*, one *OutputModel*, and one *owl : Thing* respectively.

In turn, the *FileDataSource* and the *WebDataSource* classes are not defined by any object property. The *FileDataSource* object is defined by exactly one URI *filePath* datatype property describing the local file path, at most one string *fileFormat* datatype property with the file format, and

[1]https://www.w3.org/XML/Schema
[2]https://json-schema.org/

TABLE 1. TCC's classes, properties and facets.

Class	Properties	Facets
Comparable		
SimulationSet	*hasSimulation*	≥ 1 *Simulation*
Simulation		*Comparable*
	id	1 *unsignedInt*
	name	1 *string*
	description	1 *string*
	created	1 *dateTime*
	modified	1 *dateTime*
	hasComparable	≥ 1 *Simulation*
	hasScenario	≥ 1 *Scenario*
Scenario		*Comparable*
	id	1 *unsignedInt*
	name	1 *string*
	description	1 *string*
	created	1 *dateTime*
	modified	1 *dateTime*
	numberOfSteps	1 *unsignedInt*
	hasComparable	≥ 1 *Scenario*
	hasStep	≥ 1 *Step*
Step	*step*	1 *unsignedInt*
	isCompleted	1 *boolean*
	hasService	≥ 1 *Service*
	hasPrevious	≤ 1 *Step*
	hasNext	≤ 1 *Step*
Service		*Comparable*
	hasComparable	≥ 1 *Service*
	hasPrevious	*Service*
	hasNext	*Service*
	hasInputModel	≤ 1 *InputModel*
	hasInputDataSource	≤ 1 *InputDataSource*
	hasOutputModel	≤ 1 *OutputModel*
	hasResult	≤ 1 *Thing*
InputModel	*hasInputModel*	*InputModel*
OutputModel	*hasOutputModel*	*OutputModel*
InputDataSource		
FileDataSource		*InputDataSource*
	filePath	1 *anyURI*
	fileFormat	≤ 1 *string*
	parseFileTo	≤ 1 *anyURI*
WebDataSource		*InputDataSource*
	requestURL	1 *anyURI*
	requestMethod	1 *string*
	requestHeader	*string*
	requestBody	≤ 1 *string*
	requestBodyContentType	≤ 1 *string*
	responseContentType	≤ 1 *string*
	responsePath	≤ 1 *string*

one URI *parseFileTo* datatype property defining the semantic model to which the file content should be translated to.

The *WebDataSource* class is defined by exactly one URI (*resquestURL*) and a *requestMethod* string with the HTTP request method. A *requestHeader* string may also be defined with the request header information. Optionally, at most one *requestBody* string may be defined with the message body, whenever it makes sense; as well as a *requestBodyContentType* string. Similarly, at most one *responseContentType* string may be set, and in case the response is a JSON or XML structure, the user may also set the *responsePath* string to get the intended value.

As already stated, TCC is written in OWL-DL syntax. OWL-DL provides the maximum expressiveness possible, maintaining computational completeness, decidability, and the availability of reasoning algorithms [45]. TCC's expressiveness is *ALCQ(D)*, i.e., it allows: to demonstrate attributive language (*AL*), which includes atomic negation, conceptual intersection, universal constraints, and limited existential quantification; complex conceptual negation (*C*); qualified cardinality constraints (*Q*); and the use of data type properties and values *(D)*.

Finally, instances (or individuals) are the objects of the classes that can be classified and validated by the ontology. The following section presents a case study where the use of TCC is demonstrated. It features the ontology instantiation, including its evaluation in which the previously defined requirements are validated.

V. ONTOLOGY EVALUATION

The following case study was developed to evaluate and test the requirements established for TCC in subsection IV-A. To this end, it is considered a scenario where a DR event is applied to reduce the operating costs of the network while returning a fair compensation of the resources involved. The modeled scenario considers historical data for August 2018, with a granularity of 15 minutes. It includes consumption data from 144 consumers with varying profiles (domestic and industrial), generation data from 43 renewable energy sources (solar and wind), and data from 1 regular supplier and 5 backup suppliers to be used whenever the regular supplier is not able to fully satisfy the grid needs. In terms of scalability, this scenario is based on a large amount of data.

