Quality of Ontologies in Interoperating Information Systems

Robert M. Colomb

Technical Report 18/02 ISIB-CNR
Padova, Italy, November, 2002

National Research Council

Institute of Biomedical Engineering

ISIB-CNR

Corso Stati Uniti, 4

35127 Padova

Robert M. Colomb

School of Information Technology and Electrical Engineering
The University of Queensland
Queensland 4072 Australia
colomb@itee.uq.edu.au

Work performed partly while visiting LADSEB-CNR; Corso Stati Uniti, 4; Padova, Italia,

Abstract

The focus of this paper is on quality of ontologies as they relate to interoperating information systems. Quality is not a property of something but a judgment, so must be relative to some purpose, and generally involves recognition of design tradeoffs. Ontologies used for information systems interoperability have much in common with classification systems in information science, knowledge based systems, and programming languages, and inherit quality characteristics from each of these older areas. Factors peculiar to the new field lead to some additional characteristics relevant to quality, some of which are more profitably considered quality aspects not of the ontology as such, but of the environment through which the ontology is made available to its users. Suggestions are presented as to how to use these factors in producing quality ontologies.

Quality of Ontologies in Interoperating Information Systems

Robert M. Colomb

School of Information Technology and Electrical Engineering

The University of Queensland

Queensland 4072 Australia

colomb@itee.uq.edu.au

Work performed partly while visiting LADSEB-CNR; Corso Stati Uniti, 4; Padova, Italia

Short Title: Quality of Ontology

Index terms: information systems interoperation, ontology, quality

Abstract

The focus of this paper is on quality of ontologies as they relate to interoperating information systems. Quality is not a property of something but a judgment, so must be relative to some purpose, and generally involves recognition of design tradeoffs. Ontologies used for information systems interoperability have much in common with classification systems in information science, knowledge based systems, and programming languages, and inherit quality characteristics from each of these older areas. Factors peculiar to the new field lead to some additional characteristics relevant to quality, some of which are more profitably considered quality aspects not of the ontology as such, but of the environment through which the ontology is made available to its users. Suggestions are presented as to how to use these factors in producing quality ontologies.

Introduction – information systems quality and its contribution of ontology.

The focus of this paper is on quality of ontologies as they relate to interoperating information systems. Since part of our argument is that use of high-quality ontologies increases the quality of interoperable information systems we are also concerned with quality of information systems more generally.

Let us start with some very general considerations. Quality is not a property of something but a judgment, so must be relative to some purpose. Information systems generally have many stakeholders, who have quite different views on quality.

Management wants a system that delivers a maximum benefit at minimum cost, where benefit is defined in terms of the operational effectiveness of the organization, not in terms of the system itself or even its functionality.

End users also want a system that delivers a maximum benefit at minimum cost, but benefit is defined in terms of enhancement of their individual and small group effectiveness, and cost generally in terms of ease of use. For very powerful systems, ease of use might take into account the skill increase expected from a quantum of training.

The engineers who build and maintain the system see the management and end user quality judgments as objective functions on which to make design tradeoffs. The engineering design tradeoffs are the first place we begin to see the sorts of technical issues with which this paper is concerned.

Very few systems are based on radically new designs. The usual approach is to begin with a general framework having a number of parameters which are often correlated. The management and end-user quality judgments are represented in terms of design principles or rules of thumb which assist the engineers in setting the parameters and managing their interaction. An important element of quality at this level is the tools and techniques which assist the engineers. Such tools and techniques tend to improve quality partly because they make it easier for the engineers to work, thereby reducing costs; or increase the engineers' capability, thereby increasing attainable functionality.

We leave it to others to explore the issues of quality in information systems and in conceptual modeling generally. Our focus, ontology design, is a complex of tools and techniques which come into play when the engineers need to make information systems interoperate. This can either involve modifying existing applications or creating new ones, and can involve interoperation among systems implemented in the same organization (a large hospital, say) or in autonomous organizations (the semantic web).

