Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

A MODELING FRAMEWORK FOR COMPLEX SYSTEMS

María Julia Blas[1], and Silvio Gonnet[1]

[1]Instituto de Desarrollo y Diseño INGAR, CONICET-UTN, Santa Fe, Santa Fe, ARGENTINA

ABSTRACT

This paper presents a modeling framework for defining abstractions of real-world complex systems promoting the development of discrete-event simulation models based on DEVS. An ontology, a metamodel, and a reasoner are combined in one single structure to allow an upgrade of an abstraction model to an implementation model. Our motivation is to reduce the effort related to the modeling part when specifying DEVS models for complex systems described from an abstraction of reality built over a research question. Applications include an easier introduction to M&S for students of any scientific field that can define an abstraction model with an easier introduction to DEVS models (from formalization to implementation).

1 INTRODUCTION

Complexity exists at many levels within a system, including structural, behavioral, and knowledge levels, and emergent behavior is an expected consequence of complexity (Szabo et al. 2023). Many challenges arise from the fact that systems expand in scale and complexity, including modeling and representation of such systems, their implementation, the integration of artificial intelligence models within traditional modeling and simulation, and so on. Regarding building a representation for simulation purposes (i.e., Modeling and Simulation -M&S-), these new levels of complexity challenge the modeling task by requiring a more detailed abstraction model to deal with it.

M&S comprise two parts *i) modeling* residing on the *abstraction* level and *ii) simulation* residing on the *implementation* level. *Abstraction* focuses on an aspect of reality (often called "phenomenon") to reduce the complexity of the reality being considered (Zeigler et al. 2018). It is mainly devoted to specifying what we need to model starting from reality. Ontologies are frequently used to capture reality in a computational form. An ontology is an explicit specification of a conceptualization (Gruber 1993). It is an abstract and simplified worldview that includes concepts, relationships, and constraints for representing a domain. Unlike other disciplines (where ontologies are used to describe an assumed objectively observable system), in M&S ontologies are one instrument able to be used for capturing a system from reality according to a research question which bounds the scope and resolution of models (Turnitsa et al. 2010). More importantly, they should capture ontological, epistemological, and teleological considerations for further composability and interoperability. That is, as ontologies are representations of specifications, they need to consider how these representations are differentiable by computers since when models and/or simulations are placed together (composability) they should share context (interoperability) (Turnitsa et al. 2010). One approach for achieving such a shared context is defining architectural support for the ontologies used for M&S purposes. The authors of (Ruy et al. 2016) identify three types of ontologies: foundational, core, and domain. When these types are combined, accurate worldviews can be built by enriching specific domains from independent domains across a well-defined architecture.

On the other hand, *implementation* is intended to make it easier to work out the implications of the abstraction and implement it in reality. Hence, it is mainly devoted to defining how we are going to build a simulation model to represent as closely as possible the abstraction previously defined. In this regard, the Discrete Event System Specification (DEVS) is a popular formalism for modeling complex dynamic systems using discrete-event abstraction (Zeigler et al. 2018). Due to its general systems basis, DEVS provides a set of (mathematical) elements for modeling time-varying systems. DEVS formalism can be used in either of two forms *i)* as a mathematical formalism for specifying models using set theory notation, or *ii)* as an executable representation for implementing models using a programming language. In theory, both strategies should be consistent with each other (since executable models should be grounded in formal models). However, it is well known that ensuring that an implementation conforms to a formalization is not straightforward (Sarjoughian et al. 2015). Moreover, since there is no common format used by all tools, porting DEVS models between tools implies rewriting the model (Van Tendeloo and Vangheluwe 2017).

This paper presents a modeling framework for defining abstractions of real-world complex systems promoting the development of discrete-event simulation models based on DEVS. At the core, two different components are used *i)* an ontology-based component and *ii)* a metamodeling-based component. The first is defined by a multi-tier ontological model named FCD-OntoArch (Foundational-Core-Domain Ontological Architecture for Sciences) supporting the description of real-world scenarios (from any domain) using different levels of modeling depending on the (science) representation goal. Such ontological support provides ontological consistency and makes modeling decisions easier. The latter is defined by a metamodel architecture based on Model-Driven Engineering (MDE) principles for structuring (and implementing) DEVS simulation models. Such metamodeling support defines the simulation language. By combining these components, an algorithm is defined as a reasoning engine to make mapping inferences based on a set of well-defined rules. These inferences are used to recommend actions regarding how the DEVS models should be defined starting from a real-world abstraction built by the modeler. Our motivation is to reduce the effort related to the *modeling* part when specifying DEVS models for complex systems that can be described from an abstraction of reality built over a research question. Hence, the main contribution is the overall two-fold modeling framework composed of both modeling components and the algorithm used to create a mapping between *abstraction* and *implementation* (in the form of DEVS formal models).

The remainder of this paper is structured as follows. Section 2 presents a literature review of existing approaches used to support both abstraction and implementation modeling. Section 3 introduces the modeling framework by explaining how each component was deployed. Section 4 presents a case study and a discussion regarding the results of our proposal. Section 5 is devoted to conclusions and future work.

2 RELATED WORK: EXISTING APPROACHES

In applied sciences, models are built based on abstraction. Most simulation models start from the abstraction of a phenomenon contextualized (in a particular world) with a research question (Turnitsa et al. 2010, Zeigler 2019, Robinson 2014). Such an abstraction is frequently related to the conceptual modeling task since "conceptual modeling involves abstracting a model from the real world" (Robinson 2014). However, such a model also needs to be executed on simulation software. That is often called "reduction to a concrete form" (Zeigler 2019), meaning that the abstraction needs to be formalized and implemented in a particular simulation language. Hence, the differentiation between what is *modeling-oriented* and what is *implementation-oriented* is highly important for M&S.

Ontologies definition allows an unambiguous specification of the structure of knowledge in a domain, enables knowledge sharing and reuse, and consequently, makes automated reasoning about ontologies possible (Orgun and Meyer 2008). Furthermore, ontologies are widespread in the engineering field, and M&S is no exception. Several ontologies have been defined over the years for dealing with M&S. Given that this paper is focused on the use of ontologies for event-based dynamic systems as a vehicle for having feasible representations of abstractions (i.e., dealing only with the *modeling-oriented* part of M&S), this section only includes the most relevant ontologies related to this topic (DESO and DeMO). In our case, the *implementation-oriented* part is directly defined by DEVS to delegate the simulator perspective on a formalized language. As we will show in Section 3, such an *implementation-oriented* part is outside the scope of the ontological model.

