A Framework to Create Ontologies for Scientific Data Management

Leonardo Salayandia and **Paulo Pinheiro** and **Ann Q. Gates**
University of Texas at El Paso
500 W. University Drive
El Paso, Texas 79968

Abstract

Scientists often build and use highly customized systems to support observation and analysis efforts. Creating effective ontologies to manage and share data products created from those systems is a difficult task that requires collaboration among domain experts, e.g., scientists and knowledge representation experts. A framework is presented that scientists can use to create ontologies that describe how customized systems capture and transform data into products that support scientific findings. The framework establishes an abstraction that leverages knowledge representation expertise to describe data transformation processes in a consistent way that highlights properties relevant to data users. The intention is to create effective ontologies for scientific data management by focusing on scientist-driven descriptions. The framework consists of an upper-level ontology specified with description logic and supported with a graphical language with minimal constructs that facilitates use by scientists. Evaluation of the framework's usefulness for scientists is presented.

Introduction

Scientific data processing systems can be complex. A clear understanding of processes that dictate how systems create data products is necessary for scientists to use those data products with confidence. For example, a reputable geophysicist that cannot determine if a system is generating properly treated datasets cannot use that system or the datasets produced for her work. Determining whether datasets are properly treated can be divided in two types of concerns: 1) whether the geophysicist agrees with the processes used to treat those datasets, and 2) whether the system implementation is valid with respect to the intended process. The latter concern falls in the realm of systems engineering and it is out of the scope of this work.

The work presented addresses the problem of engaging scientists in creating ontologies that describe data capture and transformation processes. Creating these types of ontologies is a difficult task that requires collaboration among domain experts, including scientists and knowledge representation experts. However, the technical complexities of creating formal ontologies may hinder the level of input from scientists, which is critical given the intention of capturing process knowledge that can help scientists evaluate data products for their work.

A framework is presented that specifies an abstraction to document, from a scientist's point of view, how data is captured and transformed. Abstraction refers to the general notion of describing real-world things by using relevant properties from a specific point of view and ignoring irrelevant properties (Gates et al. 2011). The framework defines an abstraction in the form of an upper-level ontology that includes three general concepts that can be specialized to describe data capturing and transformation. The first general concept is `data`. `Data` refers to things that can be used directly or indirectly as evidence for scientific findings. For example, the output of a sensor, the content of a technical report, records in a database, or a digital elevation map are `data`. Raw output of a sensor may be formatted in a way that is hard for a person to interpret, and hence, it would unlikely be used directly as evidence for scientific findings. However, the output of a sensor may represent field observations that can be filtered and treated to create spatio-temporal map renderings. Maps are more adequate for human interpretation and could potentially provide evidence for scientific findings.

The second general concept is `method`. `Method` refers to things that can be used to systematically transform `data`. `Method` examples are extrapolation software, interpretation of data by an expert, and visualization software. Notice that systematic `data` transformation refers to activities that are expected to produce consistent results under similar conditions; it does not refer to activities that are automated, as oppose to human-driven. For instance, extrapolation and visualization software are automated `methods`, i.e., given that required preconditions exist, these `methods` produce outcomes without human intervention. Interpretation of data by an expert, on the other hand, is a human-driven `method` that assumes consistent behavior by a person.

The third general concept is `container`. `Container` refers to things that can be used as acquirers or placeholders of `data`. For example, a sensor, a database, a file, a physical document, or a person are `containers`. Unlike `methods`, `containers` do not transform `data` and instead serve as providers or receivers of `data`. Notice that `containers` can be digital entities or real-world entities. For instance, a database and a file are digital enti-

ties, while a physical document and a person are real-world entities. In addition, `containers` can play the roles of `data` provider, `data` receiver, or both. For instance, a sensor would typically play the role of a `data` provider, while a person could play both the role of `data` provider and receiver. Notice that a person could be classified as a `method` instead of a `container` if the person is transforming `data`.