Conceptually, the aggregator shall perform the energy scheduling of the network, taking into account all restrictions of its users. After scheduling, the users who reduce their consumption according to what is established by the aggregator are rewarded. Thus, it is necessary to determine what will be the fair remuneration value for each individual. For this are made clusters based on the amount of power cut. Finally, to remunerate the end-user, the maximum rate of each group will be determined. The consumption reduction is made in certain consumers' devices, detailed in their DR contracts. These devices can be air conditioners, sockets, refrigerators, washing machines, among others. To ensure that a sensitive device is not affected, it is possible to assign each device a degree of priority where the highest priority level means that it should be cut only as a last resort, while those with lower priority can be considered more often.

To be able to simulate the described scenario, three web services were selected. The first service will be the scheduling optimization, which will be followed by the aggregation (clustering) service and after by the service that determines the remuneration applicable to each group. Services should run sequentially, where part of the output from the first tool is used by the second, and so on.

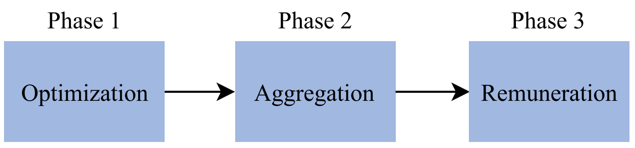

FIGURE 4. Case study execution phases.

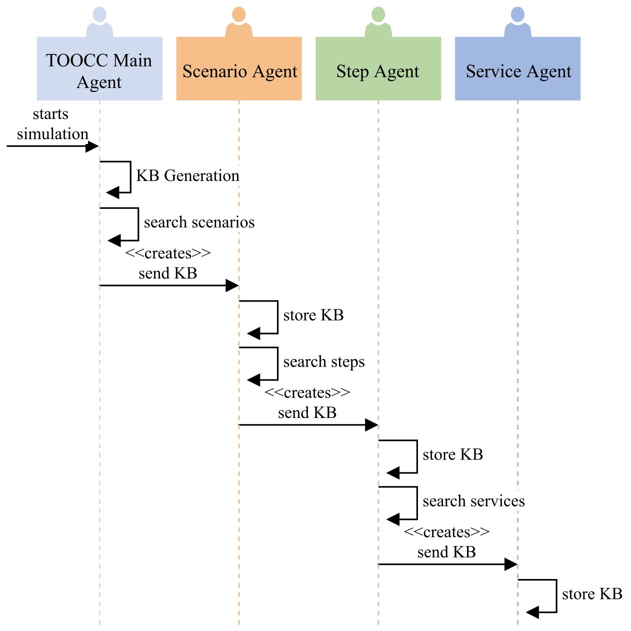

FIGURE 5. Sequence diagram for multi-agent communications.

```
1  :simulation-set rdf:type tcc:SimulationSet ,
       owl:NamedIndividual ;
2    tcc:hasSimulation :simulation-1 .
3
4  :simulation-1 rdf:type owl:NamedIndividual ,
       tcc:Simulation ;
5    tcc:hasScenario :scenario-1 ;
6    tcc:id "simulation-1" ;
7    tcc:name "Service to Service (S2S) Simulation
         " ^^xsd:string ;
8    tcc:descrition "The simulation demonstrates
         the Service to Service integration where
         the output of a Service is mapped to be
         the input of the next one."^^xsd:string ;
9    tcc:created "2019-07-25T16:26:24"^^xsd:
         dateTime ;
10   tcc:modified "2019-07-25T16:26:24"^^xsd:
         dateTime .
11
12 :scenario-1 rdf:type owl:NamedIndividual , tcc:
       Scenario ;
13   tcc:hasStep :step-1-scen-1 , :step-2-scen-1 ,
          :step-3-scen-1 ;
14   tcc:numberOfSteps "3"^^xsd:unsignedInt ;
15   tcc:id "scenario-1" ;
16   tcc:name "Service to Service (S2S) Scenario"
         ^^xsd:string ;
17   tcc:descrition "The scenario demonstrates the
          Service to Service integration where the
          output of a Service is mapped to be the
         input of the next one."^^xsd:string ;
18   tcc:created "2019-07-25T16:26:24"^^xsd:
         dateTime ;
19   tcc:modified "2019-07-25T16:26:24"^^xsd:
         dateTime .
```

Listing. 1. General scenario configuration.

```
1  :step-1-scen-1 rdf:type owl:NamedIndividual ,
       tcc:Step ;
2    tcc:hasService :service-optimization-
         algorithm ;
3    tcc:hasNext :step-2-scen-1 ;
4    tcc:step "1"^^xsd:unsignedInt .
5
6  :service-optimization-algorithm rdf:type owl:
       NamedIndividual , tcc:Service ;
7    tcc:hasNext :service-aggregation-algorithm ;
8    tcc:hasInputModel :InputOptimizationAlgorithm
          ;
9    tcc:hasInputDataSource :input-file-opti-algo
         ;
10   tcc:hasOutputModel :
         OutputOptimizationAlgorithm .
```

Listing. 2. First phase configurations.

