Applied Ontology 10 (2015) 243–258
DOI 10.3233/AO-150158
IOS Press

Position paper

Ontology and database schema: What's the difference?

Michael Uschold
Semantic Arts, 7405 125th Place SE, Newcastle, WA 98056, USA
Tel.: +1 425 830 9475; E-mail: michael.uschold@semanticarts.com

Abstract. This paper analyzes the similarities and differences between an ontology (focused on meaning), and a database schema (focused on data). We address questions about purpose, representation, creation, usage and semantics of each. We distill out twenty-five features that characterize these two representational artifacts, the majority of which are relevant to both. Each has a strong semantic heritage using formal logic to build conceptual models of some subject matter. And while there are differences in 90% of the features, the differences are mostly historical, not technical. We identify pros and cons for each, and notice that there is usually no free lunch. The disadvantage that you think you are getting rid of may show up elsewhere in a different and unexpected way. We close by considering how ontology contributes to enterprise data integration. The emergence of using URIs as global identifiers (e.g. in OWL) dramatically enhances data integration as well as schema reuse and sharing. The primary focus on meaning helps ontology break through a lot of unnecessary complexity that exists in large traditional databases and greatly simplifies the process of integration. Ontology is providing a glimmer of light at the end of the tunnel for enterprise-wide data integration.

Keywords: Ontology, schema, database schema, conceptual model, logical schema, physical schema, triple store, relational database, data integration

Accepted by: Leo Obrst

1. Introduction

We address the frequently asked question: what is the difference between an ontology and a database schema? This paper adopts a conversational perspective focusing on operational and pragmatic issues rather than attempting to offer deep theoretical insights into formal matters. Because they are by far the most prevalent, we will focus mostly on relational schema. Some but not all of our remarks will apply beyond the relational model, where there are many ways to specify schema and represent data. JSON an XML schema are popular. There are also deductive databases object-oriented databases and graph databases.

- An *ontology* is a model that clarifies and specifies a set of meanings in a formal language. Those meanings reflect the ontologist's understanding of the target subject matter, regarding the kinds of things there are and how those things are related to each other.
- A *database schema* defines the structure of a database in a formal language. There are three kinds: conceptual, logical and physical. While we recognize that there are important differences, for con-

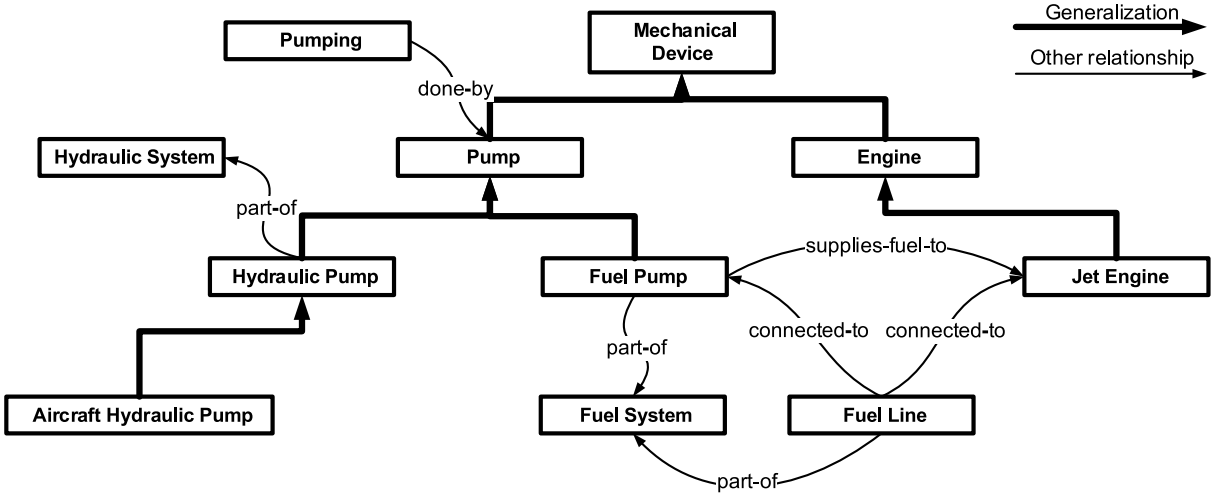

Fig. 1. An ontology generally has a taxonomy as a backbone with multiple link types each having a precise meaning. Cardinality constraints are not shown.

venience, we will conform to common usage and use the term "database schema" loosely to refer to all of these. We will call out differences as appropriate.

The fundamental similarity between an ontology and a relational database schema is that (at the conceptual level) both consist of set of type definitions expressed in a formal notation. There is significant overlap with ontologies and early work on conceptual schema for database systems. Many of the same researchers worked on both. For example, there was an ACM workshop in 1980 on data abstraction, databases and conceptual modeling.[1]

Consider the simple example in Fig. 1, showing a graphical view of an ontology which has a taxonomy as a backbone, plus various relationships each with a well-defined meaning. This could be easily expressed in a formal ontology language such as OWL. Figure 2 shows a logical schema in a graphic notation based on Information Engineering (IE) notation, described in Finkelstein (2006). Although the graph structures are identical, a deeper look reveals a wide range of important differences. Many arise from the fact that the database and ontology communities evolved fairly independently for very different purposes.

From decades of personal experience and careful analysis of the similarities and differences between an ontology and a database schema, the following five questions emerged as the basis for comparison.

1. What is it for?
2. What does it look like?
3. How do you build one?
4. How is it implemented and used?
5. Where are the semantics?

These questions are addressed in turn in Sections 2–6 of this paper. We consider how the questions are answered differently for a database schema vs. for an ontology. In Section 7 we summarize the analysis and draw some conclusions.

