Uta Priss
School of Computing, Napier University, Edinburgh, UK

A Semiotic-Conceptual Framework for Knowledge Representation

Abstract: This paper argues that a semiotic-conceptual framework is suitable for knowledge representation because it combines conceptual structures with semiotic aspects. The advantages of such a framework are discussed and explained using an example from an ontology language.

1 Introduction

Conceptual structures have been widely used to formalise knowledge representation. Conceptual theories employ classes or sets of objects, instances or features which are arranged in class hierarchies or fuzzy concept clusters. Soergel's (1985) descriptions of thesauri, Barwise & Seligman's (1997) classifications, a variety of formal ontologies, Sowa's (1984) conceptual graphs and even standard object-oriented formalisms (Atkinson et al., 1989) can all be subsumed under ontology languages in the widest sense and can all be characterised as conceptual theories.

In contrast to ontology languages, database and programming languages often have features which are not normally modelled in conceptual theories. These features are:

- Dynamic process-orientation: the data structures and even the language used to describe the data can change on a daily basis.
- Versions and evolution of models: because of frequent changes, mechanisms are provided that allow comparisons between different versions of the represented structures and that allow to fall-back to a prior version in case of problems with the current version.
- Socio-temporal-spatial anchoring in external contexts: metadata are provided that describe the external contexts of the data.
- Expression of modal aspects (beliefs and attitudes): structures can be strongly influenced by idiosyncratic preferences of their creators.
- Explicit modelling of user-interaction: different types of users are distinguished, such as programmers and end-users. Interfaces are designed for each type of user. Mechanisms for managing users are integrated within the representation system.

All of these relate to real-time interactions of users with knowledge representation systems, either during the design stage or in end-user applications. We argue in this paper that these features are not conceptual but instead *semiotic* in the sense of Peirce (1894) and that *semiotic-conceptual frameworks* are better suited for many knowledge representation applications than just conceptual ones.

2 A semiotic-conceptual framework

The American philosopher and polymath, Charles Sanders Peirce, studied signs and their relevance to communication. His research provided the foundation of semiotics, pragmatics and many aspects of modern logic. (Although the credit for these is sometimes mistakenly given to other researchers who publicised and further expanded on Peirce's ideas.) Peirce defines a sign as a triadic relation of a representamen (or physical form), an object (or denotation or meaning) and an interpretant (context or interpreting mind). A sign means something to somebody in some context. Semiotics has been widely applied in the area of user-interface design emphasising the user- and contextual aspects of interfaces. But to our knowledge, Peirce's theory has not yet been systematically applied to knowledge representation languages.

We have developed a mathematical formalisation of Peirce's semiotics (Priss, 2004), which describes semiotic aspects of knowledge representation. We have shown that this semiotic framework can be combined with standard conceptual theories in a semiotic-conceptual framework. The advantage of this approach is that semiotic aspects are added to standard conceptual theories. The semiotic-conceptual framework incorporates the reasoning mechanisms and descriptive methods of established conceptual theories instead of re-inventing them. But these standard theories are enhanced with semiotic expressivity. Or, in other words, conceptual, ontological languages are combined with semiotic, procedural aspects as occurring in programming or database languages.

The conceptual formalisms which we chose for our semiotic-conceptual framework are conceptual graphs (CGs) according to Sowa (1984) and formal concept analysis (FCA) according to Ganter & Wille (1999). In contrast to many ontology languages which are special-purpose-built and only used by a few researchers, both CGs and FCA are used by a larger number of researchers in AI. CGs have even evolved into an ISO standard. A variety of software tools for operating with FCA and CGs and for translation between CGs and other formats (such as KIF) exist. CGs are also closely related to RDF formalisms, which are employed by a large number of researchers working on the Semantic Web.

These conceptual structures are enhanced by semiotic operators pertaining to context management, identity management, user-initiated assignments and speech-act-like modes. Elements of knowledge representation systems are considered as signs, which have both a formal, logical side (their conceptual structures) and a semiotic side. On the logical side, equality and truth values are of major importance. On the semiotic side, the important relationship among signs is not equality, but instead different degrees of *synonymy* (Priss, 2004). Signs can either be *synonyms* (with similar meanings), *strong synonyms* (with indistinguishable meanings), *polysemous* (with equal representamens), *identical* (pointing to the same identifier), or *equal* (strong synonyms with equal representamens). These degrees of synonymy cannot be automatically calculated but depend on user judgement. Each sign exists within contexts and refers to denotations which may have a specific identity (such as object identifiers in object-oriented modelling). Both contexts and identities can be managed by a knowledge representation system and can support user judgements or alert users to possible problems and contradictions.