DESO (Guizzardi and Wagner 2010) is a foundational ontology for discrete event system modeling derived from the foundational ontology UFO. The main purpose of such ontology is to provide a basis for evaluating discrete event simulation languages. The authors claim that “a discrete event system model may be expressed at different levels of abstraction”. In this context, they preset two ontologies derived from UFO: *i)* the Design-Time Ontology DESO-U that describes a discrete event system by defining the entity types (i.e., the instances that are part of the system), and *ii)* the Run-Time Ontology DESO-I that deals with individuals of different types from the simulator perspective.

The Discrete-event Modeling Ontology (DeMO) (Silver et al. 2011) is a web-accessible ontology for discrete-event modeling. When building this ontology, the authors try to capture as much knowledge about the discrete event modeling domain, resulting in four main concepts: *DeModel*, *ModelConcepts*, *ModelComponents,* and *ModelMechanisms*. Such concepts are related as follows: “*A DeModel is built from model components and is “put in motion” by model mechanisms*” (Miller et al. 2004). The *ModelComponent* concept is used to describe the model elements used for building a *DeModel*. Such components correspond to the elements of n-tuples traditionally used in literature to formally define the models. On the other hand, the *ModelMechanism* concept is used to specify rules of engagement adopted by specific modeling techniques (e.g., EventSchedulingMechanism and Clock-SettingMechanism).

However, unlike other disciplines where ontologies are used to describe an assumed objectively observable system, in M&S ontologies should capture a system from reality according to a research question that bounds the scope and resolution of models. This is consistent with the notion of abstraction (or conceptual modeling). Both cases detailed above exhibit the semantic meaning of what is occurring in an implemented simulation or simulation component, not for the conceptual model. As Turnitsa et al. (2010) enunciated, they are correct for their role (i.e., describing a functional utility), not for describing the modeling decisions and assumptions that went into their design.

Table 1 classifies DESO and DeMO along with our proposal to highlight the main differences. As we will show in the following section, our proposal embeds the event-based description as one piece of a well-defined ontological structure and, at the same time, isolates the abstraction model description by making it independent of the simulation language (for us, DEVS).

Table 1: Differences between DESO, DeMO, and our proposal. All approaches use a discrete-event view.

Ontology	**Provides support to define…**			
	Abstract Model		Implementation Model	
	Modeling utility	Functional utility	Formalization level	Software level
DESO	◐	●		◐
DeMO		●	●	
This paper	●		●	●

3 THE TWO-FOLD MODELING FRAMEWORK FOR M&S OF COMPLEX SYSTEMS

Figure 1 shows the high-level architecture of the modeling framework. As the figure shows, two components are used to support the model definitions: *FCD-OntoArch* and the *Multi-level Layered Conceptualization of DEVS*. Then, a reasoning engine based on an algorithm is introduced as a vehicle for moving forward from abstraction to implementation. Each component is detailed in the following sections.

3.1 FCD-OntoArch: The Ontological Model

FCD-OntoArch is a multi-tier architecture based on modularization, reuse, and specialization of concepts and relationships across levels (Olsina 2021). Such an architecture considers Foundational, Core, Top-Domain, Low-Domain, and Instance levels. These levels promote an accurate design within the overall schema according to the following rules:

rule i) A new ontology located at level Core, Top-Domain, or Low-Domain must guarantee correspondence of its elements with the elements defined at the immediately higher level.

rule ii) Ontologies placed at the same level –except Foundational and Instance levels- can be related to each other, but they must guarantee their joint definition (as a whole) does not invalidate the principles of the next higher level.

rule iii) At the Instance level, only individuals of particular things can be found.

These rules allow maintaining an ontological structure to represent real-world scenarios with different levels of abstraction depending on the goal. By *rule i)*, the design promotes terms and relationships defined at the lower levels to be semantically enriched with the terms and relationships placed at the higher levels. Then, *rule ii)* allows the terms and relationships of ontologies placed at the same level act as complement each other, maintaining correlation with the definitions of the ontologies of the higher levels. Finally, by *rule iii)*, any individual placed at the lowest level will be an instance of a particular class at higher levels. These rules together allow using FCD-OntoArch for M&S purposes as an ontological approach that combines consistency between several domains (due to a well-defined structure of layered abstractions) with an accurate modeling template for redefining the missing (new) domains in an easier way.

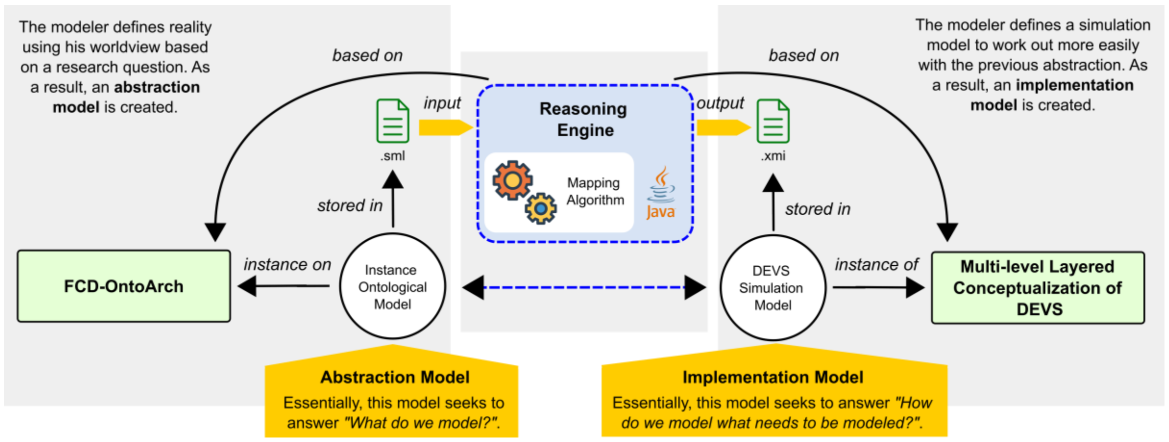
The modeler defines reality using his worldview based on a research question. As a result, an abstraction model is created.
based on
input
.sml
stored in
Reasoning Engine
Mapping Algorithm
Java
output
.xmi
stored in
based on
The modeler defines a simulation model to work out more easily with the previous abstraction. As a result, an implementation model is created.
FCD-OntoArch
instance on
Instance Ontological Model
DEVS Simulation Model
instance of
Multi-level Layered Conceptualization of DEVS
Abstraction Model
Essentially, this model seeks to answer "What do we model?".
Implementation Model
Essentially, this model seeks to answer "How do we model what needs to be modeled?".