[1] http://dblp.uni-trier.de/db/conf/sigmod/pingree80.html.

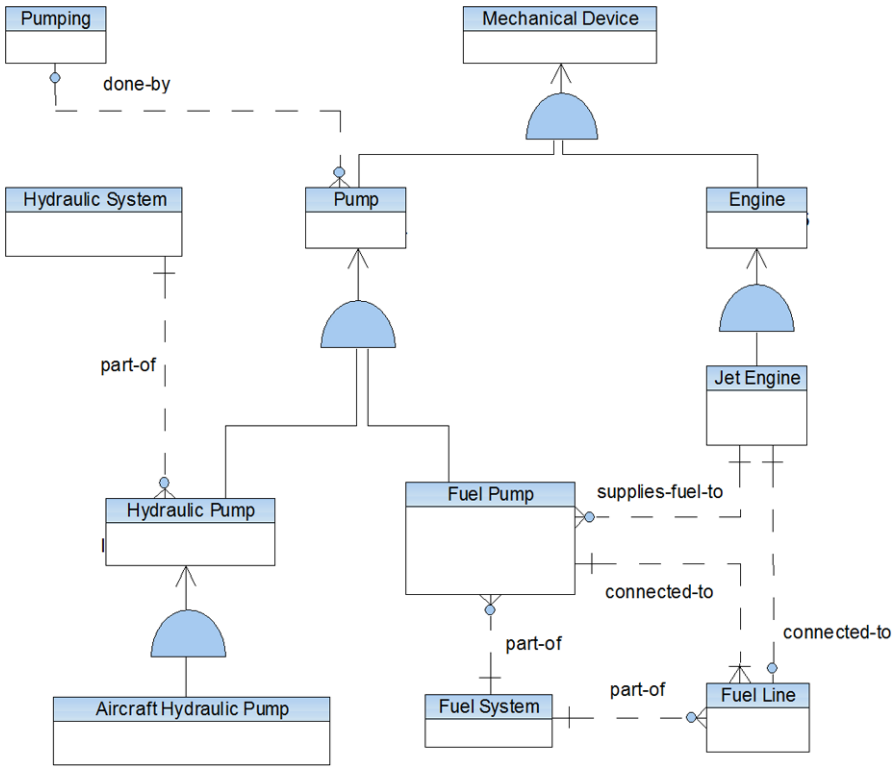

Fig. 2. Logical schema in IE notation. Boxes denote entities. Dashed lines connecting two entities denote relationships and cardinality constraints.

2. What is it for?

2.1. Purpose

The fundamental focus of an ontology is to specify and share *meaning*.

The fundamental focus for a database schema is to describe *data*.

A relational database schema has a single purpose: to structure a set of instances for efficient storage and querying. The structure is specified as tables and columns. An ontology can also be used to structure a set of instances in a database. However, the instances would usually be represented in a [possibly virtual] triple store, or deductive database rather than directly in a relational database.

Ontologies have a broader range of purposes including human communication, interoperability, search, and software engineering – Uschold and Gruninger (1996). The structure in an ontology is specified in as a network of objects and relationships. One or more parties commit to using the terms from the ontology with their declared meaning. To some extent, this is also true for conceptual or logical schemas.

The different roles played by an ontology vs. a database schema are responsible for a variety of other differences. For example, ER diagrams certainly *can* and are used to get people to agree on terminology and meaning of things that the data is about (the primary purpose of an ontology), but structuring data is their primary purpose. ER diagrams are typically used in the conceptual modeling phase, which abstracts away from data – but people typically don't create ER diagrams unless they are going to build a relational

database. A possible exception to this is to use an ER tool to build conceptual models that are exported into an ontology language.

The goals of sharing and reuse have been recognized as important by the database community long before ontology modeling languages became popular. For example, as part of the ANSI/SPARC project, Jardine (1977) proposed three levels of schema. The external schema corresponded to user views; it addressed an API to promote sharing and reuse that is independent of representation. The internal schema addressed efficient storage and access methods and would be called a physical schema today. The conceptual schema addressed what data is stored and how the data is inter-related. The word 'ontology' was not used in those days, but the conceptual level was about capturing the meaning of the data and its relationship to the world.

2.2. Instances and data

The *raison d'etre* for a database is to store and retrieve data. In a relational setting, data is represented in tables. A row corresponds to an instance of some entity in the ER diagram. By contrast, because the main focus for ontologies is on meaning, ontologies can be useful even when there are no instances or data of any kind. The purpose of such an ontology is to communicate a shared meaning. A second difference is that ontologists commonly create so-called 'abstract' classes with the explicit intention of never having any individuals be directly asserted as members. Direct assertions are meant to be into more specific subclasses. Membership into the abstract classes is inferred. Except at the conceptual level, there is nothing comparable in an ER diagram. All entities are intended to have data about individual instances.

2.3. XML schema

The primary focus of an XML schema is to define and constrain the structure of a set of documents, all expressing the same kind of information. Meier (2003) describes how in recent years, large collections of XML documents are being stored in specialized XML databases.

3. What does it look like?

What a database schema or ontology looks like directly depends on the notation used, and the expressivity. Related to this is a question about the information content of the schema or ontology – i.e. what kind of information is being expressed?

3.1. Notation

3.1.1. Syntax

ER diagrams such as found in Fig. 2, are the most common syntax for creating a database schema. Although these diagrams have semantic force, there is no standard serialization syntax.

There are many notations for ontologies, most are logic-based languages. There are many ad hoc ways to graphically depict ontologies. Unfortunately, unlike ER diagrams, none of the ontology diagramming conventions are standardized at this time. Gasevic et al. (2004) describe tools for converting UML and possibly other standard modeling notations to ontology languages such as OWL.[2]

[2]See: http://www.w3.org/2007/OWL/wiki/UML_Concrete_Syntax.

