On the Use of Ontologies for Defining, Generating and Exploring the Resulting Simulations of Application Level Protocols

Mieczyslaw M. Kokar[1][a] and Jakub J. Moskal[2][b]
[1]*Department of Electrical and Computer Engineering, Northeastern University, 360 Huntington Avenue, Boston, U.S.A.*
[2]*VIStology, Inc., Mashpee, MA, U.S.A.*

Keywords: Generic Simulation System, Ontologies, Simulation Requirements, Simulation Generation, Application Level Protocol, Simulation Querying and Exploration.

Abstract: This paper presents a simulator generator designed to aid in the development of domain-specific protocols that enable semantic communication, where messages include annotations specifying the meaning of individual fields. Furthermore, the field types are dynamic, meaning they are incorporated into messages based on the domain's underlying ontology. To experiment with such domain-specific semantic protocols, designers require data for protocol evaluation. Simulation is a common solution to this need. Simulations are produced by simulators—software systems that take inputs and generate results. For typical applications, generic simulators are often available. This paper introduces a system, named *dg*, that generates simulators for semantic communication protocol designers, based on specifications (inputs and constraints) provided in an ontology. Additionally, we demonstrate how the same ontology can be used to explore simulation results. Finally, since our approach involves policies that account for information uncertainty in both simulator implementation and result querying, we propose initiating an effort to develop an uncertainty-related extension to the SPARQL query language.

1 INTRODUCTION

When developing a communication protocol tailored to a specific domain, experimental data is often needed to evaluate the effectiveness of the protocol. This data is typically generated through simulation. One approach to ensuring that the simulation is reliable is to adhere to established systems engineering principles (Adcock, 2007). The first step is to create detailed specifications for the simulation system, followed by analyzing these specifications, then designing and developing the system in a way that satisfies all the requirements. Additionally, the development process must account for the adequacy of the model used by the simulator to generate inputs and outputs. Overall, designing a simulator for a system under development is a complex and resource-intensive task, requiring significant time and human effort.

In some cases, the simulators already exist and can be used with minor modifications to the simulator system parameters. These simulators are typically available for common scenarios. However, this paper focuses on scenarios that require simulations tailored to specific requirements, for which no suitable simulators currently exist. Thus, our focus is to streamline the simulation development process. To achieve this objective, we are pursuing a simulator generator approach. We have developed a simulator generator, *dg*, which can accept specification requirements for a simulator, instantiates the specified simulator, and generates the simulation results according to the specification. The heart of this generator is an ontology and an ontology reasoner (interpreter) that takes specifications expressed in ontological terms and generates simulators accordingly. Another aspect that is covered in this paper is the support for the comprehension of the results generated by the simulator. This objective is also achieved by the use of the ontology. Ontology provides language that both the user and the ontological reasoner can understand. The user can issue queries using ontological terms and the queries are interpreted and answered by the query engine that *dg* interacts with.

This paper is structured as follows. First, in Section 2 we provide a brief overview of the works that

[a] https://orcid.org/0000-0001-9243-3089
[b] https://orcid.org/0000-0001-6467-773X

Kokar, M. and Moskal, J.
On the Use of Ontologies for Defining, Generating and Exploring the Resulting Simulations of Application Level Protocols.
DOI: 10.5220/0013066700003838
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2024) - Volume 2: KEOD*, pages 227-234
ISBN: 978-989-758-716-0; ISSN: 2184-3228
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

have addressed the problem of simulations, especially those that use ontologies to express the specifications of the required capabilities and their attributes. Section 3 presents a class of problems that are covered by our simulator generator. Section 4 describes a scenario that we are using to show the functionality of our system. Section 5 provides an overview of the ontologies for representing knowledge of both the domain introduced in Section 4 and the ontologies for representing the more generic modeling aspects, including a top-level ontology for representing systems. The structure of *dg* is introduced in Section 6. In Section 7, we show examples of simulation results that are outputs of the simulator generated by *dg* for the scenarios described in Section 4. This is followed by the demonstration of the querying capabilities for supporting comprehension of simulation results in Section 8. Finally, in Section 9, we describe the conclusions from this study and the potential directions of future research.