Figure 4 illustrates which services run in each phase and their dependencies.

After the scenario configuration, and the `Start` button is pressed, TOOCC's knowledge base (KB) is filled with the simulation data. The KB is then queried by `TMA` to initialize all necessary agents with the knowledge they need to proceed. A simplified representation of this process is illustrated in Figure 5.

After initializing the `ScenA`, `TMA` sends it the KB data about `:scenario-1`. Listing 1 presents a snippet of `:scenario-1` definition in Turtle.

In line 1, it is possible to observe the definition of an individual (`:simulation-set`) of type `tcc:SimulationSet`. In line 2 the object property `tcc:hasSimulation` indicates that there is only one simulation defined for this case study (i.e., `:simulation-1`). If multiple simulations where defined, this object property would have more individuals, separated by commas (as defined in the first requirement). In the same way, if more scenarios were defined for this case study, the object property `tcc:hasScenarios` (line 5) would have more individuals (as determined in the second requirement). The `:scenario-1` individual has three steps configured (lines 12 to 13) fulfilling the third requirement. For each step, a *StepA* agent is initialized, and the knowledge about each particular step is sent to the respective agent for its execution. As it is possible to verify, the first three requirements defined in the subsection IV-A are present in Listing 1. Listing 2 shows the `:step-1-scen-1` instantiation, as well as the definition of the `:service-optimization-algorithm`.

The link between the optimization service running at this stage and the aggregation service running at the next stage is achieved through the `tcc:hasNext` relationship (line 7). The same happens with the `tcc:Step`,

```
1  :InputOptimizationAlgorithm rdf:type owl:Class
       ;
2    rdfs:subClassOf tcc:InputModel , [
3      rdf:type owl:Restriction ;
4      owl:onProperty tcc:hasInputModel ;
5      owl:cardinality "1"^^xsd:nonNegativeInteger
           ;
6      owl:onClass :CutLimitIn
7    ] ,
8    [ rdf:type owl:Restriction ;
9      owl:onProperty tcc:hasInputModel ;
10     owl:cardinality "1"^^xsd:nonNegativeInteger
           ;
11     owl:onClass :ConsumptionIn
12   ] ,
13   [ rdf:type owl:Restriction ;
14     owl:onProperty tcc:hasInputModel ;
15     owl:cardinality "1"^^xsd:nonNegativeInteger
           ;
16     owl:onClass :ProductionIn
17   ] , [ ... ] .
```

Listing. 3. Excerpt of input model of the first execution phase.

where the execution order is also established through the same relationship (line 3), together with the `tcc:step` property (line 4), which indicates the order in which the phase will execute. For each service defined in the `tcc:hasService` object property (line 2), the *StepA* agent creates a *ServA* agent, being the service information transmitted to the latter. If the `tcc:hasService` object property would have two services defined, then these would run concurrently. This property fulfills the fourth requirement set in subsection IV-A. Additionally, for the `:service-optimization-algorithm` individual, the relationships are constructed for the input data model `:InputOptimizationAlgorithm` (line 8) and the output data model `:OutputOptimizationAlgorithm` (line 10) to be able to communicate with the external services. Here is also demonstrated the fifth requirement defined in subsection IV-A.

Listing 3 demonstrates how the semantic input data model is built for the scheduling service `:service-optimization-algorithm`.

In this model, it is possible to view some of the fields required for execution, such as: `:CutLimitIn` (line 6), `:ConsumptionIn` (line 11), and `:ProductionIn` (line 16). These fields inform the algorithm of the characteristics of the network players, so that it can perform a more efficient scheduling based on consumption and production profiles.

Listing 4 shows a small excerpt of the data instantiated with the `:CutLimitIn` model.

