Integrating product catalogs via multi-language ontologies

Manfred A. Jeusfeld[1]

Tilburg University, CRISM/Infolab, Postbus 90153, NL-5000 LE Tilburg,
manfred.jeusfeld@uvt.nl

Abstract. A vertically integrated market consists of potentially competing companies who supply each other with products and services. Information technology can be employed to improve the business transactions and the flow of information within the market. In this paper, we show a solution on integrating massively heterogeneous product catalogs into multi-language and multi-role product groups organized as ontologies. Besides product classification, the method also addresses the classification of product properties. The product catalog data structure is decomposed into a set of label-value pairs which are then classified into the multiple ontologies which is the basis for query formulation. Our approach has been realised within the European research project MEMO (Mediating and Monitoring Electronic Commerce) using the construction industry in an extensive case study. The method can also be applied to related areas like multi-media databases or digital libraries.

1 Introduction

Enterprise application integration requires to understand the heterogeneity of services and data required for coupling the systems. Federated databases are one technology to achieve that but it focusses on data structure integration rather than semantic integration.

In this paper, we present an approach to link the content of databases on a fine grain to an ontology that allows to look up the data and even to formulate queries on top of the ontology. Essentially, the content of the database is viewed as a collection of label-value pairs rather than tuples. These label-value pairs are reified, i.e. regarded as identifiable objects, and then classified into the ontology. The ontology is featuring attribute concepts to which the label-value pairs are classified into.

The method requires the databases to be of a certain format: all attributes in a relations are functionally dependent on a single key attribute. Product catalogs are a prominent example of such relations. The information provider has to take into account that potential information consumers have different roles in the enterprise. Moreover, there might be different catalogs exporting information about the same product. To summarize, the obstacles to be addressed in this paper are:

- The structure problem: an information provider structures the product details in proprietary format. How can a customer match her query against multiple product catalog structures?
- The semantics problem: a natural language term can be ambiguous. How can a query be formulated such that there is no ambiguity in the interpretation of a search term?

There are other obstacles like the unique identification of products. This is not covered by our paper. We assume that products are identified using a unique coding scheme like the EAN code. If that is not available, techniques from data warehousing have to be employed to generate unique object identifiers.

2 Classifying product details

Our solution to bridge the structural and semantic gap between the publishers of product catalogs and the customers is organized in three steps. First, to model the key terms of each customer group are represented in so-called ontologies. Second, the product catalog structure is related to

the ontologies by classification links. Finally, a query mechanism is deployed to extract answers from the product catalogs based on queries expressed in terms of an ontology.

We use the representation framework of Telos [Mylopoulos et al., 1990] as implemented in ConceptBase [Jarke et al., 1995] to uniformly represent all meta models, schemas, and data.

2.1 A multilingual ontology model

The ontology meta model in figure 1 features the central entity 'Concept'. A concept can be related to other concepts, e.g. the 'sbk43_2' (tile) concept is related to the concept 'sbk43' (floor covering). Any concept has translations into multiple languages. Concept attributes are special concepts which are about describing a property of an entity. For example, 'hcp23' (size) is a term to describe the physical dimension of an entity.

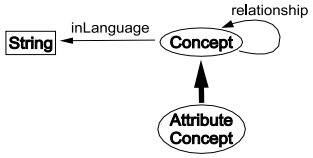

Fig. 1. Ontology meta model

Example ontologies are instantiated from the ontology meta model as shown in figure 2. The SBK ontology is an ontology for architects developped in the Scandinavian domain and being taken over by more and more European countries. It classifies products according to their function in a building. Formally, the term 'SBKConcept' is a subclass of 'Concept' since any SBK concept is also a concept. The HCP ontology is an attempt to classify product attributes. Hence, the class 'HCPConcept' is a subclass of 'AttributeConcept'.

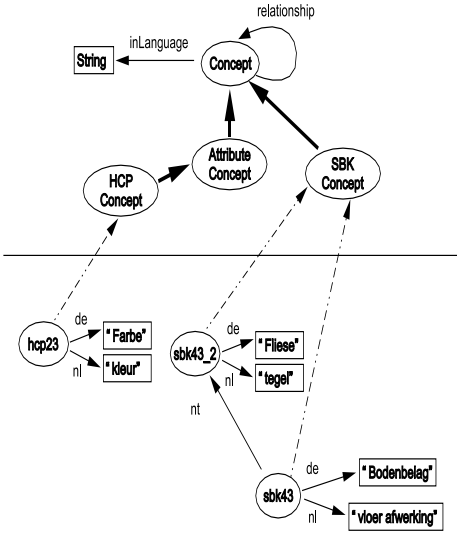

Fig. 2. Example ontologies

2.2 Linking product catalogs to ontologies

A product catalog is published as a relational table where the table structure is completely under the control of the product vendor[1]. The requirement for such a table data structure are:

- A product catalog contains one field which *identifies* the product about which a statement is made.
- A product catalog contains one or more fields which contain *product group* codes. A product group is subsuming products of the same kind, e.g. tiles.
- The remaining fields are *describing* some properties of the product.

