The need for ontologies: Bridging the barriers of terminology and data structure

Leo Obrst*
The MITRE Corporation, 7515 Colshire Drive, McLean, Virginia 22102-7508, USA

Patrick Cassidy
MICRA, Inc., 735 Belvidere Ave., Plainfield, New Jersey 07062, USA

ABSTRACT

This chapter describes the need for complex semantic models, i.e., ontologies of real-world categories, referents, and instances, to go beyond the barriers of terminology and data structures. Terms and local data structures are often tolerated in information technology because these are simpler, provide structures that humans can seemingly interpret easily and easily use for their databases and computer programs, and are locally salient. In particular, we focus on both the need for ontologies for data integration of databases, and the need for foundational ontologies to help address the issue of semantic interoperability. In other words, how do you span disparate domain ontologies, which themselves represent the real-world semantics of possibly multiple databases? We look at both sociocultural and geospatial domains and provide rationale for using foundational and domain ontologies for complex applications. We also describe the use of the Common Semantic Model (COSMO) ontology here, which is based on lexical-conceptual primitives originating in natural language, but we also allow room for alternative choices of foundational ontologies. The emphasis throughout this chapter is on database issues and the use of ontologies specifically for semantic data integration and system interoperability.

INTRODUCTION

Differing terminology and different database structures within various communities create serious barriers to the timely transfer of information among communities. Barriers to accurate transfer of information between communities, also called semantic interoperability, can be so large that they present a practical impossibility, given the limited resources available for most tasks. Nationwide, lack of semantic interoperability is estimated to have cost U.S. businesses over $100 billion per year in lost productivity (http://en.wikipedia.org/wiki/Semantic_interoperability). Such estimates do not consider loss of opportunity for timely action due to slow information sharing, nor do they measure lost opportunities that would arise from existence of a standard of meaning that would allow new, independently developed programs to interoperate accurately. The effects of the lack of semantic interoperability on national security are difficult to measure, but they add to this economic inefficiency.
The difficulty is not in transferring bits and bytes of data but in transferring information between separately developed systems in a form that can be correctly machine-interpreted and used for automated inference. Optimally, the information to be transferred must be in a neutral form and independent of the original purpose for gathering and recording the data. If recorded in such a form, it can be reused in an automated process by any system regardless of its origin. Computational ontologies, developed over the past 30 yr, provide the current state of the art for expressing meanings in such an application-neutral form. This report discusses why adoption of a common foundation ontology (FO) throughout an enterprise can enable quick and accurate exchange of machinery-interpretable information without modifying local users’ terminologies or practices, and the principle is briefly explored as it relates to sociocultural and geospatial information.

A foundation ontology as described here is an ontology that represents the most basic and abstract concepts (classes, properties, relations, functions) that are required to logically describe the concepts in any domain ontology that is mapped to it. The principles described here apply to sharing of information among all domains, including geospatial information.

The need for an ontological approach can be better understood by examining the limits of some common approaches to interoperability. Commercial vendors have developed systems that reduce this inefficiency. The syntactic and structural processes are called “data integration” or “enterprise information integration.” Among the tactics proposed is the process of “data warehousing.” In this process, data from multiple databases are extracted, transformed, and loaded into a new database that provides an integrated view of the local enterprise. This tactic is practical only within a single enterprise that has a central director that can enforce data sharing and adequately fund the integrated database. In addition, it typically does not allow real-time updates of the integrated database. Furthermore, there are no explicit semantics associated with this approach; the semantics remains implicit, individuals must inspect the structural schemas, the appropriate data dictionaries, and the application procedural code that surrounds access to or processing of the data from the database in order to understand the intended semantics.

Here is an example. One agency may have a database that includes a table of terrorist organizations, with individual members related to each organization by the column (attribute) “member” in the “terrorist organization” table. One record might specify a member of such an organization and the member’s position in the organization. The program using that database may also access lists of people trying to enter the United States, and it will immediately notify border agents to detain anyone who appears in the “member” relation of a terrorist organization for questioning. Another agency may have a database with a table of individuals considered “persons of interest.” A column “member-of” might specify the organizations to which an individual belongs. The program using that database may be searching for social relations among persons of interest, perhaps using Structured Query Language (SQL) or another query language. A common membership could indicate a significant link. If those two agencies share their data, neither program will recognize the significance of the alternative way of specifying an organization’s memberships. The two organizations represent the relations in an inverse fashion, so even a cross-reference table that maps columns of identical meaning would be unable to catch this relationship. The meaning (semantics) of social relations are locked in the procedural code of the programs that perform actions based on the implied, but not explicit, meaning of the data tables. However, if both databases are properly mapped to an ontology, even simple ontology formalisms, such as the Web Ontology Language (OWL) (Bechhofer et al., 2004), will automatically generate the inverses of the specified relations. Consequently, the attempt of that potential terrorist trying to enter the country could be recognized immediately and automatically, in spite of differences in the way information is encoded in different databases.

Another situation may arise where one agency has a database of social relations containing the field “CloseRelatives,” which lists all relatives with up to two links of family relations (i.e., parent, uncle, aunt, brother-in-law, etc.), but another agency may only list specific relations individually. In the ontology, the specific relations will be subrelations of the more general relation (e.g., an uncle will also be recognized as a “close relation”). A query for “CloseRelatives” on the database will return the more specific relatives from other databases mapped to the ontology, using the inference mechanism of the ontology. Other examples and scenarios illustrating the advantages of using ontologies for database integration are provided later in this paper. Appendix A will describe the similarities between ontologies and databases. Appendix B will describe the use of ontologies for federated queries across multiple databases. Appendix C will provide an example of a query translation from the ontology to multiple databases using that ontology.

When different geospatial communities develop ontologies or databases derived for different purposes, sharing their data will require the logical descriptions of geospatial entities as well as other concepts to be carefully specified. In order for logical inference to work consistently among different ontology-based applications, the concepts, especially the relations, to be shared in common must be logically represented in terms with an agreed-upon meaning, and in formats that, if not identical, can be accurately translated. One common concept is that of “distance.” However, to date, there is little agreement even on a common representation for units of measure, among which units of distance, time, and force (gravity, geomagnetism) will be used frequently in geospatial applications. Note that there are existing or emerging ontologies for units of measure, e.g., the emerging OASIS (Organization for the Advancement of Structured Information Standards) Quantities and Units of Measure Ontology Standard (QUOMOS, 2011) Technical Committee, the National Aeronautics and Space Administration (NASA) Quantities, Units, Dimensions, and Data Types (QUDT, 2011) ontology, the Cyc Quantity and Units of Measure representation, and the Ontolingua physical quantities ontology (Ontolingua,
The need for ontologies

1994), but as of yet, there is no common agreement on any of those ontologies.

For precise reasoning, representation of geographical features such as land masses, atmosphere, and bodies of water will depend on having a common representation for substances such as rock, sand, air, and water, for light and heat, and for objects such as vegetation or artifactual structures (buildings, roads, bridges), as well as for spatial concepts.

In some geospatial applications, entities such as countries are represented as literal strings that may be interpreted in different ways by the procedural code of different applications. This leaves a lot of room for error in interpretation when the different aspects of a country may need to be reasoned with; distinctions may need to be made between the spatial region of the geographical territory, the physical objects (structures, mineral resources) within the geographical territory, the government controlling that territory, the population or citizenry, and the abstract notion of a country as an agent that can perform certain actions and enforce its laws.

Later herein, we discuss the reasons for adopting a common foundation ontology (FO) that contains at least the primitive elements used in the different applications, which will provide a basis for accurate interoperability among those applications. Examples of such primitive elements, in addition to units of measure, that may frequently be required are: the geometric primitives of the Open Geospatial Consortium’s Geospatial Markup Language (GML, 2011) standard, spatial relations such as those formalized in the Region Connection Calculus (Randall et al., 1992; Renz, 2002; Obrst, 2002; Cohn and Gotts, 1994), time relations such as those formalized in the OWL Time ontology (Hobbs and Pan, 2004; Fernández-López and Gómez-Pérez, 2004; Hayes, 1996), geographical relations such as those in the Geographical Ontologies of Bitters (2002), and others.

In some cases, specifications for the intended meanings of terms may require some observational or measuring procedure to be identified, and that procedure must then be related to the term to make the intended meaning unambiguous. One example from the geographic information system (GIS) domain can be seen in the description by Schuurman and Leszczynski (2006, p. 711–712) of the way in which the term “crown closure” (the density of forest canopy) can be defined using the same words by two different organizations, but it can be measured by different procedures, resulting in different values for that property for the same section of forest. The terms can only be made unambiguous for computational purposes, and its uses related to each other, by including a reference to the measuring procedure as part of the logical description.

Certain mathematical procedures may also need representation as primitives. As mentioned by Fonseca et al. (2003), in translating information from one GIS application to another, it may be necessary, for example, to convert a Triangulated Irregular Network (TIN) representation of a region to a representation using isolines. In such a case, some interpolation technique(s) will be needed to convert the different kinds of representation.

For information to be converted accurately and consistently by means of a foundation ontology, the interpolation procedure used will need to be specified and represented in the ontology, so that any GIS application will be able to accurately transfer its information in a form usable by others, and be able to reason with either representation as required. As with measuring procedures, addition of an interpolation procedure to the ontology might require adding some additional primitive ontology elements beyond those found in the common foundation ontology, but interpolation procedures may themselves be constructible from more basic primitives, and they may not require additions to the primitive inventory of the foundation ontology. Such procedures for specialized fields may require new primitive elements, but they may themselves be constructible from primitives already present in the foundation ontology. If required, new primitives for specialized purposes may not need to be included in the most basic foundation ontology, but they can be included in a midlevel extension that would be used only by domain ontologies that require use of such procedural primitives.

For the purposes of this paper, and the prototype addressing the sociocultural semantic modeling that engendered the need for this paper, we chose the Common Semantic Model (COSMO) foundational ontology (Cassidy, 2008), which is largely based on the development of semantic primitives behind the common words and concepts of natural language.

Similar to the COSMO approach, Barker, Porter, and Clark (2001) described the use of a foundation ontology, though their set of primitives appears to be smaller than our interpretation of the set needed for a broadly adequate foundation ontology. After a basic foundation ontology is adopted, new applications may be mapped to it that have primitive elements not decomposable into elements already in the foundation ontology, and those missing elements will need to be added in order to support interoperability of the new application with other applications already mapped to the foundation ontology. By mapping to such a common foundation ontology, all geospatial applications may share the same information without loss of conceptual precision, and they may share it with other communities that use the foundation ontology as the basic standard of meaning.

