Ontology Evolution: Not the Same as Schema Evolution

Natalya F. Noy[1] and Michel Klein[2]

[1]Stanford Medical Informatics
Stanford University
Stanford, CA, 94305
noy@smi.stanford.edu

[2]Vrije University Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
michel.klein@cs.vu.nl

Abstract

As ontology development becomes a more ubiquitous and collaborative process, ontology versioning and evolution becomes an important area of ontology research. The many similarities between database-schema evolution and ontology evolution will allow us to build on the extensive research in schema evolution. However, there are also important differences between database schemas and ontologies. The differences stem from different usage paradigms, the presence of explicit semantics, and different knowledge models. A lot of problems that existed only in theory in database research come to the forefront as practical problems in ontology evolution. These differences have important implications for the development of ontology-evolution frameworks: The traditional distinction between versioning and evolution is not applicable to ontologies. There are several dimensions along which compatibility between versions must be considered. The set of change operations for ontologies is difference. We must develop automatic techniques for finding similarities and differences between versions.

1 Ontologies Are Ready To Evolve

If we trace the evolution of ontology research in Computer Science, we can see that as the field grows and matures, the focus of research and the questions that the researchers address shifts as well. The progress naturally reflects the shift from more theoretical issues in ontology research to the issues associated with the use of ontologies in real-world large-scale applications.

The active research in ontologies started with defining what a **formal ontology** is and what **requirements** an ontology must satisfy (Gruber 1993). The term "ontology" came to refer to a wide range of formal representations from taxonomies and hierarchical terminology vocabularies to detailed logical theories describing a domain. Later, the research focus shifted to the development of **representation languages** for defining and exchanging ontologies. Knowledge Interchange Format (Genesereth and Fikes 1992) and the Open Knowledge Base Connectivity protocol (Chaudhri et al. 1998) are the most prominent results of the research in this direction. Having defined what ontologies are and having decided how to represent and exchange them, researchers were faced with the next challenge: The lack of a critical mass of reusable ontologies became (and still remains) a bottleneck to achieving the vision of widespread use and reuse of ontologies. Developing the **content** of the ontologies became high on the agenda. The top-level of the Cyc ontology (www.opencyc.org) became publicly available, the Ontolingua ontology library (ontolingua.stanford.edu) grew, and large domain-specific ontologies, such as EcoCyc (Karp et al. 1996), GALEN (Rector et al. 1994), and Gene Ontology (www.geneontology.org) were developed. With the appearance of a large number of small and large ontologies in overlapping domains, the issue of ontology **merging and alignment** came to the forefront of the ontology research. A number of tools for finding similarities and differences among ontologies in a semi-automated way have recently appeared (Klein 2001). Today, ontologies are becoming an integral part of many industrial and academic applications in the fields such as supporting semantics-based search, interoperability support, configuration support, constraint specification and validation, Semantic Web applications, and others (McGuinness 2001). With this widespread

use of ontologies come the problems that the database community faced many years ago when databases became an integral part of many applications: **evolution and versioning**.
The researchers in ontology evolution can undoubtedly learn a lot from the database-schema–evolution research. Schema-evolution research includes analysis of causes of change, effects of different operations on the data and frameworks for handling different versions coherently. And in theory, many issues in ontology evolution are exactly the same as the issues in schema evolution. In practice, however, there are significant differences between ontologies and database schemas from the point of view of evolution and versioning. The content and usage of ontologies are often more complex than that of database schemas. Ontologies turn some of the theoretical problems and opportunities of database-schema versioning into practical ones. We will discuss the differences in the usage and development paradigms, in the presence of explicit semantics, in the knowledge representation. We will then discuss the practical implications of these differences on the development of a framework for ontology evolution and versioning.