Designing an ontology involves selecting a universe of discourse. More particularly, an ontology is a catalog of the contents of the universe visible in the interoperation of a group of existing or potential applications. Ontology is related to conceptual modeling in that the tasks performed and some of the tools used are similar. Ontology differs from conceptual modeling in that it is much more public and tends to be less specific. More public in that it records the agreements taken by the stakeholders which define the universe of interoperation, and therefore must be referred to by the designers of all the individual interoperating applications and integrated into the individual conceptual models. Less specific in that the ontology tends to abstract away from many implementation details through which the interoperations are produced. For example, the Electronic Data Interchange (EDI) standards record the agreements as to the types, semantics and permissible content of a wide variety of business messages including purchase orders and invoices. The standard is silent on how a purchase order is stored in the database of a complying system, and on how the purchase order is related to the subsequent invoice through perhaps a series of intermediate steps invisible to the customer.

Ontologies come at several levels of generality. The most specific are those supporting a particular group of applications, such as SNOMED[1] (medical conditions and interventions) or that supporting a particular business-to-business exchange like CorProcure[2]. SNOMED has 11 general categories of object, including anatomy, pharmaceuticals, and living organisms, with a total of more than 150,000 particulars such as *heart*, *aspirin* and the bacterium *e. coli*. CorProcure has a number of general categories of product such as office goods, and a large range of particular named products called *stock keeping units*, allowing demand to be aggregated and volume bids to be placed for sales to multiple customers at the same time.

More general ontologies support many specific groups of applications in a broad context, such as the EDI standards X.11 and UN EDIFACT, or a hypothetical ontology based on the intellectual property law and regulations of the European Union. The EDI standards specify the types, semantics and permissible content of business documents such as purchase orders or invoices. An ontology of intellectual property law would include a classification of various types of intellectual property, various types of stakeholders, and various types of permitted and forbidden uses of the types of property by the types of stakeholders.

Most general are the universal ontologies like Cyc[3] or the IEEE Suggested Upper Merged Ontology (SUMO)[4] which are intended to be application independent catalogs of things in the world, and the formal upper ontologies like OntoClean (Welty and Guarino 2001) or Bunge-Wand-Weber (BWW) (Weber 1997), which are neutral with respect to content, focusing on the forms used to represent the more specific ontologies.

Ontologies contribute to the quality of interoperating information systems partly by increasing capability. A single repository for general agreements among the interoperating parties is much less complex than a web of inter-party agreements, making feasible interoperation on a much greater scale. They contribute also by reducing costs. Storing the common elements in one place greatly reduces redundancy and consequently the effort needed to develop and maintain the interoperation.

---

[1] www.snomed.org/

[2] http://www.corprocure.com/

[3] http://www.opencyc.org/

[4] http://ontology.teknowledge.com/

We recognize that ontologies are used in a number of areas different from information systems interoperation, most particularly artificial intelligence and natural language applications. Our comments do not necessarily apply to ontologies used for these other purposes, at least not directly.

Quality of ontologies – preliminary

If quality is a judgment of an object against an intended use, then in order to evaluate the quality of a particular ontology we need to test alternatives against the requirements of the application as seen by the various stakeholders. A framework for doing this for conceptual modeling generally was published by Lindland et al. (1994), in which quality was to be assessed against three linguistic dimensions: syntactic, semantic and pragmatic. These three dimensions answer respectively the questions:

- Is the model syntactically correct?
- Does the model cover the domain of interest?
- Is the model comprehensible by the user?

The syntactic dimension is on its face straightforward. If the model contains syntactic errors, then the CASE tool supporting the language used in model should be able to point these errors out to the analyst. However, a deeper issue is how complex a syntax should be used. There are, after all, many modeling languages. In CASE-supported ontologies, the syntactic issue is richness and complexity of syntax rather than correctness.

Semantic quality is how well the ontology reflects its universe of discourse, while pragmatic quality is how easy the ontology is to understand. Assessment along these dimensions is in practice an assessment of fitness for purpose. The closer an ontology reflects its universe of discourse, the larger and more complex it will tend to be and therefore the more difficult to understand. There are comparable tradeoffs between syntax and semantics, and between syntax and pragmatics.