On the other hand, Listing 5 gives an overview of the scenario configuration for running the aggregation algorithm `:service-aggregation-algorithm` (line 8).

The second step (`:step-2-scen-1`) is the execution of the aggregation algorithm, which will create clusters of consumers to help determine the most appropriate remuneration rate. However, for this service to run, it needs to populate its input data model `:InputAggregationAlgorithm`

```
1  :iPmaxidr a csi:CutLimitIn ;
2    mat:item :iArray39iPmaxidr , (...) .
3
4  :iArray39iPmaxidr a mat:Array ;
5    mat:item :iItem2316iArray39iPmaxidr , (...) .
6
7  :iItem2316iArray39iPmaxidr a mat:Item ;
8    mat:pos "2316"^^xsd:unsignedInt ;
9    mat:val "0.0037"^^xsd:double .
```

Listing. 4. Excerpt of input data of the first execution phase.

```
1  :step-2-scen-1
2    rdf:type owl:NamedIndividual , tcc:Step ;
3    tcc:hasService :service-aggregation-algorithm
         ;
4    tcc:hasPrevious :step-1-scen-1 ;
5    tcc:hasNext :step-3-scen-1 ;
6    tcc:step "2"^^xsd:unsignedInt .
7
8  :service-aggregation-algorithm
9    rdf:type owl:NamedIndividual , tcc:Service ;
10   tcc:hasPrevious :service-optimization-
         algorithm ;
11   tcc:hasNext :service-remuneration-algorithm ;
12   tcc:hasInputModel :InputAggregationAlgorithm
         ;
13   tcc:hasOutputModel :
         OutputAggregationAlgorithm .
```

Listing. 5. Configuration of the second phase of execution.

with some of the values that compose the previous phase output data model. The use of concepts between the output model of a service and the input model of the next one is done by using the same class. An example of this process is illustrated in Listing 6.

The code shows that the input data model for the aggregation algorithm `:InputAggregationAlgorithm` (lines 13 to 19) contains the class `:Optimization-SolutionIn` (line 18). This class is composed by `:DGResOut` (line 30) and `:ReduceAmountResOut` (line 26) that result from `:OutputOptimization-Algorithm` (lines 6 and 11 respectively) of the previous step, demonstrating that the output model of a service can be used as part of the output model of another service, or completely.

Besides the input model already presented in Listing 6, a representation of the output data model `:Output-AggregationAlgorithm` is shown in Listing 7.

The next service to be performed is the remuneration service. It will assign a remuneration rate to each entity, according to the group in which it was classified in the aggregation phase. The configuration of phase three is shown in Listing 8.

As can be seen in Listing 8, unlike the previous phases, this service does not have the `tcc:hasNext` relationship, since it is the last phase that will be executed. The input data model `:InputRemunerationModel` of the third phase is represented in Listing 9.

Listing 10 presents the algorithm output data model for obtaining remuneration.

```
:OutputOptimizationAlgorithm rdf:type owl:Class
    ;
  rdfs:subClassOf tcc:OutputModel ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasOutputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :DGResOut ] ,
  [ ... ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasOutputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :ReduceAmountResOut ] .

:InputAggregationAlgorithm rdf:type owl:Class ;
  rdfs:subClassOf tcc:InputModel ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasInputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :OptimizationSolutionIn ] ,
  [ ... ] .

:OptimizationSolutionIn rdf:type owl:Class ;
  rdfs:subClassOf tcc:InputModel ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasInputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :ReduceAmountResOut ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasInputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :DGResOut ] .
```

Listing. 6. Reuse of concepts between different data models.

```
:OutputAggregationAlgorithm
  rdf:type owl:Class ;
  rdfs:subClassOf tcc:OutputModel , [
    rdf:type owl:Restriction ;
    owl:onProperty tcc:hasOutputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :BestKOut ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasOutputModel ;
    owl:someValuesFrom :AggregationListItem ] .
```

Listing. 7. Output model of the second phase of execution.

At the end of the execution the results are made available to the user so that he can analyze and draw conclusions about them. These include the results of each intermediate phase, as well as the final results extracted from the last execution phase.