With respect to the people involved in capturing and transforming scientific data, a distinction is made between the roles of process author and process implementer. While the former is concerned with the appropriateness of the process, the latter is concerned with the correctness of the system that carries out the process. In the realm of customized scientific systems, a scientist is often the process author. The framework presented is intended to capture process knowledge from the viewpoint of process authors. Consequently, process knowledge documented through this framework focuses on the identification of core activities used to systematically transform data without introducing nuances about how those activities are performed. In this context, process knowledge is appropriate to understand and answer questions related to *what* and *why* activities are performed in the production of data. This type of process knowledge does not delve into details of *how* activities are implemented or performed and does not give a full temporal specification about *when* activities are carried out with respect to each other. Hence, the type of process knowledge documented with this framework is labeled *data-centric* because the focus is on describing transformations of data with minimal emphasis on functional descriptions. Data-centric process knowledge captured through this framework is useful for a variety of people: for process authors it serves as structured documentation that can be shared with colleagues; for process implementers it is useful in the requirements elicitation and validation phases of system development; for process users it serves as a training tool; finally, for data users it serves as a provenance tool.

The framework is formalized using description logic (Baader et al. 2003), complemented with a graphical representation, and supported by software tools (Pinheiro da Silva et al. 2010); it has minimal constructs to facilitate process knowledge transfer among users, i.e., authorship and interpretation of process knowledge. In addition, the resulting ontologies are encoded in the Web Ontology Language (OWL). OWL is an open, Web-based language with community adoption that has resulted in a wide range of tools to support querying and reasoning.

The rest of this paper is organized as follows: Section 2 presents background on ontologies and their application to the production of scientific data. Section 3 presents the framework by describing the ontology used to encode process knowledge and the graphical representation that facilitates its use. Section 4 presents related work and evaluation. Finally, Section 5 presents conclusions.

Background

The expertise of a knowledge engineer is important in the creation of suitable knowledge representations, i.e., ontologies, that effectively capture relevant properties of a domain and that can be reused across applications. Guarino offers a classification of ontologies based on their intention or point of view (1997). *Upper-level ontologies* describe general concepts that can be used across domains. The intention of upper-level ontologies is to support semantic interoperability across specialized ontologies. For example, Cyc is a popular upper-level ontology (Lenat and Guha 1989).

Specialized ontologies from upper-level ontologies can be divided in two main categories: *domain ontologies* and *task ontologies*. Chandrasekaran and Josephson describe the distinction between domain and task ontologies in the context of the knowledge needed to build Knowledge Based Systems (1997). Domain ontologies capture factual knowledge about a domain; it is the knowledge operated on to produce an answer. Task ontologies capture knowledge about problem solving methods; it is the knowledge that guides the answer-seeking process.

In creating customized scientific systems to serve a specific project, the input of a scientist is necessary to complement the ground work of the knowledge engineer. Facilitating the scientist's input to build a knowledge base is the niche targeted by the framework presented. In this case, the resulting ontology captures knowledge from the perspective of a process author, i.e., the scientist designing appropriate processes to capture and transform data to serve specific scientific endeavors. Guarino calls these further refined ontologies *application ontologies* (1997).

To create an effective application ontology for a customized data capturing and transformation process, a process author specializes concepts from one or more domain ontologies, hence reusing the groundwork of the knowledge engineer. The overall methodology of building scientist-driven ontologies to build knowledge bases for data capture and transformation activities is discussed in more detail in (Pinheiro da Silva et al. 2009).

The Framework

This section describes an ontology for data processing using description logics (Baader et al. 2003) and a graphical language intended to facilitate its use. Description logics is a family of formal knowledge representation languages that are popular in providing logical formalism for ontologies and the Semantic Web. In description logics, a knowledge representation system is divided in two components: TBox and ABox. TBox defines the terminology of an application domain. ABox includes assertions about named individuals in terms of the terminology defined in the TBox. Terminology is captured in the form of concepts and roles, where concepts are sets of individuals and roles are binary relations between individuals. Concept subsumption refers to the notion of determining whether one concept is more general than another, where the universal concept denoted by $\top$ subsumes all concepts. The counter part of the universal concept is the bottom concept denoted by $\bot$.

The following subsections describe the ontology and its graphical language in parts: data, data assertion and store, data derivation, resource attachment, and process composition. In addition, the ontology is aligned to a

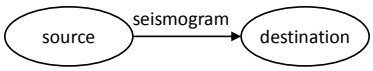

Figure 1: Graphical representation of a data concept connecting two node concepts.

provenance ontology in order to support the development of knowledge bases that can answer queries related to how data products are created. The ontology has been encoded in OWL and can be found at: `http://trust.utep.edu/2.0/wdo.owl`.