OWL is currently the only ontology language that is both a standard and also has fairly good tool support. One extremely important feature of OWL is the use of URIs as global identifiers. This has profound implications in the ability to reuse ontologies and to perform data integration. This is not an inherit characteristic of ontology vs. database schema, since most ontology languages also lack this feature. From a technical viewpoint, it may be possible for this to be added to database technology.

3.1.2. Semantics

A database schema language is not typically based on a rigorous formal semantics. One can ascribe a semantics to them if one wishes, and the process of doing so may uncover ambiguity. To avoid such ambiguity, the semantics of ontology languages are rigorously specified.[3]

3.2. Expressivity

3.2.1. General

As illustrated in Figs 1 and 2, there is a strong overlap in what can be expressed using ER diagrams compared to typical ontology languages. This includes: objects, properties, aggregation, generalization and set-valued properties. Entities in an ER diagram correspond to classes in an ontology, and attributes and relationships in an ER diagram correspond to binary relations or properties in most ontology languages. For both, there is a vocabulary of terms with natural language definitions. As a matter of historical fact, rather than technological necessity, such definitions are found in a data dictionary, separate from the database schemas. An ontology typically has such definitions inline as annotations.

There are also many specific differences in expressivity, which vary in importance. Many of the differences are attributable to the historically different ways that database schemas and ontologies have been used. One key difference is that an ontology usually has a taxonomy of kinds (e.g. classes) as a central component, its backbone (see Fig. 1). In the data modeling context, while conceptual and logical database schemas may have a taxonomy of different kinds of entities, that information is lost when creating the physical schema. So taxonomies play a relatively minor role. Taxonomic reasoning is fundamental to most ontology applications and it is not a major feature of most DBMS. That said, if recursion is allowed by the DMBS, taxonomic reasoning can be supported by view or stored procedure. It can also be simulated in various ways.

Interestingly, although ER diagrams are now very popular, the early proposals for representing conceptual schema recommended using full first order-logic plus a type hierarchy. Most of the people at the 1980 ACM conference mentioned above, used some version of formal semantics.[4]

3.2.2. Constraints

For ontologies, constraints are called axioms. From a model-theoretic perspective, axioms constrain the possible interpretations. Their main practical purpose is to express machine-readable *meaning* to support automated reasoning. Sirin (2010) and Tao et al. (2010) show how reasoning can be deployed to use the axioms as integrity constraints

For databases the reverse is true – the primary purpose of constraints is to ensure the *integrity* of the data (i.e. instances). These integrity constraints can also be used to optimize queries and help humans infer the meaning of the terms. Unfortunately, it often happens that constraints are hardwired in procedural code for efficiency. This results in loss of human readability and higher maintenance costs. Better practice is to specify constraints declaratively.

[3] See: http://www.w3.org/TR/owl2-direct-semantics/ OWL 2 Web Ontology Language Direct Semantics (Second Edition).
[4] Personal communication, John Sowa.

An important distinction here is open vs. closed world. In an open world context, the inference engine can distinguish between "no" (provably false) and "don't know" (unprovable either way). This impacts on what happens if a constraint is [apparently] violated. In the case of using ontological inference, several things can happen.

1. A contradiction may be inferred, this makes the whole ontology inconsistent.
2. An open world assumption can result in an unwanted inference. For example suppose we have an OWL object property, hasPump with the class, Pump as its range. If we mistakenly assert that a particular a hydraulic system hasPump an individual that is an instance of the class Engine, the reasoner will infer the instance of the engine to also be a Pump. It is good practice to assert disjointness axioms that would catch this by inferring a contradiction; this tells you to go fix the problem.
3. The theorem prover can 'know' that some individual is related to another individual in a specific way, but not know which individual that is. For example, an axiom might say every hydraulic system has at least one pump as a part. Suppose there is a hydraulic system, but it is not connected to any specific pump. In an open world, this is no problem, the theorem prover invents a temporary [skolem] constant that stands for that individual. In a close world, the conclusion would say that there the hydraulic system has no pump, and there would be a contradiction.

In a database context, such a cardinality constraint is handled differently. It is used in a closed world context as an integrity constraint for data validation. In the case of declaring something to have a pump that is also an engine, it would be checked before it was asserted, and thrown out as not being valid. It performs a gatekeeper function.

Cardinality and delete constraints are important kinds of integrity constraints which have highly DB-specific uses that are outside the scope of most or all ontology systems. For example, cardinality constraints are used for getting the foreign key in the right direction and to ensure that extra tables are built for many to many relationships. Delete constraints reflect dependencies, indicating what other things should be deleted, if a given thing is deleted. Such database constraints do suggest meaning, but this may be of secondary importance. By contrast, the main roles for cardinality constraints in ontologies are (1) to express meaning and (2) to ensure consistency of the ontology or of the instances or both.

Delete constraints introduce another important point about database vs. ontologies. There are no nulls in an ontology, nor in a triple store using an ontology as a schema. This is important because nulls are ambiguous. It is not clear what they mean; this causes problems.

3.3. XML schema

An XML schema is an .xsd file. There are many examples of "markup languages" that are really just .xsd files defining an XML schema. For example, the Financial Products Markup Language (FpML).[5]

4. How do you build one?

4.1. Reuse or start from scratch?

Because of the focus on meaning, and because what terms mean is often relatively stable in a given domain, existing ontologies can readily be reused when building a new ontology. Baclawski and Schneider

[5] See: http://www.fpml.org/ for an overview of FPML.

