Towards an Effective Methodology for Rapidly Developing Component-Based Domain Ontologies

Dave Kolas
BBN Technologies
1300 N. 17th Street, STE 400
Arlington, VA 22209
Email: dkolas@bbn.com

Troy Self
BBN Technologies
1300 N. 17th Street, STE 400
Arlington, VA 22209
Email: tself@bbn.com

***Abstract*—As the Intelligence Community migrates from a paradigm of using disjoint, data and application stovepipes to a paradigm of shared knowledge and networkeded component services, the cost of developing appropriate domain ontologies becomes a concern. In this paper, we present a methodology for rapidly developing composite domain ontologies by linking and reusing existing ontologies. We will compare composite domain ontologies to component-based software engineering, present a metric for measuring the compositeness of an ontology, and describe a composite domain ontology developed for real world use.**

I. Introduction

While a significant amount of research effort has been devoted to creating ontologies[1], very little has been focused on the high-level engineering of ontologies. Current research details interesting ways to extract knowledge from subject matter experts[2], ways to align and interoperate between existing ontologies[3], and ways to extract information from traditional relational databases into ontologies[4]. Current software products provide tools for building ontologies, and thus make the process less painful, but fail to explain how one should know what to build.

Many individuals and institutions in the Semantic Web research area, particularly those who started in knowledge representation work long before the "Semantic Web" idea existed, think of the ultimate solution of knowledge representation in the form of one grand ontology that can cumulatively reason about anything. The relationships in this ontology will be objectively true in all cases, resulting in an ontology that can be used for any purpose at any time and in any context. This position fails to account for the "Web" portion of the Semantic Web; many knowledge representations will be developed by many individuals for many purposes, and they may not always be exactly compatible. While the one true ontology may one day exist, until that time, a different type of engineering is required.

It is our goal in this document to describe a methodology for engineering ontologies in the real world, where ontologies are expected to interoperate. We will focus on designing ontologies that will be used for a purpose, particularly in concert with computer application software. The recommendations in this document are derived from research done on the GARCON-F program as well as the collective experience of our research group in building ontology-based applications. We hope to espouse a set of ontology engineering principles that closely parallel those of component-based software design[5]. From an abstract engineering point of view, ontologies and software components have many common characteristics:

- Both ontologies and software components will likely be used in a multitude of application situations, not all of which are anticipated by

the designer.
- Both ontologies and software components will likely evolve after their initial creation, due to changing requirements, additional use cases, etc.
- Unlike hardware, software components and ontologies can be easily replicated once developed.
- Reusing either ontologies or software components has the potential to save a considerable amount of engineering resources.
- Care must be taken to make both ontologies and software components specific enough to be useful, but general enough to be reusable.

It is this set of common characteristics that lead us to approach ontology engineering in much the same way as component-based software engineering[6]. When viewed abstractly, ontology design and software design are merely two parts of the larger task of designing an application: building an intelligent data representation, and building components that can act on that representation. Thus we will define a composite domain ontology as an ontology built from logical ontology components, much the same way as a large piece of software is built from software components.

For the remainder of this document, when we refer to an ontology, we will generally mean an OWL Web Ontology Language[7] ontology. The knowledge representation language, OWL, has several qualities that make it particularly well suited for building the types of purposeful ontologies we have discussed. First, it has a natural ability to link ontologies together. This is critical to achieving the goals of reuse. Second, it takes a delicate balance between expressivity and computational tractability, yielding ontologies that are practical. Third, the large and growing number of existing tools makes interaction with software components significantly easier. That said, we hope that much of what is described here is applicable to engineering other knowledge representations, particularly those with similar properties to OWL.

II. Ontology Design Goals

In this section, we will describe the goals of designing a composite ontology. The subsections will ideally both propose general guidelines by which ontologies should be designed and justify these guidelines. To discuss the following guidelines, a working definition of compositeness will be required. In this section, we present an intuitive, loose definition of compositeness; a more formal, measurable definition of compositeness will be presented later.

The compositeness of an ontology is defined as the degree to which its content is made up primarily of other ontologies, and the degree to which these ontologies are made up of other ontologies, etc.

With that definition established, we examine why and when composite ontologies should be created.

A. Time Savings

The most immediate benefit achieved by building composite ontologies is elimination of much of the time required to design and build them. Extracting knowledge from subject matter experts can be a difficult and expensive process, and encoding that knowledge into a structured form can be quite difficult as well. By building an ontology from the building blocks of other ontologies, it is possible to significantly reduce the amount of time required to build the ontology.