1.1 Causes of Ontology Change

Like database schemas, ontologies inevitably change over time. To examine the cause of changes, we will start with a popular definition of an ontology proposed by Gruber (Gruber 1993). Formally, an ontology is *an explicit specification of a conceptualization of a domain*. Therefore, changes to any of the three elements in the definition can cause changes in an ontology: (1) changes in the domain, (2) changes in conceptualization, or (3) changes in the explicit specification.
Changes in the domain are very common and their effects are comparable to changes in database schemas. Ventrone and Heiler (Ventrone and Heiler 1991) sketch several situations in which changes in the real world (domain evolution) require changes to a database model. For example, when two university departments with different administrative structures merge, the ontology describing this domain needs to evolve to reflect this change
Changes in conceptualization can result from a changing view of the world and from a change in usage perspective. Different tasks may imply different views on the domain and consequently a different conceptualization. For example, consider an ontology describing traffic connections in Amsterdam, which includes such concepts as roads, cycle tracks, canals, bridges, and so on. If we adapt the ontology to describe not only the bicycle perspective but also a water-transport perspective, the conceptualization of a bridge changes from a remedy for crossing a canal to a time consuming obstacle.
Finally, changes in the explicit specification occur when an ontology is translated from one knowledge-representation language to another. The languages differ not only in their syntax, but also (and more important) in their semantics and expressivity. Therefore, preserving the semantics of an ontology during translation is a non-trivial task (Corcho and Gómez-Pérez 2000).

1.2 The Knowledge Model

In the rest of the article we will refer to traditional elements of an ontology: classes, slots, slot restrictions, and instances (Chaudhri et al. 1998). *Classes* are collections of objects that have similar properties. Classes constitute a subclass–superclass hierarchy with multiple inheritance. *Slots* attached to a class describe attributes and properties of the class. *Slots* are usually first-class objects. That is, a slot can exist without being attached to a particular class. Slots can be transitive or symmetric, or have inverses. Each slot has a set of restrictions on its values, such as

cardinality and range. *Instances* are individual members of classes. We use the term *instance data* to refer to instances and their slot values.

2 Differences between Ontologies and Database Schemas

We start our discussion with differences between database schemas and ontologies in general. We then discuss different usage paradigms for database schemas and ontologies. The last group of differences addresses knowledge-representation issues. We discuss here only the differences that have direct implications on developing a framework for ontology evolution and versioning.

2.1 Ontologies are data, too

The main goal for schema evolution and versioning support in databases is to preserve the integrity of the *data* itself: How does the new schema affect the view of the old data? Will queries based on the old schema work with the new data? Can old data be viewed using the new schema?

The same issues are certainly valid for instance data in ontologies. We can view ontologies as "schemas for knowledge bases." Having defined classes and slots in the ontology, we populate the knowledge base with instance data.

However, there is a major second thrust in ontology evolution: Ontologies themselves are data to the extent to which database schemas have never been. Ontologies themselves (and not the instance data) are used as controlled vocabularies, to drive search, to provide navigation through large collections of documents, to provide organization and configuration structure of Web sites. A result of a database query is usually a collection of instance data or references to text documents, whereas a result of an ontology query can include elements of the ontology itself (e.g., all subclasses of a particular class). Therefore, when considering ontology evolution, we must consider not only the effect of ontology changes on the way applications access instance data, but also the effect of these changes on queries for the ontology contents itself.

2.2 Ontologies themselves incorporate semantics

Database schemas and catalogs often do not provide explicit semantics for their data. Either the semantics has never been specified, or the semantics were specified explicitly at database-design time, but the specification has not become part of database specification and is not available anymore. Therefore, with databases, we need specific protocols for resolving conflicting restrictions when the schema changes. These protocols are usually part of a schema-evolution framework (Banerjee et al. 1987). Ontologies, however, are logical systems that themselves incorporate semantics. Formal semantics of knowledge-representation systems allow us to interpret ontology definitions as a set of logical axioms. We can often leave it to the ontology itself to resolve inconsistencies and do not need to do anything about them in the evolution framework. For example, if a change in an ontology results in incompatible restrictions on a slot, it simply means that we have a class that will not have any instances (is "unsatisfiable"). If an ontology language based on Description Logics (DL) is used to represent the ontology (e.g., OIL (Fensel et al. 2000) and DAML+OIL (www.daml.org/language; Hendler and McGuinness 2000)), then we can use DL reasoners to re-classify changed concepts based on their new definitions.