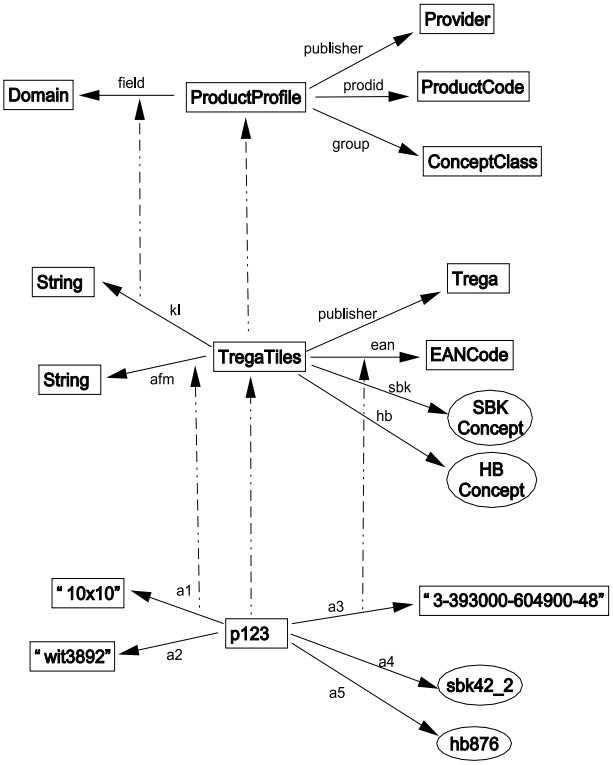

Fig. 3. Abstract model of product catalogs

Figure 3 displays the abstract model of product catalogs used in this paper. The bottom level is the raw data (i.e. product tuples). The name `p123` is a tuple identifier. The middle level is the product catalog schema with `TregaTiles` being the relation name. The top level is the product catalog data model. The link between `ProductProfile` and `Domain` represents roughly the relational data model, the right hand side list our specializations, namely that the is a product identifier and there are firlds product grouping. The publisher field will be used in section 3 to m,anifest ownership and traceability.

2.3 Classifying product properties

The challenge with product catalogs is not so much the classification of products into multiple product ontologies. Different areas like the chemical industry have developped hierarchical codes

[1] Other formats like XML have been proposed but they should rather be seen as exchange formats. As product catalogs stem from the internal databases of an information provider, it is justified to start from a table structure.

for product groups that any supplier of a product catalog can employ to classify her product offerings. A typical user [Callahan and Koenemann, 2000] limits the search space of a query by categories (product groups) and *then* proceeds with restrictions on the attributes. Hence, the challange is to classify product profile fields like a1 and a2 into attribute concepts of ontologies.

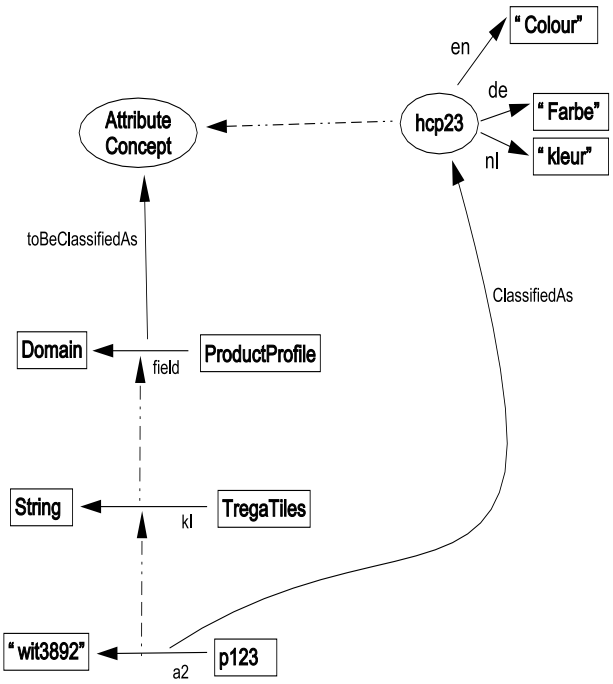

Fig. 4. Classifying product properties into attribute concepts

Our restriction to product catalogs implies a certain structure of the database tables: each attribute is functionally dependent on key attribute (the product identifier). So essentially we restrict ourselves to databases that contains tables for entity types but not relationship types. This appears a strong restriction but the reader should keep in mind that we provide a new form of integrating such databases via a dedicated 'knowledge level' in the form of ontologies.