Just as in software, the time savings gained by reusing another ontology is not totally cost-free; incorporating another ontology subjects the new ontology's designer to the design decisions of the component ontologies. However, as with software, the time spent on tuning the incorporation of an ontology into a larger ontology is most often vastly less than the time spent developing a new ontology from scratch.

B. Interaction with Software

As the number and scope of Semantic Web applications increases, so too will increase the symbiosis between these applications and their related ontologies. For instance, in the GARCON-F program, the client UI runs within ESRI's ArcMap software, providing the capacity to display semantic

annotations in a geospatial context. This software expects specific geospatial and temporal ontologies: GeoRSS for geometries and OWL-Time for temporal information. Any additional ontological information is processed dynamically by the software.

Consider the case of an ontology designer creating a new ontology to be used with this system. It is likely that whatever domain he or she is attempting to represent requires geospatial and temporal information; otherwise, the software would not be applicable. The ontology designer can choose between creating their own definitions for space and time, or merely adopting GeoRSS and OWL-Time. The former is possible by adding ontology translations to the system, but the latter makes the client software instantly able to process data from the new ontology with no software changes required.

Many applications that deal with Semantic Web data function this way. A few ontologies are treated specially, being used for a particular purpose. Other ontologies provide extra information or linkage to other applications. In this paradigm, reusing ontology components provides significantly greater tool interoperability.

C. Reasoning and Translation

Even after an ontology has been created, additional layers of reasoning are often added. These might come in the form of additional SWRL rules, translations, or linkages between ontologies. For instance, there may be existing translations from GeoRSS to another spatial representation. These translations then inherently work on any ontology that incorporates GeoRSS for spatial representation. This is very much like the automatic interoperability of software components described above. In a sense, this type of reuse driven by composite ontologies multiplies the effectiveness of software reuse with composite ontologies. If there are three tools that use three different ontologies for one particular purpose, and there are existing mappings between these ontologies, then a designer of a new ontology can choose between any of the three and get the automatic benefit of the translations and the software.

D. Scope and Applicability

It is not our intention to suggest that all ontologies should be highly composite. We generally expect the compositeness of an ontology to vary in accordance with the simplicity or abstractness of the task for which the ontology is designed. The more abstract an associated task is, the higher the level of appropriate compositeness. Consider the GeoRSS ontology mentioned previously. This ontology has a very particular, focused purpose. The goal for the GeoRSS ontology is to attach point and polygon spatial data to other ontological entities. Because this purpose is narrow in scope and widely applicable, it makes sense for the GeoRSS ontology to be simple and not composite.

On the other hand, consider an ontology for air defense, as developed earlier in the GARCON-F program. The goal for an air defense ontology might be to annotate rich data about air defense scenarios, to reason about these annotations, and to generate reports about them. This ontology must be large in scope but only narrowly applicable. This is the scenario in which it makes sense for an ontology to be very highly composite.

Obviously there is a large amount of space between these two extremes, and finding the appropriate level of compositeness for an ontology is part of the process of creating it. We will attempt to give guidance on this as well in the following sections.

III. Measuring Compositeness in Ontologies

A. Relationship to Software Reuse Metrics

Software metrics are effectively used to alleviate quality and performance concerns during development[8]. We believe that analogue metrics can be developed to measure quality, reuse, and performance in ontologies. These metrics could eventually be applied to measure these characteristics in existing ontologies as well as to guide the planning and development of new ontologies.

In software, reuse and compositeness is measured by quantifying abstract constructs of the software development language, such as lines of code and the number of references between components[9]. While we envision an eventual set of metrics for

measuring quality and reuse of ontologies, we begin in this paper by defining a single metric for measuring the compositeness of an ontology.

The goal of a compositeness metric for an ontology is to have a quick estimate of the ontology's modularity. The more composite the ontology is, the more likely it is that one could repurpose significant parts of the ontology and thus minimize further development.

B. Definitions

- The compositeness of a given ontology X is the function $C(X)$.
- The size of a given ontology X is the function $S(X)$.
- An ontology X importing an ontology Y is defined by the relation $I(X, Y)$.
- The set of all ontologies imported by X is $I(X)$.