2.3 Ontologies are more often reused

A database schema defines the structure of a specific database and other databases and schemas do not usually directly reuse or extend existing schemas. The schema is part of an integrated system and is rarely used apart from it. The situation with ontologies is exactly the opposite: Ontologies often reuse and extend other ontologies, and they are not bound to a specific system. Therefore, a change in one ontology affects all the other ontologies that reuse it, and, consequently, the data and applications that are based on these ontologies. Even seemingly monotonic changes, such as additions of new concepts to an ontology, can have adverse effects on the other ontologies that reuse it. If we add a concept that already exists in the reusing ontology, no logical conflicts arise, but the reusing ontology contains two representations of the same concept. We will need to specify an equivalence statement to reflect this fact.

2.4 Ontologies are de-centralized by nature

Traditionally, database schema development and update is a centralized process: Developers of the original schema (or employees of the same organization) usually make the changes and maintain the schema. At the very least, database-schema developers usually know which databases use their schema. By nature, ontology development (and, therefore, evolution) is a much more de-centralized and collaborative process. As a result, there is no centralized control over who uses a particular ontology. It is much more difficult (if not impossible) to enforce or synchronize updates: If we do not know who the users of our ontology are, we cannot inform them about the updates and cannot assume that they will find out themselves. Lack of centralized and synchronized control also makes it difficult (and often impossible) to trace the sequence of operations that transformed one version of an ontology into another. Recently, ontologies have become a cornerstone of the Semantic Web (Berners-Lee et al. 2001), which has the model of distributed, reusable, and extendable ontologies at its core. The envisioned huge scale of the Semantic Web and even more de-centralization in ontology development and maintenance greatly exacerbate the problem: In today's Web, we can neither know who uses an ontology that we maintain or how many users there are, nor prevent or require others to use a particular ontology.

2.5 Ontology data models are richer

An inherent part of any schema-evolution methodology is a detailed consideration of the effects of each change operation on the data: What happens if a new superclass is added to a class in an object-oriented database, if an instance variable is removed from a class, if a domain of an instance variable is changed, and so on. The number of representation primitives in many ontologies is much larger than in a typical database schema. For example, many ontology languages and systems allow the specification of cardinality constraints, inverse properties, transitive properties, disjoint classes, and so on. Some languages (e.g., DAML+OIL) add primitives to define new classes as unions or intersections of other classes, as an enumeration of its members, as a set of objects satisfying a particular restriction. Therefore, any detailed treatment of ontology changes must include a much more extensive set of possible operations.

2.6 Classes and instances can be the same

Databases make a clear distinction between the schema and the instance data. In many rich knowledge-representation systems it is hard to distinguish where an ontology ends and instances begin. The use of metaclasses—classes which have other classes as their instances (Chaudhri et

al. 1998)—in many systems (e.g., Protégé (Noy et al. 2000), Ontolingua, RDFS (Brickley and Guha 1999)) blurs or erases completely the distinction between classes and instances. . In set-theoretic terms, metaclasses are sets whose elements are themselves sets. This means that "being an instance" and "being a class" is actually just a *role* for a concept. For example, the "Lonely Planet for Amsterdam" is a specific instance of the class "Travel guides" in a bookstore; at the same time, however, it is a class of which the individual copies of the book are instances. Therefore, analysis of schema-change operations, which considers only effects on instance data, is not directly applicable to ontologies.