With the present case study, it is possible to verify the fulfillment of several TCC's requirements identified in Section IV-A. It shows the possibility of creating a scenario with several steps that will execute sequentially, and where each step can consider one or more services simultaneously, to improve the simulation performance. The interoperability of those services is achieved through the knowledge exchanged between the input and output data models.

```
:step-3-scen-1
  rdf:type owl:NamedIndividual , tcc:Step ;
  tcc:hasService :service-remuneration-
      algorithm ;
  tcc:hasPrevious :step-2-scen-1 ;
  tcc:step "3"^^xsd:unsignedInt .

:service-remuneration-algorithm
  rdf:type owl:NamedIndividual , tcc:Service ;
  tcc:hasPrevious :service-aggregation-
      algorithm ;
  tcc:hasInputModel :InputRemunerationAlgorithm
      ;
  tcc:hasOutputModel :
      OutputRemunerationAlgorithm .
```

Listing. 8. Configuration of the third phase of execution.

```
:InputRemunerationAlgorithm rdf:type owl:Class
    ;
  rdfs:subClassOf tcc:InputModel ,
  [
    rdf:type owl:Restriction ;
    owl:onProperty tcc:hasInputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :AggregationSolutionIn ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty tcc:hasInputModel ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
        ;
    owl:onClass :CostIn ] ,
  [ ... ] .
```

Listing. 9. Input model of the third phase of execution.

```
:OutputRemunerationAlgorithm rdf:type owl:Class
    ;
  rdfs:subClassOf tcc:OutputModel , [
    rdf:type owl:Restriction ;
    owl:onProperty tcc:hasOutputModel ;
    owl:someValuesFrom :RemunerationListItem ]
        .
```

Listing. 10. Output model of the third phase of execution.

In the simulation process, it was possible to verify that TCC effectively achieves its purpose. Moreover, the simplicity of TCC design enables a good performance, and it was proven that the system can execute scenarios with a large amount of data.

TCC has the flexibility to model different problems in the scope of PES, taking into account different perspectives, roles, and objectives, as can be seen by this case study and by others already published [15], [16]. This article is distinct from previously published works, as it presents the application ontology defined for the TOOCC tool and uses a case study to illustrate its use. At the same time, this case study not only evaluated TCC, but also demonstrated the execution of a complex simulation scenario in which three web services are integrated to simulate a DR program in a local community.

VI. CONCLUSION

The large-scale implementation of distributed energy sources, as well as the targets imposed worldwide to face the new climate paradigm, are causing severe changes in the sector, which are continually adapting to meet the new challenges. The development of decision support tools to address the problem as a whole is one of the key challenges in PES.

TOOCC contributes to increase interoperability between heterogeneous systems that study, experiment, and test the PES domain. This work introduces TOOCC's application ontology. TCC supports the scenarios' definition and results comparison while easing the interoperability among the several systems. It has been developed considering a level of abstraction and flexibility that allows its evolution. At the same time, TCC can include the semantic or syntactic models of the various integrated tools, as long as these models are publicly available or shared by the external tools.

The case study presents a DR program in which the consumer's remuneration (per power unit) depends on the cluster to which he is assigned. In turn, this cluster depends on the amount of energy that the consumer can make available to the network. The main purpose of the case study is to evaluate the presented application ontology, while demonstrating how a simulation is configured and how the interoperability is achieved by mapping the output of a service to the input of the next one. During the case study, it was possible to demonstrate several requirements defined for TCC.

As future work, the next step intends to show the benefits of using this ontology for the results (models) comparison, by exploring the reasoning capabilities of the system. For this purpose, two different scenarios will be considered: i) using different simulation tools aimed at the study of similar problems; and ii) using the same system with different inputs. This way, it will also be possible to demonstrate the simultaneous execution of distinct simulations and scenarios.

REFERENCES

[1] European Commission. *2020 Climate & Energy Package*. Accessed: Jun. 2017. [Online]. Available: https://ec.europa.eu/clima/policies/strategies/2020%7B_%7Den

[2] *Energy Challenges and Policy*. Accessed: Jul. 2017. [Online]. Available: http://ec.europa.eu/europe2020/pdf/energy2%7B_%7Den.pdf

[3] P. T. Manditereza and R. Bansal, "Renewable distributed generation: The hidden challenges—A review from the protection perspective," *Renew. Sustain. Energy Rev.*, vol. 58, pp. 1457–1465, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032115016597

[4] Seetharaman, K. Moorthy, N. Patwa, Saravanan, and Y. Gupta, "Breaking barriers in deployment of renewable energy," *Heliyon*, vol. 5, no. 1, Jan. 2019, Art. no. e01166.