2 LITERATURE REVIEW

Simulation is a well-established discipline of both research and engineering. Simulation has gained an exponential recognition with the occurrence of the Digital Twin technology (Grieves, 2014). Conceptually, it is part of Systems Engineering. It is typical to start development of software for controlling engineered systems by simulating their core functionality - the decision derivation function, especially in the domain of Cyber Physical Systems (CPS) (Koulamas and Kalogeras, 2018).

Since the focus of this paper is the generation of simulators from models of the simulated process, where the model is expressed as an ontology, it would be advantageous to use systems engineering standards in the development of such simulators and ontologies, and for evaluation of simulation systems. Towards this end, some researchers relied on the ISO/IEC/IEEE 15288 (Systems and Software Engineering, System Life Cycle Processes) standard (ISO, 2023). However, it does not address simulation directly.

Some researchers (cf. (Barth et al., 2023)) relied on the ISO/IEC 25010 standard related to the quality of software systems. The authors have proposed a number of criteria for assessing the quality of simulation models. Since they could not identify such criteria in the existing standards, they adopted the attributes of the Software Product Quality Model that are part of the ISO 25010 standard (ISO-IEC, 2024).

The work on standardization related to simulation started at The Society for Modeling & Simulation International. It eventually has moved to the IEEE resulting in the Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) (IEEE, 2000), still being continued, and the IEEE Standard for Distributed Interactive Simulation – Application Protocols (IEEE, 2012), which focuses on the distributed processing aspect of simulations. However, those standards in progress are not much used in the current research on the construction of simulation systems. For instance, (Liu et al., 2020) propose a process for developing an ontological framework for digital twin modeling of CNC machine tools. However, the ontology shown in this paper is relatively generic and not based on any specific standards.

(Singh et al., 2021) focuses on identifying the minimum data structure to model data for DT using ontologies to define the semantics, restrictions and data structure for DT to domain applications. It proposes a multi-step process to develop an ontology and then discusses a case study of modeling data for a specific use case. The ontological model is mapped to a relational data representation for further processing.

(Van Ruijven, 2013) proposes an ontology for systems engineering, by means of a set of information models based on a modeling methodology based on a simplification of the ISO 15926 part 2 data model. It was implemented using an RDF tool. The future version is supposed to be implemented in OWL; the RDF version is not sufficient for the approach we are presenting in this paper.

(Jeleniewski et al., 2023) presents a semantic model developed to articulate the interdependencies between process parameters in manufacturing settings. It promotes the use of Ontology Design Patterns developed by the Semantic Web community to support the reusability of ontologies (Hildebrandt et al., 2020). It also incorporates the OpenMath standard, another sign of commitment to the standards based development. Consequently, the approach is very well aligned with some of the standards, although not the simulation standards. While it is focusing on the representation of processes, it does not consider a use case that is directly relevant to our approach.

(Reif et al., 2023) describes the use of ontologies for the use case of composing multiple simulations to achieve a simulation objective. This is a very interesting idea, although it is a composition of simulation results, rather than composition of simulation processes into one simulation process, which would be much more in line with the focus of this paper.

In summary, we were somewhat successful in identifying standards for modeling simulations, although none of them were satisfactory for our needs since none of them were designed for the purpose of generating simulators. Similarly, we have identified ontologies for representing simulations, however, similarly to the standards, they were not developed for the use case as in our approach. In effect, we put emphasis on the concept of ontological representations of inputs to the simulators and constraints relevant to the domain of our application. The approach presented in this paper does not cover such a wide scope as some of the other approaches, yet it is able to successfully generate simulators for a relatively narrow domain of simulation problems.

3 THE CLASS OF PROBLEMS

The objective of the simulated process discussed in this paper is to establish the priorities of the messages to be transmitted by an overloaded communication channel, assuming that the highest priority messages will be selected first. Consider a time slot S and a set of messages to be transmitted $\{m_1, \ldots, m_s\}$. Moreover, assume the messages include the semantic content that can be classified along k aspects (field types). These fields will be referred to as *variables*: $X_1, \ldots, X_k$. The objective is to order the messages for transmission based on a user-defined *policy*. The policy consists of two parts. First, the user defines the priorities for the specific values for each of the variables, $f_i(x_{ij}), x_{ij} \in X_i$. This is feasible for the problem considered in this paper since the variables are all discrete, i.e., the values of the variables are all objects. Second, the user defines preferences for the specific field types; they represent the importance of the specific field type to the user. The preferences are defined by the policy and represented as weights, $\bar{w} = (w(X_i), \ldots, w(X_k))$.