3 Implications for Evolution and Versioning of Ontologies

The differences between ontologies and database schemas that we have just outlined, have direct practical implications for any methodology for ontology evolution and versioning. We discuss the following implications in the rest of this section:

(1) The traditional distinction between versioning and evolution is not applicable to ontologies.
(2) Defining *what* constitutes compatibility between different versions becomes a more salient issue since there are several dimensions to compatibility (e.g., preservation of instance data, preservation of answers to ontology queries, consequence preservation, etc.).
(3) The set of change operations that we must consider in classifying effects of ontology changes is much wider. In addition, we must consider the effects of these operations along different dimensions of compatibility
(4) We need techniques for determining compatibility between different versions even if we do not have a trace of the changes that led from one version to another.

3.1 Ontology Versioning and Evolution is Change Management

Database researchers distinguish between *schema evolution* and *schema versioning* (Roddick 1995). Schema evolution is the ability to change a schema of a populated database without loss of data (i.e., providing access to both old and new data through the new schema). Schema versioning is the ability to access all the data (both old and new) through different version interfaces.

For ontologies, however, we cannot distinguish between evolution, which allows access to all data only through the newest schema, and versioning, which allows access to data through different versions of the schema. Multiple versions of the same ontology are bound to exist and must be supported. Not knowing how an ontology is being reused means not being able to "force" the reusing ontologies and applications to switch to a new version. Ideally, developers should maintain not only the different versions of an ontology, but also some information on how the versions differ and whether or not they are compatible with one another. For example, the ontology-versioning mechanism in SHOE (Heflin and Hendler 2000) enables developers to declare whether or not the new version is backward-compatible with an old version (that is, applications and agents can use the new ontology in place of the old one). However, some applications may continue to use the old versions and upgrade at their own pace (or not at all).

The management of changes is therefore the key issue in the support for evolving ontologies. Hence, we will combine ontology evolution and versioning into a single concept defined as *the ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology*. This ability consists of methods to distinguish and recognize versions, specifications of relationships between versions, update and change procedures for ontologies,

and access mechanisms that combine different versions of an ontology and the corresponding data. We use the term "ontology evolution" for this concept through the rest of this article.

3.2 Compatibility of Ontologies Has Several Dimensions

In order to determine which changes to an ontology are backward-compatible, we need to determine what compatibility means. In databases, backwards-compatibility usually means the ability to access all of the old data through the new schema. In other words, no instance data is lost as a result of the change.

For ontologies, query results can include not only instance data but also elements of the ontology itself. Therefore, we cannot express compatibility only in terms of preservation of instance data. Consider a situation in which a new class is added to an ontology as a subclass of an existing class. This change has no effect on instance data and will not change or invalidate answers to existing queries that return only instance data. However, if queries are about the *ontology itself* (e.g., a list of subclasses of a specific class), the answers to existing queries change. This issue becomes even more complicated with ontology languages that support automatic classification (e.g., DAML+OIL): When a class is added to an ontology, a reasoner can re-classify existing concepts and instances, possibly invalidating existing data or applications.

Here are some of the dimensions that we must consider when determining whether a new version of an ontology is compatible with the old one:

- Instance-data preservation—no data is lost in transformation from the old version to the new one.
- Ontology preservation—a query result obtained using the new version is a superset of the result of the same query obtained using the old version.
- Consequence preservation—if an ontology is treated as a set of axioms, all the facts that could be inferred from the old version can still be inferred from the new version.

When we characterize the effects of change operations we need to take these (and, possibly, other) dimensions into account.