3 Applications of a semiotic-conceptual framework

A main result of the semiotic-conceptual framework described by Priss (2004) is that the difference between programming and ontology languages can be explained as a mathematical property. In summary, programming involves users, contexts, histories or versions, naming

issues and management of data. Variables in programming languages are signs in Peirce's sense because variables consist of a name (representamen) which has a meaning (denotation) within a programming environment or context (interpretant). In contrast to programming languages, the signs employed by ontology languages are not full signs but merely strings because it is not necessary to distinguish between their representamens and denotations. The meaning of strings can be defined by their relationships to other signs either expressed in formal grammars (i.e., pure representamens without denotations) or expressed in logical axioms (i.e., pure denotations without representamens). Ontologies refer to large, global contexts, such as the shared knowledge among a scientific community. On the other hand, programs in programming languages often change from one context to another as soon as a user enters data. Ontology-based reasoning is string processing, which is computationally less complex than full sign processing because strings abstract from details. Programming and ontology languages represent two sides of a coin: real-world user-interaction versus abstract formal knowledge representation and reasoning.

In this paper, we argue that universal solutions to knowledge organisation, if they exist, must be based on modular approaches as to avoid redundancy and unnecessary complexity with respect to formalisms. A semiotic-conceptual framework facilitates modularising and avoidance of redundancies. The following list summarises some of the projected advantages:

1. Separation of meta-vocabularies and structures: A meta-vocabulary of conceptual/ontological structures, such as class or type, instance, context, ID, role, attribute and value, can be separated from a meta-vocabulary of semiotic/procedural aspects, such as assertion, query, question, name, representamen and interpretant.
2. Joint interfaces: If the difference between database languages and object-oriented modelling languages is identified as their semiotic mode: query versus assertion, then it may be possible to design a single interface for both. In this way, it should be possible to reduce the number of tool-specific formalisms that users need to learn.
3. Separation of reasoning/derivation from communication/design: Conceptual structures and semiotic aspects fulfil different functions: conceptual structures are used for reasoning and detection of inconsistencies. They show consequences of design decisions. Semiotic aspects facilitate sign communication (between users and machines but also among artificial agents) and management of sign conditions. They provide the vehicle through which users can enter judgements and decisions into a formal system.
4. Each sign component has separate structures: Linguistic structures (lexica, grammars, namespaces) are recognised as separate representamen structures which are connected to concepts via signs. Interpretants (or contexts) are defined in a minimalist way and contain only as much information as needed for particular signs.
5. Containment and management of ambiguity: The vagueness and ambiguity of linguistic structures can be managed by asserting that these can only occur between different interpretants but not within a single interpretant. For programming languages, this means that changes in interpretants can to some degree be automatically detected.
6. Re-use of linguistic and conceptual structures: Because terms (or representamens) are distinguished from concepts, terms and concepts can be re-used in a systematic manner. This can be achieved by using facets (Priss, 2000). Re-use of linguistic structures significantly reduces the time required for users to learn programming or ontology languages.