C. Desirable Properties of a Compositeness Metric

If an ontology imports another ontology, its compositeness is greater than the imported ontology.

$$I(A, B) \rightarrow C(A) > C(B)$$

If two ontologies import the same ontology, and one of the importing ontologies is smaller than the other, that ontology has higher compositeness.

$$I(A, B) \wedge I(C, B) \wedge S(A) < S(C)$$
$$\rightarrow C(A) > C(C)$$

If two ontologies that are otherwise identical import different ontologies, the one that imports the more composite ontology will have higher compositeness.

$$I(A, B) \wedge I(A', C) \wedge C(B) > C(C)$$
$$\rightarrow C(A) > C(A')$$

If one ontology that is otherwise identical to a second ontology imports an ontology that the second does not, that ontology will have higher compositeness.

$$(I(A, B) \wedge I(A', B) \wedge I(A', C)$$
$$\wedge \forall X : (I(A, X) \wedge X \neq C))$$
$$\rightarrow C(A') > C(A)$$

D. A Compositeness Metric

To meet the desired properties, we define a compositeness metric as illustrated in Equation 1.

$$C(X) = \sum_{y \in I(X)} C(y) + \frac{\sum_{y \in I(X)} S(y)}{S(X)} \quad (1)$$

IV. Building Composite Ontologies

This section will define a process for creating effective composite ontologies. By following this process, ontology engineers should create an ontology with maximum portability and reusability.

A. Start With the Application Domain

The most important aspect of creating a useful ontology is starting with a particular application in mind. By starting with an application, the knowledge engineer immediately creates a scope that the ontology will need to fulfill, and thus prevents the ontology from slowly expanding to represent unnecessary concepts and relationships. This process is analogous to software requirements gathering. In order to build an effective ontology, the knowledge engineer must answer the following questions:

- What questions/queries should the user or software client be able to ask of the ontology?
- What should the instance data look like?
- What types of inference will the ontology need to provide?

If a concept or relationship is not part of any of these three sets, then it need not be included in the ontology.

B. Divide the Goals

Once the overall scope of the ontology has been established, the next step in the process is to divide the scoped relationships and concepts into logical subcomponents. These components could be both aspects of the data (geolocation, temporal information, provenance information) or dividable subparts of the overall domain (vehicles, ground systems, etc for air defense). When attempting to partition the ontology, the following questions should be asked:

- Could this partition be successfully reused without the other partitions?
- Would making this partition allow a piece of software to work without understanding the other partitions?
- Are there known existing ontologies that fulfill part of the overall scope?
- Can some part of the overall ontology be viewed as a specialization of another part?

By answering these questions, the knowledge engineer should be able to tentatively create partitions of the ontology, and create a dependency graph between the parts.

C. Identify Reusable Ontologies

Once partitions have been identified, all attempts should be made to fill them in with suitable existing ontologies. The applicability of a given ontology component should be evaluated by how well it fits the partition, how widely it is used, and the quality of its construction.

It is entirely possible that an ontology that does not perfectly fit a partition may be desirable nonetheless, especially if it is already in wide use. The ontology component breakdown may be revisited during this step to accommodate existing ontologies with slightly larger or smaller scope.

D. Develop New Component Ontologies

Once existing ontologies have been worked into the overall ontology, the remaining pieces must be created. Other work addresses this part of the process; here we only advise careful attention to the scope of the ontology created in the first step. If a particular concept will never be directly inserted as data, queried for, or inferred over, it should not be included.

E. Link Component Ontologies Together

The final step is linking the ontology components together. This is accomplished by importing ontology components from the other components, mirroring the dependency graph created. As in software, care should be taken to avoid circular dependencies. While a circular dependency will not prevent compilation as in software, it will significantly reduce reusability of any of the components involved.

When appropriate, ontology components should link directly into the components they inherit from via subclass, subproperty, or restrictions. This provides the cohesion between parts necessary for effective use of the overall ontology. Occasionally, if multiple components are fulfilled by existing ontologies, glue components will be required to contain these relationships between parts. It is preferable to create a new linkage component rather than directly changing an existing ontology.

V. Example: Building a Raid Mission Planning Ontology

A. Geospatial Semantic Annotation Tool

The Geospatial Semantic Annotation Tool (gSAT) is a platform for capturing imagery annotations using ontologies[10]. Previously, gSAT had been used for annotating imagery intelligence in the Air Defense domain. The purpose of this exercise was to rapidly develop a new domain ontology that could be used within gSAT without changing any of the software. The new domain was Raid Mission Planning as defined by the United States Marine Corps for Counter-Insurgency (CoIn) operations. A raid mission is a military mission where forces quickly advanced on a chosen target and then leave. Examples of a raid can include attacking a known enemy location to eliminate its threat capability or evacuation procedures, such as removing non-essential staff from an embassy.