3.3 Ontology-Change Operations and Effects

The set of possible change operations for ontologies is larger than the traditional sets of database schema-change operations (Banerjee et al. 1987). There are two causes of the differences between these two sets. The first cause is the richer knowledge model for ontologies: We must add operations that deal with changes in slot restrictions, with slot attachment, and so on. The second cause is the use of *composite operations* which few researchers in the schema-evolution community have addressed (with the notable exception of Lerner (Lerner 2000)). Consider for example a change in the domain of a slot from a class to its superclass. Our model for traffic connections in Amsterdam may have used a slot `speed-limit` only for roads. To change the domain of the `speed-limit` slot to include thoroughfares (both roads and canals), we need to "move" the slot up the class hierarchy (imposing a speed limit for boats as well). If we treat this operation as a sequence of two operations, removing the slot from the `Road` class and then adding it to the `Thoroughfares` class, we would have to delete all the values of the `speed-limit` slot for all instances of `Road` after the first operation. However, after the second operation, all instances of `Road` can have the `speed-limit` slot again. The composite effect of the two operations does not violate the integrity of the instance data, whereas one of the operations does. Therefore, the algebra of ontology-change operations must include these composite operations since their compound effect on schema evolution (1) is predictable and (2)

can belong to a completely different class of operations than each of the simple operations that constitute it.

In Table 1 we sketch a set of possible ontology-change operations based on the knowledge model we described in Section 1.2 and discuss their effects. More specifically, we consider the effects with respect to the instance-data–preservation dimension, i.e. we consider whether instance data can still be accessed through the changed ontology. We classify the operations effects as:

- Information-preserving changes—no instance data is lost ("+" in the table)
- Translatable changes—no instance data is lost if a part of the data is translated into a new form ("~" in the table). We illustrate some translatable changes in Figure 1.
- Information-loss changes—it cannot be guaranteed that no instance data is lost ("–" in the table).

We also assume that there is no automatic classification.

3.4 There are Two Modes of Evolution

Characterizing effects of specific changes on compatibility between versions of an ontology is important. However, because of the extremely distributed nature of ontologies, we must also account for the fact that we will not always have the trace of changes that led from one version to another. Therefore, we distinguish two modes of ontology evolution: *traced* and *untraced* evolution. Traced evolution largely parallels schema-evolution where we treat the evolution as a series of changes in the ontology. After each operation that changes the ontology (e.g., add or delete a class, attach a slot to a class, change restrictions on slots, etc.), we consider the effects on the instance data and related ontologies, depending on the dimension of compatibility we use. The resulting effect is determined by the combination of change operations.

With the untraced evolution, all we have are two versions of an ontology and no knowledge of the steps that led from one version to another. We will need to find the differences between the two versions in an automated or semi-automated way. The problem of finding the differences between (versions of) ontologies is in fact very close to the problem of ontology merging. In both cases, we have two overlapping ontologies and we need to determine a mapping between their elements. When we are merging ontologies, we concentrate on *similarities*, whereas in evolution we need to highlight the *differences*, which can be a complementary process. In addition, in the case of ontology evolution we need to make much more "liberal" assumptions in determining which concepts are the same. For example, if we are merging two ontologies that came from independent sources, we cannot assume that two classes named `University` in the two sources refer to the same concept: One could refer to the organizational structure of a university and the other to a university campus. However, if two different versions of the same ontology both contain a class named `University`, it is much more likely that these classes do indeed refer to the same concept. We can reuse many of the heuristics and algorithms (with lower thresholds) that ontology-merging tools use to develop semi-automated interactive ontology-evolution tools for untraced ontology evolution. PROMPT (Noy and Musen 2000) is one example of an ontology-merging tool that analyzes the structure of the ontology, classes and relations among them, as well as user actions, to present suggestions for possible merge to the user. The ontology-merging research also provides analysis of possible ways to translate between ontologies. OntoMorph (Chalupsky 2000) provides a powerful rule language to represent complex transformations from one ontology to another and an engine for applying these transformations. The ONION ontology-articulation approach (Mitra et al. 2000) is based on