[5] F. P. Sioshansi, Ed., *Evolution of Global Electricity Markets—New Paradigms, New Challenges, New Approaches*. Amsterdam, The Netherlands: Elsevier, 2013. [Online]. Available: https://www.sciencedirect.com/book/9780123978912/evolution-of-global-electricity-markets

[6] P. Faria, Z. Vale, and J. Baptista, "Demand response programs design and use considering intensive penetration of distributed generation," *Energies*, vol. 8, no. 6, pp. 6230–6246, 2015. [Online]. Available: https://www.mdpi.com/1996-1073/8/6/6230

[7] A. Conejo, M. Carrion, and J. Morales, *Decision Making Under Uncertainty in Electricity Markets*, vol. 153. New York, NY, USA: Springer, 2010, doi: 10.1007/978-1-4419-7421-1.

[8] T. Pinto and Z. Vale, "AiD-EM: Adaptive decision support for electricity markets negotiations," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 6563–6565, doi: 10.24963/ijcai.2019/957.

[9] G. Santos, T. Pinto, H. Morais, T. M. Sousa, I. F. Pereira, R. Fernandes, I. Praça, and Z. Vale, "Multi-agent simulation of competitive electricity markets: Autonomous systems cooperation for European market modeling," *Energy Convers. Manage.*, vol. 99, pp. 387–399, Jul. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890415003969

[10] T. Pinto, Z. Vale, T. M. Sousa, I. Praça, G. Santos, and H. Morais, "Adaptive learning in agents behaviour: A framework for electricity markets simulation," *Integr. Comput.-Aided Eng.*, vol. 21, no. 4, pp. 399–415, May 2014, doi: 10.3233/ICA-140477.

[11] H. Morais, P. Vancraeyveld, A. H. B. Pedersen, M. Lind, H. Jóhannsson, and J. Østergaard, "SOSPO-SP: Secure operation of sustainable power systems simulation platform for real-time system state evaluation and control," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2318–2329, Nov. 2014.

[12] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, "EPOCHS: A platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Trans. Power Syst.*, vol. 21, no. 2, pp. 548–558, May 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1626358/

[13] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, and J. Thorp, "GECO: Global event-driven co-simulation framework for interconnected power system and communication network," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1444–1456, Sep. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6200399/

[14] S. Scherfk. *Mosaik Documentation*. Accessed: Nov. 2019. [Online]. Available: https://media.readthedocs.org/pdf/mosaik/latest/mosaik.pdf

[15] B. Teixeira, T. Pinto, F. Silva, G. Santos, I. Praça, and Z. Vale, "Multi-agent decision support tool to enable interoperability among heterogeneous energy systems," *Appl. Sci.*, vol. 8, no. 3, p. 328, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/3/328

[16] B. Teixeira, F. Silva, T. Pinto, G. Santos, I. Praça, and Z. Vale, "TOOCC: Enabling heterogeneous systems interoperability in the study of energy systems," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Jul. 2017, pp. 1–5.

[17] T. Pinto, G. Santos, and Z. Vale, "Practical application of a multi-agent systems society for energy management and control," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst. (AAMAS)*. Richland, SC, USA: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2378–2380.

[18] IEEE PES. (2010). *IEEE PES Multi-Agent Systems Working Group*. [Online]. Available: https://site.ieee.org/pes-mas/upper-ontology/

[19] G. Santos, T. Pinto, and Z. Vale, "Ontologies for the interoperability of heterogeneous multi-agent systems in the scope of power and energy systems," in *Trends in Cyber-Physical Multi-Agent Systems. The PAAMS Collection—15th International Conference*, F. De la Prieta, Z. Vale, L. Antunes, T. Pinto, A. T. Campbell, V. Julián, A. J. Neves, and M. N. Moreno, Eds. Cham, Switzerland: Springer, 2018, pp. 300–301.