To overcome the opaqueness of meanings in traditional databases within the federal government, a few relatively isolated divisions have attempted approaches to ensure that shared information would be recognized by all programs sharing the data. However, a strategy for retrieving information from multiple databases in a form suitable for automated inference has not yet achieved widespread use.

The difficulty of reaching common agreement on one standard of meaning has been a subject of speculation and debate. Systems that are not based on ontologies have limited expressiveness and do not support the logical inferencing that will permit full representation of meanings. This may inhibit potential users who do not find the representation models they need or prefer. The ontologies that are sufficiently expressive are time-consuming to learn effectively. Without publicly available
programs demonstrating their usefulness, motivation to undertake the cost of implementing them has limited most programs to a small exploratory phase.

For ontologies, this latter point suggests that it is only a matter of time until convincing utility demonstrations are developed to provide the motivation for the necessary resources. Another factor is an almost universal misunderstanding among those who have not studied ontology technology regarding bridging terminology barriers. Because of a lack of understanding of the new technology and misleading focus on linguistic phenomena, it is still widely, but incorrectly, believed that overcoming these barriers will be impossible. An additional factor is that individual ontology systems are not used by a large community and thus the number of third-party vendors of utilities that make ontology simple is small. Only in the past few years have vendors developed programs that make use of ontologies easier, and, thus far, mostly for the OWL ontology language. Another issue is the lack of agreement among the ontology research community regarding the content of the foundation ontology. One attempt to avoid the latter problem is to view the foundation ontology as a “defining vocabulary,” which is being pursued in developing the COSMO ontology (Cassidy, 2008), in parallel with the sociocultural data model.

Most efforts at retrieving information from multiple databases in a form suitable for automated inference suffer from a lack of explicit, expressive, and application-neutral data representation. One well-developed information exchange system, the National Information Exchange Model (NIEM, 2011), grew from a Department of Justice initiative to exchange information about criminal suspects to a broader program to support exchanging information across the federal government (e.g., to the Department of Homeland Security). Although NIEM is widely used, it does not have the expressiveness of an ontology, and, consequently, other projects within the federal government are still attempting to develop alternative approaches in order to move beyond the simple message-passing supported by NIEM formats (i.e., Information Exchange Package Documentation).

One initiative (Wisnosky, 2010), apparently similar in several ways to the one we suggest here, is now being undertaken within the Business Mission Area of the U.S. Department of Defense. In that approach, a common core vocabulary based on primitives is logically specified by an ontology that constrains the meanings of the terms; the common vocabulary provides the basis for translating data among local data stores. This system is being developed incrementally, but details and examples of performance are not yet available for public inspection.

Several industry groups have developed specialized information exchange standards, which are also commonly aimed at accurate information exchange (e.g., UCore [2009], the National Building Information Model Standard [2011], and the Standard for the Exchange of Product model data [STEP, 2011], International Organization for Standardization [ISO] standard 10303: see http://www.iso.org/iso/iso_catalogue.htm).

The primary alternative to the use of a common foundation ontology that has been explored by a number of groups is ontology mapping or ontology alignment (Kalfoglou and Schorlemmer, 2003, 2004; Ehrig, 2005; Euzenat and Shvaiko, 2007; Musen and Noy, 2002; Stoutenburg, 2009). There have been conferences and workshops devoted to that technique. However, the accuracy of automatic mapping is low (usually less than 60%, often much less) and insufficient to support automated inference in mission-critical applications. Semi-automated mapping techniques may be useful to align independently created ontologies to the common foundation ontology when the domain ontology is not created originally from components provided by the foundation ontology. In the context of geospatial ontologies, matching techniques have been studied by Cruz and Sunna (2008) and Sunna and Cruz (2007), who also found that conceptual heterogeneity among classification methods required human inspection to deduce proper alignments for most classes. For the purpose of automated inference, the difficulties related to automated alignment are considerably worse than for many of the cases studied because alignment between the all-important semantic relations in formal ontologies is very complicated, and it can be challenging even for experienced ontologists.

To achieve generalizability beyond simple message passing, the federal government has embarked on the Community of Interest (COI) paradigm. The COI paradigm enables two or more organizations that form a COI to develop specific vocabularies and models of those vocabularies to share information. Some organizations, such as the Enterprise Vocabulary Team (EVT) under the Air Force CIO office (Parmelee, 2007) have adopted or are adopting principled approaches to developing COI vocabularies and models. The EVT is developing semantic models of the COI vocabularies, which are expressed in OWL, and then it is generating downstream products (e.g., XML schemas) for the actual exchange of data.

Other approaches, including some with ontologies in their architectures, do exist. For example, several projects within the federal government are aimed at developing a “core” model for information that represents concepts common to many domains and that can be used as a tool for integrating different domain models. The Department of Defense (DoD) Core taxonomy (Hayes, 2005), NIEM, Universal Core (UCore), Command and Control (C2) Core, the Maritime Information Exchange Model (MIEM, 2008), the Federal Enterprise Architecture Reference Ontology (FEA-RMO), and other efforts within the intelligence community are examples. Except for a few ontologies used in the intelligence community, these efforts do not use ontologies, but instead they use data schemas. They do not have the expressiveness required to unambiguously represent the meanings of concepts from different domains of interest. Examples of this expressiveness problem are given in the body of this chapter and in Appendix B. Exceptions to the attributions here are the ontologies developed for the semantic layers of UCore and C2 Core (Winters and Tolk, 2009; Smith et al., 2009), which, however, are recognized only as products affiliated with the standards and not direct products of those standards.
These other approaches do not provide a general solution to exchanging information among databases with heterogeneous types and purposes. Some form of information exchange system must be developed to allow true federated (or consistent distributed) queries from a requester with the proper permissions to the databases to operate on one of the federal networks without requiring the local systems or practices to be modified to accommodate the external query beyond granting access. The following section describes the method, based on existing technology, which is likely to achieve that goal.

THE ONTOLOGY SOLUTION

An ontology solution requires both expressive ontologies and a well-constructed ontological architecture (Obrst, 2010), the latter of which includes the use of foundational (or sometimes called “upper”), midlevel, utility, domain, and subdomain ontologies. In this section, we discuss the level of expressiveness of the ontologies needed for complex applications and the nature of the fully founded ontological architecture needed.

An Expressive Ontology

An adequately expressive ontology enables information to be exchanged in a form that contains the elements needed (the classes, relations, functions, instances, and potentially rules) for a computer system to properly interpret and automatically use the information without prior knowledge of, or interaction with, the system that created the information. In Figure 1, the level of expressiveness (ability to describe the meaning of a concept) required is that of a logical theory represented minimally in OWL-DL, but preferably in OWL-Full, with, in addition, rules expressed in a rule language such as the Semantic Web Rule Language (SWRL), the Rule Interchange Format (RIF), or in logic programming, or a hybrid representation (Samuel et al., 2008).

In Figure 1, “terms” are natural language words or phrases that act as indices into the underlying meaning, i.e., the concept (or composition of concepts) and the syntax (e.g., string) that stands in for or is used to indicate the semantics (meaning). “Concepts” (referents, categories) are units of semantics, i.e., the node (entity) or link (relation) in the mental or knowledge representation model and a placeholder for the real-world referent. Both weak and strong ontologies use the subclass_of relation between a child concept and its parent. The primary distinction between a weak and a strong ontology is that a “weak” ontology is expressed in a knowledge representation language that is not based on a formal logic. Why is this important? It means that a machine can only read and process a weak ontology (e.g., currently, that means models in an entity-relation language or the Unified Modeling Language [UML]). It cannot semantically interpret the ontology, i.e., ingest the ontology and perform automated reasoning on it. So, a weak ontology is not semantically interpretable by machine; a “strong” ontology is. The figure depicts the correlation between complexity of the semantic model—ranging from taxonomy, thesaurus, and conceptual model to logical theory—and complexity of the potential application. As the expressiveness of the
semantic model increases, so does the possibility of solving more complex problems. For applications that require great semantic precision, i.e., where approximate or loose characterizations of the semantics simply will not accomplish what is needed, more expressive models (ontologies) are required, and only logical theories are expressive enough to enable complex applications such as semantic interoperability (Obrst, 2003; Daconta et al., 2003; Obrst et al., 2007, 2010a, 2010b; Gruninger et al., 2008) and complex decision support for rich situational awareness and course-of-action (COA) analysis (Stoutenburg et al., 2007a, 2007b; Obrst, 2010).

This ability to automatically and correctly interpret data without contact with the creators of the data is critical because massive amounts of data are generated and accessible, making it a practical impossibility for data consumers to consult with more than a few data source managers to resolve the ambiguities that are present in information when it is not represented in a form as expressive and unambiguous as an ontology. This is especially important when time-critical information must be interpreted and forwarded to the person or system that can take the appropriate action. For 20 yr, ontology technology has been developing rapidly; it is now being deployed to solve practical problems. For example, the Cyc ontology has been used to integrate clinical data at a major medical facility (Lenat et al., 2010). A separate system “SIRI” (http://siri.com) is a “virtual personal assistant” that uses an ontology for information management, and it is now included as an app for the iPhone. As will be described later herein, an ontology, together with an interface to local data, can serve as a “translation utility” that allows local groups to develop and pursue their tasks independently while still being able to accurately share information with other independent groups.

The use of a common foundation ontology to enable sharing of information has been suggested previously. As early as 1998, Guarino (1998) provided theoretical arguments for the use of a common foundation (“top-level”) ontology in information sharing, and also explained why a post hoc attempt to map domain ontologies that were developed independently can provide a misleading apparent overlap, because domain ontologies are often insufficiently detailed to rule out many unintended models, some of which may appear identical between domains.

Need for a Foundational Ontology

The COSMO (Cassidy, 2008) ontology is only one of several existing foundational ontologies that might serve as the starting point for developing a common foundation ontology for one or more communities. Others are OpenCyc (2011), Suggested Upper Merged Ontology (SUMO, 2001) (Niles and Pease, 2001), Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) (Masolo et al., 2003), Basic Formal Ontology (BFO) (Grenon, 2003a, 2003b; Grenon and Smith, 2004), and OCHRE (Schneider, 2003). An overview of other foundational ontologies is presented in chapter 6 of the book by Oberle (2006). Another discussion of the ways in which foundation ontologies can be used for interoperability can be found in Borgo and Lesmo (2008). Comparisons of foundational ontologies can be found in Semy et al. (2005), Mascarci et al. (2006), Grenon (2003a), etc. Finally, there is the Upper Ontology Summit, which was an effort to correlate existing upper ontologies (Obrst et al., 2006).