The classification into ontologies is realized in two steps. First, the products, indentified by their keys, are classified into the concepts mentioned in the product group attributes of the original table. This step is straightforward and does not add any information. Second, the describing properties are mapped to so-called *attribute concepts*(see figure 4).

Attribute concepts are concepts occuring in some (natural) language to describe properties of things. Typical examples are size, color, and fire resistance. They share the fact that some objects (like products) may have fillers for those properties. One might argue that the field names of the product catalogues already carry this information but

- field names in tables are often abbreviated and hard to read,
- some fields in some tables contain several values, in particular when the field type is a string type,
- some values are scattered to more than one fields, e.g. the number value and the unit code,
- subtype relationships between fields cannot be expressed, e.g. the length and the height a product are both specializations of the 'dimension' of the product,
- field names are expressed in some natural language, e.g. English, which makes access via equevalent terms in other languages difficult.

We assume that the describing catalog fields are classified into the attribute concepts by tuples

$$TOBECLASSIFIEDAS(R, j, AC)$$

where R is the name of the product catalog and j is the position of the describing attribute and AC is the attribute concept. Note that the combination (R, j) of the catalog name and the position of the attribute identifies the attribute at the schema level. Figure 4 denotes this graphically by the link `toBeClassifiedAs` attached at field attributes of `ProductProfile`. ConceptBase uses expressions like `TregaTiles!kl` to denote the attribute `kl` of priduct catalog `TregaTiles`. For sake of readability, we will use a relational representation in this paper.

The attribute concepts in our ontology are identified and have links to their translations in as many natural languages as shall be supported by the system. What remains to be done is the classification of product catalog fields into attribute concepts. This is done by generating two types of so-called *universal catalog items*. Let $R(ID, G_1, G_2, ..., A_1, A_2, ...)$ be a product catalog table with grouping attributes G_i and describung attributes A_j. Let further $R(id, ..., g_i, ..., a_j, ...)$ be some tuple of that catalog.

1. For each g_i generate a tuple $C_1(id, R, i, g_i)$. This realized the mapping of products to their product group: product id is classified into product group g_i due to the grouping attribute (R, i).
2. For each a_j and each $TOBECLASSIFIEDAS(R, j, AC)$ generate a tuple $C_2(id, R, j, a_j, AC)$. This classifies describing attributes at the tuple level into the attribute concepts of the ontology: the describing attribute (R, j) with value a_j is classified into attribute concept AC.

The second tuple C_2 realizes the attribute classification. The method is loss-less in the sense that the original relation R can be reconstructed from the relations C_1 and C_2. In figure 4, the second attribute `a2` of tuple number 123 is classified into attribute concept `hcp23`. The subsequent Datalog program patterns show that C_1 and C_2 can be constructed from R and vice versa:

```
C1(ID,R,i,Gi) :- R(ID,...,Gi,...).
C2(ID,R,j,AJ,AC) :-
   TOBECLASSIFIEDAS(R,j,AC),
   R(ID,...,AJ,...).

R(ID,G1,...GM,A1,...,AN) :-
      C1(ID,R,1,G1),...,C1(ID,R,M,GM),
      C2(ID,R,1,A1,_),...,C2(ID,R,N,AN,_).
```

Note that the same attribute (R, j) can be classified into several attribute concepts AC. We assume that each describing attribute (R, j) is classified into at least one such attribute concept.

The parameter `ID` is the product identifier (e.g. the EAN code of the product). The parameter `Gi` is the i-th grouping attribute of product catalog `R`, the parameter `AJ` is the j-th describing attribute of `R`. We include position numbers in the universal catalog items to ensure the *reconstructability* of the original relation. Each grouping attribute G_i of a product catalog R leads to the generation of one rule `C1(ID,i,Gi)`. Analogously, each describing attribute is matched by one rule `C2(ID,R,j,AJ,AC)`.

