A Pattern for Modeling Computational Observations

Cogan Shimizu[1,2]✉, Pascal Hitzler[2], and Charles F Vardeman II[3]

[1] Wright State University, USA
[2] Kansas State University, USA
[3] University of Notre Dame, Notre Dame USA

Abstract. Knowledge graphs (KG) are an established method for heterogeneous data integration and have begun powering complex software agents. However, it is important to understand where the data in the knowledge graph originates, especially within the context of synthetic research agents and other trustworthy AI systems. In this paper, we propose an ontology design pattern for tracking the provenance and context of computational observations, as well as a proposing a supporting, simplified conceptual framework for modeling abstract and concrete versions of the same underlying notion.

1 Introduction

Knowledge graphs (KG) are an established way of integrating data from multiple, heterogeneous sources [10]. Recently, they have begun powering complex software agents. However, as the complexity – and pervasiveness – of these agents grows, it is important to understand where the data in the knowledge graph originates. This is particular important within the context of synthetic research agents and other trustworthy AI systems. When the AI agent is operating on data of specious origin, it is important to propagate this downstream through all the downstream actions.

In particular – and the focus of this paper – we want to track the provenance and lineage of data produced from computational models. This requires that we understand both the models themselves and the results that they produce – which we call "computational observations," as well as the context in which these models are executed,

However, the space of computational models is quite large. Building an entire domain ontology would be very difficult and, not to mention, contentious. As such, we have opted to develop an ontology design pattern [5] that represents, in a generalized case, the interplay between computational models, executions, and observations. Such a pattern will enable downstream ontology or knowledge engineers to quickly incorporate modeling best practices into their own KG, and will enable them to easily align their KGs to other KGs that reuse the same pattern.

The primary contributions of this paper are:

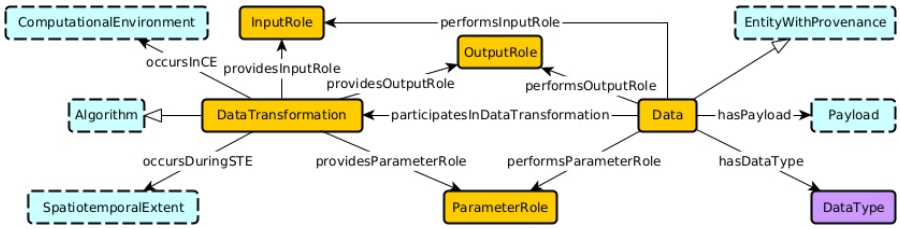

Fig. 1: The schema diagram for the Data Transformation Ontology Design Pattern (ODP). Gold boxes represent concepts that are central to the pattern. Blue boxes with dashed borders are interfaces to other patterns or represent concepts with significant complexity, but outside the scope of this pattern. Purple boxes represented controlled vocabularies, which simply means they are comprised of a finite set of individuals. Black-filled arrows represent object or data properties; and open arrows represent subclass relationships.

1. a simplified framework for conceptually modeling abstract and concrete concepts representing the same underlying notion; and
2. a pattern for modeling computational observations.

The remainder of the paper is organized as follows. In the next section, we discuss background and related work. Section 3 provides a brief discussion on a basic, conceptual pattern for modeling abstract and concrete concepts for the same notion. In Section 4 we present our pattern and provide its formalization. Finally, in Section 5, we conclude with future work.

2 Related Work

There is some existing literature that focuses on how to capture the results of modeling and simulation software. As far as the authors are aware, there is not directly corresponding work for directly modeling computational observations. Below, we describe some tangential work that could be used in a limited fashion, and how it relates to our pattern described in Section 4.