Lindland et al. recogise this issue with their consideration of *feasibility*. Syntax must be rich enough, semantics must be complete enough and pragmatics must be understandable enough. They say “Adding the notion of feasibility lets you trade off between benefits and drawbacks”.

The Lindland et al. framework gives a way to design usability tests. The argument of the present paper is that we also need to be able to describe the differences among the alternatives in order to be able to determine what characteristic of the alternative ontologies lead to particular quality judgments. In other words, we need a technical vocabulary to describe the ontological design space.

The technical aspects of quality with which we are concerned in this paper therefore have to do with the general frameworks and correlated parameters of ontologies and the design principles used by the engineers to make their tradeoff decisions. We are further concerned with actual or possible tools available to the engineer to increase capability or reduce costs. In order to see these frameworks, principles and tools, we need to distinguish a vocabulary to talk *about* ontologies from a vocabulary to *represent* ontologies. Every ontology has a representation language, but part of the task of studying quality is to develop a vocabulary about them, which requires some thought.

One way to develop vocabulary in a new field is to identify and adapt relevant vocabulary from adjacent more established fields. Neighbors of ontology include the branch of information science concerned with the development of classification systems, knowledge-based systems, and programming languages. In all cases, it is essential that the system being evaluated be comprehensible to the humans who are building it, if not to the end users, so that cognitive considerations are important.

In the following, we collect a relevant vocabulary from the neighbouring disciplines, then consider some quality issues specific to ontologies. Finally, we look at how this design vocabulary can be used to help assess the quality of particular ontologies.

Lessons from information science

Classification *systems*

One important characteristic of ontologies is that they often include subsumption hierarchies (is-a networks). The analogous concept in information science is the classification system, where the network is based on the broader term/ narrower term relationship. Classification systems are used to organize information repositories, including libraries. If the repository consists of a collection of documents, then each document is attached to exactly one of the leaf nodes in the hierarchy, and thereby to the chain of broader terms associated with that leaf node. There is a very long history of engineering of such systems, with design principles given in such texts as Rowley (1992) and Colomb (2002).

A critical design parameter in a classification system is *specificity* – the average number of documents associated with a leaf node. This parameter is based on cognitive principles, namely the number of documents a user is prepared to sift through in the result of a query on the document repository. Once the specificity of the system has been established, the size of the collection determines another parameter, *size* – the number of leaf nodes. In an information retrieval system, the designers might determine that a user making a query with a single term would tolerate an average of 50 documents in a query result, so the system would have a specificity of 50. If there were 1 million documents in the repository, then there would have to be 20,000 descriptor terms, so the system would have a size of 20,000.

If the classification system is intended to be understood by humans, then the size of the system can be a problem. Large classification systems such as used in libraries, for example, have hundreds of thousands of leaf nodes. Humans are capable of comprehending only a small number of alternatives, generally well below 100. For this reason, classification systems are generally hierarchical. If we call the average number of subclasses in a class the *width* of the classification system, then the size of the system determines the *depth* of the system, or the average number of subsumption steps from the root to a leaf. In the library world, the Library of Congress classification system (LOC) and the Dewey Decimal Classification System (DDC) are of similar size, both intended to classify all books published in the past and foreseeable future. The LOC system is wider than the DDC. Many classes in the LOC have more than 20 subclasses, while in the DDC each class has at most 10 subclasses. The LOC is therefore shallower than the DDC. It has a depth of about 7 compared with more than 10 in the DDC.

So the design parameters for a classification system for a collection of documents of a given size include *specificity*, *size* of the classification system, *width* and *depth* of the hierarchy. Specificity and size are (anti-)correlated, as are width and depth. The driving parameters specificity and width are both derived from cognitive considerations, so that a system is of low quality if its specificity or width deviate too far from the cognitively most comfortable.