The ontology level adds additional query capabilities based on the new relations C_1 and C_2. First, an attribute value can be retrieved via its attribute concept identified AC. Since this has links to multiple translations, the product can be queried using terms of any natural language. Second, attribute values hidden in a complex attribute (e.g. size and fireresistance on one text fields) can be retrieved by using the relevant attribute concept that the user is interested in. Third, and most important, the query evaluator can exploit sub-typing relationships between attribute concepts formulated in the ontology. For example, queries on the 'dimension' of a product return both 'length' and 'height' properties. retrieved by the specific attribute concept

Figure 5 shows how data from multiple product catalogs are integrated. The classification method sketched above has to be executed on both catalogs and then allows to access the data

using ontology terms rather than table field names. When the source tables come from different organization, the table name R can be include into the C_2 table. Then, the answer can be traced back to the original source table.

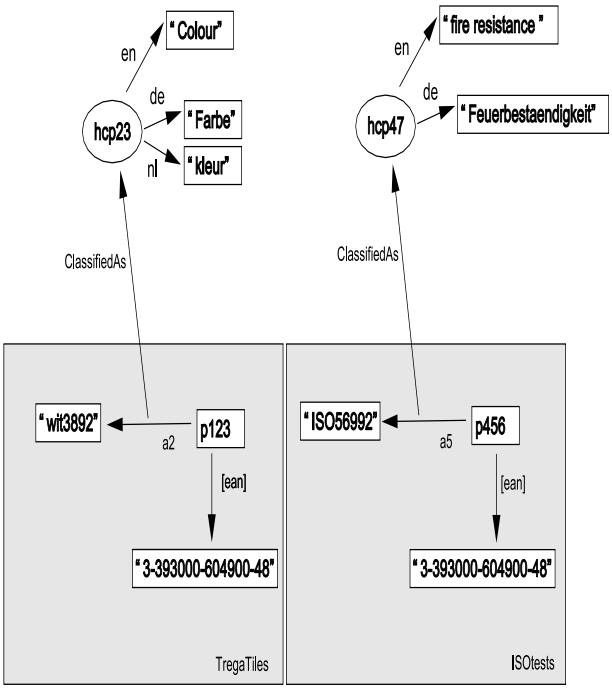

Fig. 5. Integrating product data from multiple catalogs

Figure 6 visualized the treatment of complex attribute values. Here, textual attribute is classified into two attribute concepts (for color and for fire resistance).

Complex attributes are likely in product catalogs from which human-readable representations (flyers, web pages) are generated. Our method makes their content accessible via a semantic layer.

2.4 Querying via the ontology

The Datalog rules show that the C_1 and C_2 relations do not remove information. But the question is: What is the benefit? First, the universal catalog representation allows the integration of arbitrary many product catalogs. It is even possible that information about a given product is provided via multiple sources, e.g. the vendor's product catalog and the quality data supplied by some independent institute.

The more important benefit is however that the universal catalog items can be accessed via the ontology. This makes the user independent from the terms used by the supplier to represent the catalog schema. Moreover, the user can navigate in the network of relationships between ontology concepts in order to specialize or generalize the query. We discuss a few examples to show the added value of the ontology.

We assume a predicate $P(G_1, rel, G_2)$ to denote that product group G_1 is related to a product group G_2 by an ontological relation rel. A possible value for rel is nt (narrower term). Further we assume a predicate $T(N, L, A)$ which returns for a concept name N in language L its identifier A in the ontology.

Example 1: Show products of a given product group 'tile' and product groups narrower than 'tile'.

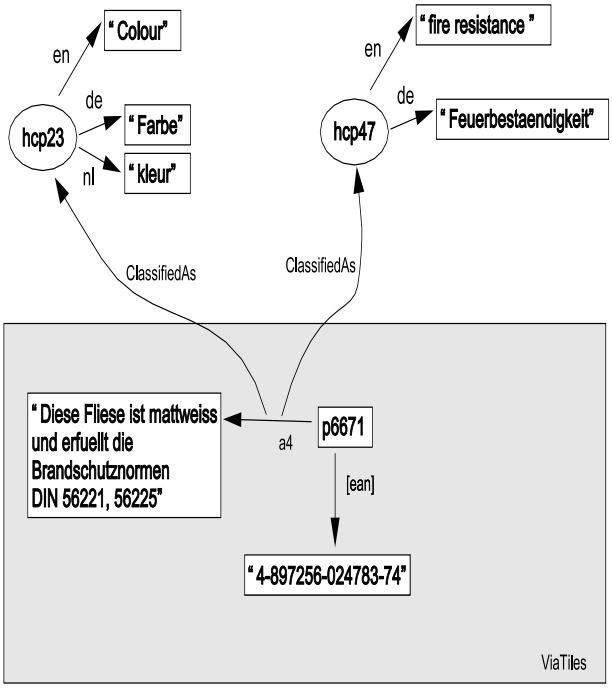

Fig. 6. Integrating product data from multiple catalogs

```
Q1(ID) :-
  T('tile','english',G),
  C1(ID,_,_,G).
Q1(ID) :-
  T('tile','english',G),
  P(G,nt,G1),
  C1(ID,_,_,G1).
```

The example can easily be extended to cover also broader product groups and to cover product groups that have a distance larger than 1 to the given group 'tile'. The Datalog rules are just for showing the feasability of the query. A real user wouldn't code Datalog but just select the product group from the ontology.