Figure 1: High-level architecture of the modeling framework. Components highlighted in green are used to support both abstraction and implementation models. The reasoning engine is used to support the set of mapping rules used as guidelines for the M&S of a real-world phenomenon (abstraction model) using DEVS models (implementation model).

Figure 2 depicts the FCD-OntoArch architecture using the set of conceptual components already included in the structure as ontologies placed at a specific level of abstraction. Quick level identification is given through naming convention (e.g., the ontology at the foundational level is called Foundational Ontology -FO for short-, ontologies at the core level are named Core Ontologies -CO-, and so on). At the foundational level, an ontology called *Thing FO* is included (Olsina 2023). Foundational ontologies are representations of primitive top-level concepts, independent or neutral of any domain. In this context, *Thing FO* has a minimum set of terms that refers to particular and universal concepts of the world. As a result, the ontology includes 17 terms as concepts, 15 terms as concept attributes, 3 axioms specified in first-order logic, and 12 terms as non-taxonomic relationships well balanced with the taxonomic relationships. For details regarding the *Thing FO* ontology, please see (Olsina 2023).

Regarding other abstraction levels included in FCD-OntoArch, the architecture includes: *i)* at the core level: *Process CO* (Becker et al. 2021), *Goal CO*, *Situation CO*, *Project CO*, and *Particular Event CO* (*PEvent CO*) (Blas et al. 2022); *ii)* at the top-domain level: *Test TDO* (Tebes et al. 2021), *Functional Requirements TDO* (*FRs TDO*), *Non-Functional Requirements TDO* (*NFRs TDO*), and *Measurement and Evaluation TDO* (*MEval TDO*); and *iii)* at the low-domain level: *Metrics LDO* and *Indicators LDO*.

Since this paper is intended to build simulation models based on DEVS, the *PEvent CO* ontology is mainly used as the foundation of the reasoning engine. A brief description of *PEvent CO* is presented below.

Such a description includes all the features required to follow our proposal. Due to space reasons, supplementary ontologies composing the FCD-OntoArch are not discussed. At this point, it is critical to mention that this does not imply other ontologies are not used during reasoning. Moreover, as described before, all features of the ontologies placed on top (and on the same level) of *PEvent CO* are guaranteed. For ontologies placed below *PEvent CO*, the *PEvent CO* definition is also kept directly by enriching specific concepts with such notions.

3.1.1 PEvent CO

PEvent CO is a core ontology based on *Thing FO* that incorporates the notion of events as assertions taking place due to the behavior of the entities. Hence, it is based on the existing terms and relationships of *Thing FO* by the definition of new elements at the core level to facilitate the representation of event-based behaviors. Figure 3 presents the terms (i.e., concepts, concept attributes, taxonomic relationships, and non-taxonomic relationships) included in *PEvent CO*.

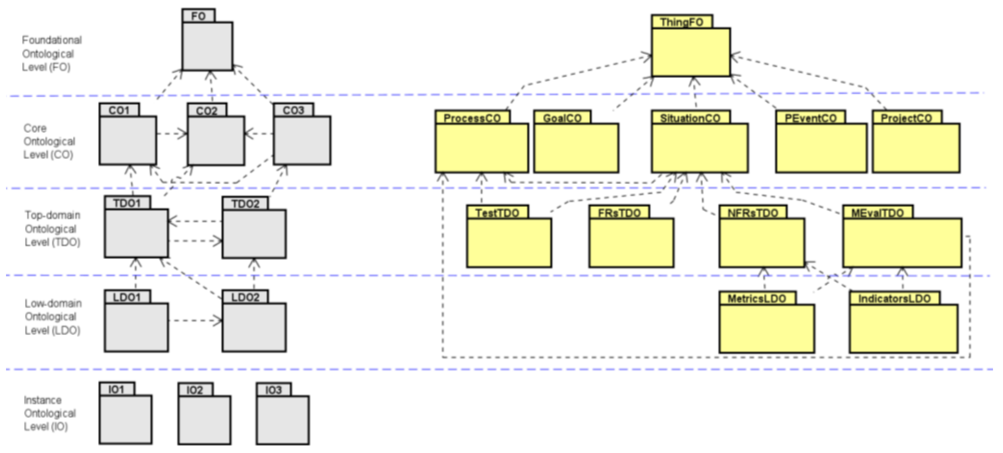
Foundational Ontological Level (FO)
Core Ontological Level (CO)
Top-domain Ontological Level (TDO)
Low-domain Ontological Level (LDO)
Instance Ontological Level (IO)
FO
CO1
CO2
CO3
TDO1
TDO2
LDO1
LDO2
IO1
IO2
IO3
ThingFO
ProcessCO
GoalCO
SituationCO
PEventCO
ProjectCO
TestTDO
FRsTDO
NFRsTDO
MEvalTDO
MetricsLDO
IndicatorsLDO

Figure 2: Five-tier ontological architecture defined through the set of ontologies already included as conceptual components placed at each level.

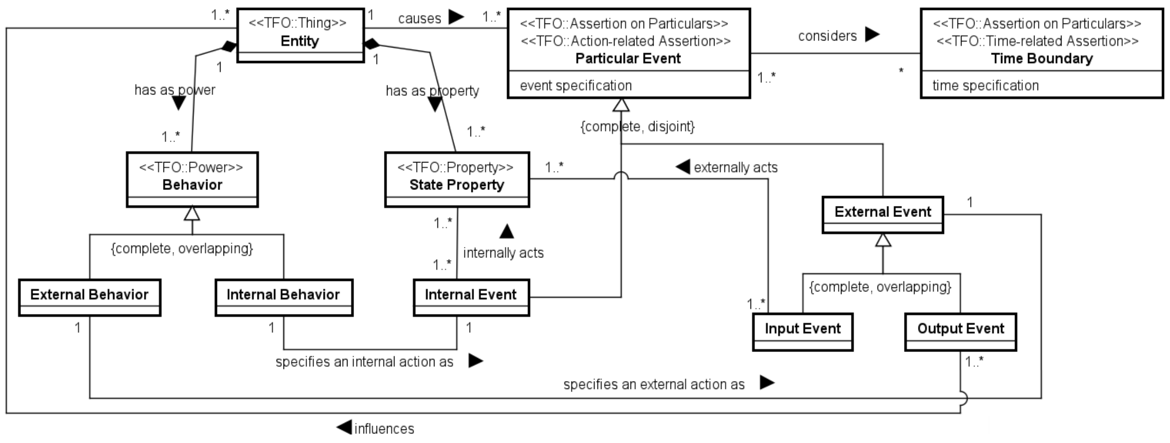

Figure 3: The *PEvent CO* ontology. Note that TFO stands for *Thing FO*. Stereotypes are used to show how the new concepts semantically enrich other concepts defined at the foundational level in *Thing FO*.