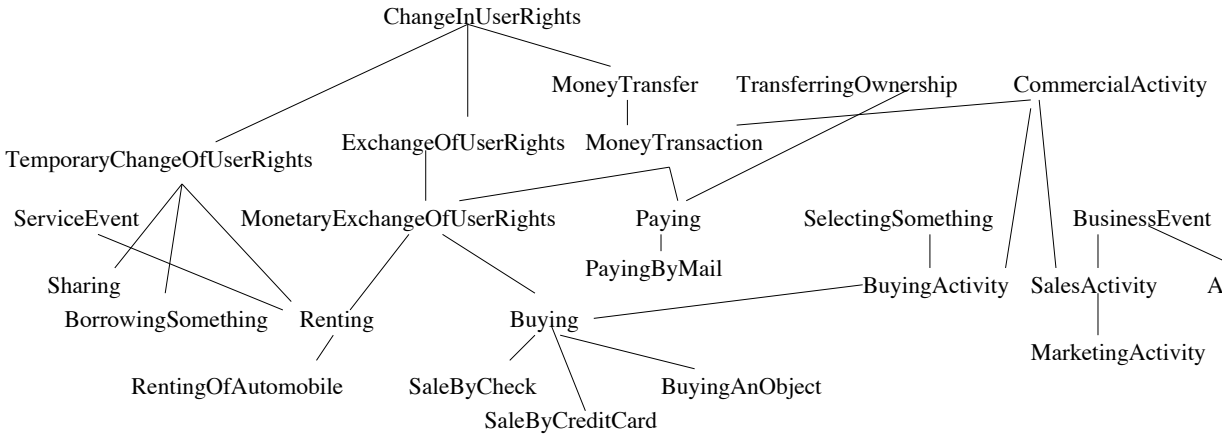

Fig. 1 CYC's terms for Buy/Sell

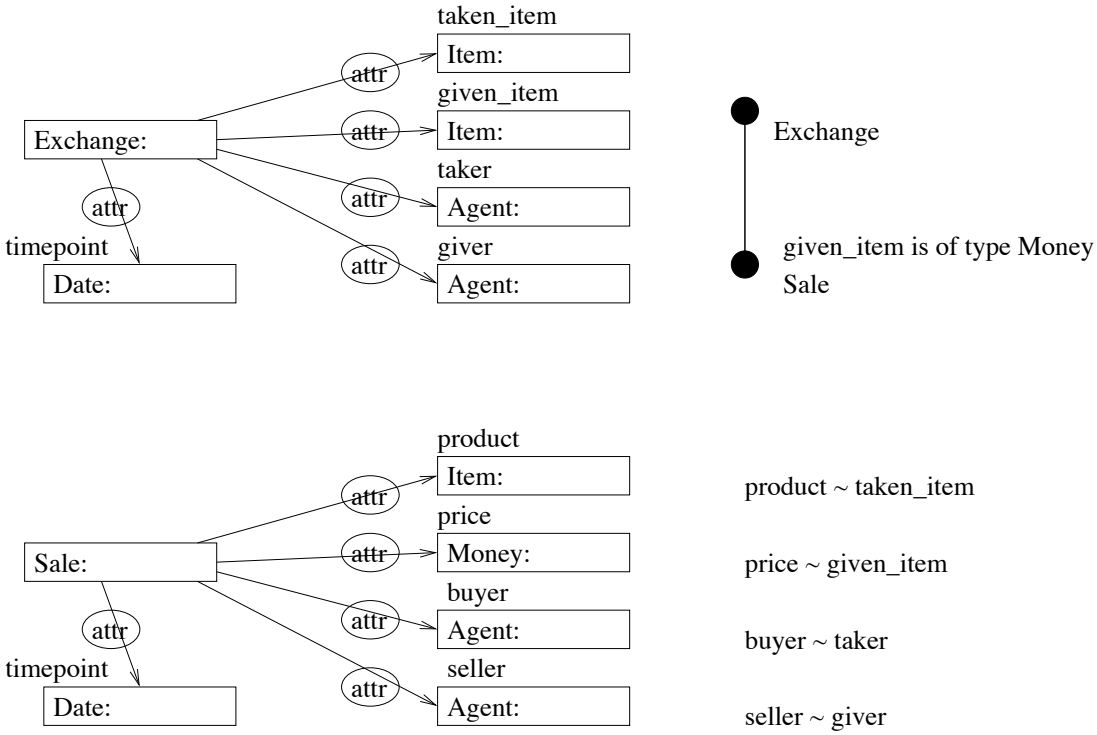

Fig 2. A semiotic-conceptual schema for Buy/Sell

Figures 1 and 2 provide an example that illustrates these points. It should be noted that the graphical representations in figure 2 refer to internal structures. They are not meant as an interface for end-users. Therefore, usability questions are not relevant at this point.

The first example in figure 1 shows a section of the concept hierarchy of the AI ontology CYC (www.cyc.com) which refers to the concepts of "buying" and "selling". The full concept hierarchy in CYC actually consists of several parallel hierarchies (for generalisation, instantiation and type). We believe that CYC needs so many different hierarchies because no distinction is made between conceptual and semiotic relations. A part of the type hierarchy is depicted in

figure 1. The hierarchy is obviously rather "messy". CYC advocates would argue that structures as in figure 1 are only used for automated reasoning, and are not meant to be human readable. But if ontologies represent core structures of human knowledge, should they not be human-readable?