In Figure 2, the layers represent the upper, midlevel, and domain (or lower) ontology layers. Sometimes the upper and midlevel ontologies are called foundational ontologies, and there can be multiple such in each layer. In our view, “upper” or “foundational” ontologies are typically abstract and make assertions about constructs such as identity criteria, parts/wholes, substance and constitution, space and time (and endurants and perdurants), necessary properties, dynamic properties, attributes spaces, etc., that apply to all lower levels; hence, they span all midlevel and domain ontologies (Obrst, 2010). These foundational ontologies may consist of levels, as discussed as “levels of reality” in a number of papers (Poli, 2003, 2010; Poli and Obrst, 2010). “Midlevel” ontologies are more specific, making assertions that span multiple domain ontologies. “Domain” or “subdomain” ontologies address specialized portions of the real world, i.e., about specific aspects of electronic commerce, military situational awareness, intelligence analysis, geographical, geophysical, and geopolitical portions of the world, which can get very fine grained. “Utility” ontologies are specialized ontologies that address very common domains shared by many domains and can be represented at different levels in the figure: e.g., units of measure, time, biology experimental construct ontologies, etc. “Super-domain” ontologies are those ontologies that act as upper or foundational ontologies for specific complex domains, such as biology, medicine, physics, services, etc. Sometimes, it can be hard to substantively differentiate midlevel, utility, and superdomain ontologies, since many ontologies can be characterized at each of those levels.

Figure 2. Ontological architecture.
The use of ontologies for sharing of geospatial information has been previously discussed by Schuurman and Leszczynski (2006). However, as they point out (p. 711), the formalization of the basic GIS data that must precede mapping to a common ontology has been only slowly adopted. Whether the GIS data are currently structured as a relational database, or as an ontology, or in some other form, we wish to convey that the potential for accurate interoperability provides a reason to undertake the additional effort needed for information modelers to describe their data using the logical form provided by a common foundation ontology.

Some vendors claim to provide systems that use an ontology to integrate relational databases. For database integration, commercial vendors such as Cycorp (http://www.cyc.com) and HighFleet (formerly Ontology Works, http://www.highfleet.com) offer systems that have a built-in foundation ontology and utilities to link that ontology to multiple databases. Availability of an existing foundation ontology does not eliminate the need to create specific domain ontologies and ontology extensions that can map the local data stores to the foundation ontology and through the foundation ontology to other data stores. Several foundation ontologies are available for free public use, and some commercial vendors provide abilities beyond their ontologies, such as utilities for the process of linking the foundation ontology to a database. If, in the database integration process for the federal government, open-source programs are desired, the utilities for using the foundation ontology must be developed based on a foundation ontology selected or developed for that purpose. However, this linkage utility will only be developed once, and will not greatly increase the cost of integration beyond the effort required for developing the mappings of local data stores.

Whether using an existing vendor’s integration utility or developing an open system to minimize the long-term costs and maximize the open-source community’s participation, both involve mapping each local data store to a common representation of meaning (requiring some expressive ontology), and, when required, converting information in one terminology and format into information with the same meaning in another terminology and format, using the foundation ontology as a common standard of meaning.

Using a common ontology is a practical method to avoid many-to-many mappings of multiple databases. As depicted in Figure 3, $n$ databases can be mapped to each other by a common ontology near linearly with only $2n - 1$ mappings to the common ontology; while without a common reference interlingua, $n^2 - n$ individual mappings would be required among the databases. As a result of this well-known difference, almost every project that attempts to integrate multiple knowledge organization systems uses a common core to which mappings are made and through which terminology conversions can be accomplished. Most of these projects address only local integration concerns, so the resulting mappings of one project are not generalizable beyond that project.

Compounding the lack of compatibility of local integration projects is the fact that few have used a common standard of meaning with the expressiveness of an ontology. As a result, it is inevitable that one group’s common meanings are unsatisfactory to others. To avoid the problem of limited local orientation or limited expressiveness, a translation utility is needed, which is based on a common foundation ontology with an inventory of basic concepts that is large enough to allow definitions of most specialized concepts found in databases. The task is to develop a translation utility and apply it to integrate multiple databases. The task includes the development of domain ontologies that extend the basic concepts represented in the foundation ontology, and thereby accurately represent the meanings of the data in the integrated databases. Such an ontology-based system will perform

**Figure 3. Complexity of semantic integration with and without ontologies:** An ontology allows for near-linear semantic integration (actually $2n - 1$) rather than near $n^2$ (actually $n^2 - n$) integration. Each application/database maps to the “lingua franca” of the ontology, rather than to each other.
conversions rapidly, accurately, and automatically. This will enable a true federal-wide federated search for information using local terminology mapped to the common ontology. For situations where a database has been developed without reference to a common foundation ontology, the conceptual schema used to develop the database can serve as a record of the conceptual elements of importance for the application, accelerating the process of post hoc mapping. Fonseca et al. (2003) discussed issues in relating conceptual schemas to their corresponding ontological representation.

Some scenarios showing how an ontology can query across multiple databases are given in Appendix B. The access to a particular store of information will depend upon the permission level a querying community has to access other data stores, and the development of mappings to those data stores.

In this paper, we illustrate the principles by describing the way in which a common foundation ontology can support accurate federated search among diverse databases. However, the same principle can be used to allow data from one source to be entered into multiple databases, by translating the assertions in the data from one data source into the terminology and format of all of the local databases that may wish to use and store that data. This principle also supports accurate communication among multiple intelligent agents or among ontology-based applications generally, and a query of multiple databases is merely one procedure that may be used by a large number of different applications for data sharing. The principle of a common foundation ontology allows great flexibility in the programs and communications methods used locally or within small communities; it is only necessary, when communication to a broad audience is desired, for one of the local agents to translate the information into the logical vocabulary of the common foundation ontology, and then from there to any other system that wishes to access the information. The communications methods among local groups of databases can therefore be specialized and optimized for local efficiency, and translation to the common logical foundation is required only for information to be communicated from or to systems outside the local community.

For brevity, we consider the case where the data reside only in the local databases, and queries to the remote databases are created at query time. This can be done by translating a query posed to one of the federated databases, transmitting the translations to the other databases that are receiving the replies, and translating them back to the terminology of the original database.

Alternative architectures have been suggested. In one alternative (Critchlow et al., 1998), the ontology is used to translate local databases into a common warehouse database that represents all the desired data. Data are entered locally, but the federated search is accomplished by query of the warehouse database. This alternative architecture can reduce potential inefficiency of automatically generated queries by creating a single warehouse database for which optimum forms for queries to the warehouse can be identified and used. Another alternative architecture uses a Data Access Mediation Service that accepts queries and returns answers. It also provides effective translations between local terminologies by links to two other services—a semantic map and an association—and the local data stores. In that architecture, the single ontology mapping is replaced by two types of translation service—the semantic map, which specifies the data source and elements in the source that satisfy the query, and the cached associations, which translate the source data back into the terminology of the requester.

Either of these alternative architectures may be more efficient than the simple query translation enabled by a common foundation ontology. However, the simple translation method should minimize demands on the local data managers and create the ability to federate queries across databases quickly. This architecture will be an intermediate phase while a more efficient method is developed to query across multiple diverse data stores; it will ease the transition from the existing structures of legacy databases to the time when the schemas of databases are generated directly from an ontology.

In all these architectures, a common foundation ontology (or its procedural equivalent) is essential to serve as an unambiguous standard for the meanings of the local data and enable accurate query translation and data retrieval. The common foundation ontology will provide the standard for meanings of information extracted from free text and the bridge between the structured data in databases and the unstructured data in text. A common foundation ontology will allow the creation of new databases by developing an ontology as the conceptual data model and automatically generating the database schema from the ontology. This procedure will be no more costly than traditional database development, as discussed later herein. If each local ontology is developed using the basic concepts in the common foundation ontology, the databases generated will automatically map to each other. Automatic generation of database schemas permits local terminology to be used in the local ontology and database. The local terminology is mapped to the common foundation ontology by linkage of the local ontology to the common foundation ontology. This is all invisible to the user. The method of generating a database directly from an ontology had already been implemented commercially in the HighFleet system. Use of a translation service to federate existing databases is a tactic that can deal with legacy databases while allowing database developers to become familiar with the use of ontologies as they work with existing data stores.

For those only beginning to formalize their data in an ontology format, Appendix A describes considerations and principles that should be kept in mind. There are now many books, articles, and online tutorials that can assist beginners who are learning the basic techniques and tools; new resources appear rapidly. There is also a large body of literature and annual conferences addressing the problem of relating separately developed ontologies to each other. Variously, this technical thread goes by ontology “matching,” “alignment,” and “mapping” (Sowa, 2010; Kalfoglou and Schorlemmer, 2003, 2004; Ehrig, 2005; Euzenat and Shvaiko, 2007; Stoutenburg, 2009; Obrst, 2010), and, of course, a potential
ontologies of ontologies (Blair et al., 1992; Guha, 1991; McCarthy, 1990, 1993; McCarthy and Buvac, 1997; Gabbay, 1996; Akman and Surav, 1996, 1997; Giunchiglia and Bouquet, 1997, 1998; Giunchiglia and Ghidini, 1998; Obrst et al., 1999; Lenat, 1998; Meseguer, 1998; Menzel, 1999; Menzel, 1999; Basin et al., 2000; Blackburn, 2000; Bouquet et al., 2003; Obrst, 2010; Obrst and Nichols, 2005; Haase et al., 2006, and the papers therein). These approaches do not use the tactic of focusing on semantic primitives, as does the COSMO ontology used for this project.

The task of converting existing relational databases into ontology format has also been investigated. There are simple systems such as D2RQ described by Bizer (2009), which converts an RDB to an RDF graph, as well as RDB2Onto described by Laclavik (2006), and there are also more complicated systems, such as R2O (Barrassa et al., 2004), in which a separate query system is developed to deal with potential mismatches among different databases.

Cost Considerations

The cost of this method of achieving interoperability for newly developed databases is no greater than the cost of traditional database development because development of the domain ontology for a new database replaces developing the conceptual and logical data models. Because the number of databases can be expected to increase at least at the rate of economic expansion, -3%–6% per year, within 10 to 20 yr, the number of new databases being created will exceed legacy databases. Therefore, it is significant that practical semantic interoperability for the majority of databases can be achieved with no additional development cost, beyond the minimal cost of achieving agreement on the common foundation ontology.

For legacy databases, retroactive mapping will be practical, but the cost is likely to be a significant fraction of the cost of developing a new database, because each element (table, field, or restriction) must be mapped to the foundation ontology. A “triage” approach can be taken, where critical databases are mapped immediately, less critical ones are mapped when the databases are refactored or are merged into a data warehouse, and unimportant databases (those not important for federated search purposes) are left unmapped. With the anticipated savings in interoperability costs, this triage approach could quickly pay for the development cost for a translation utility and mapping of legacy databases.