This problem can be formalized as a Multi-Objective Constrained Optimization Problem (MOCOP) in which the priorities for each of the variables are treated as optimization objectives, i.e., the objective is to select the messages for transmission in such a way that they maximize (or minimize) the specific objectives functions for each of the variables. However, since the particular objectives may push the selection in opposite directions - what is good for one variable may be bad for another - there is a need to reconcile the possibly conflicting objectives. One possible approach to solving this problem is to select solutions that are *Pareto optimal*, i.e., such that none of the objective functions can be improved in value without degrading some of the other objective values.

In general, the problem of finding Pareto-optimal solutions is NP-hard, which is one of the reasons that we are simulating a simpler problem. We follow an approach to the reconciliation of the multiple optimization objectives using the so-called *utility function* which prioritizes particular objectives. In our implementation we used the *linear scalarization function* $\sum_{i=1}^{n} w_i(f_i(x))$. This function is used by the Simulator to solve the message selection decision. It is effective and guaranteed to find Pareto optimal solutions when the Pareto front is convex, but it can fail when the Pareto front is non-convex. Proper selection of weights is crucial, and multiple scalarization runs with different weights are often needed to approximate the Pareto front comprehensively. This is one of the justifications for developing support for analyzing multiple policies for a given domain.

Since the weights for the variables all add to 1 and thus they can be treated as probabilities for computing the decisions. The objective of the simulated process is thus to compute the expected value of the decision function for all possible hypotheses and select the one that gives the highest expected value, as shown in Equation 1, achieving the objective of the message selection decision, m.

$$max_{x \in X} \sum_{i=1}^{n} w_i f_i(X_i(x)) \tag{1}$$

The solution to this problem is subject to various constraints on both the values and, even more importantly, the relations among the particular variables. In other words, the solution defined in Equation 1 is subject to a set of constraints, $C(X)$, which are not shown explicitly in this formulation, although they are embedded in the domain ontology.

4 SCENARIO

We are demonstrating our approach on the scenario shown in Figure 1. It involves communication between Nodes, where Sources send messages to Destinations, starting at the Start time and ending at the End time. It is important to stress that only one communication channel exists whose capacity is mostly insufficient to accommodate all the messages and thus the Monitor needs to make decisions which messages to serve first, which may result in some messages being unattended.

Figure 1 shows only one instance of Source and one of Destination (in ontology they are referred to as Nodes). However, in this scenario, there are multiple Nodes sending messages to one another concurrently, resulting in a multitude of communication events requiring monitoring and prioritization. The specific Nodes belong to specific Organizations, are involved in some Activities, and deliver various Services to their clients (Destinations). The objective of the monitoring system is to decide which of the messages should be selected for transmission first. Since the Monitor runs a loop, it selects most preferable messages until the transmit buffer is full.

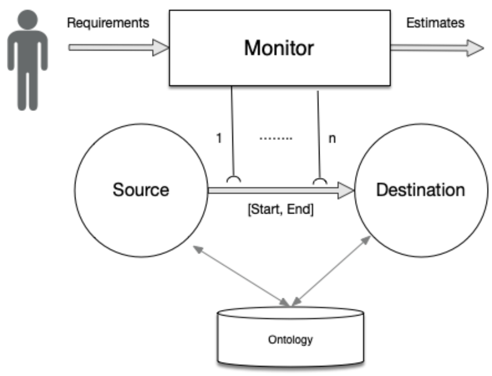


Figure 1: A surveillance scenario.

5 ONTOLOGIES USED

The ontology fragment used to provide the generic information to the generator is shown in Figure 2. Since we are using a generic foundational ontology, Nuvio (http://cogrario.org/ont/Nuvio.owl), the names of the classes to represent a simulation are taken from this ontology. *Entity* is the top class in this ontology; all other classes are either direct or indirect subclasses of this class. The *InformationContent* is the class that will represent the requirements of the domain ontology. *Process* is the class that represents things that change in time, while *Object* is for static things. The *Attribute* class is where all attributes of all of the other classes are represented. The *Quality* class is for qualitative attributes, while the *Quantity* class is for quantitative attributes. The quantitative attributes have a special structure of *Interval*. Some quantitative attributes may have *UnitOfMeasure*, while some others (like probabilities) are dimensionless.