In a semiotic-conceptual framework, the conceptual structures of figure 1 can be reduced to several smaller structures. The top left half of figure 2 shows the basic conceptual schema that underlies all terms related to buying, selling, transfer and exchange in general. This representation uses conceptual graphs, CGs, (Sowa, 1984) which are enhanced with concept names (representamens). The upper right half shows a concept lattice in the sense of Ganter & Wille (1999). It defines "Sale" as the subtype of "Exchange" where the "given_item" is of type Money. Other terms from CYC's hierarchy in figure 1 can be added in a similar manner. For example, "TransferringOwnership" refers to Exchange where the type of the "taken_item" is Possession. The lower right half of figure 2 shows a list of synonyms which are valid in the context of "Sale". For example, the "taker" of a sale (or "Sale.taker") is a synonym of "buyer". These synonyms are linguistic structures, which are not necessary for the conceptual structures but provide abbreviations for users. The lower left half of figure 2 shows the schema for "Sale" after applying the synonyms.

The 6 claims from above relate to the example as follows: First, the conceptual vocabularies of CGs and concept lattices provide for the conceptual structures in figure 2. Apart from adding concept names to the CGs, these are standard representations. The synonymy definitions follow the semiotic-conceptual framework of Priss (2004).

Second, the representations in figure 2 are independent of semiotic modes, such as query or assertion. These would be added via a user interface. For example, users could choose to design conceptual structures for "Exchange", to query for all instances of a Sale at a certain time-point or to question whether a certain Sale is recorded in a database.

Third, standard reasoning mechanisms for CGs and concept lattices can directly be applied to the structures in figure 2 and support queries and questions.

Fourth, linguistic structures such as synonym lists are separated from conceptual structures. In addition to the synonyms that can be directly mapped to conceptual structures, such as the lower right half of figure 2, a larger vocabulary can be defined which represents syntactic variations. For example, a syntactic rule can state that for the English verb "buy" the "buyer" is normally a subject whereas the "seller" is a prepositional object ("buy from ...").

Fifth, contexts can be defined for linguistic structures. For example, the different senses in the lexical database WordNet (Fellbaum, 1998) provide different contexts for each word in use, which could be combined with the synonymy lists.

Sixth, in the example, "Sale" re-uses the conceptual structures of "Exchange". Similarly, synonyms can be re-used in different contexts.

4 Conclusion

This paper contains 6 claims about the advantages of combining conceptual and semiotic structures in a semiotic-conceptual framework for knowledge representation. So far this framework has been applied to a few very simple examples, such as the Buy/Sale terms from CYC and a programming language example mentioned by Priss (2004). As a next step we plan to apply this framework to some larger examples from a variety of domains.

References

Atkinson, M.; Bancilhon, F.; DeWitt, D.; Dittrich, K.; Maier, D.; Zdonik, S. (1989). The Object-Oriented Database System Manifesto. In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pages 223-40, Kyoto, Japan.

Barwise, Jon; Seligman, Jerry (1997). Information Flow. The Logic of Distributed Systems. Cambridge University Press.

Fellbaum, C. (1998). WordNet: An Electronic Lexical Database and Some of its Applications. MIT press.

Ganter, Bernhard; Wille, Rudolf (1999). Formal Concept Analysis. Mathematical Foundations. Berlin-Heidelberg-New York: Springer, Berlin-Heidelberg.

Peirce, Charles (1894). What is a Sign. In: Houser; Kloesel (eds). The Essential Peirce. Selected Philosophical Writings. Vol 2. (1893-1913).

Priss, Uta (2004). Signs and Formal Concepts. In: Eklund (ed.), Concept Lattices: Second International Conference on Formal Concept Analysis, Springer-Verlag.

Priss, Uta (2000). Faceted Knowledge Representation. Electronic Transactions on Artificial Intelligence, Vol. 4, Section C, p. 21-33.

Soergel, Dagobert. (1985). Organizing information. San Diego, CA: Academic Press.

Sowa, J. (1984). Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, MA.