There is also an emerging field within ontological engineering that attempts to strictly characterize the costs associated with ontology development. ONTOCOM is one such attempt (Simperl et al., 2006, 2009; Bürger, 2008; Bürger and Simperl, 2008; Intitiaz et al., 2009).

### ONTOLOGY VERSUS TRADITIONAL DATA MODEL

This brief summary provides a general overview of the differences between an ontology and traditional data model. For a detailed discussion, refer to Appendix A.

Traditional data models are syntactic and structural without any explicit semantics for the tables and fields. Data are represented as strings, numbers, and simple enumerations, and are often shown as a choice among a limited number of string values for an attribute. By contrast, in a proper ontology, few things, other than entity names, would be represented as simple strings. The elements (including attributes) would be part of a class structure; they would be first-class entities in the sense of being types (categories or classes), relations, or instances of types. Each first-class entity can have many relations to other entities in the ontology, whereas strings in an enumeration do not have semantically meaningful internal structure.

For example, in a traditional data model, a list of religions as attributes for a person in one data model might include the strings “Islam,” “Sunni Islam,” “Christianity,” “Church of England,” etc. Those strings may not be related to anything else in the data model, whereas in an ontology, the meaning of these terms and rules is explicitly recorded. This definition applies to any detail required for any of the applications using any of the data stores. At a minimum, each religion would be a first-class element (which might be a subtype of “Religion” or an instance of that type); the conceptual structure containing information about the religion might be potentially useful in some application. In the traditional data model, the same string might be used differently in other tables, depending on the procedural code that interprets the data in the tables. The relations and differences may be of no significance in a given application, even though the implicit concepts are related (e.g., Sunni Islam would be a specialization of Islam). On the other hand, when that information is reused in a different application, the fact that the strings refer to first-class entities and have a direct relation to each other may be significant to the second application.

Next, in Table 1 we present a segment from one database that provides a small sampling of cases where interpretations can depend on specific applications and require careful mapping to avoid misinterpretation. This example illustrates issues that can arise from representation of the common concept of a “person,” but equally troublesome issues may be found in most databases when the data need to be shared for use beyond the application for which the original database was created. Note that MIDs to refers to the “Modernized Integrated Database” used by the U.S. Department of Defense (see ref. MIDB, 2011) (Table 1).

### Entity_Guid

A GUID (“Globally Unique Identifier”) can be uniform throughout a database, but it may not be interpretable between databases. To recognize the identity of individual entities, some method will be required to “de-duplicate” individuals that are
returned from a search. If a common unique identifier is not used in different databases, other attributes and a complex algorithm may be used to arrive at a probability that two individuals are identical. The translation utility can include such algorithms, unless simple presentation of results without concern for duplication is acceptable.

Name_First, Name_Last

These are usually unambiguous for Western European names, but they may be ambiguous for names in other cultures (e.g., Asian names, where the family name is actually the first name). The local databases must be consistent in assigning family name to “NAME_LAST.” In databases where other elements of a name are maintained (patronymic, mother’s maiden name), it will be necessary to ensure that the “last” name is consistent with the “last” name of typical American names when mapping. For single-name individuals, consistent use of the “first name” is expected.

Alias

This field may record names other than the one given at birth. In this database, there is no way to distinguish the type of alias; the bit merely indicates that it is an alias. In another database, the type of alias (married name, assumed name, nickname, pen name, nom de guerre, etc.) may be indicated by a code. The codes need to be unambiguously interpretable and stored in the ontology mappings. The type of “alias” can be important. The significance of a maiden name may be useful in a social networking application but not interpreted in other applications. In an ontology, “maiden name” will be a first-class entity with its own significance regarding relationships. The significance of an alias such as “Abu Abbas” will also be indicated as a property of a name in an ontology (again, useful for social networking), but it will be lost in most databases that simply record a name as a string.

Birth_Place

In this database, birthplace is a complex string and not encoded in any standardized way. To align with other databases that record birthplace, a parse of that string will be required to extract country, town, and other significant location information that can help identify an individual. A “birthplace” of “Jersey City” must also be recognized as being consistent with a birthplace of “New Jersey” and a birthplace of “USA.” This transitivity is needed to allow such inferences and recognize proper relations.

Birth_Date

Birth date is generally used consistently, though format conversion may be required. The ontology mappings will automatically convert to a consistent internal format. Where the birth date is approximate or not known, different databases may have different methods of coping, and knowledge of these different methods will be needed for proper interpretation and translation.

Ethnicity

Different databases may have different codes for ethnicity classifications, requiring a simple translation to a common representation, but it is possible for different ethnicity classifications to be used, in which case, the ontology would need to represent both classifications with relations between them to properly interpret and translate queries.
Individual_Religion

In this database, the religion attribute is a code. Such codes will vary among databases and need to be referenced to a common classification. As previously mentioned, religions can have subdivisions. The significance of a religion and its subdivisions in a database is likely to be buried in the procedural code. Over time, actual usage can become unknown to those who inherit the database from its original creators and application developers. For example, a database’s application may implement a procedural rule limiting any attempt to use Shiite workers in a Sunni neighborhood, reflecting the fact that Sunni Muslims and Shiite Muslims in a particular time and place were sufficiently antagonistic that they refused to associate, or would attempt to even kill each other. In an ontology, a religion is a first-class entity from which many such facts may be asserted. Some facts may be used implicitly with no trace in the database by one application.

Allegiance, Citizenship1

Both attributes refer to a country. If “allegiance” is not identical to “primary citizenship,” its significance will be interpreted within the procedural code of the local application; yet, it may be significant to other applications, if it can be interpreted properly. A difference or identity in these two attributes may be significant, but to be used properly, the definition of “allegiance” must be explicitly related in the ontology to an individual’s citizenship and behavioral tendencies.

In all cases, a common ontology and mappings from that ontology to the databases will allow the data fields to be unambiguously represented, so that fields of identical meaning will be mapped to the same element in the ontology. Fields of related meanings will also be mapped with the relations represented in a form that permits automatic retrieval of potentially significant data, even when data in the remote databases are not identical to originating community’s database.

Role of the Translation Utility

The translation utility will interpret data in one database and convert it to its equivalent representation in another database. The utility will use logical inferencing and procedural code; these mechanisms can become too detailed to present in a short paper. For example, a “LengthInMeters” column in one database might be used with a data type of decimal, whereas a different database might represent “Length” as two columns—a decimal number and a unit of length measure. The one column in the first database and the two columns in the second database would both be mapped to an instance of “LengthMeasure” in the ontology, with unit conversions performed when required. Other types of data will not have equivalents that can be easily converted. In any given database, such conversions could be programmed as integrity constraints on input or stored procedures. However, such procedures are unique to specific databases and cannot be shared because the interpretation of the results depends on the ways in which the data are used in the application. A more robust solution is to include logical rules in a domain ontology, which are the equivalent of computer program procedural functions that encode “business rules.” One such logical rule might notify the office when a certain situation is recognized, even though the person holding that office may differ for each of the different communities sharing their data. Where logical rules alone are inadequate, an ontology can call on external procedural functions, as when performing complex mathematical calculations required for translation or inference. Finally, the translation utility itself may use stored procedures for translation.

In addition to recognizing related but not identical data elements in different data stores, an ontology will permit the generation of answers that require some form of logical inference (this could be performed automatically once the local data are mapped to an ontology). For example, a query may ask for a person’s relatives, even though the local data store may have only parent-child relations, while a remote data store may have information on some of the same people, but contain only sibling relations. The ontology, using information from both stores, will not only recognize both kinds of kinship, but using domain rules, will be able to generate a list of increasingly remote relations, including uncles, aunts, and cousins. One such rule might be:

(if ?Person1 is the father of ?Person2, and ?Person3 is a brother of ?Person1, then ?Person3 is an uncle of ?Person2)

An important capability enabled by an ontological representation is the ability to create such rules that can be added to the domain ontology without recompilation of the program. This provides a simple mechanism for creating “new information” from existing information.

Such rules can be immediately translated into the ontology format and added to the domain ontology, in effect permitting changes to the translation program in a simple manner. By understanding these rules, domain experts could update the query translation utility. Use of declarative formats to simplify data structures is common to database and ontology design. However, the ability of ontologies to rapidly ingest and use inference rules provides them with the ability to adapt quickly and at less cost to additional domain knowledge from the domain experts.

In summary, by mapping local databases to a common ontology, the definition of the data can be interpreted by other applications that have data stores linked to the ontology. By linking simple concepts and strings in one data model to more complex concepts in the ontology, the utility of the data in the first database can be expanded beyond its original purpose and in ways that a different application can use to infer more information.

ONTOLOGY USE FOR DATA INTEROPERABILITY

A foundational ontology should be used to achieve data interoperability among multiple data stores. The foundation
ontology provides a set of concepts that can be combined to create a domain ontology, which represents meanings in each local data store. In this way, when one community accesses data in nonlocal data stores, a query interpretable in the community’s local data store (e.g., Structured Query Language [SQL] created by a local user interface) is automatically converted to a federated query in conceptual form using the mappings to the foundation ontology. This federated conceptual query can then be automatically converted into local queries to all accessible data stores that may have relevant information. The answers follow a reverse pathway and are converted into the terminology of the originator’s local data store and presented to the user. A sketch of one possible query method (called the Query Translation Method) is presented in Figure 4 (note: data paths are numbered in sequential order; each numbered path moves over the network from a source to a destination).

The explanations of the numbered data paths in the figure are:

1. A query entered in the standard form for some local data store (e.g., database 1 [DB1]) is forwarded to the local copy of the query interface and sent over the network to the query translation service.

2. (a1) The query translation service interprets the query in terms of the common data model, based on the ontology extensions and mappings that relate each local data store to the foundation ontology and its domain extension. The query is translated into the terminology of each local data store using the mappings, forwarded to the local data stores over the network, and received by each local data store in its own terminology. (a2) An example of the way in which a database query in one database could be translated into a comparable query in multiple other databases is given in Appendix C. This example assumes that the local data stores do not need to provide any services other than simple access to their databases. If the local data stores provide an access service other than SQL query, a different translation mechanism might be needed. (b) The translated queries from the translation service are received by each local data store via a Web service.

3. Information retrieved from each local data store is returned to the translation service in the terminology of each local data store.

4. The translation service converts the retrieved data into the terminology of the querying user (e.g., DB1 terminology) and returns the answers in the user’s terminology. The local copy of the query interface presents data to the user and, in the case of semantic mismatches (related, but not identical, data meanings), adds qualifying comments to alert the user of the meanings of the retrieved data that may not conform to the format of the querying user’s local database or local terminology.