An *Entity* represents a particular or concrete, tangible or intangible object, for which a dynamic behavior is defined explicitly using its *Properties* and *Powers*. *Entity* has the semantics of *Thing* (from the *Thing FO* ontology) implying that an *Entity* is not a particular object without its *Properties* and *Powers*. At the foundational level, the tuple *Thing-Property-Power* defines a unity that can be either *i)* semantically enriched at the CO, TDO, and LDO levels or *ii)* instantiated at the IO level. At the *PEvent CO* ontology, the *Properties* are stated explicitly as the *State Properties* of an *Entity*. *State Property* refers to the intrinsic state structure of a particular dynamic *Entity*. On the other hand, the *Powers* are stated explicitly as the *Behaviors* of an *Entity*. *Behavior* refers to how a particular *Entity* behaves under established conditions. Hence, the structural part of an *Entity* definition is centered on how its *Behaviors* change its *State Properties*, ideally, over time. Since "things have properties, these properties instantiate […] acting powers, and this ensemble of things, properties and powers cause any events that might occur" (Fleetwood 2009, p. 12), the *PEvent CO* ontology introduces the notion of *Particular Event*. A *Particular Event* is an *Assertion on Particulars* and, at the same time, an *Action-related Assertion* that explicitly states and specifies the occurrence of an *Entity* action. The *Particular Event* mechanisms should consider *Time Boundaries* besides the changes or queries used to support the states of *Entities* (detailed as *State Properties*). A *Time Boundary* is a *Time-related Assertion* specifying temporal limits and restrictions from which a *Particular Event* or series of *Particular Events* can be related and modeled. Hence, the dynamic part of an *Entity* definition is given by the tuple *Entity-Behavior-State Property-Particular Event* (i.e., a *Particular Event* determines the occurrence of a concrete object action in an explicit context or place during a range or at an instant of time).

At this point, it is important to denote how events may take place. The *PEvent CO* ontology considers two types of *Behaviors*: *Internal Behavior* and *External Behavior*. *Internal Behavior* is a type of *Behavior* that refers to what a particular *Entity* can do to act over its *State Properties*. *External Behavior* is a type of *Behavior* that refers to the occurrence of external actions over an *Entity*. Then, *Internal Behavior* refers to internal actions fired from an Entity over itself. If a *Particular Event* explicitly states and specifies the occurrence of an action acting over the *State Properties* of such an *Entity*, it is called an *Internal Event*. On the contrary, an *External Event* is a *Particular Event* that explicitly states and specifies the occurrence of an external action on an *Entity*. Two types of *External Events* are included: *Input External Event* and *Output External Event*. An *Input Event* is an *External Event* that explicitly states and specifies the occurrence of an external action that acts over the *State Properties* of an *Entity* according to its *External Behavior*. An *Output Event* is an *External Event* that explicitly states and specifies the occurrence of an external action having some impact on other *Entities*. That is, the entities cannot produce an impact between them directly. The *Behavior* of an *Entity* can produce a new event that may result in a state change in another *Entity* depending on how the latter reacts to the input event using its own *Behavior*.

The correspondence of the *PEvent CO* elements and elements defined at the immediately higher level (i.e., *Thing FO*) was specified also using a set of axioms. Such axioms were formalized in first-order logic. For example, the model includes an axiom defining that *"an entity that causes a particular event is a thing that defines an assertion on particulars"* as follows:

$$\forall \text{e}, \forall \text{ev}: [\ \text{Entity(e)} \wedge \text{ParticularEvent(ev)} \wedge \text{causes(e,ev)} \rightarrow \text{defines(e,ev)}] \tag{1}$$

Equation (1) maps the *causes* relationship (defined at *PEvent CO* level between an *Entity* and a *Particular Event*) as the *defines* relationship (defined at the *Thing FO* level between a *Thing(e)* and an *AssertionOnParticulars(ev)*). A set of twelve axioms was defined as part of *PEvent CO* to ensure an accurate integration with the foundational level.

Due to *rule ii)* defined at the FCD-OntoArch, *PEvent CO* allows the definition of event-based behaviors for all concepts of ontologies placed at the same level based on the notion of *Thing* (e.g., *Target Entity* and *Context Entity* from *Situation CO*, and *Work Entity* and *Product Entity* from *Process CO*). For ontologies placed below *PEvent CO*, the concepts can be semantically enriched by the notion of *Entity*.

The *PEvent CO* ontology of the FCD-OntoArch structure has been implemented in Telos language using ConceptBase (Koubarakis et al. 2021) as a software tool. Telos is a conceptual modeling language for representing knowledge regarding information systems based on object-oriented technology, integrity constraints, and deductive rules. Hence, the language has been used to define formally the concepts, relationships, and axioms required for instantiating a model at the Instance Ontological Level using the event-based modeling centered on *PEvent CO*. Such instances are stored on a ".sml" file.

3.2 The Multi-Level Layered Conceptualization of DEVS

DEVS models are mathematically defined, but their simulation is performed by concrete DEVS simulation systems. Given that DEVS is an abstract formalism independent of any particular implementation, when engineers want to simulate DEVS models, they need to program them in the input language of a concrete simulator, which means writing code in Java or C++, or another general-purpose programming language (Cristiá et al. 2019). Moreover, since there is no common DEVS format used by all the software tools, DEVS modelers are tied to their tools (Van Tendeloo and Vangheluwe 2017). To address these issues, Blas et al. (2021) have proposed a set of layers based on the practical implementation of DEVS formal models as part of a multi-level conceptualization. These levels were designed following the principles of MDE.

MDE is a software development methodology that focuses on creating and exploring domain models as conceptual models of all topics related to a problem-specific domain (Schmidt 2006). The methodology differentiates three types of models as engineering artifacts: domain models, platform-independent design models, and platform-specific implementation models. Supporting MDE, metamodels are used along with model transformations to provide a process that enables the automated development of new artifacts (Brambilla et al. 2017). The MDE principles have been applied in the M&S field for several years to produce well-structured and maintainable simulation models by increasing the level of abstraction through well-defined representations (Cetinkaya et al. 2011; Sarjoughian et al. 2015; Dalmasso et al. 2023).