SOSA/SSN (The Sensors, Observations, Sampling, Actuators Ontology and Semantic Sensor Network Ontology [7]) are W3C Recommendations for modeling how observations are generated from sensors. While the notion of a sensor, within these ontologies, is left intentionally ambiguous and can, indeed, be used to represent a piece of software (e.g., for simulation or forecasting purposes), it is awkward to do so, due to requirements on phenomenon and result times of observations – and whether or not a software producing a dataset is really an observation at all. Secondly, SOSA/SSN does not provide a mechanism for modeling aspects of the software or hardware, which this pattern provides.

ML-Schema [2] was proposed by the W3C Machine Learning Schema Community Group to capture the provenance of machine learning data sets, algorithms, models, software implementations, and model runs for machine learning

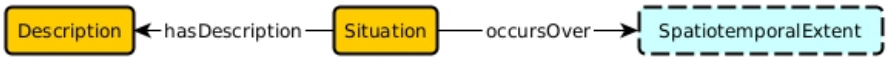

Fig. 2: A visual representation of the simplified Descriptions and Situations framework. The graphical syntax is the same as in Figure 1.

experiments. While ML-Schema captures the computational workflow associated with an ML experiment, it does not conceptualize a "model" as a surrogate for a real world phenomenon and is instead focused on capturing the type of machine learning methodology as a model. Also, because it was developed primarily as a schema for ML experiments, it lacks the generality of a ontology pattern based approach.

The Data Transformation Pattern [12] describes how data is transformed via different numerical operations (a schema diagram for the pattern is shown in Figure 1). This pattern focuses on the *dataset* level, rather than the specific data within them, for tracking how the datasets were transformed, what – or whom – performed those actions, and which datasets were used to generate or derive the new dataset. The Computational Observation pattern, instead, focuses on how computational models generate specific data (e.g., computational observations).

The Computational Environment pattern [6] is used to model the hardware and software configurations where a particular piece of code may be implemented and executed. Its focus is not on the metadata of the results of that execution, but instead on providing a human and machine-interpretable way of exchanging (arbitrary) configuration information. Indeed, we reference this concept in our own pattern.

3 Modeling Abstract and Concrete Concepts

It is frequently desirable to model a particular concept and its abstraction. That is, the difference between an algorithm and the *execution* of that algorithm. The Descriptions and Situations (DnS) [4] framework is one way to accomplish this. However, DnS can be quite challenging to approach, especially for those without significant ontology engineering experience or foundations in logics. It furthermore leverages DOLCE [3], a foundational ontology, which may not be compatible with project needs. The following is an alternative, simplified framework by which one can *conceptually* reason about the description and situation dynamic.

Figure 2 shows a schema diagram for a simplified conceptualization of the Description and Situation dynamic. In summary, a Description exists outside of time – it is the abstraction of a notion; a Situation is an instantiation of that Description that is anchored in space and time.

For example, consider the notion of an algorithm. Loosely speaking, an algorithm is a set of steps that operate on an input to produce an output, for

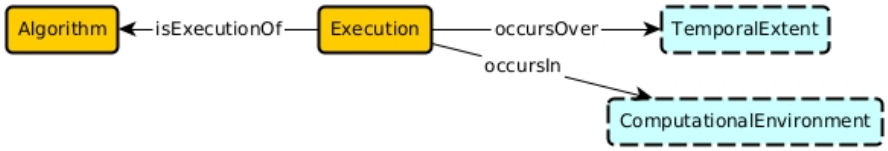

Fig. 3: A visual representation of our Algorithm example. The graphical syntax is the same as in Figure 1. Note the correspondences between it and the figure above.

example: long division or, less trivially, normalizing a series of data. Such algorithms, however, exist in the abstract; the *executions* of them, do not. They are run in particular computational environments, on some hardware, and according to some software implementation. Consider the graphical representation in Figure 3: the Algorithm corresponds to the Description; the Execution corresponds to the Situation; and we split the spatial and temporal aspects of a SpatiotemporalExtent to produce a TemporalExtent and a ComputationalEnvironment. In a more complex scenario, one might also consider an algorithm to have an input or parameter space, where the Execution would have specific inputs and parameters.