In the query translation method, all data reside in the local data stores. The local data stores are accessed in a read-only manner from the translation service; updates to the data are performed locally. There is no replication of local data. Outside of the local data stores, key components of this translation process are:

**Federated Query Using Foundation Ontology**

![Diagram of the query translation method](image)
1. Extensions of the foundation ontology express concepts in each of the local data stores, not in the foundation ontology. These extensions together constitute a domain ontology that is a model of the domain represented by the full set of connected databases. This domain ontology is defined in terms of the foundation ontology. When it must perform complex inferences to answer queries, the linkage to the foundation ontology may be used.

2. Elements in the local data store are mapped to concepts in the domain extension ontology.

3. A local data access service is connected locally to the local query interface, which accepts the query from the local user, forwards it to the translation service, receives the answers from the translation service in the terminology of the local data store, and presents the retrieved information to the user. The data access service is responsible for presenting the retrieved data in a way that notifies the user when information is similar, but not identical, to the queried data.

4. Query services (Q-service) at each local data store accept queries from the network in a format agreed to by the local data stores and returns answers to the translation service. The query format may be standard SQL, but the query service must provide a wrapper that can accept the query over the network.

Creating and Using Mappings from Databases to the Ontology

Although the mappings to the common foundation ontology must be created separately for each local data store, the translation service utility that uses the mappings to interpret local data in terms of the foundation ontology and convert it to terms conforming to a different local data store should only be developed once by a team of ontologists and programmers. Adequate foundation ontologies are available to support the translation service and do not need to be developed. The effort of an ontologist will be required only to develop the specialized domain ontologies and mappings for each set of databases to be integrated. The mappings themselves will all be extensions of the foundation ontology, and the mappings will use relations that can be interpreted by the conversion utility, based on the foundation ontology. Periodic improvements and supplements to the foundation ontology may be required as additional specialized data stores are integrated, but local data store developers need not learn the additional internal translation mechanisms or assist in their development.

The conversion to and from the conceptual form supported by the foundation ontology and its extension can be accomplished dynamically at query time. By using separate mappings from each local data store to the foundation ontology, a translation utility can perform the conversion from local form to common conceptual form and the reverse.

The mappings are created for each local data store by a team that includes an ontologist familiar with the foundation ontology and a domain expert who understands the database terms to be mapped. The mappings take the form of an extension to the foundation ontology specifying the particular elements in the foundation ontology (or its domain extension) that correspond to tables or columns in each local database. Mapping updates will be required if the local data model changes; this should be easier than development of the initial mapping. Only a single mapping of each local data store to the foundation ontology is required in order to create mappings between all of the local data stores. This takes a problem that is potentially \( n^2 \)-squared in complexity \( (n \) being the number of data stores to integrate) and reduces it to “order of \( n \)” When sophisticated interfaces can be developed to permit domain experts to find the ontology elements that correspond to the data elements they use locally, then assistance from an ontologist may no longer be needed.

There are different options for a system architecture that will implement this mechanism for data integration. These include having the translation from and to the local terminologies done centrally and the federated queries sent to each local store using the local terminology (refer to Fig. 4), or having the queries transmitted in a common terminology and translated to the local terminology at the local data store site by the local query service. In either case, Web services must be implemented in each local data node to accept queries in the required format. Alternative architectures were discussed in the section “The Ontology Solution.”

Prototyping and Implementing Ontology-Based Integration

Development of a working prototype using the query translation method will require an extension to this study. This phase of the project has developed, in addition to a preliminary version of the Sociocultural Ontology, only a simple Java program to illustrate one type of query translation. The cost of adaptation of commercial methods for data integration via ontologies needs to be weighed against the cost of developing an open-source method suitable for widespread government and commercial use.

For DoD-wide or federal-wide data integration, the cost of developing an open-source translation utility will be a small fraction of the total cost for developing new databases or mappings to legacy databases. An open-source utility addressing the issue of federated query via a common foundation ontology will permit refinements from a large community of interested DoD and non-DoD developers. For now, it is anticipated that any query mechanism currently used by each local data store can be intercepted, translated, and sent to other accessible data stores with relevant information. In addition to assisting in the development of the mappings by providing clear descriptions of their data elements, the local data stores will need to provide access by implementing a service that can accept database queries in SQL or another agreed common format. That service can be developed as part of the project developing the translation utility. It will be the same for all local data stores, unless some local system requires modification of the standard service. Thus, we do not anticipate the need to develop new user interfaces but rely on intercepting a
query at a local data store in a format such as SQL, sending it over the networks connecting the local data stores, and returning answers from remote data stores in the same format. In the long-term, development of a natural language interface for federated query will make wide access to data easier.

Development of the translation utility should be a project within the federal government, and it should be integrated with support for a single foundation ontology. The foundation ontology should be chosen from among existing ontologies, and supplemented when required. Because the translation utility is so dependent on the format and basic structure of the foundation ontology, it is not practical for an open-source generic utility to be developed by an academic or industrial group.

Integrating multiple data stores will require each local team involved in data management to clearly define each of the data elements (tables, columns, restrictions) in their system; read the documentation provided by the ontology team describing the meanings of the logical representations they create; and confirm or correct the interpretations of the ontology team.

With the mappings in hand, the ontology team and developers of the translation interface can develop the programs that will provide the desired translation capability.

Further validation testing will come when data transfers are executed in a realistic user setting. If a local data store does not have an implemented service allowing access to the data store via the network, one must be added to the local system. It may be necessary to develop local access programs tuned to specific data stores if they do not permit access via the same method (e.g., passing an SQL query over the network).

It also may be desirable to implement new display methods to return the query results from remote data stores that indicate the source of the answers. For example, when remote information may be close, but not identical, in meaning to the requested information, it may be signaled by a standardized prefix in the answers (e.g., an "[S]" to indicate that the remote store only contains supertypes of the type requested, and the answer following the [S] may or may not relate to the more specific type requested).

The Problem of De-Duplication

Specific individuals or objects in different databases may not be easily recognized as identical. A unique identifier in one database may be unrelated to a unique identifier used in another database. The methods for determining that two individuals or objects in different databases are the same (“de-duplication”) can be complicated. In favorable cases, there may be unique identifiers that are common (e.g., Vehicle Identifier Number, Social Security Number [SSN]). In general, methods may be included in the translation service to determine if two individuals returned as the answer to the same query on different databases are identical. Such methods may use multiple attributes (for people: name, address, date of birth, place of birth, SSN, passport number, telephone number, e-mail address, known relatives) to arrive at a probabilistic estimate of common identity. For an initial implement-ment of query translation, such de-duplication methods may be omitted, but the system will be most useful if information that can indicate identity is used and reported to the user.

Data Conversion Effort for Sociocultural Data

A prototype was begun, based upon the process described herein, as part of an effort to integrate data stores containing sociocultural information. Although this example derived from a project involving sociocultural information, the principle of a common foundation ontology applies to integration of information among any community that wishes to share their data and have them accurately interpreted and properly used by other computer systems. In our effort, we used the COSMO ontology (Cassidy, 2008). As mentioned earlier, other foundational ontologies could have been chosen, and in past applications, we have indeed chosen or recommended others (Semy et al., 2005). COSMO was used in this project because COSMO was initiated and has continued to be developed with the purpose of including representations of any of the basic (primitive) concepts that exist in any other foundation ontology; thus COSMO, when fully developed, will have all of the primitive ontology elements needed to build representations of more complex ontology elements in any linked ontology, regardless of the domain of interest.

The sociocultural data integration prototype included the following features:

1. The foundation ontology providing the fundamental concept representations that will be used to define domain-specific concepts was the “Common Semantic Model” (COSMO), an ontology developed as a merger of several publicly available foundation ontologies and extended to include required basic concepts not already in those ontologies.

2. Development of the Sociocultural Ontology (SCO), an extension of the COSMO, was begun to represent the specialized concepts found in data stores describing sociocultural information. The priorities for adding new concepts to the SCO were derived from lists of important sociocultural terms supplied by domain experts, who provided cases indicating the kind of sociocultural information needed for actions undertaken by U.S. military forces.

3. A mapping from the SCO to the MAP-HT (Mapping the Human Terrain, 1997) data model can be created when a required minimum number of sociocultural concepts have been added to the SCO.

4. A translation utility can be developed to convert information from the format and terminology of one mapped database into the format and terminology of another. The structure and language of this utility are not yet determined. Some lessons from the early phases of this project are discussed next.

The sociocultural information integration effort is still at an early stage and can easily adapt to the preferences or requirements of other data integration projects. Projects that are already
mature in their data models, such as the NIEM, can be treated as another form of data store to which mappings can be developed to the foundation ontology. Projects that are still in the formative stages could be fully integrated by proactive coordination with this sociocultural data integration project. Such coordination of data model development with an ontology-based SCO effort will allow all parties to find common representations. Coordination at the data model development stage will simplify the mapping effort to the foundation ontology, while still allowing local data store developers to choose any terminology or structure that they find most convenient. Residual local structures still differing from the foundation or other local representations can be mapped by the process discussed here, but the mapping effort will be reduced if coordination at the formative stages is feasible. Conversely, without such coordination, local choices for data representations might differ merely due to the absence of an accepted standard, thereby increasing the mapping effort.

**Issues in Creating Mappings of Databases to Ontologies**

This section discusses some issues in creating mappings from databases to ontologies.

**Collaborate with the Database Developers**

The first issue in creating mappings is finding a way to collaborate with the database developers to determine the meanings of the database elements to be mapped. This requires a good data dictionary explaining the meanings of the database elements. For some databases, a data dictionary is critical because the names of the data elements are not self-explanatory. For example, no data dictionary is available for one database and elements look like the one below:

In table IND (individual) attributes (columns) exist:

- CIH_Charactertraits
- CIH_Traits
- CIH_Characteristics

... and in table IND_SOURCE_INFO

- CIH_CharTraits.

Any of these elements might have a meaning identical to or related to the other, but without detailed explanations, mapping cannot be created to any other database, even if the names of the columns are identical. We cannot be certain of the intended interpretation or usage of these elements without consulting the developers or users of that database.

Other examples abound. In the same sample database, there is a table:

IND_SOURCE_INFO

with column: CIH_MembershipInOrgs

... and a table IND_SL

with column: CIH_Memberships

... and a table ORG (organization, apparently)

with columns: CIH_Membership
- CIH_Personnel.

If the CIH_Membership relation points to a list of members of an organization, then it could be related as the inverse of CIH_Memberships or CIH_MembershipInOrgs, but how do the latter two relations differ, if at all?

**Two Elements with Different Names May Have Identical Meanings**

This is the simplest mismatch to map. Consider three columns from two databases:

- CIH_dob
- BIRTH_DATE, and
- GMI_BIRTH_DATE.