The cognitive consideration is loosely that smaller (result sets or menu choices) is better. So is a more specific or narrower system generally of higher quality than a less specific or wider system? The answer is not necessarily. There are two additional design principles introduced by Bates (1986) and discussed in Colomb (2002), the principles of *variety* and *uncertainty*, which affect the quality of the system through the choice of specificity and width.

Variety is a technical parameter, first introduced in cybernetics as Ashby's law of requisite variety, *a control system must have more states than the system it is controlling*. This principle is another way of saying smaller is better, and its application leads to more specific or wider systems. This principle might lead one to classify my occupation as *University Reader* in the Australian system, and my field as *Information Systems Interoperability*.

Uncertainty opposes variety. The principle of uncertainty derives from the cognitive fact that people, even trained indexers, are not very reliable at assigning documents to very specific classes. That is, two different people or even one person at different times will tend to assign different specific classes to a given document. The problem gets worse the more specific the classes or the wider the hierarchy. If my occupation were to be classified in the American system, would I be considered as an *Associate Professor* or a *Full Professor*? Would my field be classed as a subclass of *communication*, or *database systems*, or *Web technology*?

High quality systems therefore tend to be no more specific than they need to be. Most systems that would classify my occupation would have me as a *university teacher*, and my field as *information systems*. Large systems which are necessarily more specific than is comfortable for humans will be supported by indexes, coding manuals, and other tools to help overcome the results of the principle of uncertainty, thereby maintaining quality.

One would expect that these design principles would transfer fairly directly to the taxonomies used in ontology design. Specificity is often interpreted somewhat differently, though. In many cases the objects of interest are not individuals (documents in the information science case), but types, for example customers or products. An information system may have millions of instances of types without any problem with specificity. That multiplicity is as it were outside the design horizon.

Uncertainty is central to ontologies, since they are intended to be used by multiple parties. There are thus many opportunities for misunderstanding and unintended interpretations. One way to limit uncertainty is to describe an ontology using structural relationships among meta-properties. This adds an element of redundancy somewhat like the use of types in programming languages. The users of the ontology must not only agree on the meta-properties but also on the structural relationships underlying the terms in the ontology itself. The structural relationships can in many cases be described in axioms which give necessary and sufficient conditions for the ontological terms.

OntoClean (Welty and Guarino 2001) makes extensive use of this approach. They put forward meta-properties such as *identity* (the ability to identify instances) and *unity* (the ability to put together the parts of complex objects) and use them to guide the structural relationships used in subsumption networks. In particular, their approach gives a principled solution to the famous problem of Clyde the elephant. (Clyde is an elephant. Elephant is a species. Is-a is a transitive relationship. Therefore the fallacious conclusion Clyde is a species. The OntoClean approach prevents subsumption from being used in this case since the identity criteria for instances of elephant are different from the identity criteria for instances of species.)

Coherence

Uncertainty appears in classification systems as difficulty humans have in consistently assigning an instance to a class. The way the subclasses are defined can make a big difference to the difficulty of assigning individuals to them. Plato, in *The Phaedru*s, perhaps the first systematic presentation of classification as a knowledge representation technique, advises that the subclasses should divide the superclass "at the joints". In other words, the subclasses should be divided according to a systematic, easily agreed upon set of characteristics. This method appears in the information science literature as *theoretical analysis* (Rowley 1992, Colomb 2002).

One way to achieve coherent subdivision of a class is to define the subclasses formally using SQL views or defined concepts (see eg Donini et al. 1996 for terminology) in description logics. The set of subclasses should be disjoint and exhaustive (partition the superclass). Other ways include basing the subclasses on the set-instance or part-whole relationship. For example, one can classify material about computers by CPU type (defined concepts), or material about famous computers by the computer's name (Difference Engine, Enigma, Eniac, HAL – set/instance relationship), or material about computer systems by subsystem (cpu, monitor, disk, printer, modem – part/whole relationship). The alternative is a heterogeneous collection of names (primitive concepts in description logics), such as is found in many subclasses of Yahoo!. At one stage (in 2001), the *Computers and the Internet: WWW* class included as subclasses *Books* and *Java*, among others. The obvious question to ask is where would a book about Java be classed.