Data

The `data` concept was identified through interaction with the Earth science community and it is intended to represent things that can be used directly or indirectly as evidence for scientific findings. In a more pragmatic sense, `data` refers to content being captured, transformed, analyzed, used, or created in a process (Salayandia et al. 2006). The `data` concept is formalized in the description logics equations below. `Data` is defined in relation to the `container` and `method` concepts, which will be further described in the following subsections.

$$N \equiv C \sqcup M \quad (1)$$

$$D \equiv \forall isOutputOf.N \sqcap \forall isInputTo.N \quad (2)$$

$$\bot \equiv (D \sqcap C) \sqcup (D \sqcap M) \sqcup (C \sqcap M) \quad (3)$$

Equation 1 defines the `node` concept, denoted by N, as the union of the `container` and `method` concepts denoted by C and M respectively. Equation 2 defines the `data` concept denoted by D as things that are *output of* `node` and things that are *input to* `node`. The roles *isOutputOf* and *isInputTo* are chosen to indicate data flow. Notice that equation 2 requires `data` concepts to have both *isOutputOf* and *isInputTo* relations to `nodes`. Equation 3 defines `data`, `container`, and `method` concepts to be disjoint.

Figure 1 illustrates the graphical representation of the `seismogram` concept, which has been subsumed by the `data` concept[1]. The label of the directed edge corresponds to the name of the concept. The direction of the edge represents flow. In this case, `seismogram data` is flowing from the `source node` to the `destination node`.

Data assertion and storage

The `container` concept, denoted by C, represent acquirers or placeholders of `data`. In describing a `data` capture and transformation process, `containers` play the role of asserting `data` (e.g., a sensor capturing data from the field and providing it), storing `data` (e.g., a file server receiving data dump files for archiving), or both (e.g., a person reading a document and reporting on it). The relation between `data` and `container` concepts is formalized as follows:

$$C \equiv (\exists hasOutput.D \sqcup \exists hasInput.D) \sqcap \neg M \quad (4)$$

$$\bot \equiv \exists isOutputOf.C(a) \sqcap \exists isInputTo.C(a) \quad (5)$$

As mentioned above, the roles *isOutputOf* and *isInputTo* are chosen to indicate data flow. The roles *hasOutput* and *hasInput* are corresponding inverse roles. With respect to `data` and `containers`, `data` *isOutputOf* `container` represents `data` being *asserted by* a `container`, while `data` *isInputTo* `container` represents `data` being *stored in* a `container`.

Equation 4 states that `container` represents things that assert `data` or things that store `data` and that are not `method`. Equation 5 specifies that things cannot be asserted and stored from/to the same `container`. This restriction refers to the intention of the `container` concept to represent things that do not alter `data`. As such, having any one `data` concept being asserted and stored from/to the same `container` would be ineffectual.

Notice that the formalism does not specify `container` or `data` structure. For example, a database may be used to assert and store datasets in some system. The schema of the database and the formats of the datasets are not specified here. The intention is to facilitate process description by separating concerns between process author and process implementer, focusing on the point of view of the former.

`Container` concepts are represented graphically as ovals. Figure 1 shows the graphical representation of two containers labeled `source` and `destination`. `Source` represents a `container` that is asserting `seismogram data`, while `destination` represents a `container` that is storing `seismogram data`. Notice that this specification does not violate equation 5 above, since `seismogram` is not being asserted and stored by the same `container`. While it is still true that this specification does not have an effect on `seismogram`, such specification may be useful to indicate that `data` is being moved from one `container` to another, e.g., a dataset moved from main to archival storage.

Data derivation

The `method` concept, denoted by M, was originally identified through interaction with the Earth science community, and it is intended to represent discrete activities carried out to systematically transform `data` (Salayandia et al. 2006). `Data` is derived from previously existing `data` through the application of a `method`. Methods can be manual, e.g., a person analyzing and interpreting a dataset; methods can be automated, e.g., using extrapolation software with pre-established parameters to process a dataset; finally, methods can be hybrid, e.g., using software that requires inputs from a person to process a dataset. The relation between `data` and `method` concepts is formalized as follows:

$$M \sqsupseteq (\exists hasOutput.D \sqcup \exists hasInput.D) \sqcap \neg C \quad (6)$$

With respect to `data` and `methods`, `data` *isOutputOf* `method` represents data being *derived by* a `method`, while `data` *isInputTo* `method` represents data being *used by* a

[1]Notice that subsumed concepts can be referred by the name of the more general concept, e.g., the `seismogram` concept is a `data` concept. This convention is used throughout the paper.