All three columns appear to represent the data of birth for an individual, and the mapping appears obvious. However, there may be different formats for the date, so a translation of format may be required. If no format conversion is required, this translation could be done with a lookup table. If conversion is required, a simple data model with the translation provided by an XSLT (“Extensible Stylesheet Language Transformations”) format conversion method may suffice. However, other conversions will not be so simple, and adopting the more general solution of mapping to an ontology will serve the complicated as well as the simple cases.

Even for this simple case, an ontology can be helpful. In one database, there may be an integrity constraint that individuals cannot be paid for work performed before they were born, but across different databases, that kind of integrity constraint cannot be performed. When multiple databases are mapped to an ontology, the logical rules of the ontology could detect anomalies that might not be detectable in one database. In two different databases, a person with a certain SSN might have two different birth dates. Knowing that a person can have only one birth date allows the discrepancy to be detected by logical inference within the ontology. Such consistency checking can be performed automatically within an ontology when data conversion is being performed. Within a single database, it can only be performed at data-entry time.

**Data with Similar Meaning Can Need Conversion**

A case where conversion of data elements is required occurs when measurements are expressed in different units in different databases. An XSLT procedure could perform conversions, even if the mappings of databases are created among data models rather than ontologies.

**No Off-the-Shelf Ontology Extension for Specific Use**

Each local data store that is to be integrated into a coordinated query system will have many concepts, data structures,
and terms specific to that local system and cannot be expected in any preexisting ontology. In favorable cases, some similar knowledge may be represented from a previous data store, and only minor additions or changes to an existing ontology mapping may be required to represent similar data in a new agency or enterprise. That state of development of ontology mapping is unlikely to occur until years of widespread use of a common system have passed. For now, neither vendors nor in-house developers can develop a proper mapping of a local data store to a common foundation ontology without an effort specifically directed at representing the local data structures used in the enterprise. The size of the effort will depend on the size of the local data model (the number of distinct types of data elements). Input from the developers and managers of the local data store will be essential in creating an accurate mapping to the common foundation ontology by clarifying ambiguities in the local data model.

How the Foundation Ontology Bridges Multiple Domains

A foundation ontology serves as the common conceptual and logical language through which different human languages or technical terminologies can be translated into each other. The present study was focused on converting terms used in different databases, but the same principle applies to information in other forms, such as in free text. Widely varying use of terms by different communities and in different contexts makes ordinary human languages unsuitable for representation in a form that computers can properly interpret. The foundation ontology overcomes such incompatibility by providing a relatively small common “defining vocabulary” of basic concepts that can be used in combinations to describe any of the more specialized concepts of interest to different communities. Use of the common foundation ontology allows precise specification of the similarities and differences in the use of terms, regardless of the context or community in which they are used, and thereby facilitates accurate exchange of information. By converting local knowledge in local terminology into its representation using the foundation ontology, translations into multiple other terminologies can be provided. Use of a common foundation ontology as the means of translation allows the local user control over community terminology without sacrificing the ability to accurately transfer information among different communities. The development of “common core” vocabularies that are not structured as ontologies suitable for automated reasoning does not provide the functionality required for this task.

The principle of using a relatively small controlled vocabulary to provide easily understood definitions of a much larger set of terms has been established practice in the dictionary publishing industry. The *Longman Dictionary of Contemporary English* (1987) uses a controlled vocabulary of 2100 words, with which it defines all of the 65,000 words in the dictionary. A study of the Longman vocabulary showed that the minimum required vocabulary is actually only 1200 words (Wilks et al., 1996; Guo, 1989). The notion of a basic set of primitive concepts as the basis for human language understanding has also gained experimental support from recent studies of brain activity using functional magnetic resonance imaging (fMRI) (Mitchell et al., 2008). The COSMO ontology, chosen as the foundation ontology for this study, is structured specifically to serve as a conceptual analogue of the linguistic defining vocabulary, and it is being developed to include concept representations corresponding to the full set of 2100 Longman defining vocabulary words. This ontology should allow creation of normalized precise logical specifications for any information in databases or other information sources. The benefit of using a relatively small set of ontology elements as the basis for defining terms or concepts from many fields is that the restricted size makes it easier for developers of interfaces to understand and use the common foundation ontology. The more widespread it is used, the more effective such a common standard of meaning will be.

ACKNOWLEDGMENTS

The views expressed in this paper are those of the authors alone and do not reflect the official policy or position of the MITRE Corporation or any other company or individual.

APPENDIX A. ONTOLOGIES AND DATABASES: SIMILARITIES AND DIFFERENCES

This appendix describes ontologies and databases and their similarities and differences by focusing on their respective goals and design processes.

Introduction

Ontologies are about vocabularies and their meanings with an explicit, expressive, and well-defined semantics, possibly machine-interpretable. Ontologies limit the possible formal models of interpretation (semantics) of those vocabularies to the set of meanings a modeler intends (i.e., close to the human conceptualization). None of the other “vocabularies,” such as database schemas or object models with less expressive semantics, does that.

The approaches with less-expressive semantics assume that humans will look at the “vocabularies” and supply the semantics via the human semantic interpreter (your mental model). Additionally, a developer will code programs to enforce the local semantics that the database/database management system (DBMS) cannot.

1. They may get it right.
2. Other people will have to read that code, interpret it, and see if it will do what it should do.
3. The higher you go in terms of data warehouses, marts, etc., the more semantic error creeps in.

Ontologies model generic, real-world concepts and their meanings, unlike either database schemas or object models, which are typically specific to a particular set of applications and represent limited semantics. A given ontology cannot model any given domain completely. However, in capturing real-world and imaginary (i.e., a theory of unicorns and other fantastic beasts) semantics, you are enabled to reuse, extend, refine, and generalize, etc., that semantic model.
It is suggested to reuse ontologies; database schemas cannot generally be reused. A database conceptual schema might be used as the basis of an ontology; however, that would be a leap from an entity-relation model to a conceptual model (i.e., a weak ontology) to a logical theory (strong ontology). Similarly, one could begin with a taxonomy or a thesaurus and migrate it to an ontology.

Logical and physical schemas are useless, since they incorporate non-real-world knowledge (and in non-machine-interpretable form). By the time the physical schema is achieved, there are only relationships and key information; the little semantics available was thrown away at the conceptual schema level.

The methodologies for ontologies and databases are similar. The database designer or knowledge/ontology engineer must consider an information space that captures certain kinds of knowledge. However, a database designer does not care about the real world, but about constructing a specific local container/structure of data that will hold the user’s data in an access-efficient way.

A good database designer works with users and generates use cases and scenarios based on the expected user interaction. Similarly, ontologists work with domain experts and/or subject matter experts (SMEs) and get a sense of the semantics that they require. A good ontologist will analyze the data available (bottom up) and analyze what the domain expert says (top down). In many cases (e.g., intelligence analysis), the ontologist will not only ask the SME about the kinds of questions that are asked for the tasks, but also the kinds of questions that should be asked (which are impossible to get answered currently by using mainstream database and system technology).

The Database Design Process: Three Stages

There are three stages to the database design process. They include:

1. In interaction with prospective users and stakeholders of the proposed database, the database designer will create a conceptual schema using a modeling language and tools based on entity-relation (ER) models, extended ER models, or object-oriented models using UML.
2. Once this conceptual schema is captured, the designer will refine it to become a logical schema, sometimes called a logical data model; it will still be in an ER language or UML. The logical schema typically results from refining the conceptual schema using normalization and other techniques to move closer to the physical model that will be implemented to create the actual database. This is done by normalizing the relations (and attributes, if the conceptual schema contains these) using the same ER and UML languages.
3. Finally, the design will refine the logical schema to become the physical schema, where the tables, columns, keys, etc., are defined, and the physical table is optimized in terms of index elements and sectors in the database to place the various data elements.

A data dictionary may be created for the database. This expresses the meaning of various database elements in natural language documentation. The data dictionary is only semantically interpretable by individuals, since it is written in natural language. The most expressive real-world semantics of the database creation process exist in the conceptual schema and data dictionary. The conceptual schema may part of the documentation for the process of developing the database (i.e., an artifact of that process). The data dictionary will be kept as part of the documentation for the process of developing the database conceptual schema and data dictionary. Unfortunately, the underlying physical database and its schema may change dramatically without the original conceptual schema and data dictionary being comparably changed. This is also the case with UML models used to create object-oriented systems and sometimes to define enterprise architectures.

Databases and Ontologies: Integrity Enforcement

Databases typically try to enforce three kinds of integrity.

1. Domain integrity (not the same notion of “domain” used in logic/ontologies): Domains are usually data-type domains (i.e., integers, strings, real numbers, or column-data domains).
2. Typically, symbolic objects are not entered into a database, just strings. So on data entry or update, some program (or the DBMS) will make sure that if a column is defined to contain only integer data, the user can only enter integer data.
3. Referential integrity: This refers to key relationships, primary and foreign. This kind of integrity is structural, making sure that if a key is updated, other keys dependent on it are updated appropriately: add, delete, update (usually considered an initial delete, followed by an add).
4. Semantic integrity: This represents real-world constraints, sometimes called “business rules,” that are held over the data. Databases and DBMSs cannot usually do this (even with active and passive triggers), so auxiliary programming code usually enforces this (e.g., “no other employee can make more than the CEO” or other cross-dependencies).

You cannot check database consistency like an ontology, using a logical knowledge representation language. For databases, you can only enforce the three types of integrity. For an ontology, you can check consistency in two ways:

1. syntactically (proof theory) and
2. semantically (model-theory).

This consistency checking can be done at two levels:

1. Prove that your KR language is sound and complete at the meta-level.
   - Sound (“\(\Phi \vdash A\)” implies “\(\Phi \models A\)”): The proof system will not prove anything that is not valid.
   - Complete (“\(\Phi \models A\)” implies “\(\Phi \vdash A\)”): The proof system is strong enough to prove everything that is valid.
2. Check the consistency of a theory (ontology) at the object level.
   - This is like negation consistency: There is no A such that both “\(\Phi \vdash A\)” and “\(\Phi \vdash \neg A\)” (i.e., a contradiction).

The Ontology Design Process

In creating common knowledge (as opposed to deep domain knowledge), intuition and understanding of the world can be used to develop the ontology. A good background in formal ontology or formal semantics helps, because then you have already learned:

1. a rigorous, systematic methodology;
2. formal machinery for expressing fine details of world semantics;
3. an appreciation of many alternative analyses, pitfalls, errors, etc.; and
4. complex knowledge about things in the world and insight into your pretheoretical knowledge.

In linguistics, it is said that although everyone knows how to use natural language like English, very few know how to characterize that knowledge or about prospective theories about that knowledge. Native
speakers do not have good subjective insight into how they do things; they just do them.