However, a coherent method of subclassification is not a guarantee of high quality. Coherence interacts with specificity. The different subclasses can differ very much in specificity. Since the width of the hierarchy is strongly constrained, the further decompositions of the more specific subclasses are much shallower than the more general, so that the system is unbalanced. Unbalanced systems can be undesirable because in a menu system it can take many more choices to reach some leaf nodes than others. Where a classification system is used as the basis of a coding system, the shallower branches leave code space unused. So in applications where balance is important, a high quality system may be subdivided on a less coherent basis, but will compensate by providing extra assistance to the people making the decisions classifying instances. Coherence and balance are anti-correlated design parameters.

It may be that in ontology work, the need for coherence is stronger than the need for balance. There is a good case to be made that the revision of WordNet according to the OntoClean method (Gangemi et al. 2002) improved the quality of WordNet[5] by regularizing the subsumption relationships using the meta-properties of their system.

Faceted systems

Uncertainty influences the quality of information science classification systems in yet another way – the consistency of subdivision among the different branches of the hierarchy. Suppose we have a classification system for Olympic athletic teams. First teams are classified by sport. Then each sport is subclassified by the country from which the team comes. The assignment of individuals to classes is easier if each sport is subclassified by the same set of countries. One benefit of this kind of approach is that we can easily rearrange the hierarchy, subdividing first by country then by sport, for example. The two independent subdivisions are called *semantic dimensions* or *facets* (Rowley 1992, Colomb 2002).

The cost of this faceted method is that there may be possibly many empty classes – Jamaica may have a bobsled team, but Egypt does not. In many cases, the predominance of empty classes makes the method infeasible – for example if we subdivide sport by type of event, the subdivisions of swimming are entirely inapplicable to those for equestrian events, and vice versa.

However, high quality systems intended to classify complex objects tend to be built using the faceted approach, for example the SNOMED[6] system for classifying medical conditions and interventions. This kind of issue appears in ontologies that support multiple inheritance. The quality of these might improve if note were taken of faceted design methods.

Lessons from knowledge-based systems

Quality in knowledge-based systems was addressed by Debenham (1989). What is meant by knowledge-based system in that work is a deductive database, that is a relational database with possibly recursive view definitions. The objects of design are called data, information and knowledge. *Data* is the individual atomic names that are stored in the database (the names of individuals in description logic terms). *Information* includes both the schemas defining the database relations (roles) and the tuples stored in the database (role membership assertions). *Knowledge* is the view definitions, which are expressed as Horn clauses (complex concepts).

There are a large number of specific design principles in the work, but they are all expressions of the fundamental principle of avoiding redundancy – each object in the universe of discourse is represented in the knowledge-based system in one place and one place only. This is an extension of the database design principle of normalization, where the schemas are designed to store various kinds of dependencies non-redundantly. The reason for normalization is that it avoids update anomalies – a change in a dependency can be recorded by a single update without introducing inconsistencies. The reason given for Debenham’s design principles is maintainability – a change in an object in the universe of discourse can be represented in the knowledge-based system by one possibly complex change in the data, information and/or knowledge.

For example, universities have degree programs, each of which has a set of rules. There will often be aspects of rules which apply to more general groups of students – say undergraduates versus postgraduates, coursework versus research degrees, degrees offered by different Faculties. If the common rule sets applying to each group of students is factored out of the rule definition, then a general change to say a research degree type of program will automatically propagate to each specific research degree.

The price paid for normalization is increased complexity of structure (more tables) and increased complexity of processing (more joins). Applications where performance is a quality issue often denormalize the most-used tables. Some database systems support view materialization, where a denormalized table is represented as a view whose population is computed eagerly and where the redundant updates are performed automatically. Similarly, application of Debenham’s design principles tends to

[5] http://www.cogsci.princeton.edu/~wn/online/

introduce additional complexity both of structure and processing. In the university rule example, the rules expressed in this way will be very difficult for a student to understand since the rules relevant to a particular student will be scattered through a number of modules. Quality will be enhanced if the student-rule interface is able to automatically incorporate all the modules into a single relevant story.