(2009) describe OOR, one of several on-line libraries of ontologies. There are specialized search tools to find ontologies about specific topics, or with specific terms in them.[6] Despite this, ontology reuse is not as widespread as it should be, due to the disparate needs of ontology developers, as well as the wide variation in quality and purposes of the ontologies.

However, OWL ontology developers do have a powerful and convenient infrastructure for reusing existing ontologies, it is based on global URIs that refer both to ontologies as a whole, and to their elements (e.g. classes, properties and individuals). There is also an explicit mechanism to import other ontologies. Because OWL is a standard, any tool from any vendor may be used to import and reuse any other OWL ontology. This is in stark contrast with relational database technology.

Relational schema developers in industry typically do not make much use of schema reuse infrastructure. There are many patterns that get reused, but otherwise schema developers mostly start from scratch. In principle, conceptual and logical schemas which explicitly capture much useful semantic information could be reused. However the semantics are left behind when the physical schema is produced and the logical and conceptual schema typically do not evolve along with the physical schema. This greatly limits reuse.

There is one other factor that makes it easier to reuse an OWL ontology vs. a database schema. There is no distinction between the data and the metadata. In a triple store, it's just triples. Relational schema formalisms are much less portable, as they differ across different vendors.

The main difference here is due to there being a standard ontology representation language. There may be business reasons, but there are no technical reasons why standards could not be created for relational schema languages. Similarly, the non-standard ontology languages do not enjoy the benefits of a convenient reuse infrastructure in the same way.

4.2. Normalization

Database design principles include a set of rules for keeping the database schema normalized (e.g. to minimize duplication and avoid data anomalies). This is pervasive in the industry, everyone who builds a logical schema follows these rules. The guidelines are in natural language, and although some tools have been created to support this process, hardly anyone uses them – usually it is a manual effort.

There are no similarly pervasive guidelines for building ontologies that are commonly used. Guarino and Welty (2009) created guidelines for building ontologies that help ensure a logically consistent hierarchy of classes (OntoClean). Welty et al. (2004) describe a prototype that demonstrated the utility of this method in an industrial setting. OntoClean is fairly difficult to understand and apply, which may be part of why it is not more used. Völker et al. (2005) describe an approach to automate some of this analysis. These tools are not widely used in practice.

Rector et al. (2012) and Rector (2003) describe techniques analogous to database normalization, chiefly for the purpose of enable collaborative development of modular ontologies with guarantees to avoid collisions.

There is a rich body of experience in how to build good ontologies, however quite a lot of it is in the academic literature, in the heads of working ontologists, or in long email discussions that never see the light of day. Some of the core ideas have been distilled out and captured in a series of informal notes published by the W3C Semantic Web Best Practice and Deployment Working Group.[7] Allemang and Hendler (2011); Arp et al. (2015) and Fernandez-Lopez and Corcho (2010) have all written books on

[6] See: http://swoogle.umbc.edu/ or http://watson.kmi.open.ac.uk/WatsonWUI/ to search for ontologies on the web.
[7] See: http://www.w3.org/2001/sw/BestPractices/.

ontological engineering, some geared at researchers, some at practitioners. There has been literature and an ongoing series of workshops on ontology design patterns,[8] for example: Gangemi and Presutti (2009). There are also ontology engineering tutorials regularly offered at academic and industry conference. Nevertheless, common practice is highly varied among individuals. There is no 'bible' on ontological engineering that is widely used across either academia or industry.

4.2.1. Optimization

Optimization is a fundamentally important step for developing a database schema A logical schema is transformed into a physical schema. Optimizations are done manually using the ER diagramming tool. The optimizations are geared for the SQL engines that perform the queries. *Importantly*, the schema is optimized for a particular set of queries that are expected to be run on a regular basis. Enforcement of DB integrity constraints is expensive; therefore some constraints identified during modeling are left unstated in the physical database schema. This results in loss of captured meaning. It also allows for "dirty read/write" operations that must be carefully monitored to limit impact on business.

Ontologies are used in conjunction with logic-based inference engines that are built for specific ontology languages (e.g. OWL) and many are used in conjunction with triple stores using SPARQL for queries, see DuCharme (2013).

There are two separate phases of optimization. First, for the theorem prover, and second for SPARQL access to the triple store. The inference engine designers put optimizations in reasoners based on expected patterns of ontology use in general. The optimizations are generally independent from any particular ontology. By contrast, database schema optimizations are performed directly on the schema and are geared for specific kinds of queries.

People who build and use large scale ontologies learn which kinds of combinations of constructs cause the reasoner to run slowly. Knowing this, they can 'optimize' their ontology by avoiding those constructs. For example, adding any of: property inverses, individuals, hasValue restrictions, or disjoint constraints in OWL can dramatically slow down the inference engine. This is analogous to optimizing a physical database schema. Care must be taken when doing these performance tweaks. Removing things that help say what is true about the subject matter being modelled can compromise the semantic integrity and clarity of the ontology. This can be mitigated with annotations documenting the purpose of such tweaks.

In the case of a physical schema, much more dramatic changes occur as semantics are left out of the schema in favor of query performance. For example, much relationship information is lost: names are no longer in the schema, though they may be kept in the system catalogs. The connection between classes is not explicit and many-to-many relationships are replaced by extra tables. Something similar happens with triple stores, the entire schema of a triple store is one big extra table of this sort. Fig. 3 shows the physical schema that results from transforming the logical schema in Fig. 2. The scalability of relational database technology achieved by the optimizations comes at a cost: reduced flexibility and agility.