Ontology design and development uses typical software development lifecycle notions, but enriches them with the following processes:

1. Incremental, staged ontology development and deployment (typically breadth-first) with feedback from user/developer community at each stage, as time warrants.
   a. Define versioning, change management.
   b. General process: Ontology Lifecycle Management;
2. Form team (three roles: ontologists, domain expert liaison, domain expert/SME partners) and develop project plan;
3. Identify stakeholders (end users, developers);
4. Identify existing data stores and systems and analyze them;
5. Investigate existing model resources for potential harvesting;
6. Develop competency questions, use cases, scenarios, requirements; and
7. Develop architecture and select tools, design; and
8. Iterate: build, test and review (per stage), refine, deploy.

Initially, the ontology development process seeks requirements primarily by developing competency questions:

What questions (queries) do you want answered by the newly developed ontologies?

These questions will help drive out specific use cases. Those use cases will be refined to create specific scenarios and requirements. Competency questions will be used to determine if the ontology development effort is complete (i.e., if reasonable responses to the queries are given, with sufficient detail, as judged by knowledgeable domain experts, then you are done).

Concurrently, simultaneous bottom-up and top-down analysis is performed. Nearly every ontology development effort requires accessing and using data stored in databases or data obtained from existing software applications. Bottom-up analysis analyzes existing data stores and legacy systems that you want to cover to capture their semantics. This may require the creation of conceptual models (local ontologies), if those stores and systems do not have existing models or those models are insufficient. This process must address syntactic (relational vs. XML, other file or message syntax), structural (database [DB] schema, XML schema, message schema), and semantic heterogeneity and interoperability, including potentially different business rules, conversions, etc.

Top-down analysis includes consideration of queries that cannot be now asked, even using existing data sources and legacy systems. Examples include: link or social network analysis, temporal (time-based) queries or time series, part/whole decomposition, spatial-temporal flow (supply chain) of distributed material, equipment (biological, chemical, nuclear weapons), suppliers, etc.

An extended example may be a transportation ontology for a government agency concerned with logistics, which would include:

1. Determine coverage needed (competency questions, use cases, requirements) and project plan. The coverage may include transportation modes; vehicles, conveyors, and subclasses, operational characteristics; organizations and people; facilities and locations; cargo; services and related entities; travel and transportation routes; related and affecting notions; designations and identifiers; events associated with transportation and travel; others as deemed appropriate. The coverage will be decided by referring to the competency questions that have been developed, any use cases and scenarios used to obtain the competency questions, and any requirements distilled from all of these sources.
2. Determine existing models, resources for harvesting, and prospective integration with existing upper and middle ontologies.
3. Analyze resources and existing data stores with respect to coverage requirements.
4. Design incremental breadth-first stages:
   • Develop core subclass hierarchies with basic relations.
   • Expand the representation of facilities, routes, and cargo.
   • Address in-depth organizational, governance, and control concepts for transportation, including travel planning and itineraries, effects of regulatory policy on transportation activities, and some physical factors such as transportation impediments (weather and terrain).
5. Implement each stage with review and feedback, gauge competency question results. When the competency questions can be answered reasonably, with the appropriate correctness, detail, and level of granularity as adjudicated by domain experts who understand the data and applications, then the ontology development process is completed.

Ontologies vs. Databases

Often with nonontological approaches to capturing the semantics of data, systems, and services, the modeling process stops at a syntactic and structural model, and even throws the impoverished semantic model away to act as historical artifact. It is completely separated from the evolution of the live database, system, or service, and is still only semantically interpretable by an individual who can read the documents, interpret the graphics, supply the real-world knowledge of the domain, and who understands how the database, system, or service will be implemented and used. Ontologists want to shift some of that “semantic interpretative burden” to machines and have them mimic human semantics (i.e., understand what we mean). The result would bring the machine up to the human, not force the human to the machine level. By “machine semantic interpretation” we mean structuring and constraining in logical, axiomatic language the symbols humans supply; the machine will conclude, via an automated inference process, what an individual would in comparable circumstances.

The knowledge representation language that enables this automated inference must be a language that makes fine modeling distinctions and has formal or axiomatic semantics for those distinctions, so no direct human involvement will be necessary—the meaning of “automated inference.”

The primary purpose of the database is for storage and ease of access to data, not complex use. Software applications (with the data semantics embedded in non-reusable code via programmers) and individuals must focus on data use, manipulation, and transformation, all of which require a high degree of interpretation of the data. Extending the capabilities of a database often requires significant reprogramming and restructuring of the database schema. Extending the capabilities of an ontology can be done by adding to its set of constituent relationships. In theory, this may include relationships for semantic mapping, whereas semantic mapping between multiple databases will require external applications.

APPENDIX B. SCENARIOS FOR ONTOLOGY USE IN FEDERATED QUERY

This appendix illustrates examples where an ontology’s more detailed semantic relations can provide the links between databases, allowing retrieval of relevant data that are not represented directly in less-expressive data structures.

Part Relations

A database may contain information specifying which entities are parts of others, but the meanings of the “part” relations may take
different semantically or lexically distinct forms in different databases. There is an abstract, generic notion of “part” discussed in the philosophical literature, as explored by Casati and Varzi (1999); but in practical applications more specific notions of “part” may be used, such as:

1. a finger is part of a hand,
2. a person is part of an organization (group membership),
3. Sunnis are a part of the Muslim community (subgroups),
4. a division is part of a company (business),
5. a company is part of a division (military),
6. oxygen is part of water (chemical composition),
7. salt is a part of seawater (substance mixtures),
8. a word is part of a sentence (syntactic rule),
9. an angle is part of a polygon (definition),
10. an engine is part of a car (necessary part),
11. an engine is a component of a car (a component is a separately manufactured part),
12. a module is part of a spacecraft (major segment),
13. a subroutine is part of a program,
14. taking a step is part of walking (process part),
15. buying a ticket is part of taking a train (standardized script part), and
16. the Normandy invasion was part of the Second World War (event part).

The meanings of many of these are sufficiently different from the others that the use of a generic “part” relation to mean all these things would cause serious error if the assertions using those relations were used in a system that performed automated inference. Both an ontology and a traditional data model will allow specifying specialized “part” relations for semantically distinct types. In either case, one may assert, for example, that <TheNormandyInvasion wasaSubeventOf TheSecondWorldWar>, where the relation “was aSubeventOf” is used only to relate events to other Events, but the ontology will also allow us to specify that “was aSubeventOf” is a subrelation of the more generic “isaPartOf.” Therefore, if "X isaSubeventOf "Y, then it is also true that "X isaComponentOf "Y. This permits cases where one database may use the generic “isaComponentOf” relation on heterogeneous objects, and another may use a more specialized “part” relation. The fact that the second relation will answer queries on the first would be very difficult to do without the use of subrelation inferences available in an ontology.

For a specific example, a relation “isaComponentOf” may be defined in one database, with the intended interpretation being that "X isaComponentOf "Y means that "X is a separately manufactured unit that is assembled with other units to produce a "Y. In the ontology, “isaComponentOf” would be a subrelation of “isaPhysicalPartOf.” A second database may not make any distinction between components and other kinds of “parts” (e.g., an arm “isaPartOf” a body). A query for physical parts from a user of the second, less specific database could still return “components” from the first database without misinterpreting the more specific intended meaning of “component” in the first database. Thus, in one database, an engine may be part of a car, and, in another, an engine may be a component of a car. Both of these can be properly interpreted by an ontology without confusing the different intended meanings of “part” and “component.”

This simple inference is useful in cases where different databases might have more than one of these meanings of “part” in a single table, whereas in another database, only the more specific “component” relation might be used. A query from users of the first database for “parts” could automatically return “components.” Inferences on a hierarchy of relations (attributes) are built into many ontology systems (including the Web Ontology Language [OWL]), but they would need to be specially programmed with an XML schema definition (XSD) data structure. In the ontology, the proper inferences are performed automatically.

Unit Conversion

In two different databases, an attribute “height” may be represented as a decimal number. In database 1, the units are assumed to be meters. In database 2, the units are assumed to be feet. Although the intended meanings of the two attributes are similar, one cannot substitute one for the other. A mapped ontology can recognize the relations between the two attributes, and a translation program accessing the ontology mappings can convert the height values in one database to equal values in the units of the other database. An outline of an example of translation is shown in Appendix C.

Specialization-Generalization Mismatches

In database 1, there may be a table representing “Armored_Vehicles” with no subtypes of that type explicitly represented. In database 2, there may be tables for “Tanks” and “Armored_Personnel_Carriers.” A query in database 1 terminology for Armored_Vehicles must be able to retrieve Tanks and Armored Personnel Carriers from the second database. However, these mappings are not 1:1. A query for “Tanks” in the second database could retrieve all “Armored_Vehicles” from the first database. Assuming that there are no attributes that can distinguish among armored vehicles, the returned query would need to indicate in some standardized manner that the returned value (Armored_Vehicle) is an instance of a supertype of the requested type and may not actually be of the type (Tank) requested. A prefix or suffix on the returned value would be a simple method to signal such a potential mismatch, if no other mechanism were provided by the user interface.

Transitive Relations

There are relations that are “transitive” in this sense: If R is a transitive relation, then ?X R ?Y and ?Y R ?Z imply ?X R ?Z. For example, the “sibling” relation is transitive. If A is a sibling of B and B is a sibling of C, then A is a sibling of C. Such inference rules are automatically executed in an ontology, allowing the generation of new knowledge from existing knowledge. When transitive relations are used in more than one database, new knowledge can be created from the combination of the databases that is not available from any one database alone.

Some “part” relations are transitive:
X is a physical part of Y.
X is a subregion of Y.
X is a textual part of Y (where X and Y are texts).
However, some “part” relations are not transitive:
J is a member of Organization X.
A crowbar (an object) is composed of Steel (a substance).

Transitive relations must be individually marked as such in the ontology.

APPENDIX C. AN EXAMPLE OF QUERY TRANSLATION

This appendix describes a method by which a query against a database mapped to an ontology could be translated as a semantically equivalent query to other ontology-mapped databases.

Introduction

This appendix presents a simplified hypothetical example of the logical mechanism by which a Structured Query Language (SQL) query on one database mapped to the foundation ontology would be
translated into an equivalent query on other mapped databases. The actual translator program is not developed yet. This represents an outline of a small part of the kind of code that would be developed for the simple case of unit conversion in a query. The translator program would be developed once to integrate many databases. A more detailed illustration of query translation was created as a short Java program ("TestQuery.java"). That program illustrates the steps required for translation to remote databases from an SQL query posed to one of the federated databases.