Operation	Effect	Comment on effect
Create class C	+	No data is lost
Delete a class C	–	Instances of class C have a less specific type (they have become instances of the superclass of C)
Create slot S	+	No data is lost
Delete a slot S	–	The values of slot S for all instances are lost
Attach a slot S to a class C	+	No data is lost
Remove a slot S from a class C	–	The values of slot S for instances of C are lost
Add a subclass–superclass link between a subclass SubC and a superclass SuperC	+	SubC has new slots inherited from SuperC—in most cases, equivalent to adding slots[1]
Remove a subclass–superclass link between a subclass SubC and a superclass SuperC	–	SubC no longer has the slots that it inherited from SuperC. The values for these slots for instances of SubC are lost
Re-classify an instance I as a class	+	No data is lost
Re-classify a class C as an instance	_	Instances of C are less specifically typed
Declare classes C_1 and C_2 as disjoint	_	Instances that belonged to both C_1 and C_2 are invalid
Define a slot S as transitive or symmetric	_	Slot values for S that violated the transitivity or symmetry property are invalid
Move a slot S from a subclass SubC to a superclass SuperC (“move a slot up the hierarchy”)	+	Class SubC still inherits slot S (and the instances preserve all the values for slot S)
Move a slot S from a superclass SuperC to a subclass SubC (“move a slot down in the hierarchy”)	–	Class SuperC no longer has slot S. The values for slot S for instances of SuperC are lost
Move a slot S from a class C_1 to a referenced class C_2 (see Figure 1a)	~	No data is lost if the values of the slot are moved
Encapsulate a set of slots into a new class (see Figure 1b)	~	No data is lost if the values of the slot are moved
Change a superclass of a class C to a class higher in the hierarchy (“move a class up the class hierarchy”)	–	C no longer has the slots that it inherited from its direct superclass. The values for these slots for instances of C are lost
Change a superclass of a class C to a class lower in the hierarchy (“move a class down in the class hierarchy”)	+	C has possibly inherited additional slots. No data is lost
“Widen” a restriction for a slot S (e.g., increase the number of allowed values, decrease the number of required values, add a class to the range or replace an existing class in the range with its superclass, etc.)	+	All the existing slot values are still valid
“Narrow” a restriction for a slot S (e.g., decrease the number of allowed values, increase the number of required values, remove a class from the range or replace it with a subclass, etc.)	_	Slot values that violated the narrower restrictions are invalid
Merge classes: the superclasses, subclasses, and slots of the merged class are the union of the superclasses, subclasses, and slots of the original	~	No data is lost if values of slots are moved. However, see comment at “Adding subclass link”
Split a class in several classes: the operation can specify which of the new classes the instances of the old class belong to based on a slot value	~	No data is lost if values of slots are moved

Table 1. Ontology-change operations and their effects on instance data.

[1] If slot restrictions inherited from SuperC are incompatible with restrictions for the same slot that already existed for SubC, then slot values may become invalid (and the operation would be a data-loss operation)

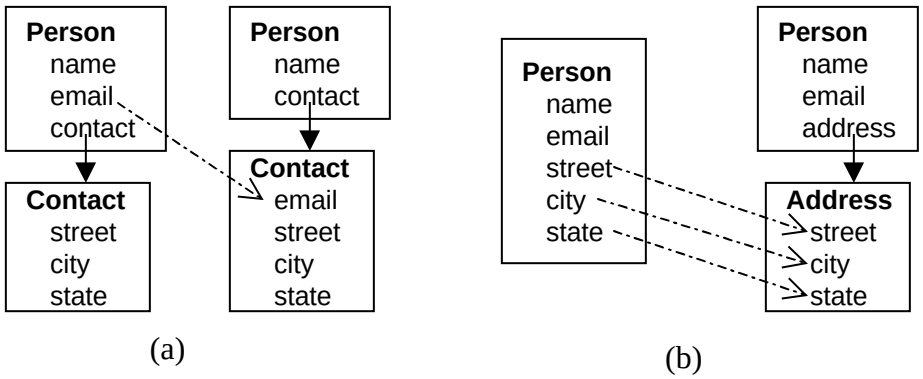

Figure 1. Composite slot operations: (a) Move a slot S from a class C_1 to a referenced C_2 class: The slot email is moved to the Contact class; (b) Encapsulate a set of slots into a new class. Slots street, city, and state are encapsulated into a class Address.

specifying "semantic bridges" between ontologies. In ONION, the ontologies remain separate (as we would want to do if we keep the old versions of ontologies available) and the articulation rules provide the mapping between ontologies.