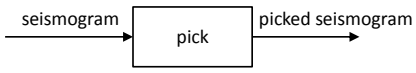

Figure 2: Graphical representation of a method concept using and deriving data concepts.

method. Equation 6 states that method includes things that *derive* data or things that *use* data and that are not container.

Method concepts are represented graphically as rectangles. Figure 2 shows the graphical representation of the seismogram data concept that is used by the pick method concept to derive the picked seismogram data concept.

Notice that the formalism does not directly specify the relation of a data concept being *derived from* another data concept. Instead, the focus is on the relations *used by* and *derived by* through an intermediary method concept. The intention is to explicitly identify the activities involved in data derivation and to simplify for the process author the specification of data derivation in a wide range of execution models. In other words, defining relations of data *derived from* data leaves out references to methods used to carry out such derivations. Furthermore, specifying data *derived from* data relations would require process authors to specify logical expressions to match the execution model and implementation of methods. For example, consider a method m that uses data concepts x1 and x2 and derives data concept y. In one implementation, m executes until x1 and x2 are available. The data derivation specification in this case would be: y *derived from* (x1 and x2). In another implementation, m executes when at least one input is available. The data derivation specification in this case would be: y *derived from* (x1 or x2). The specification for both implementations is the same with the proposed formalism: x1 *used by* m, x2 *used by* m, y *derived by* m.

It is expected that all methods will *use* at least one data concept and *derive* at least one data concept; after all, the intention is to specify a process about data capture and transformation. However, this restriction is relaxed in equation 6 through the use of an inclusion instead of an equality. The intention is to support incremental description of processes where a process author may identify a method concept before identifying data concepts to be used and derived.

The formalism does not define cardinality restrictions or control structures between data and method. Hence, the specification of Figure 2 does not specify whether one or more seismogram items are to be used by pick and whether one or more picked seismogram items will be derived. Also, the formalism does not specify triggering conditions for a method, and in the case of a loop, it does not specify the number of iterations to be performed.

In contrast to containers, methods are intended to change the state of data. A convention used here is that data concepts are linked to relevant states of data. If the relevant state of data changes, then data is no longer represented by the same concept. The process author determines which states of data are relevant. For example, with respect to Figure 2, a given dataset could be referred to as seismogram or as picked seismogram depending on the state of the dataset with respect to the data transformation process.

Through data assertion, derivation, and store, processes are effectively represented as connected graphs where leaf nodes are containers, intermediate nodes are methods, and connecting edges are data. Notice that containers are defined as things that necessarily *assert* or *store* data, while methods are defined as things that can *use* or *derive* data. The implications with respect to process graphs is that all container nodes must be connected, while method nodes can be disconnected. Similarly, data concepts are defined as things that necessarily are *asserted* or *stored* in containers, or things that necessarily are *used* or *derived* by methods. The implications with respect to process graphs is that all directed edges need to be connected to at least one node on each end. Finally, the restriction imposed by equation 5 above implies that process graphs cannot include container nodes with edge loops.

Resource attachment

Resource attachment refers to the inclusion of references in a process specification, e.g., a reference to a user's manual document. The attachment concept, denoted by A, is introduced to represent references to resources that can be attached to concepts used in a process specification, i.e., data, container, and method concepts. The process concept is also used; it will be described further in the next subsection. The formalism is as follows:

$$A \equiv \exists isAttachedTo.(D \sqcup C \sqcup M \sqcup P) \quad (7)$$

$$\bot \equiv A \sqcap (D \sqcup C \sqcup M) \quad (8)$$

Equation 7 defines attachment as things that are *attached to* data, container, method, or process concepts. Equation 8 defines attachment as things that are not data, container, or method.