Additionally, the ontology must be able to represent the aspects of the specific scenario described in Section 4. A partial view of the ontology for representing some of the domain aspects of the scenario is shown in Figure 3. The simulated process is an instance of class *Service*. The information channel that is surveilled is shown as *Stream*. The Requirements originate from the system User, as shown in Figure 1. The class *Priority* represents the priorities assigned by the user to the specific information sources and to the particular variables and their values. The figure seems

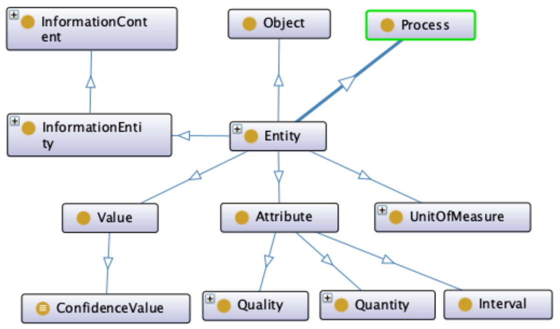


Figure 2: A fragment of the simulation generator ontology.

self-explanatory since the classes are linked via OWL *properties* with informative names. A mapping of the classes of the domain ontology to the classes of Nuvio is shown in Table 1. The left column of the table lists the classes of the Domain Ontology, while the right column represents the classes of Nuvio. The Domain Ontology classes are related to the Nuvio classes via the OWL *subClassOf* relation, i.e., each class in the left column is *subClassOf* the Nuvio class located in the same row in the right column.

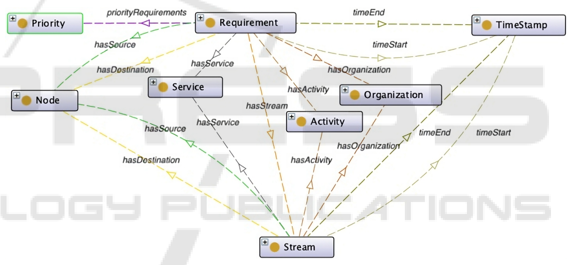


Figure 3: Domain ontology: a partial view.

Table 1: Mapping of Domain Ontology to Simulator ontology.

Domain Ontology	Nuvio Ontology
Requirement	InformationEntity
Stream	InformationContent
Service	Process
Activity	Process
Node	Object
Organization	Object
ChanceNode	ConfidenceValue
TimeStamp	Quantity
Priority	xsd:decimal

Note that some of the Nuvio classes from Figure 2 do not appear in Table 1. This is because some of them are parts of the subclass chains. E.g., *Entity* is the top class and thus none of the Domain Ontology classes are directly subclasses of this class. The same applies to the other Nuvio classes from Figure

2. More specifically, *Service* – the process to be simulated – is the subclass of *Process*. The simulation variables are represented by *Quantity*, a subclass of *Attribute*. Their values are captured by the instances of the class *Value*. In general they may have dimensions, represented by the class *UnitOfMeasure*. The requirements for the Simulator are associated with *Service*. They are instances of the *Requirement* class, which is aggregated by *RequirementList* (not shown in the figure). The formal relation between the Simulator (Process) and the Requirement can be formalized as a triple ⟨*SimulatorObject rdf*:*type Requirement*⟩. Finally, the *ChanceNode* domain class, which appears in the table, is not shown in Figure 3 because it originates from a separate Uncertainty Ontology, which is omitted here due to space constraints.

6 SIMULATION GENERATOR

A representation of *dg* is shown if Figure 4. The main component of *dg*, Simulator Generator, reads in the ontology that has imported the domain ontology, and user inputs. The most important aspect of this process is that the generator acquires the variables for the simulator from the Domain Ontology. In this case, the variables are *Activity, Node (Source, Destination), Organization*. The (structural) constraints that the Simulator must satisfy are imported with the ontology, too. *dg* then generates the Java object - *Simulator*. In OWL terms, the *Simulator* is of *rdf:type Requirement*. Thus it satisfies the constraints of requirements defined by the user. While the current implementation is limited to the structural constraints, it can be extended to include the procedural aspects by expressing them in the OpenMath standard.