In this regard, the practical views of the multi-level layered conceptualization of DEVS were defined as follows (Figure 4): *i) formalization* is the level that offers the DEVS formalism as a domain language allowing to define a DEVS formal simulation model (atomic or coupled) for any event-based system; *ii) platform-independent implementation* is the level focused on the common concepts included in most Object-Oriented (OO) implementations of DEVS software tools; and *iii) platform-specific implementation* is the level that provides a concrete realization of the platform-independent implementation that can be deployed in a target M&S environment based on an OO programming code.

Each layer was designed as an independent modeling level. Layers *i)* and *ii)* are supported by metamodels. In Figure 4, these metamodels are depicted as *DEVS Formal Specification (DFS)* for level *i)* and *DEVS OO-Implementation (DOI)* for level *ii)*. A detailed version of the *DFS* metamodel can be found in (Blas and Gonnet 2023). Across these metamodels, traceability relationships are included (i.e., the *trace* association). Such a traceability relationship implies two concepts (placed at different modeling levels) define the same element (i.e., a piece of a DEVS model) from two distinct perspectives. Linking concepts from different metamodels allows us to obtain "new knowledge" from the DEVS model definition. On the other hand, the modeling level placed at layer *iii)* is linked to OO programming languages, allowing the introduction of DEVS implementations as Java or C++ code. Hence, the structure of the conceptualization allows for defining model-to-model and model-to-code transformations among distinct modeling levels employing traceability associations. Now, the complete definition of a DEVS simulation model *D* is given by the set of models instantiated from the conceptual modeling levels as *D = {formalization model, platform-independent implementation model, platform-specific implementation code}*. The metamodels required to support the conceptualization depicted in Figure 4 were implemented with the existing modeling technology of the Eclipse platform (The Eclipse Foundation 2024). Such metamodels were designed as a set of UML diagrams restricted by OCL constraints. An instance of the metamodel is stored in an *xmi* file.

3.3 The Reasoning Engine

The reasoning engine that upgrades the abstraction model to an implementation model founded on a DEVS formal model definition is based on an algorithm. Such an algorithm was implemented in Java (Figure 1). The algorithm starts defining the high-level structure of the DEVS model. Based on an *Instance Ontology*, the reasoner detects the main component with the semantic of *Thing* composed of several elements of type *Entity*. Such a component is upgraded to a *CoupledModel*. Then, the basic mapping is centered on transforming each *Entity* into an *AtomicModel*. A pseudocode detailing the main steps of the algorithm described below can be found here.

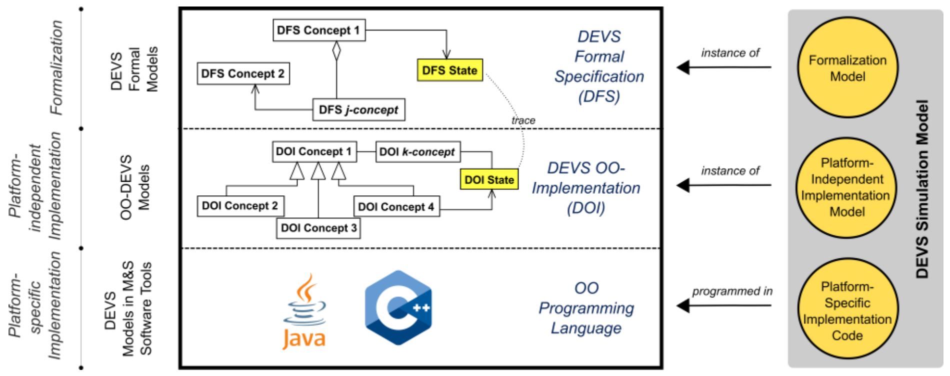

Figure 4: Multi-level layered conceptualization of DEVS proposed in Blas et al. (2021). Each metamodel is sketched as a set of concepts to show domain independence. As an interoperability example, both include the *State* concept. Even though both *States* refer to the same simulation model, each level employs the concept differently (following the intended meaning at the modeling level).

For each *AtomicModel* required, an *AtomicStructuralPart* and an *AtomicBehavioralPart* are created. These parts are concepts included in the *DFS* metamodel to classify the tuples used during formalization. Over the *AtomicStructuralPart*, the *State* is defined using all the *State Properties* of an *Entity*. Each *State Property* (linked by the predicate *hasAsProperty* to the *Entity*) is used to create a *StateVariable*. The structure of *StateVariables* is defined following the *structural description* detailed at the ontological level. On the other hand, around the *AtomicBehavioralPart*, the required functions *InternalTransitionFunction*, *ExternalTransitionFunction*, and *OutputFunction* are defined with the following guidelines:

- For the *InternalTransitionFunction*, the predicate *hasAsPower* related to the *Entity* is used. Taking the *Event* defined with the *specifiesAnInternalActionAs*, the *NextStateSpecification* is built. Such a state specification is designed following the *StateProperties* over which the *Event* is linked by the *internallyActs* predicate.
- For the *OutputFunction*, both *causes* and *influences* predicates are used. At this point, the *causes* predicate is linked to the actual *Entity*, while the *influences* predicate links the actual *Entity* to a secondary *Entity*. Then, an *OutputSpecification* is defined. To complete such a specification, an *OutputPort* is created (if it does not exist).
- For the *ExternalTransitionFunction*, the algorithm uses the *InputEvents* that are linked by the *externallyActs* predicate to the actual *Entity*. Such a relationship is based on the notion that an *InputEvent* is caused by an external *Entity* over the actual *Entity* (i.e., due to the *causes* predicate). Then, if it does not exist, an *InputPort* is created. Also, the *NextStateSpecification* is defined using the *event specification* property and the *StateProperties* over which the event *internallyActs*.

Once all the *AtomicModels* are defined, couplings are built. At this point, *ExternalInputCouplings* and *ExternalOutputCouplings* are defined following the location of the *Entities* on their containers (i.e., components with the semantic of *Thing* identified during step 1). For *InternalCouplings*, the *OutputEvents*, along with the *causes* and *influences* predicates, are used to create an *InternalCoupling* between two Entities (the one that *causes* the event and the one that is *influenced* by it). Finally, over each *CoupledModel*, a *CoupledStructuralPart* is created to collect all *AtomicModels* and couplings defined in it.