This example presents a case that does not require an ontology to implement; a less-expressive data model might represent the kinds of relations used here. However, there will be other cases, particularly for transitive and inverse relations or where inference rules (e.g., business rules) are needed, where an ontology or its expressive equivalent is needed. This example exhibits a part of the mechanics of a hypothetical translation program that is more complex than simple one-to-one conversion.

The full query translator will consist of a program that can:

1. access an ontology that contains the domain model and mappings for the set of databases to be queried (the access is via an Application Programming Interface [API] implemented by the Ontology Management Program [OMP]);
2. perform manipulations on the information retrieved from the ontology;
3. receive queries from a user interface;
4. transmit queries to remote databases;
5. receive answers from remote databases; and
6. send the answers, translated into the terminology of the user, back to the user's interface.

The domain ontology for any given set of databases to be integrated will be an extension of the foundation ontology. It will contain representations of all of the terms in the full set of databases, among which translations are to be performed. The extension ontology will also contain the mappings from the databases to the extension ontology.

Other issues, such as de-duplication, can only be resolved on a case-by-case basis. The translator may have an index for simple 1:1 correspondences of elements in the different databases, making ontology query unnecessary for those cases.

For retrieval of information from remote data stores when the mappings are too complex to return data in the exact form in which they appear in the database of the originating query, information returned to the user from the remote database should indicate that it is related but not identical to the kind of information stored in the local store. Relations may be: D1 is a subtype of D2, D2 is a subtype of D1, D2 and D1 are both subtypes of a common type, D1 and D2 have overlapping but not identical types and cannot be clearly distinguished. Those cases are not illustrated here.

This translation mechanism is only an outline, and it has not been implemented for a Common Semantic Model (COSMO)–based translator. In this illustration, relations are used as variables. Some ontology query languages do not permit such use. To properly map table columns to ontology relations, the logic will need to be quasi–second order, but tactics to accomplish that effect are available for first-order reasoners. Alternatively, a procedural code that is equivalent to the logical operations specified here might be used in the translator; the ontology could be stored as another database. Additional complexity occurs when data stored in a table in one database are stored in multiple tables in a second database. In that case, the foreign keys must find corresponding elements in the other tables of such a database and will need to be identified by the ontology query mechanism. That situation is not treated here.

This simple case of measure translation is only one step more complex than a simple table lookup for one-to-one translations. More complex relationships exist among database elements and ontology elements (e.g., when a table is used to represent more than one type of entity, and some of the columns are relevant to description of one type of entity but not of another).

The Translation Program

In this example, the translation program uses Java language conventions. The ontology access will be performed using an API specific to the Ontology Management Program (OMP), but in this example, ontology queries will be shown in the Knowledge Interchange Format (KIF) language, which is interpretable by several public programs, including the SigmaKEE (Sigma Knowledge Engineering Environment, 2011; used by SUMO) implementation of Vampire (the First-Order Logic theorem prover, 2009) and the HighFleet (formerly Ontology Works) Integrated Ontology Development Environment (IODE) system.

1. The Query

A simple query is given for the height of people having the last name “Cassidy” in a local database where people are represented in a table called INDIVIDUAL, and a person’s height is in a column called “height,” the unit of measure is assumed to be meters. This example does not query for attributes of a specific individual, but for attributes of all persons having the same last name. If the attributes of individuals were to be queried, either a unique identifier would be used throughout the set of databases or some set of attributes would be used for de-duplication. Where necessary, de-duplication procedures might match individuals to some adequate level of confidence by a probabilistic matching algorithm. This example can represent any query for a particular attribute on individual entities of any particular type having some other attribute in common.

(Q:DB1) Select HEIGHT from INDIVIDUAL where NAME_LAST = “Cassidy”

2. The Query Is Received

The query is received from the Data Access Service (mechanism unspecified here) over the network by the translation program, along with the name of the original database to which the query was posted. For simplicity, the original database is called “DB1” and is received as the value of the string variable “sourceDB.”

3. The Translator Interprets the Query

The translator interprets the query by constructing the (preexisting) unique names for the database table and column by string concatenation, using a double underscore to separate database name, table name, and column name. In the program, the names of the ontology elements are stored as strings.

The pseudo-call QueryOntology used here returns an array of variable names corresponding to the return value(s) of a KIF query. More than one table and more than one column in the set of databases can correspond to a type or relation (e.g., attribute) in the ontology. In the following, the lines preceded by a double slash represent comments in the program, and those lines without the initial double slash represent computer code that can be executed as a program.

// Define some Java variables used.

String rellname[1000]; // space for the relation names
String rel2name[1000]; // space for other relation names
String tablename[100]; // space for the table names
The need for ontologies

String remoteTables[100]; // tables in the remote databases that correspond to the queried table
String remoteCols[100]; // columns (attributes) in the remote databases corresponding to the queried attribute
String remoteNumCols[100]; // columns (attributes) in the remote databases that are numbers representing a measure
String remoteMeasCols[100]; // columns (attributes) in the remote databases that represent complete measures, including the unit of measure and quantity
String remoteID // name of the unique identifier column for a remote table

// ********** Get strings for query ************
String variable “sourceDB” and the query string are received along with the call to the translator program (not shown); the query string is parsed to get the column name for the return value, the table name, and the column name(s) in the “where” clause. The names of the table and columns used in the ontology are then constructed.

String qtable = sourceDB + “__” + “INDIVIDUAL”;
String qcol = qtable + “__” + “HEIGHT”;
String idcol = qtable + “__” + “NAME_LAST”;
String idval = “Cassidy”;

// The method “QueryOntology” represents the API call to query an ontology for whatever ontology management program is being used.
// Get the corresponding ontology elements for the table and the columns:
// note that the ontology query returns an array of class names in the ontology that correspond to the table in the query. Usually (and in this case) only one Type in the ontology is represented by the table referenced in the query.

classname = QueryOntology(correspondsToTable, ?TABLE, qtable);

// The relations in the ontology that correspond to the database columns in the query are similarly found by the API call.
// Note that the idcol (unique id column) may be specific to a particular table.
// or may (as with SSN) show up in multiple tables. We need to cover all cases.

rel1name = QueryOntology(correspondsToDB_Column, ?ID, idcol);
rel2name = QueryOntology(correspondsToDB_Column, ?REL, qcol);

// Next, it is necessary to determine if the corresponding ontology elements are directly mapped, or whether some form of conversion is required. In the present case, the field value is a simple number, but it represents a measure. Therefore, it will need to be converted to the standard form in the ontology for measures of that type.
// The ontology is queried to determine if the present element needs conversion. For this simple case, we only ask whether the terminology relation is a “dbMappedMeasureRelation,” which means that it is a number representing a measure. We only illustrate the query for one corresponding ontology relation (rel2name[0], which has a value of ‘DB1__INDIVIDUAL__HEIGHT’); there may be more than one corresponding relation.
// The query call in this case returns only a true or false response.

boolean isaMeasure = QueryOntologyTF(isanInstanceOf, rel2name[0], dbMappedMeasureRelation);

// The answer for this case is “true.” That means that the type of measure and units needs to be determined, and a conversion into the ontology standard units must be performed.
// These queries are known to return unique values, so arrays are not necessary.

String measureType = QueryOntologyStr(quantifiesMeasure, rel2name[0], ?measure);
String measureUnit = QueryOntologyStr(hasUnit, rel2name[0], ?munit);
String measureUnit = QueryOntologyStr(hasUnit, rel2name[0], ?munit);

// We can also get the type of measure and its units, but these are not needed in the translation.
// They may be useful for more detailed reasoning.

String measureType = QueryOntologyStr(hasMeasure, rel2name[0], ?mtype);
String measureUnit = QueryOntologyStr(hasUnit, rel2name[0], ?munit);

// The actual number in the “height” attribute may need conversion, but that depends on whether the remote databases store the comparable data in the same or different units.
// We can postpone conversion until we know it is necessary.

// Now we can inquire whether other accessible databases have this attribute recorded for the class of Person.
// First, find tables in all connected databases that represent an Individual person:
// Since there may be more than one Type in the ontology that is represented in the local table being queried, the array classname may have a length greater than one. The query below would have to be iterated over all Types in the ontology that are returned in “classname[],” but for this example, we find that there is only one,
// so we only query for classname[0]. The return value will be an array of table names, potentially more than one in any given database, and likely more than one in the full set of integrated databases.

remoteTables = QueryOntology(correspondsToTable,classname[0], ?TABLES);

// The name(s) of the column(s) in the full set of integrated databases that correspond to the ontology relation need to be discovered. This query gets all corresponding columns,
// not only those in the relevant tables. However, only the columns in the relevant
tables will be used. We have to iterate over all of the \( n \) relations
that correspond to the column in the query, though typically there
should only be one.

In this case we are dealing with a measure, which may be
represented by a bare number
in one column (as in the table in the query-originating DB), or
by a column value
that includes the unit of measure. We need to search
for both methods of representation.

\[
\text{remoteNumCols} = \text{QueryOntology(quantifiesMeasure,}
\text{?measureCols, measureType)};
\]
\[
\text{remoteMeasCols} = \text{QueryOntology(correspondsToDB_}
\text{Column, rel2name[n], ?COLUMNS)};
\]

The individual columns are linked to the tables in which they appear
by an ontology relation.
Using this, we select only those columns that appear in the relevant
tables, thereby
not querying irrelevant tables (the relation “height” might be used
for entities other
than that in the Person table). An alternative to cross-checking the
returned columns above
with returned tables is to perform the column query for one
database table at a time,
iterating over all tables returned by the table query. In that case, the
column queries
would use a more complex expression transmitted to the ontology,
one for each
table “remoteTables[n]” (each of which corresponds to the
ontological type of the
// table DB1_INDIVIDUAL, which was queried by the original
query, in this case
// corresponding to the ontological type “Person”). The query would
then look like the
following, but the carriage returns would be omitted; they are here
just for clarity:

\[
\text{remoteNumCols} = \text{QueryOntology((and}
\text{(isaColumnInDBTable ?measureCols}
\text{remoteTable[n]})))
\]

An example of cross-checking of tables and columns is
omitted for brevity
because it is a simple iterated string comparison in Java.

To determine whether any conversion of the number in the measure
column is
required, to translate to some other database, the query above for
“measureUnit”
is repeated for each corresponding remote database column that
contains a bare number.
If the unit of measure is the same, no conversion is necessary. If
the unit of measure
is different, the conversion will need to be done by some mathematical function. Such
conversions may be simpler in the procedural code of the translation
program, but the
conversion factors should be encoded in the ontology in any case,
to minimize dependence on
procedural code.

************** Correlating the NAME_LAST column
**************
The need for ontologies

Manuscript accepted by the society 17 February 2011

Printed in the USA