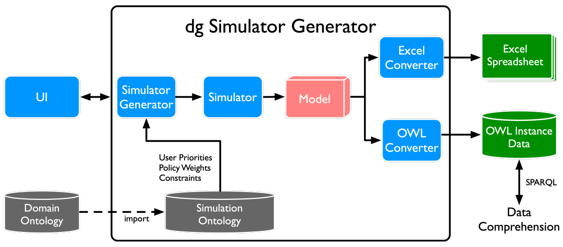
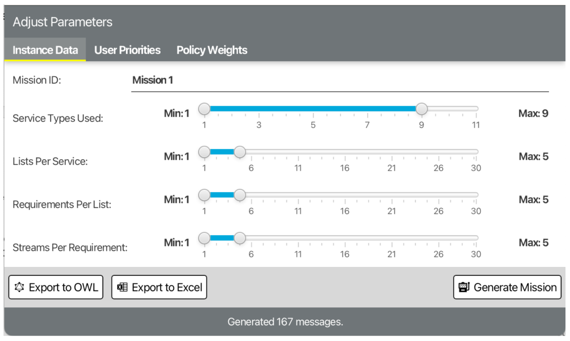


Figure 4: The dg simulator generator.

The user of *dg* uses a graphical UI to provide the Requirements, as described in Section 5. Figure 5 shows the setup menu which allows the user to set the general parameters of the simulation. This GUI allows the user to roughly set the size of the simulated data by focusing on Services – the number of Service Types that are available in the ontology, followed by the number of Requirement Lists, the number of Requirements per list, and the number of Streams per requirement. Since the idea of this simulation generator is to make data generation random and unbiased, the actual numbers and the values of the variables depend on the user who sets the min/max boundary for each parameter that is then randomly picked by the generator (uniform distribution).



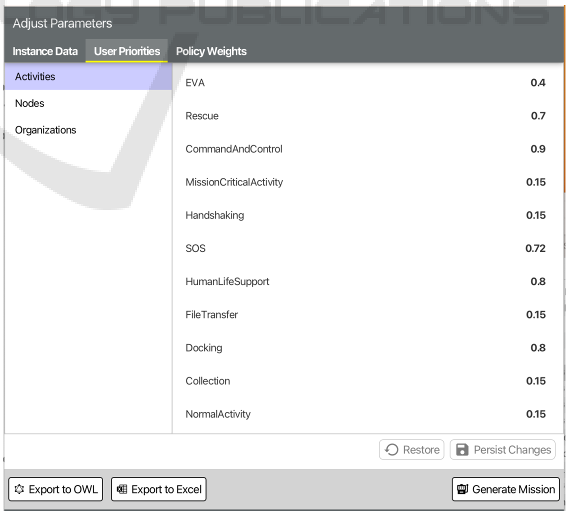
Mission ID: Mission 1

Service Types Used:	Min: 1	Max: 9
Lists Per Service:	Min: 1	Max: 5
Requirements Per List:	Min: 1	Max: 5
Streams Per Requirement:	Min: 1	Max: 5

Generated 167 messages.

Figure 5: The dg setup menu.

In the scenario discussed in this paper, the random variables include Services, Activities, Nodes (Sources and Destinations) and Organizations. These variables are provided by the Domain Ontology, shown in Figure 4, rather than being hard coded in *dg* itself. This is the main point of this paper – the Simulator Generator is controlled through ontology. The weights can be adjusted in the 'Policy Weights' tab.



EVA	0.4
Rescue	0.7
CommandAndControl	0.9
MissionCriticalActivity	0.15
Handshaking	0.15
SOS	0.72
HumanLifeSupport	0.8
FileTransfer	0.15
Docking	0.8
Collection	0.15
NormalActivity	0.15

Figure 6: The dg GUI for defining random variables.

7 SIMULATION EXAMPLES

In this section, we show some of the simulations that use *dg* for experimenting with simulation policies. The main objective of these simulations is to demonstrate how the user can explore the simulation space in order to understand the communications scenarios. The scenarios are not under the user's control since they depend on the communication scenario. However, the user can get an understanding of which variables and values should be given more priority in the policy so that the communications resources are better utilized to achieve the best information delivery results. In *dg*, this policy aspect is encoded in the weights and controllable through the *dg* GUI, as shown in Section 6. In our ontology, these weights are encoded as individuals of the Probability class.