The graphical representation of attachments is considered non-essential from the point of view of process description, i.e., a process graph composed of data, containers, and methods is sufficient to describe the capture and transformation of data while attachments provide additional information based on the structure of the process graph. As a result, the intention of the graphical representation of attachments is merely to signal the presence of additional resources for a specific part of the process. One possible graphical representation of attachments is to use a coloring scheme to signal the presence of attachments in data, containers, and methods.

Process composition

Process composition refers to the linking of two process description graphs. The intention is to support a mechanism by which processes can be described at multiple levels where one process is considered a subprocess of a more general process. The process concept, denoted by P, is introduced to formalize process composition as follows:

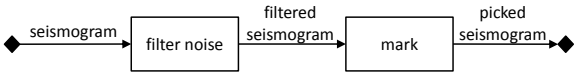

Figure 3: Graphical representation of a sub-process.

$$D' \sqsubseteq D, \;\; C' \sqsubseteq C, \;\; M' \sqsubset M, \;\; A' \sqsubseteq A \tag{9}$$

$$P \equiv D' \sqcup C' \sqcup M' \sqcup A' \tag{10}$$

$$P \sqsupseteq \forall isAbstractedBy.(M \sqcap \neg M') \tag{11}$$

$$M \sqsupseteq \forall isDetailedBy.(P \sqsupseteq M' \sqcap \neg(P \sqsupseteq M)) \tag{12}$$

Equation 9 introduces subsumptions of the `data`, `container`, `method`, and `attachment` concepts. Notice that in the case of `method` a proper subsumption is specified to indicate that there is at least one item that is not included in the subsumed concept. Equation 10 defines `process` to be the graph defined by the subsumed concepts. Equation 11 defines the relation where `processes` can be *abstracted by* `methods` that are not included in the definition of the `process`. Equation 12 defines the inverse relation where `methods` can be *detailed by* `processes` that do not include those `methods` in their definition. The intention is to support multiple levels of process description where a `method` represents an activity that is treated as black box in one general level and further described in another more specific level.

Figure 3 shows the graphical representation of a subprocess corresponding to the method illustrated in Figure 2. To maintain consistency between the super process and the subprocess, a leveling DeMarco structure similar to that of Data Flow Diagrams (DFD's) is supported, where the same number of inputs and outputs to a method node at the super level are maintained in the subprocess graph (Davis 1990). Special diamond-shaped nodes are used to maintain graphical consistency across levels. In addition, semantic consistency is maintained across process description levels by referencing `data` concepts across levels instead of duplicating concepts. In OWL encoding, this is accomplished by assigning unique identifiers to concepts, i.e., URIs, and using them for reference.

Aligning to a provenance ontology

Data provenance supports the analysis of how data is created or derived, e.g., which sources were used, who encoded the data, what equipment was used, and more. Capturing data provenance at an adequate level of granularity, however, is not a trivial problem (Stephan et al. 2010). Aligning a provenance ontology to the proposed framework is beneficial in determining an adequate level of granularity from the point of view of the scientist. Currently there are several languages and models for capturing data provenance and there is work underway to establish a recommendation from the World Wide Web Consortium (W3C) (`http://www.w3.org/2011/prov/`). The Proof Markup Language (PML) (McGuinness et al. 2007) ontology is used as a case study to leverage the perspective of process authors to capture data provenance.

PML defines concepts and roles for representing provenance about data. First, `identified thing` refers to entities from the real world. These entities have attributes that are useful for provenance, such as name, description, create date-time, authors, and owner. Three key subsumptions of `identified thing` motivated by provenance representation concerns are `information`, `source`, and `inference rule`. `Information` refers to information at various levels of granularity and structure. `Source` refers to `identified things` from which `information` is obtained. A `source` could be a document, an agent, a web page, among others. `Inference rules` are `identified things` that are used to derive conclusions from premises. Examples of `inference rules` are algorithms used to manipulate data or decision trees used to deduce outcomes from existing information. In PML, explanations of how `information` came to be are described in terms of justifications, in the sense that `information` existence is justified because some set of actions took place. Justifications are described in terms of `information`, `source`, and `inference rule` concepts.