[20] PQ Soft. (2018). *PSCAD: Transients Simulation Software By the Manitoba HVDC Research Centre*. Accessed: Feb. 2018. [Online]. Available: http://www.pqsoft.com/pscad/index.htm

[21] General Electric. (2018). *PSLF*. Accessed: Feb. 2018. [Online]. Available: https://www.geenergyconsulting.com/practice-area/software-products/pslf

[22] University of Southern California. (2018). *The Network Simulator—ns-2*. Accessed: Feb. 2018. [Online]. Available: https://www.isi.edu/nsnam/ns/

[23] A. Patel and S. Jain, "Present and future of semantic Web technologies: A research statement," *Int. J. Comput. Appl.*, pp. 1–10, Jan. 2019. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/1206212X.2019.1570666, doi: 10.1080/1206212X.2019.1570666.

[24] A. Gyrard, M. Serrano, and P. Patel, "Building interoperable and cross-domain semantic Web of things applications," in *Managing the Web of Things: Linking the Real World to the Web*. Amsterdam, The Netherlands: Elsevier, Feb. 2017, pp. 305–324.

[25] M. R. Saeed, C. Chelmis, and V. K. Prasanna, "Automatic integration and querying of semantic rich heterogeneous data: Laying the foundations for semantic Web of things," in *Managing the Web of Things: Linking the Real World to the Web*. Amsterdam, The Netherlands: Elsevier, Feb. 2017, pp. 251–273.

[26] Java Agent Development Framework. *Homepage*. Accessed: Nov. 2019. [Online]. Available: https://jade.tilab.com/

[27] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems With JADE*. Hoboken, NJ, USA: Wiley, 2007. [Online]. Available: https://www.wiley.com/en-us/Developing+Multi+Agent+Systems+with+JADE-p-9780470058404

[28] *The Foundation for Intelligent Physical Agents*. Accessed: Nov. 14, 2019. [Online]. Available: http://www.fipa.org/

[29] D. Man, "Ontologies in computer science," *Didactica Math.*, vol. 31, no. 1, pp. 43–46, 2013. [Online]. Available: http://www.math.ubbcluj.ro/~didactica/pdfs/2013/didmath2013-06.pdf

[30] M. Gruninger and J. Lee, "Introduction," *Commun. ACM*, vol. 45, no. 2, pp. 39–41, Feb. 2002, doi: 10.1145/503124.503146.

[31] G. Santos, Z. Vale, P. Faria, and L. Gomes, "BRICKS: Building's reasoning for intelligent control knowledge-based system," *Sustain. Cities Soc.*, vol. 52, Jan. 2020, Art. no. 101832. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210670719314295

[32] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic Web rule language combining owl and RuleML," W3C, Tech. Rep., 2004. Accessed: Jan. 2019. [Online]. Available: http://www.w3.org/Submission/SWRL

[33] M. K. Bergman, "A brief survey of ontology development methodologies," Tech. Rep., 2010. [Online]. Available: https://www.mkbergman.com/wp-content/themes/ai3v2/files/2010Posts/a-brief-survey-of-ontology-development-methodologies.pdf

[34] N. F. Noy and D. L. Mcguinness, "Ontology development 101: A guide to creating your first ontology," Stanford Knowl. Syst., Stanford, CA, USA, Lab. Tech. Rep. KSL-01-05 and Stanford Med. Inform. Tech. Rep. SMI-2001-0880, Mar. 2001.

[35] Standford University. *Protégé*. Accessed: Jan. 2020. [Online]. Available: https://protege.stanford.edu/

[36] W3C. *Owl*. Accessed: Jan. 2020. [Online]. Available: https://www.w3.org/TR/owl-guide/

[37] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. (2014). *RDF 1.1 Turtle*. [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/

[38] Ontology Engineering Group. *Linked Open Vocabularies*. Accessed: Jan. 2019. [Online]. Available: https://lov.linkeddata.es/dataset/lov/

[39] M. Lefrançois, J. Kalaoja, T. Ghariani, and A. Zimmermann, "SEAS knowledge model," Smart Energy Aware Syst., Deliverable 2.2, Tech. Rep. ITEA2 12004, 2016. [Online]. Available: http://www.maxime-lefrancois.info/docs/SEAS-D2_2-SEAS-Knowledge-Model.pdf

[40] L. Daniele, F. den Hartog, and J. Roes, "Created in close interaction with the industry: The smart appliances reference (SAREF) ontology," in *Formal Ontologies Meet Industry*, R. Cuel and R. Young, Eds. Cham, Switzerland: Springer, 2015, pp. 100–112.