For the first example, consider a case where the most important variable is the Activity in which the nodes are involved. In this case, the policy should place greater emphasis on the weight assigned to the Activity variable. As shown in Table 2, the weight associated with Activity is 0.88, while all other weights are set to 0.02, adding up to 1.

The distribution of the weights may seem a bit exaggerated, however this came as a result of interacting with the simulator. In fact, it is an artifact of the decision algorithm. Notice that the decisions are derived by optimizing the expected value, which is a linear combination of weighted probabilities. Our team, and other people who tried to set policies according to their intents, were surprised by the low sensitivity to the policy weights. We all expected that the decisions of the Monitor would favor a specific variable whenever the weight associated with the variable is somewhat higher than those of the others. However, since in this particular simulation example there were seven variables, the impact of just one of them is diminished by the combination of the other variables. Our conclusion from these experiments was that users need to be trained for effectively setting recognition policies and *dg* can serve as a useful tool in such training.

Table 2: Weights: Policy: Activity matters the most.

Activity	Service	Org	Src	Dest	Start	End
0.88	0.02	0.02	0.02	0.02	0.02	0.02

Figure 7 shows the result of one of the simulations that followed the "Activity matters the most" policy described above. While it is difficult to show a representation of an 8D sample space, it is rather clear from this picture that the Activity aspect was prevailing in the decisions. This conclusion can be reached because for each of the seven groups of samples the priority generated by the Simulator were distributed uniformly for all of the activity instances of the Activity types. In other words, the other six aspects did not have much impact on the derivation of the priority of the particular messages.

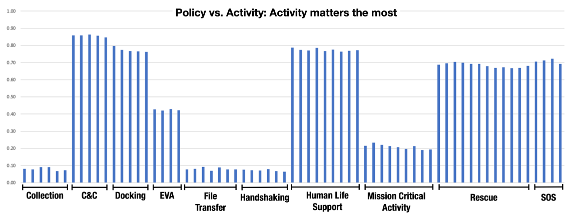


Figure 7: Simulation result: Activity is the most important aspect to the user.

Table 3: Weights: Policy: All variables are equal.

Activity	Service	Org	Src	Dest	Start	End
0.15	0.15	0.15	0.15	0.15	0.15	0.1

Figure 8 illustrates the decisions derived by the Monitor when the policy treats all input variables as equally important (c.f. Table 3). In this case, as with the previous scenario, the data were grouped and ordered by Activity type. By examining the distribution of the expected values across the Activity groups, it is evident that the distribution is not uniform.

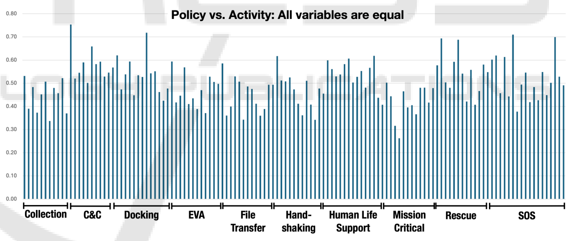


Figure 8: Simulation result: All input variables are equally important to the user.

8 SIMULATION QUERIES

In this section we show some examples of how ontology, supported by an ontology querying tool, is useful for the understanding of the various aspects of the simulation results.

In the following we show two examples of SPARQL queries and query results. The first example shows a query that is of a similar kind as in (Fang, 2019), i.e., the query is asking to retrieve the bindings that satisfy the condition of the type $?p \leq s$ referred to above.

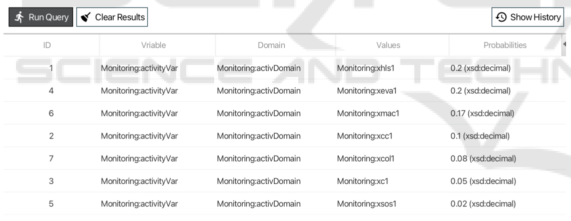
Example 1: *Show the probability distribution of variable Activity.*

```
SELECT *
WHERE {
  ?Vriable rdf:type Monitoring:ChanceNode;
  Monitoring:hasDomain ?Domain.
  ?Domain nuvio:aggregateOf ?Values.
  ?Values Monitoring:hasPriority ?Probabilities
}
```

Listing 1: SPARQL query.