To present the alignment of the framework and PML, the PML concepts of `information`, `source`, and `inference rule` are denoted by I, S, and IR respectively. Also, the concept `assert container` denoted by AC is introduced. The description logic formalism follows:

$$D \sqsubseteq I \tag{13}$$

$$AC \equiv \exists hasOutput.D \sqcap C \tag{14}$$

$$AC \sqsubseteq S \tag{15}$$

$$M \sqsubseteq IR \tag{16}$$

Notice that concepts `data`, `container`, and `method` are respectively subsumed by the PML concepts of `information`, `source`, and `inference rule`. In the case of `container`, the concept is more closely subsumed by the `source` concept if we only consider `containers` that *assert* `data` and not `containers` that *store* `data`. The reason is that PML focuses on describing justifications of `information`, which includes
containers that assert `data`; however, such justifications do not specify where `information` (or `data`) is stored after it has been derived, and hence, `store containers` are not included in the subsumption.

The alignment formalism includes subsumption equations instead of equalities because provenance-related concepts are more general than their corresponding framework concepts in the following sense: data can be transformed through a systematic process, in which case, the presented framework can be used to document the process. Alternatively, data can be generated through an ad-hoc approach, in which case, the framework cannot be used. In both cases, however, provenance about the generated data can be encoded with PML.

Evaluation

Petre suggests graphical languages do not guarantee clarity and that secondary notation is necessary to be effective in capturing complex processes. However, secondary notation in graphical languages requires training and experience from users to detect perceptual cues about important information (1995). The graphical language chosen for this framework is similar in nature to Data Flow Diagrams (DFD's), favoring language simplicity over expressiveness. Given the intention of using the framework to capture process author concerns instead of process implementer concerns, language simplicity is preferred. Figure 4 illustrates feedback from users with respect to ease of use. This is the result of a survey performed among 18 subjects from diverse engineering and science backgrounds, including college students, faculty, researchers, and professionals that either were involved in capturing and transforming data or were users of scientific data products. Over 75% of subjects agreed the graphical language was easy to use to describe process knowledge, while over 45% of subjects agreed the graphical language was easy to use to interpret process knowledge.

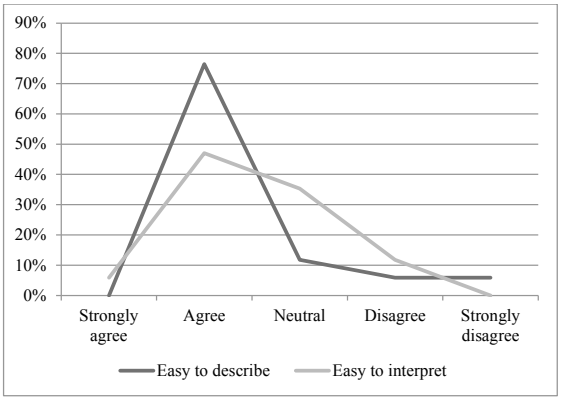

Figure 4: Ease of use.

Intuitiveness of the terminology was also evaluated in the survey. Figure 5 shows over 85% subjects found `Data` and `Method` concepts intuitive. However, `Container` was not as well received. One suggested alternative was `Data Container`. More interestingly, several subjects agreed that `Container` was not an intuitive term but could not think of a better alternative.

Finally, the overall usefulness of the framework was gauged using a scale from 1 to 10 that ranged from *too simple to be useful* to *too complicated to be useful*. The scale was designed without an exact middle, and hence, forced subjects to choose an inclination. As illustrated in Figure 6, subjects with engineering or programming expertise favored the *too simple to be useful* side, while scientists favored *too complicated to be useful*.

Conclusion

By focusing on abstractions that support process authors, a framework is presented that scientists can use to create

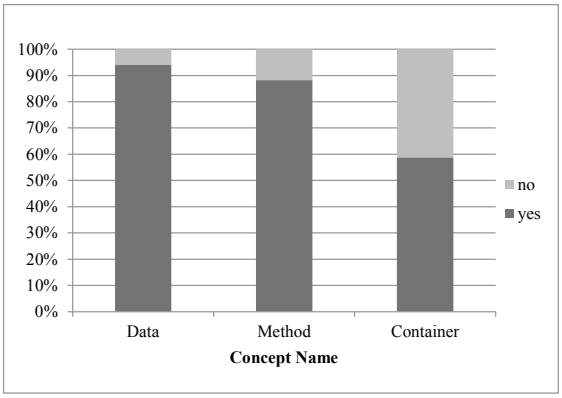

Figure 5: Intuitiveness of terminology.