4 RESULTS AND DISCUSSION

4.1 Case Study: Using M&S for Health Patient Monitoring

To show how a real-world abstraction can be defined using the ontological part of our framework, we employ a case study centered on monitoring health patients with Tuberculosis in Argentina. Tuberculosis

(TB) surveillance incorporates all the components that involve the management of the person with TB from diagnosis to the completion of treatment (World Health Organization 2012). Any healthcare provider (or health service) that detects TB cases must notify the World Health Organization. Case detection means that TB is diagnosed in a patient and is reported within the TB surveillance system to receive a TB treatment. Each detected case should be classified according to the history of previous treatment (if exists). According to the Ministry of Health (Secretaría de Salud 2020), the tracking required is based on studying new, relapsed, and transferred (e.g., due to relocation) patients or, in worst cases, patients with lost tracking (i.e., patients that do not return to the health center) or with a failed treatment. Once a treatment has started, information about treatment is updated to follow up on the patient status as follows: treatment completed, patient cured, patient follow-up missed, patient deceased, treatment failed, patient transferred or unknown.

Considering the real-life system, several parts can be identified (e.g., healthcare provider, patient, treatment, and so on). However, when a research question is proposed for studying such a system with an M&S approach, an abstraction model (based on that question) can be defined. For our case study, the research question proposed was "How effective are the healthcare providers on the follow-ups of patients?". Following such a basis, we define an abstraction model *IOTBSurveillance* as a new ontology on the *Instance* level of FCD-OntoArch. The main characteristics of such a model (stored as *IOTBSurveillance.sml*) were the following:

- The *TB Surveillance System* is based on a *Particular Situation* (from *Situation CO*) including two entities: *Patient* and *Healthcare provider*. The last two have the semantic meaning of *Target Entity* (from *Situation CO*) and Entity (from *Event CO*).
- The *Healthcare provider* manages diagnosis and treatment of *Patients* (i.e., produces the laboratory exam to get a diagnostic, if the diagnostic is positive for TB produces a treatment, and so on). A simplification introduced at this level of abstraction is that a *Patient* starting a new treatment on a *Healthcare provider* is not attached to its previous history.
- A *Patient* who starts treatment is classified by the *Healthcare provider* as one of the following cases: NEW, RELAPSE, TRANSFERRED, LOST TRACKING, or FAILURE.
- A *Patient* who ends a treatment is classified by the *Healthcare provider* as one of the following cases: TREATMENT COMPLETED, CURED, FOLLOW-UP MISSED, DECEASED, FAILED, TRANSFERRED or UNKNOWN.

The engine detailed in Section 3.3 was executed over the *IOTBSurveillance* model to get a partial formalization of the DEVS model representing such a situation. Then, the *DEVSTBSurveillance* model was created. Such a model was stored in the *DEVSTBSurveillance.xmi* file. As a result, a DEVS simulation model structured following the formalization metamodel was obtained. Both models can be found here. Figures 5 and 6 show a representation of both models using diagrams. For space reasons, the figures only include part of the models.

Even when the description includes only two components (patient and healthcare provider), the number of elements defined in the abstraction model is not small. Then, this model allows us to show how all these elements are upgraded to discrete-event elements that follow the formal definition of DEVS models. Of course, a complete formalization of such an abstraction is not possible due to the level of detail involved in the ontology regarding, for example, set values. Even so, a 73% definition is available directly from the algorithm. Such a percentage is obtained as *number of concepts instantiated during the algorithm execution / total number of concepts required*. Over such a DEVS definition, the modeler can complete the missing elements. Hence, an easier introduction to DEVS is encouraged with our framework.

4.2 Strengths and Limitations

Robinson (2019) identifies "grand challenges" in conceptual modeling for discrete-event simulation in the following areas: conceptual modeling itself, developing conceptual modeling frameworks, and the representation of conceptual models. Our two-fold modeling framework helps in this direction by promoting the design of the conceptual model directly from an abstraction model well-structured according to an ontological foundation.

The main benefit obtained from establishing the ontological foundations of the core concepts of a real-world abstraction model used for M&S purposes is a clarification of its semantics as a whole, as complete as we can, through a large-scale structure that allows combining several domains at multiple levels according to their ontological meaning (in philosophy). Therefore, the ontological architecture encourages modularity, extensibility, and reuse of ontological elements at different levels. That follows the well-known contributions of ontology-based systems (Yang et al. 2019), such as enabling interoperability and communication among multiple disciplines, describing concepts and their relationships explicitly and accurately to avoid incompleteness and ambiguity, providing core and basic concepts as a reference to describe other concepts, and sharing a common understanding of a domain (in this case, an abstraction model for M&S). In this way, our framework allows building abstraction models for complex scenarios by reusing existing foundational and core models in new domains or enriching the structure with missing core models to improve the real-world view.

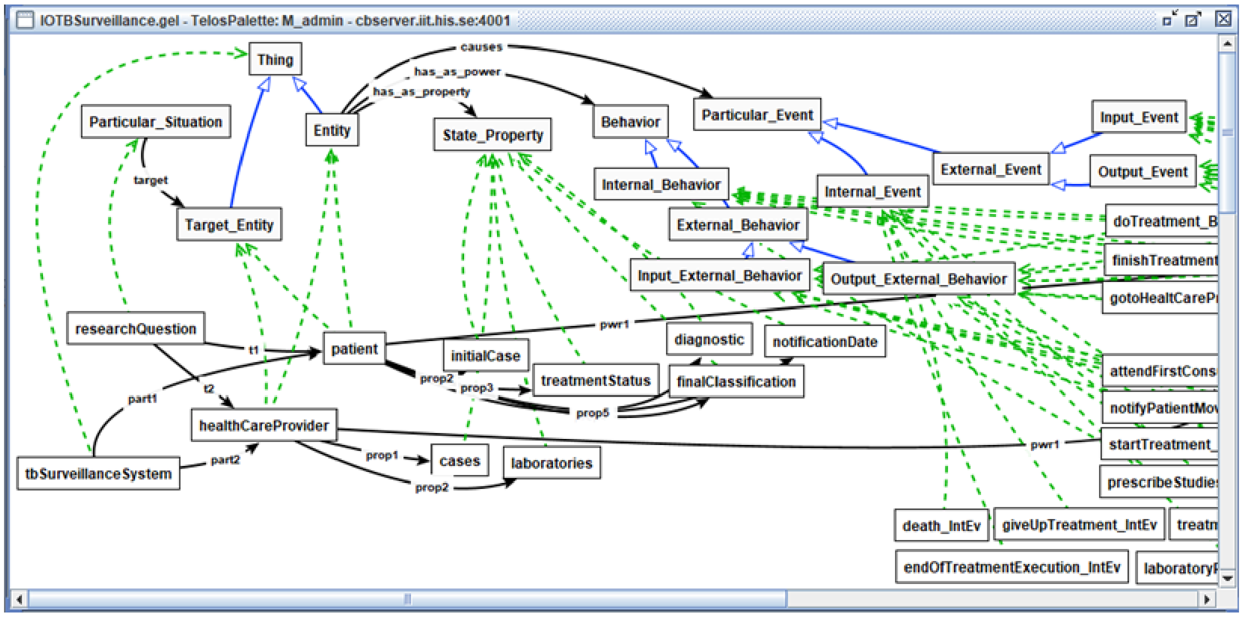

Figure 5: Part of the situation defined in the *IOTBSurveillance* represented directly from ConceptBase.