3.1 Formalization of the Simplified Framework

We provide a brief formalization of this framework.

Description is the abstract representation of a particular concept or notion. This could, for example, be an algorithm or a recipe. Intuitively, the Description is a *template*.

$$\top \sqsubseteq \mathsf{hasDescription}.\mathsf{Description} \tag{1}$$

Situation is the concrete instantiation of a particular Description. This could, for example, be the execution of an algorithm or the act of following a recipe. Intuitively, the Situation is a template that has been "filled out."

$$\mathsf{Situation} \sqsubseteq \forall\ \mathsf{occursOver}.\mathsf{SpatiotemporalExtent} \tag{2}$$

$$\mathsf{Situation} \sqsubseteq =1\ \mathsf{occursOver}.\mathsf{SpatiotemporalExtent} \tag{3}$$

$$\mathsf{Situation} \sqsubseteq =1\ \mathsf{hasDescription}.\mathsf{Description} \tag{4}$$

4 A Pattern for Computational Observations

The Computational Observation ODP is driven by the interplay between the three core concepts: ComputationalModel, ComputationalModelExecution (CME), and ComputationalObservation. The first two correspond, respectively, to the Description and Situation from Figure 2. That is, a ComputationalModel is an abstract representation of the space of all computational models, and the CME

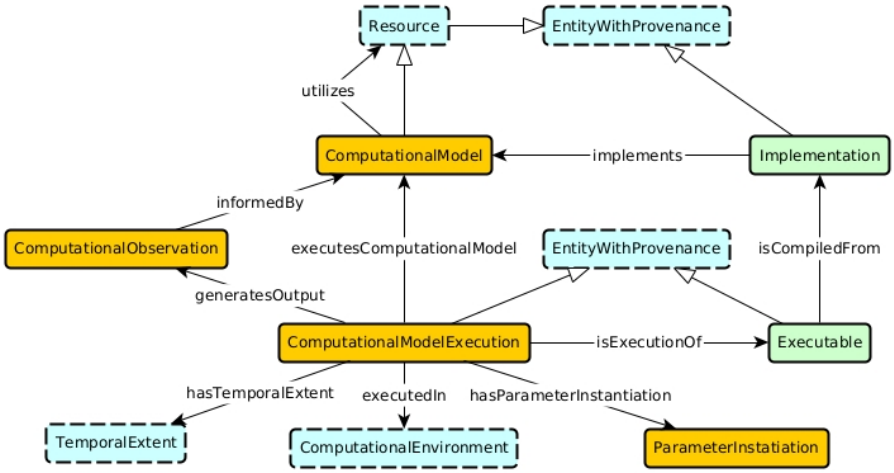

Fig. 4: The schema diagram for the Computational Observation ODP. It uses the same graphical syntax as in Figure 1.

is a concrete instance from that space (e.g., parameters and inputs have been selected). The CME is then comprised of a set of generated outputs: the ComputationalObservations.

The rest of the pattern is relevant metadata important to describing the entire process or pipeline. ComputationalModels are implemented into a codebase (Implementation) and are compiled into executables (Executable). An execution of the ComputationalModel requires that we capture the temporal extent (i.e., when it was executed) and the computational environment (i.e., where it was executed). Below, we provide further descriptions of each concept and its relevant axioms (listed alphabetically).

ComputationalEnvironment is the encapsulation of hardware and software configurations of the computer used to execute the ComputationalModelExecution. This interface may be satisfied by the ODP of the same name, as in [6].