[41] G. Santos, T. Pinto, Z. Vale, I. Praça, and H. Morais, "Electricity markets ontology to support MASCEM's simulations," in *Proc. Int. Workshops Practical Appl. Agents Multi-Agent Syst.*, Seville, Spain, J. Bajo, J. M. Escalona, S. Giroux, P. Hoffa–Dkabrowska, V. Julián, P. Novais, N. Sánchez-Pi, R. Unland, and R. Azambuja-Silveira, Eds. Cham, Switzerland: Springer, Jun. 2016, pp. 393–404, doi: 10.1007/978-3-319-39387-2_33.

[42] J. Cuenca and F. Larrinaga. (Feb. 2019). *DABGEO: Domain Analysis-Based Global Energy Ontology*. [Online]. Available: https://innoweb.mondragon.edu/ontologies/dabgeo/index-en.html

[43] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *Knowl. Eng. Rev.*, vol. 11, no. 2, pp. 93–136, Jun. 1996.

[44] M. Rodrigues, R. R. Silva, and J. Bernardino, "Linking open descriptions of social events (LODSE): A new ontology for social event classification," *Information*, vol. 9, no. 7, p. 164, 2018.

[45] M. K. Smith, C. Welty, and D. L. McGuinness. (2004). *OWL Web Ontology Language Guide*. [Online]. Available: https://www.w3.org/TR/owl-guide/

BRÍGIDA TEIXEIRA received the B.Sc. and M.Sc. degrees from the Polytechnic Institute of Porto, Porto, Portugal, in 2014 and 2019, respectively. She is a Researcher with the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD), Polytechnic Institute of Porto. Her research interests include multiagent systems, ontologies, decision support systems, and smart energy grids.

GABRIEL SANTOS received the bachelor's and M.Sc. degrees in informatics from the Polytechnic Institute of Porto, Porto, Portugal. He is currently pursuing the Ph.D. degree with the University of Salamanca, Salamanca, Spain. He is also a Researcher with the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD), Polytechnic Institute of Porto. His research interests include multiagent systems, ontologies, electricity market negotiations, decision support systems, and smart energy grids.

TIAGO PINTO (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Polytechnic Institute of Porto, Porto, Portugal, in 2008 and 2011, respectively, and the Ph.D. degree from the University of Trás-os-Montes e Alto Douro, Vila Real, Portugal, in 2016. He is also an Invited Assistant Professor with the School of Engineering, Polytechnic Institute of Porto (ISEP/IPP), and a Researcher with the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD). His research interests include multiagent simulation, machine learning, automated negotiation, smart grids, and electricity markets.

ZITA VALE (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Porto, Porto, Portugal, in 1993. She is currently a Professor with the Polytechnic Institute of Porto, Porto. Her research interests focus on artificial intelligence applications, smart grids, electricity markets, demand response, electric vehicles, and renewable energy sources.

JUAN M. CORCHADO (Member, IEEE) was born in Salamanca, Spain, in 1971. He received the Ph.D. degree in computer sciences from the University of Salamanca and the Ph.D. degree in artificial intelligence from the University of the West of Scotland. He was the Vice President for Research and Technology Transfer, from 2013 to 2017, and the Director of the Science Park with the University of Salamanca, where he was also the Director of the Doctoral School, until 2017. He has been elected twice as the Dean of the Faculty of Science with the University of Salamanca. He has been a Visiting Professor with the Osaka Institute of Technology, since 2015, and a Visiting Professor with University Teknologi Malaysia, since 2017. He is the Director of the Bioinformatics, Intelligent Systems, and Educational Technology (BISITE) Research Group, which he created, in 2000. He is the President of the IEEE Systems, Man and Cybernetics Spanish Chapter and the Academic Director of the Institute of Digital Art and Animation, University of Salamanca, where he is currently a Full Professor. He also oversees the master's programs in digital animation, security, mobile technology, community management, and management for TIC Enterprises with the University of Salamanca. He is a member of the Advisory Group on Online Terrorist Propaganda of the European Counter Terrorism Centre (EUROPOL). He is also an Editor and the Editor-in-Chief of specialized journals such as the *Advances in Distributed Computing and Artificial Intelligence Journal*, the *International Journal of Digital Contents and Applications*, and the *Oriental Journal of Computer Science and Technology*.