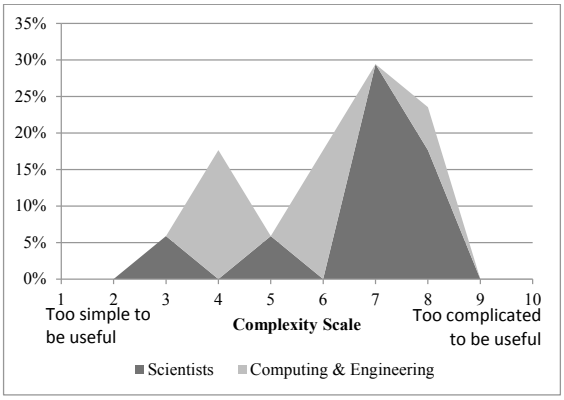

Figure 6: Overall usefulness.

ontologies that describe the processes of how data is captured and transformed into products that support scientific findings. The framework consists of an ontology formalized with description logic and aligned to a provenance ontology. In addition, the framework supports a graphical language similar in nature to Data Flow Diagrams in order to facilitate its use for authoring and interpreting processes about data capture and transformation.

The alignment of the framework to a provenance ontology provides the additional benefit of using the scientist's perspective to scope the granularity at which data provenance should be acquired, a non-trivial problem in complex data applications.

The framework is currently in use in an interdisciplinary setting with colleagues from the fields of Environmental Sciences and Earth Sciences. Evaluation with respect to the usefulness of the framework is positive. Concept terminology improvements are still needed to increase intuitiveness for scientists. A Java-based tool is available to support the authoring and interpretation of processes with this framework and can be downloaded from `http://trust.utep.edu/wdo`.

Acknowledgments

This work has been funded in part by the National Science Foundation (NSF) under CREST grant no. HRD-0734825. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Chandrasekaran, B., and Josephson, J. R. 1997. The ontology of tasks and methods. Technical Report SS-97-06, AAAI.

Davis, A. M. 1990. *Software Requirements: Analysis and Specification*. Upper Saddle River, NJ, USA: Prentice Hall Press.

Gates, A. Q.; Pinheiro da Silva, P.; Salayandia, L.; Ochoa, O.; Gandara, A.; and Del Rio, N. 2011. Use of abstraction to support geoscientists' understanding and production of scientific artifacts. In Keller, G., and Baru, C., eds., *Geoinformatics: Cyberinfrastructure for the Solid Earth Sciences*. Cambridge University Press. 266–.

Guarino, N. 1997. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In *Information Technology, International Summer School*, SCIE-97, 139–170. Springer Verlag.

Lenat, D. B., and Guha, R. V. 1989. *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st edition.

McGuinness, D. L.; Ding, L.; Pinheiro da Silva, P.; and Chang, C. 2007. Pml 2: A modular explanation interlingua. In *Proceedings of the 2007 Workshop on Explanation-aware Computing*, ExaCt-2007.

Petre, M. 1995. Why looking isn't always seeing: readership skills and graphical programming. *Commun. ACM* 38:33–44.

Pinheiro da Silva, P.; Salayandia, L.; Gandara, A.; and Gates, A. Q. 2009. Ci-miner: semantically enhancing scientific processes. *Earth Science Informatics* 2(4):249–269.

Pinheiro da Silva, P.; Salayandia, L.; Del Rio, N.; and Gates, A. Q. 2010. On the use of abstract workflows to capture scientific process provenance. In *Proceedings of the 2nd Conference on Theory and Practice of Provenance*, TAPP'10. Berkeley, CA, USA: USENIX Association.

Salayandia, L.; Pinheiro da Silva, P.; Gates, A. Q.; and Salcedo, F. 2006. Workflow-driven ontologies: An earth sciences case study. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, 17–. Washington, DC, USA: IEEE Computer Society.

Stephan, E.; Halter, T.; Critchlow, T.; Pinheiro da Silva, P.; and Salayandia, L. 2010. Using domain requirements to achieve science-oriented provenance. In McGuinness, D.; Michaelis, J.; and Moreau, L., eds., *Third International Provenance and Annotation Workshop*, volume 6378 of *IPAW'10*, 301–303. Springer, New York.