ComputationalModel (CM) is an *abstract* notion intended to connect the conceptual, mathematical and algorithmic models as surrogates for *real world phenomena* that is the intended target of an observation. That is, we use it to represent the space of CMs. Every CM is a Resource, which allows us to chain together CM input and outputs.

$$\mathsf{ComputationalModel} \sqsubseteq \mathsf{Resource} \tag{5}$$

$$\mathsf{ComputationalModel} \sqsubseteq \geq 0\ \mathsf{utilizes.Resource} \tag{6}$$

ComputationalModelExecution (CME) is the concrete notion of a ComputationalModel. That is, it corresponds to the Situation in Figure 2. We directly link the CME to the ComputationalModel via an exact cardinality restriction on

the executesComputationalModel property. This differs from the notion of an Executable (below) in that the Executable is the artifact that can be executed many times, and the CME is the actual act of execution, and it thus exactly corresponds to that Executable. Furthermore, it will always have exactly one TemporalExtent, and at least one ComputationalEnvironment.[4] ParameterInstantiations are sometimes necessary when complex CMEs generate their own parameters as part of the execution (more details are provided below). Finally, CMEs generate ComputationalObservations, and they will always generate at least one.

$$\mathsf{CME} \sqsubseteq \forall\ \mathsf{generatesOutput.ComputationalObservation} \tag{7}$$

$$\mathsf{CME} \sqsubseteq \geq 0\ \mathsf{generatesOutput.ComputationalObservation} \tag{8}$$

$$\top \sqsubseteq \forall\ \mathsf{hasTemporalExtent.TemporalExtent} \tag{9}$$

$$\mathsf{CME} \sqsubseteq = 1\ \mathsf{hasTemporalExtent.TemporalExtent} \tag{10}$$

$$\top \sqsubseteq \forall\ \mathsf{executedIn.ComputationalEnvironment} \tag{11}$$

$$\mathsf{CME} \sqsubseteq \exists\ \mathsf{executedIn.ComputationalEnvironment} \tag{12}$$

$$\top \sqsubseteq \forall\ \mathsf{hasParameterInstantiation.ParameterInstantiation} \tag{13}$$

$$\mathsf{ParameterInstantiation} \sqsubseteq \exists\ \mathsf{hasParameterInstantiation}^{-}\mathsf{.CME} \tag{14}$$

$$\mathsf{CME} \sqsubseteq \geq 0\ \mathsf{hasParameterInstantiation.ParameterInstantiation} \tag{15}$$

$$\top \sqsubseteq \forall\ \mathsf{isExecutionOf.Executable} \tag{16}$$

$$\mathsf{CME} \sqsubseteq = 1\ \mathsf{isExecutionOf.Executable} \tag{17}$$

$$\mathsf{CME} \sqsubseteq = 1\ \mathsf{executesComputationalModel.ComputationalModel} \tag{18}$$

ComputationalObservation (CO) is the core *concept* of this pattern. Notably, we do not mandate that the CO is an EntityWithProvenance, as it is already directly modeled via the generatesOutput property. Indeed, we specify that it is *inverse existential* to state that any CO must have been generated by a CME. We also do not specify any particular way to formulate what the value of the CO is, as we want the pattern to be sufficiently generalized. Finally, we realize that informedBy is a very informal term – we anticipate that the exact label for the relationship be adapted to a particular use-case; we simply want to use a placeholder to indicate that a relationship exists here.

$$\mathsf{ComputationalObservation} \sqsubseteq = 1\ \mathsf{informedBy.ComputationalModel} \tag{19}$$

$$\mathsf{ComputationalObservation} \sqsubseteq \exists\ \mathsf{generatesOutput}^{-}\mathsf{.CME} \tag{20}$$

EntityWithProvenance is eponymous; it indicates that the entity in question has desirable metadata, such as who generated the entity, when it may have

[4] We leave this only as an existential restriction as it is up to the user of this pattern – and the conceptualization of the computational environment – whether or not *distributed* computing scenarios count as multiple computational environments.

been generated, and what was used to derive the entity. We recommend using PROV-O [9] to satisfy this interface.