So a high-quality system will be designed in the awareness of redundancy, and where redundancy needs to be introduced into the design, it will be properly controlled. The complexity introduced will be compensated. In fact, the university rule example shows that we can view this principle as a call for the use of ontology in the design of knowledge-based systems (the different groups of students form an ontology) as well as a call for the avoidance of redundancy in ontologies.

Programming languages

Design of programming languages has a long history and a large literature. Some relevant design tradeoffs include expressiveness versus simplicity, orthogonality versus efficiency, and support for re-usability versus one-off cost.

Expressivity is an interesting issue. The minimum model for computation is extremely simple – a Turing machine is a CPU and memory with a few instructions. There is nothing that can't in principle be programmed with the minimum model. Programming languages with more facilities simply make it easier to do certain kinds of things, so expressivity does not refer to what you can say at all, but what you can say easily. Expressivity is very closely related to re-usability – the more than minimal features of a programming language are re-usable structures.

The cost of expressivity is complexity. The more re-usable structures there are in a programming language, the more difficult it is for the programmer to find the right one to use, and to learn to use it. Similarly to the variety/uncertainty tradeoff in information science, a high quality programming language will tend to have its complexity limited to a set of features useful in a class of applications, so that the benefit of learning is high.

For ontologies, this tradeoff casts light on the design issues around specificity discussed above. Addition of a subclass increases the cost of learning how to use it as well as the cost of acquiring and maintaining its instances. For example, an application supporting membership in a professional society may find that many of its members have an address that indicates that they work for the same organization. It would be possible to model the organizational affiliation in the address, but the model would be very complex (organizations often have many addresses, for example, and also can merge or split, while the member continues to work and receive post at the same address). So we have another reason to expect that a high-quality ontology would not be more specific than necessary. Nor would it have more other features than necessary.

Orthogonality in a programming language has to do with the interaction of its various features. Two features are orthogonal to the extent that they are not correlated. For example, a language may have a range of data types and may have a function constructor. If a function can return any type, then the function constructor and the data types are orthogonal. A language with orthogonal features is easier to use than one with correlated features, since the programmer does not have to learn the correlations. The cost of orthogonality is efficiency. It may be extremely difficult to implement certain types as values of functions, so that a program making use of these difficult-to-implement constructions may be extremely large or extremely slow, and thus unsatisfactory. So high-quality programming languages tend to have limited orthogonality, especially if they are more expressive (there are more features to interact). They will also tend to have optimizing implementations to preserve as much orthogonality as possible.

Orthogonality is related to the faceted classification systems discussed above. In ontology terms, these systems allow complex objects to be classified along several relatively simple dimensions rather than one complex hierarchy.

Reusability is related to expressivity, as described above. But a programming language can make it easier to re-use not only language features, but programs developed by a particular user community. This sort of facility is often called *abstract data types*, and in object-oriented languages is often implemented as the ability to associate methods with classes. The most easily re-used abstract data types generally have clear specifications, so that a programmer can understand what the method is intended to do without knowing anything about its implementation. Historically, the most successful types are mathematical and statistical function libraries. The relational algebra and is relatives in SQL are good examples, as are the various breeds of logic programming derived from particular subsets of the predicate calculus. In fact, the most successful abstract data types tend to be incorporated as increased expressiveness in programming languages.

Experience has shown that it is extremely difficult to develop easily used abstract data types, since it amounts to developing new mathematics. This is why the tradeoff for reusability is one-off cost. It is much cheaper to develop programs for a specific application than it is to develop libraries for use in many applications. A high-quality programming language/environment will provide support for libraries of abstract data types.

The whole point of ontology is reusability. An ontology is intended to be a collection of descriptions of object types which support an ecology of interoperating information systems. So a high-quality ontology environment will permit the re-use of as much as possible of the successful abstract data types associated with its classes. A high-quality class will tend to have methods associated with it implementing useful operations on its objects. A high quality ontology will support meta-properties which will insure that the methods of its classes are reliably applicable to instantiations. This is essentially the justification for the use of the OntoClean methodology of ontology design (Guarino and Welty 2002). The methodology permits the more extensive and reliable use of subsumption and abstract data types associated with the part-whole relationship, among others.