The query can be expressed in SPARQL as shown in Listing 1. The query is referring to the ontology namespace Monitoring and selects the values of the domain of the ChanceNode (from the ontology not shown in this paper), which is associated with the class Activities in the ontology. The Activities class is an aggregate of Activity, each of which has the Priority value generated by the Simulator *dg*. The Monitoring ontology imports the UncertatintyOntology described earlier.

The query is executed by *PolVISor IDE*, a SPARQL execution IDE developed by VIStology. It supports loading an ontology, running an inference engine, e.g., HermiT, the reasoner that can also be invoked from Protege, and provides an interface for the authoring of policies expressed in SPARQL. A screenshot in Figure 9 shows the result of the execution of the above query.

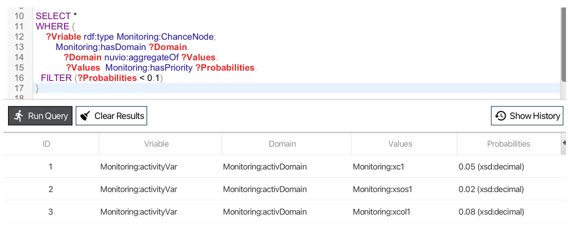


ID	Vriable	Domain	Values	Probabilities
1	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xhls1	0.2 (xsd:decimal)
4	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xeva1	0.2 (xsd:decimal)
6	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xmac1	0.17 (xsd:decimal)
2	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xcc1	0.1 (xsd:decimal)
7	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xcol1	0.08 (xsd:decimal)
3	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xc1	0.05 (xsd:decimal)
5	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xsos1	0.02 (xsd:decimal)

Figure 9: Query result for Example 1.

Example 2: *Show the Activity instances whose probability is less than 0.1.*

A screenshot from PolVISor IDE in Figure 10 shows both the SPARQL expression that represents this query and the result of the execution of the query.



```
SELECT *
WHERE {
  ?Vriable rdf:type Monitoring:ChanceNode;
     Monitoring:hasDomain ?Domain.
        ?Domain nuvio:aggregateOf ?Values.
         ?Values  Monitoring:hasPriority ?Probabilities.
 FILTER (?Probabilities < 0.1)
}
```

ID	Vriable	Domain	Values	Probabilities
1	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xc1	0.05 (xsd:decimal)
2	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xsos1	0.02 (xsd:decimal)
3	Monitoring:activityVar	Monitoring:activDomain	Monitoring:xcol1	0.08 (xsd:decimal)

Figure 10: Query Example 2 and the result of its execution.

While such queries can be very useful for browsing through the simulated (or real) data, it requires the mapping of the natural language expressions shown in Examples 1 and 2 above to SPARQL, which requires some knowledge of this language and the ontology that captures the domain knowledge, plus the results of logical inference on the original data. While in these examples the names of the random variables and the names of SPARQL variables (the ones with the "?" mark as the first character) were selected in such a way that they indicate the meaning of these variables, SPARQL does not care about such choices. Moreover, SPARQL does not include any functions that are useful for computing various probabilistic and statistical attributes. We claim that it would be desirable, to add such features to pure SPARQL. In particular, an agreed upon Uncertainty Ontology would have to be used as the first ingredient of the solution. Second, a number of functions specific to the probability and statistics domains would need to be included. The functions would perform quantitative computations, such as generating random data according to specific probability distributions (e.g., Binomial, Exponential, Normal, Bernoulli), and calculating joint probability distributions, expected values, and similar metrics. Additionally, functions for statistics and statistical testing could be included. Note that quantitative computations are not well-suited for inclusion in the ontology, making functions associated with SPARQL a more appropriate place for them.

9 CONCLUSIONS

In summary, this paper has shown the value of ontology in the process of developing simulators to support the experimentation with the semantic communication. In particular, we have shown that ontology can play different roles. First, it can be used to setup generation of simulators of various systems. The simulators, like the one described in this paper, would take an ontology as input. The simulator would then provide a User Interface to setup the various parameters that can control the simulated variables and the amount of the simulated data. The main structure of the simulator code remains unchanged, even though different systems are simulated thanks to the use of ontology and ontology reasoners to interpret the simulation requirements.