The second phase of optimization occurs after an OWL ontology has been checked for consistency with the theorem prover and it is loaded into a triple store. From that point, inference is limited to whatever the vendor happens to provide, and what can be done 'manually' using SPARQL. Here the situation can in principle be similar to SQL. In both cases, there is now a single immutable schema which precludes any opportunity for further performance gain. It is still early days for SPARQL technology and triple stores in terms of commercial deployment. It remains to be seen how much additional flexibility can be enjoyed for ontologies, while retaining adequate performance.

[8] See: http://ontologydesignpatterns.org/wiki/WOP:2015.

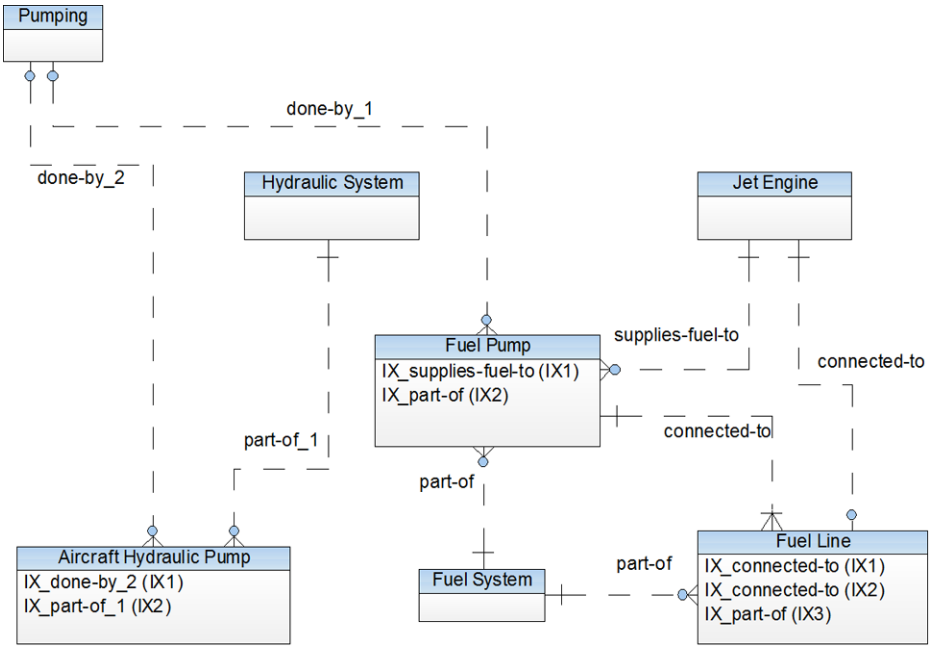

Fig. 3. A physical schema in IE notation. Compared with Fig. 2, relationship information is lost. For example, the relationship names are gone.

4.3. XML schema

While many markup languages in xsd may be created from scratch, it is common to build complex markup language in a modular fashion, so that many definition are reused in many places across different .xsd files. An important limitation is the lack of global URIs, so reuse is limited to be within a given set of .xsd files.

5. How is it implemented and used?

5.1. Change management, agility and flexibility

As noted above, databases are highly optimized for a specific set of queries. If one wishes to pose other kinds of queries, then two things often happen: (1) it is awkward and time-consuming to build the query and (2) it takes a long time to execute the query. The effect of this is rigidity. When the database schema is in place, and the DBMS is up and running, then you are locked into a restricted set of queries that can be posed in a cost-effective way.

For various reasons, it is difficult to evolve the database schema and applications that use it:

- much re-programming and re-structuring is required,
- the data dictionary and conceptual model (and possibly the logical model) go out of date,
- the semantics are hardwired in procedural code which is error prone,
- the semantics are left behind,
- applications are tightly coupled with the data schema.

When using an ontology as a data schema, there is no restriction to a specific set of queries. This is a huge advantage. Anyone can ask any kind of query at any time using the same query formalism (e.g. SPARQL). A relational implementation could provide views to sustain arbitrary query patterns as well, but there would be a cost for their computation.

Evolving an ontology is analogous to evolving a database schema. If new things are added, and nothing else is changed, it is relatively straightforward. Existing data need not change. If there are significant changes to the existing schema components, then the data needs to be carefully migrated to the new schema. This is routine practice in the DB world, and there is much tool support (e.g. ETL tools).

The use of an ontology as a data schema is of growing importance, yet there is less tool support. One can use semantic mappings from the existing ontology to the new ontology and use automated inference to help migrate the data. This is a slow expensive process that typically involves writing scripts or SPARQL.

Both ontologies and database schemas are difficult and expensive to evolve. Given the greater semantic clarity, however, an ontology can be easier to modify and maintain than a database schema. You cannot maintain what you cannot understand. Also, there is typically much looser coupling between ontologies and applications that use them than between applications and the database schema that they use. Queries can be at the semantic level, and can be used across multiple applications more readily. One big advantage that ontologies, combined with automated theorem proving, could have over a relational schema is that entailment could be used to evaluate the potential effects of changes.

Versioning for ontologies is relatively immature. There has been some research, but most commercial ontology tools have only rudimentary support for versioning, if any.

5.2. Processing engines

For both database schema languages and ontology languages, there are processing engines that can handle complex logical expressions. For DB queries, SQL engines are highly specialized and tuned for answering queries, reasoning with views and ensuring data integrity. SQL engines are also generally standardized. For ontologies, the processing engine is normally based on a logic-based theorem prover, specific to the language. For example, OWL is a standard, and there are several theorem provers to choose from. The fundamental role of a reasoning engine is to derive new information.

One very important use of inference is to check the logical consistency of an ontology. We saw this earlier in the hydraulic system and pump example. It is a very important way to root out certain kinds of bugs.

Note that deriving new information and checking consistency can take place with or without instances. If there are instances, then inference can help with data integrity. Classically, such mixing of types with instances does not take place with database schema and data. This is mainly due to much greater scale and performance requirements for database systems.