3.5 There are Two Modes of Evolution

Characterizing effects of specific changes on compatibility between versions of an ontology is important. However, because of the extremely distributed nature of ontologies, we must also account for the fact that we will not always have the trace of changes that led from one version to another. Therefore, we distinguish two modes of ontology evolution: *traced* and *untraced* evolution. Traced evolution largely parallels schema-evolution where we treat the evolution as a series of changes in the ontology. After each operation that changes the ontology (e.g., add or delete a class, attach a slot to a class, change restrictions on slots, etc.), we consider the effects on the instance data and related ontologies, depending on the dimension of compatibility we use. The resulting effect is determined by the combination of change operations.

With the untraced evolution, all we have are two versions of an ontology and no knowledge of the steps that led from one version to another. We will need to find the differences between the two versions in an automated or semi-automated way. The problem of finding the differences between (versions of) ontologies is in fact very close to the problem of ontology merging. In both cases, we have two overlapping ontologies and we need to determine a mapping between their elements. When we are merging ontologies, we concentrate on *similarities*, whereas in evolution we need to highlight the *differences*, which can be a complementary process. In addition, in the case of ontology evolution we need to make much more "liberal" assumptions in determining which concepts are the same. For example, if we are merging two ontologies that came from independent sources, we cannot assume that two classes named `University` in the two sources refer to the same concept: One could refer to the organizational structure of a university and the other to a university campus. However, if two different versions of the same ontology both contain a class named `University`, it is much more likely that these classes do indeed refer to the same concept. We can reuse many of the heuristics and algorithms (with lower thresholds) that ontology-merging tools use to develop semi-automated interactive ontology-evolution tools for untraced ontology evolution. PROMPT (Noy and Musen 2000) is one example of an ontology-merging tool that analyzes the structure of the ontology, classes and relations among them, as well as user actions, to present suggestions for possible merge to the

user. The ontology-merging research also provides analysis of possible ways to translate between ontologies. OntoMorph (Chalupsky 2000) provides a powerful rule language to represent complex transformations from one ontology to another and an engine for applying these transformations. The ONION ontology-articulation approach (Mitra et al. 2000) is based on specifying "semantic bridges" between ontologies. In ONION, the ontologies remain separate (as we would want to do if we keep the old versions of ontologies available) and the articulation rules provide the mapping between ontologies.

4 Conclusions and the Next Steps

With ontologies becoming an integral part of many industrial and academic applications, support for ontology evolution and versioning is the next logical step in the ontology research. This research can undoubtedly benefit from the many years of research in database-schema evolution, including analysis of change effects, frameworks for handling different versions, interfaces to different versions, and so on. However, if we compare ontologies and database schemas from the point of view of evolution, we will find important differences as well. Although some of these issues do not form theoretical distinction between schema evolution and ontology evolution, they constitute a substantial difference in practice, and have significant practical implications on the way ontology evolution and versioning can be handled. We need to maintain all versions of an ontology, to define different dimensions of compatibility depending on what must be preserved, to be able to find differences between versions without a trace of how one version evolved into the other, and to define a set of ontology-change operations and consider the effects of these changes. The research in ontology evolution is in its very early stages. Our analysis of the problems could be a starting point for addressing various open issues in ontology evolution. We analyzed the effects of ontology-change operations very briefly and only with respect to the instance-data–preservation dimension. A much more detailed analysis of these operations and their effects along all dimensions of compatibility is needed. This analysis should go alongside with a more precise specification of the necessary transformations on data or ontologies for translatable changes. Other open issues include identification of change, deprecation of outdated ontologies, algorithms for finding differences between versions automatically. We hope that our analysis will help direct the research and demonstrate that ontology evolution is even more complicated than schema evolution.