Executable is the artifact that can be executed, and is generally produced by a compiler. As such, we indicate that it is an EntityWithProvenance so that this may be captured. We also state that an Executable can only be compiled from exactly one Implementation.

$$\mathsf{Executable} \sqsubseteq \mathsf{EntityWithProvenance} \tag{21}$$

$$\exists\ \mathsf{isCompiledFrom.Implementation} \sqsubseteq \mathsf{Executable} \tag{22}$$

$$\mathsf{Executable} \sqsubseteq \forall\ \mathsf{isCompiledFrom.Implementation} \tag{23}$$

$$\mathsf{Executable} \sqsubseteq =1\ \mathsf{isCompiledFrom.Implementation} \tag{24}$$

Implementation is the formulation of the computational model into some codebase (e.g., the hosting repository). In our axiomatization, we state that the Implementation is an EntityWithProvenance, allowing us to model, for example, contributors. Furthermore, we provide scoped domain and range restrictions for the implements property, as well as state that an Implementation implements ComputationalModels. This axiom can be changed depending on the requirements – it is foreseeable that one to many ComputationalModels can be implemented in the same repository (existentiality), or only one (exact cardinality restriction of 1).

$$\mathsf{Implementation} \sqsubseteq \mathsf{EntityWithProvenance} \tag{25}$$

$$\exists\ \mathsf{implements.ComputationalModel} \sqsubseteq \mathsf{Implementation} \tag{26}$$

$$\mathsf{Implementation} \sqsubseteq \forall\ \mathsf{implements.ComputationalModel} \tag{27}$$

$$\mathsf{Implementation} \sqsubseteq =1\ \mathsf{implements.ComputationalModel} \tag{28}$$

ParameterInstantiation is the set of parameters the defines the, typically mathematical, *model space* for an execution of a computational model. For instance, in Newton's Law of Cooling: $\dot{Q} = h * A(T(t) - T_{env})$, the heat transfer coefficient h, objects surface area A, and the temperature of an objects surrounding environment T_{env} would provide part of a ParameterInstantiation needed to understand the results of the execution of a computational model. Additional details such as time step used in solving the mathematical model would also be part of the ParameterInstantiation.

Resource is an arbitrary piece of data that the computational model will require in order to run. These might be data sources or parameters. Note that a ComputationalModel is also a Resource. This allows us to model how ComputationalModels might feed into each other.

TemporalExtent is a straightforward concept, representing the length of time that the ComputationalModelExecution would have run. It is, however, left as an interface in the pattern, as we do not want to mandate a particular conceptualization of time. This interface could, for example, be satisfied using xs:duration or the Time ontology [8] for more complex modeling, depending on the needs of the use-case.

5 Conclusion

Modeling the process or pipeline by which computational observations are generated (i.e., the results from executed numerical models or simulations) is an important aspect of trustworthy data. For example, the adage "Garbage In, Garbage Out" comes to mind. How can we trust operations on unknown data that is untrustworthy? The Computational Observation ODP is an attempt to fill this space. By describing the entire generation pipeline, from the particular resources used to implement a computational model, to the hardware, software, and parameter configurations used in the execution of said computational model we can more readily understand *how* certain results were achieved in modeling and simulation scenarios.

We furthermore presented a simplified, conceptual framework for discussing abstract and concrete notions of the same underlying concept. That is, the difference between, for example, an algorithm and its execution; or a recipe and the act of cooking. In this way, a computational model and its execution must be similarly modeled.

We have identified the following items as next steps in our work.

1. We intend to instantiate and connect it to instantiations of both the Data Transformation and Computational Environment Patterns, to create a Modeling & Simulation Ontology.
2. The Computational Observation pattern will be added to the next version of MODL (The modular ontology design pattern library – [11]).
3. Create a cross-walk from the Computational Observation pattern to the CodeMeta Project [1].