5.3. Executability

Unlike an ontology, automated inference is not leveraged to check internal consistency of a database schema. At runtime, a schema is embedded in the database, it is not a separately executing artifact. Ontologies executing provide a wide variety of services in applications that use them. Uschold (2008) describes how in some cases, the ontology drives the application.

Another important difference is that it is common place for users to load an ontology as a separate artifact and then query it using the same query language as for the database (i.e. triple store). You can

ask what are the subclasses of a given class, or about cardinality constraints for a given property. The same is true for a deductive database using say Flogic as the ontology language, – Kifer and Lausen (1989). This is partly due to the fact that you can also load the ontology along with the instances in one single database. In the case of OWL, neither the language nor the infrastructure recognizes a formal distinction between data and metadata. There is an analogy for database schema; one may be loaded, and its structure queried via a system catalog. The difference is that it does not happen in the same database using the same query language.

5.4. Performance

Until recently, there was a substantial difference in performance for a DBMS vs. an ontology-based system with a triple store back end. The gap is much smaller now, and often it depends more on the specific type of data you have, than it does on the underlying technology. Even where there is worse performance for a triple store, the added flexibility may be the greater concern – especially since triples stores continue to get faster. There is a growing number of large scale commercial deployments using triple stores. Two well-known examples are the BBC, Raimond et al. (2010), and Garlik, Harris et al. (2009).

The fundamental tradeoff is flexibility vs. performance. Flexibility is obtained through greater ease in maintaining and evolving ontologies and relatively looser coupling with applications. Relational DB technology scales better when the data naturally fits into a table structure.

5.5. XML schema

XML data is not restricted to tables. There is no similar penalty in flexibility as there is for relational databases. The standard query language is XQuery, and there are a number of fast engines available in commercial products. An XML schema is a computational artifact that is used to ensure that the XML documents are conformant.

6. Where are the semantics?

Conventional DB technology really shines in certain areas, particularly when speed and scalability are of paramount importance, and where the data fits nicely into tables and where things are expected to be stable. In comparing ontologies and database schemas, we noted various challenges faced by current database technology. It is difficult to understand the meaning of the schema components. This, in turn makes it hard to reuse a schema and also makes maintenance difficult. This in turn increases the difficulty and cost of creating a new schema, as well as evolving and maintaining an existing schema. This in turn reduces agility.

The root cause of these challenges is the lack of explicit semantic information. The semantic information largely exists in the conceptual and logical schema as well as in the data dictionary. It is intended for use by humans, so they can communicate requirements and design specifications. The semantics is never intended for machine processing. The conceptual and logical schema are rarely maintained. Worse, the semantics are thrown away when creating the physical schema. Furthermore, the integrity constraints are often hard-coded in procedural code. In summary, in mainstream practice, the semantics are lost, tossed or out of date.

By contrast, the whole focus for ontologies is on explicitly encoded semantics that can support automated reasoning. Not all ontologies are built equal, of course, it is possible to build bad ontologies whose semantics are not very useful.

An interesting question is under what circumstances an ontology and a database schema expressing information on the same subject matter are logically equivalent.

6.1. XML schema

There are dozens or hundreds of industry markup languages that are expressed as an XML schema. Creating these markup languages entails a considerably amount of analysis and representing of subject matter, analogous to conceptual modeling for a database schema and an ontology. At a glance, some parts of an XML schema bear a strong resemblance to an ontology. Elements correspond to ontology classes and relational tables. Sub-elements connected to their element in a conformant XML document correspond to columns in a relational table, and triples in an triple store. Many attributes in an XML schema naturally map to OWL datatype properties. There are also cardinality constraints that look very similar.

With the increased interest in semantics and ontologies, people began to wonder: can we convert an XML schema to an ontology? Although XML was initially heralded as being very good at expressing meaning, that meaning is accessible only to people reading the markup, not to machines. Antoniou and Van Harmelen (2008) and Cover (1998) explain this very well. An XML Schema is fundamentally about the structure of a document, and not about meaning. For this reason, attempts to automatically convert an XML schema to an ontology provide results of limited utility. The trickiest part is deciding what in the XML schema is semantically relevant and worth extracting.

While there are fully automated xsd2owl converters, the end result is better thought of as a transliteration. This can be helpful to those who prefer viewing owl files over xsd files. Nevertheless, the converted owl file may not look or behave much like an ontology that a human would build, nor that captures the semantics of the subject matter. Doing that requires a lot of manual effort.

7. Summary and conclusion

7.1. Summary by the numbers

To compare ontologies and database schemas, we asked a series of questions. In answering these questions, we identified over two dozen aspects and features that shed light on the similarities and differences. Forty percent of the features were unique to one, playing a minor if any role for the other. Fifty percent of the features were much more strongly associated with one than the other. Twelve percent of the twenty four features were fundamental to both.

The three features or aspects fundamental to both ontologies and database schema are:

1. Use of a formal language for representation.
2. Expressivity: representing types, properties, aggregation, generalization and constraints.
3. The use of constraints for consistency checking.

The features or aspects that are fundamental or common for ontologies but are secondary or optional for a database schema are:

4. Representing and sharing meaning for some subject matter.
5. Taxonomy of types.
6. Wide range of purposes other than data storage and retrieval.
7. Natural language definitions included in the main artifact.
8. Constraints to assist in specifying meaning.
9. Constraints and automated reasoning to check consistency in the artifact (ontology) itself.