Ontology-specific quality issues

We have seen that ontologies used for information systems interoperability have much in common with classification systems in information science, knowledge based systems, and programming languages, and that they inherit quality characteristics from each of these older areas. The quality characteristics generally involve recognition of design tradeoffs, in particular the interaction of adequacy for human cognition (principle of uncertainty, ability to distinguish alternatives, ease of learning and so on), with technical factors (principle of variety, control of redundancy, implementability, reusability and so on).

Ontologies are of course a field in themselves. Key differentiating factors include

- They are independent of specific applications within a general context.
- They are often public, in the sense that they are independent of any particular organization deploying applications.
- They must integrate with a large variety of applications and interfaces, increasingly including natural language interfaces.

These special requirements lead to some additional quality issues, some of which are more profitably considered quality aspects not of the ontology as such, but of the environment through which the ontology is made available to its users.

The three requirements suggest that an ontology may be of poor quality if it is so tied to a particular natural language that it is difficult to integrate it with applications or interfaces using a different natural language. Similarly, it may be of poor quality if it is so tied to a particular programming language that it is difficult to integrate it with applications written in different programming languages. So language independence is a quality dimension, both natural language for content and programming language for representation. The cost of language independence is lack of subtlety of expression. There are many features of particular language which are useful to users of that language, and it can often be difficult to express features of a particular application in a standard language.

Of course, there is no *outside* of language. The content must be expressed in some language. So an ontology must take into account the range of natural languages with which it must interface, and tailor its content appropriately to a least common denominator. Hence the loss of subtlety. Similarly, the ontology must be represented somehow – there is no *outside* of representation. An ontology will therefore be represented in a standard representation system which is widely used among the applications intended to employ it, hence again at a cost of subtlety.

Furthermore, any ontology is intended to be used in a particular context, say a particular e-commerce exchange, or a medical records exchange system within a particular region, or for advice and bookings for independent travelers in a given group of countries. The objects and messages represented in the ontology and its associated applications will be interpreted in the light of what Searle (1995) calls *background* knowledge – that is customs, expectations, business practices and regulations which are not generally explicitly represented in either the ontology or the applications. Examples include be settlement deadlines for invoices and the consequences of missing them, or the regulations and practices regarding use of private data. In particular, an airline may publish a limit of 15 kilograms for checked baggage. Whether I need to worry about removing stuff from my 18-kilogram bag depends on the background of how strongly the airline enforces the weight limit at check-in.

Changes in the background associated with the context can make a significant difference to the applications involved (think of changes in taxation or privacy regimes). Although as with any application it is impossible to foresee all possible changes, there are certain kinds of changes which can be anticipated. An ontology will have a facility for representing relevant background parameters, thereby making predictable change easier. The cost is of course complexity, since the more background is represented, the bigger the system must be.

Finally, no matter how good the ontology is, in order for it to be useful it must be deployed in an environment where it can be effectively accessed both by human users (typically systems developers) and by the applications and agents operating in its context. Therefore, there are quality issues related to the ontology server and its operating environment. The tradeoff dimension is of course cost. The better the environment, the more it costs to build it.

For the human, the ontology environment needs to provide

- Glosses on the terminology, and links to background literature to assist the user in understanding the terms and their relationships.
- Search and browse facilities supported by indexes to that people can quickly and reliable find what they need.

For the applications and agents, there needs to be

- Hooks to relate terms from particular natural languages to the terms in the ontology.
- Facility for rewrite rules to express the terms in the ontology in a variety of implementation languages.

How all this helps with quality assessment

What this paper has done is to present a vocabulary to describe what are almost physical characteristics of ontologies in terms of engineering design tradeoffs. This vocabulary enables us to describe an ontology in terms of its location in a design space. Further, all of the engineering tradeoffs are apparent at the design stage, before the ontology is actually built. This vocabulary is summarized in Table 1. Concepts are presented in pairs. The left column is the concept in the pair commonly expressed as a desired feature. The right column is the correlated concept commonly thought of as the cost of the desired feature.