Acknowledgement. Authors Shimizu and Hitzler wish to acknowledge funding for this work under the National Science Foundation Grant No. 2033521: "KnowWhereGraph: Enriching and Linking Cross-Domain Knowledge Graphs using Spatially-Explicit AI Technologies". Author Vardeman wishes to acknowledge funding for this work under the National Science Foundation Grant No. PHY-1247316: "DASPOS: Data and Software Preservation for Open Science" and Grant No. 2127548 "CI-Compass". Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors would also like to acknowledge valuable discussion with David Carral, Gary Berg-Cross, and Michelle Cheatham in the early stages of the pattern's development.

References

1. The CodeMeta Project. https://codemeta.github.io/
2. Correa Publio, G., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., Vanschoren, J., Zafar, H.: ML-Schema: Exposing the semantics of machine learning with schemas and ontologies. arXiv e-prints p. arXiv:1807.05351 (Jul 2018)
3. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening ontologies with DOLCE. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) Knowledge Engineering and Knowledge Management. Ontologies and the Semantic

Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2473, pp. 166–181. Springer (2002). https://doi.org/10.1007/3-540-45810-7_18, https://doi.org/10.1007/3-540-45810-7_18

4. Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Lecture Notes in Computer Science, vol. 2888, pp. 689–706. Springer (2003). https://doi.org/10.1007/978-3-540-39964-3_44, https://doi.org/10.1007/978-3-540-39964-3_44
5. Gangemi, A., Presutti, V.: Ontology design patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 221–243. International Handbooks on Information Systems, Springer (2009). https://doi.org/10.1007/978-3-540-92673-3_10, https://doi.org/10.1007/978-3-540-92673-3_10
6. Huo, D., Nabrzyski, J., II, C.F.V.: An ontology design pattern towards preservation of computational experiments. In: Keßler, C., Zhao, J., van Erp, M., Kauppinen, T., van Ossenbruggen, J., van Hage, W.R. (eds.) Proceedings of the 5th Workshop on Linked Science 2015 - Best Practices and the Road Ahead (LISC 2015) co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015. CEUR Workshop Proceedings, vol. 1572, pp. 15–18. CEUR-WS.org (2015), http://ceur-ws.org/Vol-1572/paper3.pdf
7. Janowicz, K., Haller, A., Cox, S., Lefrançois, M., Phuoc, D.L., Taylor, K.: Semantic sensor network ontology. W3C recommendation, W3C (Oct 2017), https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/
8. Little, C., Cox, S.: Time ontology in OWL. W3C recommendation, W3C (Oct 2017), https://www.w3.org/TR/2017/REC-owl-time-20171019/
9. Sahoo, S., McGuinness, D., Lebo, T.: PROV-o: The PROV ontology. W3C recommendation, W3C (Apr 2013), http://www.w3.org/TR/2013/REC-prov-o-20130430/
10. Shimizu, C., Hammar, K., Hitzler, P.: Modular ontology modeling. Semantic Web (2022), in press
11. Shimizu, C., Hirt, Q., Hitzler, P.: MODL: A modular ontology design library. In: Janowicz, K., Krisnadhi, A.A., Poveda-Villalón, M., Hammar, K., Shimizu, C. (eds.) Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 27, 2019. CEUR Workshop Proceedings, vol. 2459, pp. 47–58. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2459/paper4.pdf
12. Shimizu, C., McGranaghan, R.M., Eberhart, A., Kellerman, A.C.: Towards a modular ontology for space weather research. In: Blomqvist, E., Hahmann, T., Hammar, K., Hitzler, P., Hoekstra, R., Mutharaju, R., Poveda-Villalón, M., Shimizu, C., Skjæveland, M.G., Solanki, M., Svátek, V., Zhou, L. (eds.) Advances in Pattern-Based Ontology Engineering, extended versions of the papers published at the Workshop on Ontology Design and Patterns (WOP). Studies on the Semantic Web, vol. 51, pp. 299–311. IOS Press (2021). https://doi.org/10.3233/SSW210021, https://doi.org/10.3233/SSW210021