The following features or aspects are fundamental or common for a database schema but are secondary or optional for an ontology:

10. Efficient storage and querying of data (instances of types).
11. Standardized diagramming notation for conceptual model (e.g. ER).
12. Separate artifact for natural language definitions (data dictionary).
13. Constraints for data integrity.
14. Industry wide systematic methods for construction (i.e. normalization).
15. Explicit primary focus is to be scalable to huge sizes.

The following features or aspects are fundamental or common for ontologies but play a minor if any role for database schemas.

16. Abstract types intended to have no instances.
17. Reused for building new ones.
18. Reused in unanticipated ways.
19. Formal model-theoretic semantics and inference engines, logic an explicit foundation.
20. Executable.

The following features or aspects are fundamental or common for database schemas but play minor if any role for ontologies.

21. Cardinality constraints for getting the foreign keys right to ensure that extra tables are built for many to many relationships.
22. Throw away the semantics after the conceptual modeling phase.
23. Optimization for a particular set of anticipated queries.
24. Sophisticated tool support for migrating data when schema evolve (ETL tools).
25. SQL for querying data.

This is summarized in Figs 4 and 5. What conclusions can we draw from this? 60% of the features are common to both, and the three features that are fundamental to both are arguably the most important ones: what is expressed and how. Therefore one might conclude that these two entities are not so much different beasts, as they are similar beasts that evolved independently. Most of the differences can be explained by the fact that they arose for historically different purposes in separate communities.

One might also focus on the differences. After all, the features fundamental to both are only twelve percent of the countable features. Even though the common features are quite important, it is still true that almost 90% of the features show substantial differences between the two. From this perspective, one may conclude that they really much more different that they are the same.

7.2. Key observations

What are the key takeaways from this analysis? As usual, the same data can support different conclusions. In the end, the more important thing is to know and understand what the similarities and

Database Schema	Ontology
Focus on data	Focus on meaning
DB Constraints • to ensure integrity • often hint at meaning	Ontology axioms • to specify meaning • may be used for integrity
Little emphasis on ISA hierarchy	ISA hierarchy is a backbone
SQL engines • querying, views • data integrity	Theorem provers • infer new information • ensure consistency
Instances are central	Instance are optional
Data dictionary • generally separate from schema	Annotations • generally part of the ontology

Fig. 4. Key differences between a database schema and an ontology.

	Core for DB Schema	Secondary for DB Schema	Rare or Unimportant for DB Schema
Core for Ontology	1. Represented using a formal language. 2. Expressivity: types, properties, constraints. 3. Constraints for consistency checking.	4. Shared meaning of some subject matter. 5. Taxonomy. 6. Multi-purpose. 7. Embedded natural language definitions. 8. Constraints for meaning. 9. Constraints for ensuring self-consistency (not data).	16. Abstract types w/no instances. 17. Reused to build new ones. 18. Reused in unexpected ways. 19. Formal model-theoretic semantics.
Secondary for Ontology	10. Efficient querying and storage for data. 11. Standardized diagram notation. 12. Separate natural language definitions (data dictionary). 13. Constraints for data integrity. 14. Industry-wide construction guidelines (normalization). 15. Scale to huge sizes.	**More Alike?** **• Over 60% of features are common to both.** **• The 3 features core to both are the most important: what is expressed and how.**	
Rare or Unimportant for Ontology	20. Cardinality constraints for getting foreign keys right and ensuring tables created for many-to-many relationships. 21. Toss semantics after conceptual modeling. 22. Optimization for specific set of queries. 23. Sophisticated tool support for migrating data when schema evolve (ETL). 24. SQL for querying data.	**More Different?** **Only 12% of features are core to both.**	

Fig. 5. Feature comparison: more alike, or more different?

differences are so that each can be leveraged to best advantage. There is little value in arguing that one is categorically better than the other. A more useful conclusion to draw from this analysis is that each has an important role to play; the two can be used in concert. Ontologies are a good choice for implementing conceptual and logical schema, or at least they can play a useful role in constraining their design.

Both database schema and ontology have a strong semantic heritage based on formal logic. Many of the differences that we see today are not technical. Rather, they originate from factors that are historical, cultural, or affected by the business interests of vendors.

Some of the differences seem at first appear to give one technology a clear advantage over the other. However, there is usually no free lunch. The disadvantage that you think you are getting rid of may show up elsewhere in a different and unexpected way. For example, we observed that meaning is often thrown away in a database context. Something similar can happen if you load an OWL ontology into a triple store which does not implement an OWL theorem prover. Some of the semantics will fall on the floor.

Open-world semantics that some ontology languages have is very different from closed-world seman-

tics that all relational databases and triple stores have. It plays out most obviously in how constraints are interpreted. The difference is whether a constraint is playing the role of a gatekeeper, or just results in more inferences.

7.3. Data and enterprise integration

In closing, we note the important role that ontology plays in data and enterprise integration. Data integration/interoperability is a pressing need in most organizations. One reason for this, is that each application uses its own database, even when the applications are doing similar things. Also, more databases show up each time a company grows by acquisition. There are various barriers to integrating databases:

1. The lack of globally unique identifiers that span all the systems and databases in an organization.
2. The amount of work it takes to find out what the schema means, largely due to the semantics not being readily available, but also due to ...
3. The sheer size and complexity of a typical database. A major driver for complexity is that there is a random mix of schema elements that are expressing meaning in the subject area and schema elements that are only relevant to a particular application or database.

Ontology helps overcome the first barrier if you use a language such as OWL, which provides globally unique URIs. They can link both data items (like John Jones) as well as schema elements (like Patient). These URIs make it possible to have a corporate knowledge graph where both schema and data just snap together. Reuse and sharing is dramatically easier.

Ontology helps overcome the second barrier because of the primary focus on semantics. An ontology, by its very nature, is unlikely to evolve to where the semantics are less clear. In a typical relational database, the conceptual schema is rarely maintained as the database evolves.