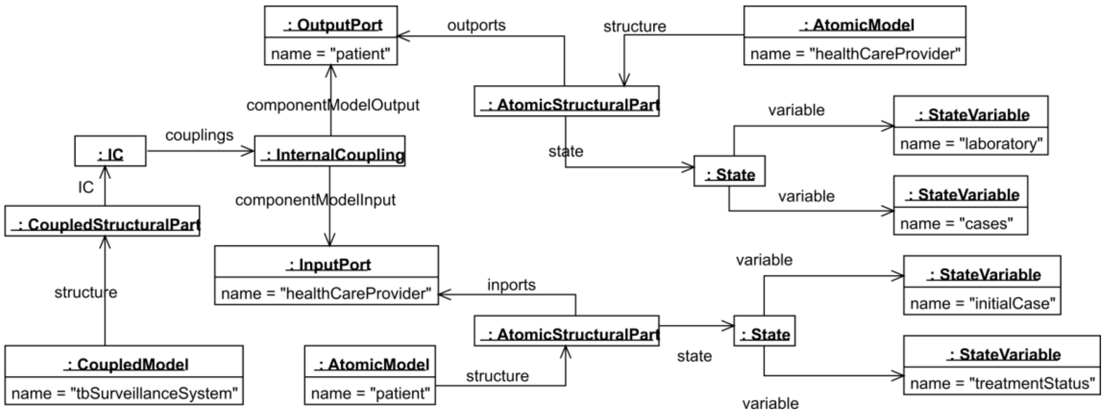

Figure 6: Part of content created in the *DEVSTBSurveillance.xmi* file represented as a class diagram describing the DEVS structural parts instantiated from the model detailed in Figure 5.

On the other hand, our research group has studied DEVS formalism as a common language for describing and handling discrete-event simulation models for several years (Blas et al. 2017, Blas et al. 2021, Blas and Gonnet 2023, Blas et al. 2023, Dalmasso et al. 2023). We are aware that there are several issues when implementing DEVS models from scratch. By using abstraction models defined over a modeling "template" (i.e., the ontological support), a common guide for formalizing DEVS models is provided. Our proposal centered on the MDE approach allows "navigating" from one modeling level to the other using translation rules defined over the different DEVS metamodels.

In summary, the resulting modeling framework leads mainly to *i)* exploiting the reuse capabilities of models at a semantic level allowing implementation compatibilities; *ii)* validating the consistency among discrete dynamic system descriptions and their DEVS formal models; and *iii)* verifying the consistency among DEVS formal models and their simulation implementations developed using specific software tools.

5 CONCLUSIONS AND FUTURE WORK

Modeling and *Simulation* are distinct activities. *Modeling* is the process of developing and using abstractions to simplify the real world for focused analysis, while *simulation* subsumes modeling activities and focuses on model execution in a simulator. In the *modeling* activity, the question "*What do we model?*" should be answered. During the *simulation* activity, such a question is reformulated to "*How do we model what needs to be modeled?*". Our motivation is to reduce the effort related to the modeling task on both parts of the M&S process. We adopt ontology-based modeling for the modeling part (i.e., for answering "*What do we model?*") and metamodel-based modeling for the simulation part (i.e., for answering "*How do we model what needs to be modeled?*"). We promote DEVS as a formal specification for defining simulation models (i.e., DEVS acts as a vehicle to describe how simulation models should be defined for studying "*what needs to be modeled*"). Hence, although the latter is presented as a particular case (we use DEVS, but ideally, you may use any other formalism or software tool for implementing the simulation models), the former is the key to representing "*what needs to be modeled*".

Even when it would seem unlikely that a single, accepted framework could emerge for studying any available domain (given that each domain has its own properties), we strongly believe that our two-fold proposal can help in this direction. By linking two different modeling strategies (ontologies and metamodels) on a single structure, we encourage simulationists to use real-world focused descriptions as a basis for building their simulation models. In future work, we want to improve our framework for including information that helps (besides *abstract and implementation modeling*) during the validation process.

REFERENCES

Becker, P., M. F. Papa, G. Tebes, and L. Olsina. 2021. "Analyzing a Process Core Ontology and Its Usefulness for Different Domains" In *QUATIC 2021: Quality of Information and Communication Technology*, edited by A. C. R. Paiva, A. R. Cavalli, P. Ventura Martins, and R. Pérez-Castillo, 183-196. Switzerland: Springer.

Blas, M. J., and S. Gonnet. 2023. "Modeling and Simulation Through the Metamodeling Perspective: The Case of the Discrete Event System Specification". In *Handbook of Model-Based Systems Engineering*, edited by A.M. Madni, N. Augustine, and M. Sievers, 1189-1228. Cham: Springer.

Blas, M. J., S. Gonnet, P. Becker, and L. Olsina. 2022. "A Core Ontology for the Representation of Entities with Event-based Behaviors". *Electronic Journal of SADIO*, 21(2):17-41.

Blas, M., S. Gonnet, D. Kim, and B. Zeigler. 2023. "A Context-Free Grammar for Generating Full Classic DEVS Models". In *2023 Winter Simulation Conference (WSC)*, 2579-2590. https://doi.org/10.1109/WSC60868.2023.10407991.

Blas, M., S. Gonnet, and H. Leone. 2017. "Routing Structure over Discrete Event System Specification: A DEVS Adaptation to Develop Smart Routing in Simulation Models". In *2017 Winter Simulation Conference (WSC)*, 774-785. https://doi.org/10.1109/WSC.2017.8247831.