Second, the ontology used in this work can be expanded to introduce many more concepts from both probability theory and statistics and then used by the simulator. In this work we developed a very basic ontology and ran simulations of very simple data prioritization algorithms. With a more complex ontology more advanced prioritization algorithms could be defined and used for the processing of information by the simulator and later used for querying.

We briefly showed the PolVISor IDE system for query editing, invoking a logical inference engine, executing queries and displaying results of queries. These capabilities could be significantly expanded to make simulation simpler and the querying more interesting. Finally, we have shown that the ontology was useful for browsing through the simulated data.

We conducted a literature search for SPARQL extensions, specifically looking for a Probabilistic SPARQL. Our search yielded limited results, as the only extension we found was pSPARQL. The pSPARQL extension provides an extension to the semantics of the FILTER function of SPARQL. We would like to see an extension that includes an Uncertainty Ontology, various attributes that are native to both probability theory and statistics, and various functions for probabilistic and statistical computations. We believe such a SPARQL extension as postulated here would be possible, feasible, and most importantly useful not only for developing simulation systems, but more generally to the software engineering community.

Finally, yet another extension to the current capabilities of *dg* seems to be possible due to the recent progress of AI and Machine Learning. Currently, we generate SPARQL queries manually. We believe it would be possible and beneficial to add the capability of automatic translation of natural language queries to SPARQL. The use of Large Language Models (LLMs) should be investigated for the use of translating natural language queries to SPARQL.

ACKNOWLEDGEMENTS

The authors wish to acknowledge that the development of the *dg* simulator was supported by a grant 80NSSC22CA141 from NASA through a subcontract from AiRANACULUS to VIStology.

REFERENCES

Adcock, R. (2007). Principles and practices of systems engineering. In *INCOSE*.

Barth, M., Ristic, M., and Jäkel, J. (2023). A role-based Metric to determine the Quality of Simulation Models. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE.

Fang, H. (2019). pSPARQL: A Querying Language for Probabilistic RDF Data. *Complexity*, 2019(1):8258197.

Grieves, M. (2014). Digital twin: manufacturing excellence through virtual factory replication. *White paper*, 1(2014):1–7.

Hildebrandt, C., Köcher, A., Küstner, C., López-Enríquez, C.-M., Müller, A. W., Caesar, B., Gundlach, C. S., and Fay, A. (2020). Ontology building for cyber–physical systems: Application in the manufacturing domain. *IEEE Transactions on Automation Science and Engineering*, 17(3):1266–1282.

IEEE (2000). IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. *IEEE Std 1516-2000*, pages 1–28.

IEEE (2012). IEEE Standard for Distributed Interactive Simulation–Application Protocols. *IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995)*, pages 1–747.

ISO (2023). ISO/IEC/IEEE 15288:2023. Systems and software engineering — System life cycle processes. `https://www.iso.org/standard/81702.html`.

ISO-IEC (2024). ISO/IEC 25002:2024. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model overview and usage. `https://www.iso.org/standard/78175.html`.

Jeleniewski, T., Nabizada, H., Reif, J., Köcher, A., and Fay, A. (2023). A Semantic Model to Express Process Parameters and their Interdependencies in Manufacturing. In *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, pages 1–6. IEEE.

Koulamas, C. and Kalogeras, A. (2018). Cyber-physical systems and digital twins in the industrial internet of things [cyber-physical systems]. *Computer*, 51(11):95–98.

Liu, J., Yu, D., Bi, X., Hu, Y., Yu, H., and Li, B. (2020). The research of ontology-based digital twin machine tool modeling. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 2130–2134. IEEE.

Reif, J., Jeleniewski, T., and Fay, A. (2023). An Approach to Automating the Generation of Process Simulation Sequences. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE.

Singh, S., Shehab, E., Higgins, N., Fowler, K., Reynolds, D., Erkoyuncu, J. A., and Gadd, P. (2021). Data management for developing digital twin ontology model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 235(14):2323–2337.

Van Ruijven, L. (2013). Ontology for systems engineering. *Procedia Computer Science*, 16:383–392.