Ontology helps overcome the third barrier, also as a direct consequence of the primary focus being on semantics. This is a recent discovery. Why does this work? A database schema in a large company often has tens or hundreds of thousands of tables and attributes. It is a daunting problem to integrate two such databases, even if you know exactly what everything means. In our firm, we start by building an ontology to be used as a data schema in a triple store. By focusing on the semantics of the subject matter and ignoring the application-specific things, we find there is typically one to two orders of magnitude reduction in the size of the ontology compared to the size of the original schema. Yet, amazingly, when we convert the original data into triples against the ontology, everything fits. It is radically easier to map a large and complex database onto a small, simple ontology than onto another large and complex database. You can continue mapping additional databases one by one, evolving the relatively small ontology as you proceed. Over time, the ontology can evolve to cover the scope of a large enterprise.

Building an enterprise-wide data model has long been a holy grail, but it never seems to work. Perhaps the reason is that insufficient attention has been paid to separating out the stable and unchanging parts of a business from all the other things that have more to do with details of specific applications (which change or are replaced on a regular basis). Ontology is providing a glimmer of light at the end of the tunnel for enterprise-wide data integration.

Acknowledgements

I am grateful to John Thompson whose high level of practical experience in both building database schema and ontologies enable him to answer my many dozens of questions on the material in this paper.

I am grateful to Phil Bernstein, Tim Wilmering, Jun Yuan, Anhai Doan, Bill Andersen, Nicola Guarino, John Sowa, Amit Sheth, Dan Carey and Dave McComb for participating in discussions or reviewing prior drafts and helping me understand and articulate the key issues in this paper. The paper also benefits from many useful comments from 3 anonymous reviewers.

References

Allemang, D. & Hendler, J. (2011). Semantic web for the working ontologist: Effective modeling. In *RDFS and OWL* (2nd edn.). Elsevier.

Antoniou, G. & Van Harmelen, F. (2008). *A Semantic Web Primer*. MIT Press.

Arp, R., Smith, B. & Spear, A.D. (2015). *Building Ontologies with Basic Formal Ontology*. Mit Press.

Baclawski, K. & Schneider, T. (2009). The open ontology repository initiative: Requirements and research challenges. In *Proceedings of Workshop on Collaborative Construction, Management and Linking of Structured Knowledge at the ISWC* (p. 18).

Cover, R. (1998). *XML and Semantic Transparency*, The XML Cover Pages. http://xml.coverpages.org/xmlAndSemantics.html. Accessed 2015-11-5.

DuCharme, B. (2013). *Learning Sparql*. O'Reilly Media, Inc.

Fernandez-Lopez, M. & Corcho, O. (2010). *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer.

Finkelstein, C. (2006). Information engineering methodology. In *Handbook on Architectures of Information Systems* (pp. 459–483). Berlin, Heidelberg: Springer.

Gangemi, A. & Presutti, V. (2009). Ontology design patterns. In *Handbook on Ontologies* (pp. 221–243). Berlin, Heidelberg: Springer.

Gasevic, D., Djuric, D., Devedzic, V. & Damjanovic, V. (2004). Converting UML to OWL ontologies. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters* (pp. 488–489). ACM.

Guarino, N. & Welty, C.A. (2009). An overview of OntoClean. In *Handbook on Ontologies* (pp. 201–220). Berlin, Heidelberg: Springer.

Harris, S., Lamb, N. & Shadbolt, N. (2009). 4store: The design and implementation of a clustered RDF store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)* (pp. 94–109).

Jardine, D.A. (1977). The ANSI/SPARC DBMS model. In *Proceedings of the Second Share Working Conference on Data Base Management Systems*, Montreal, Canada, April 26–30, 1976. Elsevier Science Inc.

Kifer, M. & Lausen, G. (1989). F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *ACM SIGMOD Record* (Vol. 18, pp. 134–146). ACM.

Meier, W. (2003). eXist: An open source native XML database. In *Web, Web-Services, and Database Systems* (pp. 169–183). Berlin, Heidelberg: Springer.

Raimond, Y., Scott, T., Oliver, S., Sinclair, P. & Smethurst, M. (2010). Use of semantic web technologies on the BBC web sites. In *Linking Enterprise Data* (pp. 263–283). Springer.

Rector, A., Brandt, S., Drummond, N., Horridge, M., Pulestin, C. & Stevens, R. (2012). Engineering use cases for modular development of ontologies in OWL. *Applied Ontology*, *7*(2), 113–132.

Rector, A.L. (2003). Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proceedings of the 2nd International Conference on Knowledge Capture* (pp. 121–128). ACM.

Sirin, E. (2010). Data validation with OWL integrity constraints. In *Web Reasoning and Rule Systems* (pp. 18–22). Berlin, Heidelberg: Springer.

Tao, J., Sirin, E., Bao, J. & McGuinness, D.L. (2010). Integrity constraints in OWL. In *AAAI*.

Uschold, M. (2008). Ontology-driven information systems: Past, present and future. In *Proceedings of the 2008 Conference on Formal Ontology in Information Systems: Proceedings of the Fifth International Conference (FOIS 2008)* (pp. 3–18). IOS Press.

Uschold, M. & Gruninger, M. (1996). Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, *11*(02), 93–136.

Völker, J., Vrandečić, D. & Sure, Y. (2005). Automatic evaluation of ontologies (AEON). In *The Semantic Web–ISWC 2005* (pp. 716–731). Berlin, Heidelberg: Springer.

Welty, C., Mahindru, R. & Chu-Carroll, J. (2004). Evaluating ontology cleaning. In *AAAI* (pp. 311–316).