Acknowledgments

We are very grateful to Dieter Fensel, Mark Musen, and Gio Wiederhold for their feedback on the earlier versions of the paper. This research was supported in part by the contract from the National Cancer Institute.

References

Banerjee, J., Kim, W., Kim, H.-J. and Korth, H.F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: *Proceedings of the SIGMOD Conference*.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American* **May**.

Brickley, D. and Guha, R.V. (1999). Resource Description Framework (RDF) Schema Specification. Proposed Recommendation, World Wide Web Consortium**:** http://www.w3.org/TR/PR-rdf-schema.

Chalupsky, H. (2000). OntoMorph: A translation system for symbolic knowledge. *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000).* A. G. Cohn, F. Giunchiglia and B. Selman, editors. San Francisco, CA, Morgan Kaufmann Publishers.

Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, AAAI Press/The MIT Press.

Corcho, O. and Gómez-Pérez, A. (2000). A roadmap for ontology specification languages. In: *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000)*, Juan-les-Pins, France, Springer.

Fensel, D., Horrocks, I., Harmelen, F.V., Decker, S., Erdmann, M. and Klein, M. (2000). OIL in a Nutshell. In: *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000)*, Juan-les-Pins, France, Springer.

Genesereth, M.R. and Fikes, R.E. (1992). Knowledge Interchange Format, Version 0.3, Reference Manual Logic-92-1, Knowledge Systems Laboratory, Stanford University.

Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition* **5**: 199-220.

Heflin, J. and Hendler, J. (2000). Dynamic ontologies on the Web. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX.

Hendler, J. and McGuinness, D.L. (2000). The DARPA Agent Markup Language. *IEEE Intelligent Systems* **16**(6): 67-73.

Karp, P.D., Riley, M., Paley, S.M. and Pelligrini-Toole, A. (1996). EcoCyc: Encyclopedia of *E. coli* Genes and Metabolism. *Nucleic Acids Research* **24**(1): 32-40.

Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In: *Proceedings of the IJCAI-20001 Workshop on Ontologies and Information Sharing*, Seattle, WA.

Lerner, B.S. (2000). A Model for Compound Type Changes Encountered in Schema Evolution. *ACM Transactions on Database Systems* **25**(1): 83-127.

McGuinness, D.L. (2001). Ontologies Come of Age. *The Semantic Web: Why, What, and How*. D. Fensel, J. Hendler, H. Lieberman and W. Wahlster, editors, MIT Press.

Mitra, P., Wiederhold, G. and Kersten, M. (2000). A Graph-Oriented Model for Articulation of Ontology Interdependencies. In: *Proceedings of the Proceedings Conference on Extending Database Technology 2000 (EDBT'2000)*, Konstanz, Germany.

Noy, N.F., Fergerson, R.W. and Musen, M.A. (2000). The knowledge model of Protégé-2000: combining interoperability and flexibility. In: *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000)*, Juan-les-Pins, France, Springer-Verlag.

Noy, N.F. and Musen, M.A. (2000). PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX.

Rector, A., Gangemi, A., Galeazzi, E., Glowinski, A. and Rossi-Mori, A. (1994). The GALEN CORE Model Schemata for Anatomy: Towards a Re-Usable Application-Independent Model of Medical Concepts. In: *Proceedings of the Medical Informatics Europe, MIE'94*.

Roddick, J.F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology* **37**(7): 383-393.

Ventrone, V. and Heiler, S. (1991). Semantic heterogeneity as a result of domain evolution. *SIGMOD Record (ACM Special Interest Group on Management of Data)* **20**(4): 16-20.