B. Raid Ontology

The concepts and relationships defined for this ontology are based on discussions with United

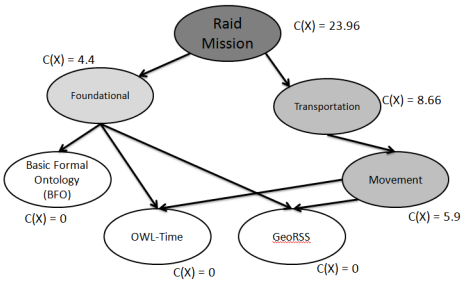

Fig. 1. The Raid Mission ontology is comprised of multiple ontologies for representing time, space, and reusable features, such as vehicles and routes. The arrows indicate that one ontology imports the other.

States Marine Corp imagery analysts. The raid ontology needed to include concepts and relationships necessary to represent an observation of a physical feature at some geospatial location at a particular time. The representation for an observation consisting of a what, where, and when were reused from gSAT. The concepts to be annotated included routes, landing zones, assault support vehicles, enemy facility types, and dangerous locations, such as potential IED areas and potential sniper positions.

Figure 1 shows the various ontology components in the Raid Mission ontology and their dependencies. Each circle represents an ontology and includes its compositeness score according to the metric defined in Equation 1. The ontologies that show $C(X) = 0$ do not import any other ontologies.

As the diagram shows, the Raid ontology is the most abstract, and directly or indirectly imports all of the other ontologies. The most foundational ontologies are at the bottom, and are imported into the Raid ontology along two different paths. Most of the components in the ontology are reused.

Reusing the ontologies as shown above allows the gSAT annotation system to create spatiotemporal semantic annotations in the Raid domain without changing any aspect of the software. This is because the software only needs to directly understand the parts of the ontology that were reused: the foundational, temporal, and spatial portions. Thus in this case the reuse of ontologies has led to 100 percent reuse of the software.

VI. Further Research

This document describes the initial research into a formal methodology for developing composite domain ontologies. The compositeness metric defined in this document requires further testing against a larger reference set of ontologies to validate its correctness. Further exploration into other metrics of ontology reuse is necessary. It is important to encorporate metrics that consider the ontology's internal complexity, cohesion, and other measures[11]. Since component-based software has multiple metrics for measuring quality, it is expected that component-based ontologies should also have multiple metrics. The compositeness metric described here only considers the size of ontologies and the single relationship of $imports$ between them. Future metrics must consider the amount of linking and semantic complexity between linked ontologies.

VII. Conclusion

In this document we have likened the process of engineering ontologies to component-based software engineering. We have demonstrated the benefits of designing ontologies this way, and defined a process for creating such ontologies. We have also defined a metric for determining how composite a particular ontology is. Our hope is that this analysis will help others create more effective ontologies in the future.

References

[1] D. Bianchini, V. De Antonellis, and M. Melchiori, "Domain ontologies for knowledge sharing and service composition in virtual districts," in *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, Sept. 2003, pp. 589–594.

[2] X. Wang, X. Wang, and F. Wang, "How to use class axioms to model ontology effectively in owl," in *Knowledge Acquisition and Modeling Workshop, 2008. KAM Workshop 2008. IEEE International Symposium on*, Dec. 2008, pp. 601–604.

[3] J. Sampson, M. Lanzenberger, and C. Veres, "Facilitating interoperability in semantic web applications using ontologies," in *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, March 2008, pp. 233–239.

[4] Z. Qu and S. Tang, "Research on transforming relational database into enriched ontology," in *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on*, Dec. 2008, pp. 749–753.

[5] L. Etzkorn and H. Delugach, "Towards a semantic metrics suite for object-oriented design," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on*, 2000, pp. 71–80.

[6] A. Farooq, A. Shah, and K. Asif, "Design of ontology in semantic web engineering process," in *High Capacity Optical Networks and Enabling Technologies, 2007. HONET 2007. International Symposium on*, Nov. 2007, pp. 1–6.

[7] M. Dean and G. Schrieber, Eds., *OWL Web Ontology Language Reference*. W3C Recommendation, February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[8] S. Sedigh-Ali, A. Ghafoor, and R. Paul, "Metrics and models for cost and quality of component-based software," in *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, May 2003, pp. 149–155.

[9] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Comput. Surv.*, vol. 28, no. 2, pp. 415–435, 1996.

[10] T. Self, D. Kolas, and M. Dean, "Ontology-driven imagery analysis," in *Proceedings of the Second International Ontology for the Intelligence Community Conference OIC-2007*, 2007.

[11] Y. Ma, X. Ma, S. Liu, and B. Jin, "A proposal for stable semantic metrics based on evolving ontologies," in *Artificial Intelligence, 2009. JCAI '09. International Joint Conference on*, April 2009, pp. 136–139.