Blas, M., S. Gonnet, and B. Zeigler. 2021. "Towards a Universal Representation of DEVS: A Metamodel-Based Definition of DEVS Formal Specification". In *Proceedings of the 2021 Annual Modeling and Simulation Conference*, July 19th-22nd, Fairfax, USA, 1-12.

Brambilla, M., J. Cabot, and M. Wimmer. 2017. *Model-Driven Software Engineering in Practice*, 2nd ed. Switzerland: Springer.

Cetinkaya, D., A. Verbraeck, and M. D. Seck. 2011. "MDD4MS: A Model-Driven Development Framework for Modeling and Simulation". In *Proceedings of the 2011 Summer Simulation Conference*, June 27th-30th, Hague, Netherlands, 113-121.

Cristiá, M., D. A. Hollmann, and C. Frydman. 2019. "A Multi-target Compiler for CML-DEVS". *Simulation* 95(1):11-29.

Dalmasso, F., M. J. Blas, and S. Gonnet. 2023. "Enriching UML Statecharts through a Metamodel: A Model Driven Approach for the Graphical Definition of DEVS Atomic Models". *IEEE Latin America Transactions* 21(1):27-34.

Fleetwood, S. 2009. "The Ontology of Things, Properties and Powers". *Journal of Critical Realism*, 8(3):343–366.

Gruber, T. A. 1993. "A Translation Approach to Portable Ontology Specifications". *Knowledge Acquisition* 5(2):199–220.

Guizzardi, G. and G. Wagner. 2010. "Towards an Ontological Foundation of Discrete Event Simulation". In *2010 Winter Simulation Conference (WSC)*, 652-664. https://doi.org/10.1109/WSC.2010.5679121.

Koubarakis, M., A. Borgida, and P. Constantopoulos. 2021. "A Retrospective on Telos as a Meta-modeling Language for Requirements Engineering". *Requirements Engineering* 26(1):1-23.

Miller, J. A., G. T. Baramidze, A. P. Sheth, and P. A. Fishwick. 2004. "Investigating Ontologies for Simulation Modeling". In *Proceedings of the 37th Annual Simulation Symposium*, April 18th–22nd, Arlington, USA, 55-63.

Olsina, L. 2021. "Applicability of a Foundational Ontology to Semantically Enrich the Core and Domain Ontologies". In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, October 25th-27th, Lisbon, Portugal, 111–119.

Olsina, L. 2023. "The Foundational Ontology ThingFO: Architectural Aspects, Concepts, and Applicability". In *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, edited by A. Fred, D. Aveiro, J. Dietz, J. Bernardino, E. Masciari, and J. Filipe, 73-99. Cham: Springer.

Orgun, M. A. and T. Meyer. 2008. "Introduction to The Special Issue on Advances in Ontologies". *Expert Systems: The Journal of Knowledge Engineering*, 25(3):175–178.

Robinson, S. 2014. *Simulation: The Practice of Model Development and Use*. 2nd Ed. New York: Palgrave MacMillan.

Robinson, S. 2019. "Conceptual Modelling for Simulation: Progress and Grand Challenges". *Journal of Simulation*, 14(1):1-20.

Ruy, F. B., R. A. Falbo, M. P. Barcellos, S. D. Costa, and G. Guizzardi. 2016. "SEON: A Software Engineering Ontology Network". In *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management*, edited by E. Blomqvist, P. Ciancarini, F. Poggi, F. Vitali, 527-542. Cham: Springer.

Sarjoughian, H. S., A. Alshareef, and Y. Lei. 2015. "Behavioral DEVS Metamodeling". In *2015 Winter Simulation Conference (WSC)*, 2788-2799. https://doi.org/10.1109/WSC.2015.7408384.

Schmidt, D. C. 2006. "Model-driven engineering". *Computer*, 39(2):25-31.

Secretaría de Salud. 2020. Instrucciones para la Notificación de Tuberculosis al Sistema Nacional de Vigilancia de la Salud SNVS 2.0. https://bancos.salud.gob.ar/sites/default/files/2020-10/instrucciones-notificacion-tuberculosis-SNVS2.0.pdf, accessed 4th April.

Silver, G. A., J. Miller, M. Hybinette, G. Baramidze, and W. S. York. 2011. "An ontology for discrete-event modeling and simulation". *Simulation*, 87(9):747-773.

Szabo, C., R. Castro, J. Denil, and S. Sanchez. 2023. "Resilience and Complexity in Socio-Cyber-Physical Systems". In *2023 Winter Simulation Conference (WSC)*, 660-670. https://doi.org/10.1109/WSC60868.2023.10408660.

The Eclipse Foundation. 2024. "Eclipse modeling project. Eclipse modeling framework". https://www.eclipse.org/modeling/emf/, accessed 27th March 2024.

Tebes, G., L. Olsina, D. Peppino, and P. Becker. 2021. "Specifying and Analyzing a Software Testing Ontology at the Top-Domain Ontological Level". *Journal of Computer Science and Technology*, 21(2):126–145.

Turnitsa, C., J. J. Padilla, and A. Tolk. 2010. "Ontology for Modeling and Simulation". In *2010 Winter Simulation Conference (WSC)*, 643-651. https://doi.org/10.1109/WSC.2010.5679124.

Van Tendeloo, Y. and H. Vangheluwe. 2017. "An Evaluation of DEVS Simulation Tools". *Simulation* 93(2):103-121.

World Health Organization. 2012. Electronic Recording and Reporting for Tuberculosis Care and Control. https://www.who.int/publications/i/item/9789241564465, accessed 4th April.

Yang, L., K. Cormican, and M. Yu. 2019. "Ontology-based systems engineering: A state-of-the-art review". *Computers in Industry*, 111(1):148-171.

Zeigler, B. P. 2019. "How Abstraction, Formalization and Implementation Drive the Next Stage in Modeling and Simulation". In *Summer of Simulation*, edited by J. Sokolowski, U. Durak, N. Mustafee, and A. Tolk, 25-37. Switzerland: Springer Nature.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*. 3rd ed. London: Academic Press.

AUTHOR BIOGRAPHIES

MARIA JULIA BLAS is an Assistant Researcher at INGAR and an Assistant Professor at UTN. She received her Ph.D. degree in Engineering from UTN in 2019. Her research interests include UML modeling and discrete-event M&S. Her email address is mariajuliablas@santafe-conicet.gov.ar.

SILVIO GONNET received his Ph.D. degree in Engineering from UNL in 2003. He currently holds a researcher position at CONICET. His research interests are models to support design processes and conceptual modeling. His email address is sgonnet@santafe-conicet.gov.ar.