Table 1

Summary of engineering tradeoffs

Dimension	at a cost of	**Dimension**	
specificity	How much each term covers	size	Number of terms needed
width	Number of terms visible at once	depth	Number of levels of hierarchy
coherence	Ease of understanding	balance	Equality of depth for all branches
maintainability	Ease of change	complexity	
expressivity	What you can say	complexity	
orthogonality	Lack of interaction of features	efficiency	
re-usability		one-off cost	
language independence	From particular natural language	loss of subtlety	Some things easier to say in particular language
standard representation		loss of subtlety	Some concepts can be hard to express in standard language
representation of background		complexity	
richness of deployment environment		cost	

One way to use this vocabulary is to develop a series of prototypes which differ along one or two design dimensions, say balance, or maintainability, or specificity and width, or specificity and orthogonality. The prototypes can be ranked according to cost using the correlated cost dimensions. They then can be assessed for suitability using a method appropriate to the particular application and relevant stakeholders. Results of the suitability analysis can be factored in with cost assessments in order to decide on a high quality design.

Another way to use the vocabulary is to assess the feasibility of being able to meet the specifications of a proposed system. Often the specifications include characteristics which are constrained by the quasi-physical laws to be correlated. It may happen that the requirements specify systems which are impossible or very difficult to build. For example, some people would prefer user manuals to be both small and specific. The ideal manual for these users consists of a single page which tells them exactly what to do in the situation presently confronting them on their screen. The engineering tradeoff dimensions can be used as a basis for negotiation with the stakeholders to attain a specification which is possible to build at a reasonable cost. We might think of this use of the dimensions as contributing to the quality of the ontology building process, in terms of producing satisfactory results on time within budget.

In conclusion, in this paper we have attempted to elucidate some of the engineering tradeoff dimensions related to quality in the ontologies used to support interoperating information systems. We have drawn some of these from ancestral technologies and some from the specific requirements of the emerging area, and indicated how these engineering dimensions can contribute to quality.

Acknowledgement

This paper owed much to discussions with Nicola Guarino of LADSEB-CNR.

References

Bates, M. J. (1986) Subject access in online catalogs: a design model. JASIS 37(6) 357-76.

Colomb, R.M. (2002) *Information Spaces: the architecture of cyberspace* Springer, London.

Debenham, J. K. (1989) *Knowledge Systems Design* Prentice-Hall, Englewood Cliffs, NJ.

Donini, F., Lenzerini, M., Nardi, D. and Schaerf, A. (1996) Reasoning in Description Logics. In Brewka, G. (ed.) Principles of Knowledge Representation and Reasoning Studies in Logic, Language and Information, CLSI Publications, Stanford, CA

Gangemi, A., Guarino, N., Oltramari, A. and Borgo, S. (2002) Cleaning-up WordNet's Top-Level. *Proc. 1st Global WordNet Conference*, 21-25 January pp. 109-121. Central Institute of Indian Languages, Mysore, India.

Guarino, N. and Welty, C. (2002) Evaluating Ontological Decisions with OntoClean. *CACM* **45**(2) 61-65.

Lindland, O.I., Sindre, G., and Sølvberg, A. (1994) Understanding quality in conceptual modelling. *IEEE Software* **11**(2) March 42-49.

Rowley, J. E. (1992) *Organizing Knowledge* 2nd Edn Gower, Aldershot, England.

Searle, J. R. (1995) *The Construction of Social Reality* The Free Press, New York.

Welty, C. and Guarino, N. (2001) Supporting Ontological Analysis of Taxonomic Relationships. *Data and Knowledge Engineering* **39**(1) 51-74.

Weber, R. (1997) *Ontological Foundations of Information Systems* Coopers & Lybrand Accounting Research Methodology. Monograph No. 4. Melbourne.