Example 2: Show products of a given group 'tile' that have some information about fire resistance or a related attribute concept.

```
Q2(ID,V) :-
  T('fire resistance','english',F),
  REL(F,F1),
  T('tile','english',G),
  C1(ID,_,G),
  C2(ID,_,_,V,F1).

REL(F,F).
REL(F,F1) :- P(F,_,F1);P(F1,_,F).
```

Like in the example before, the distance of concepts related to fire resistance can be larger than 1. Support of languages other than English is obvious.

A characteristic of our approach is that it makes no specific assumption about the value types of describing attributes in product catalogs. All values are treated as vanilla. This apparently makes the use of typed functions like comparison operators between numbers difficult if not impossible.

A possible way out is to represent a type lattice next to the attribute concepts in order to imply the value type of an attribute concept. The more type information is represented the more typing conflicts can occur and potentially endanger the universality of our approach.

3 Traceability and Ownership of Data Items

The classification of multiple product catalogs implies the merge data items from these catalogs into one uniform space. This has the advantage of universal queries ranging over data of all catalogs. However, there are two concerns that need to be addressed: ownership and traceability.

1. Suppliers of product catalogs want to execute some control who can access their product data. Specifically, they want to be able to remove some or all of their data.
2. Users of the universal product catalog need to be able to trace back answers to the sources. Answers that cannot be traced back can also not be vberified by the user.

Both requirements can be addressed by augmenting the universal catalog items C_1 and C_2 by a supplier identifier S and a tuple identifier T.

For a given product catalog $R(ID, G_1, G_2, ..., A_1, A_2, ...)$ from some suypplier S we create an ownership-aware variant $OR(S, R, T, ID, G_1, G_2, ..., A_1, A_2, ...)$ which just adds the constant values S and R and a tuple identifier which uniquely idnetifies the position of a tuple in the original relation R.

From this new base relation, we derive ownership-aware variants of the universal catalog items as follows:

```
OC1(S,T,ID,R,i,Gi) :- OR(S,R,T,ID,...,Gi,...).
OC2(S,T,ID,R,j,AJ,AC) :-
   TOBECLASSIFIEDAS(R,j,AC),
   OR(S,R,T,ID,...,AJ,...).
```

For each source product catalog `OR` and each attribute position i and j such rules `OC1` and `OC2` are generated. The augmented rules allow any universal catalog item be traced back precisely to the location where it originated from.

The supplier of some product catalog can easily compute the universal catalog items that are generated from its catalogs. Due to the logic-based definition, the effect of a change (items added, items removed) can incrementally be computed. This allows simple maintenance of the integrated product catalog from its sources.

Authorization rules based on ownership-aware variants can be employed to restrict access to certain catalog items. A simple authorization rule might be that only users of a user group gr_1 are allowed to access information generated from the product catalog R:

```
OC1(S,R,T,ID,i,Gi) :-
   user(U), member(U,gr1),
   OR(S,R,T,ID,...,Gi,...).
OC2(S,R,T,ID,ID,R,j,AJ,AC) :-
   user(U), member(U,gr1),
   TOBECLASSIFIEDAS(R,j,AC),
   OR(S,R,T,ID,...,AJ,...).
```

The authorization can also be formulated in terms of product groups (proper concepts) and attribute concepts. This allows to target the information to user groups dependening on the concepts they are interested in rather than the data structure as common with database systems.

4 Implementation Aspects

The approach has been realized using the ConceptBase system. ConceptBase has the ability to represent tuple, schema, and meta model information in the same framework based on Datalog as query language. Tuples are automatically decomposed into objects $P(o, t, i, y)$ where t is the tuple identifier, i is the position of the attribute and y is the attrubute value. The attribute itself is identified by o.

The import of product catalogs into the ConceptBase system is automatic. Just the location of the original database and the relation structure have to be specified. The classification of products in product groups is automated by the rules for C_1. The classification of describing attributes into attribute concepts requires the manual specification of the links `toBeClassifiesAs`. It has to be done once per relation.

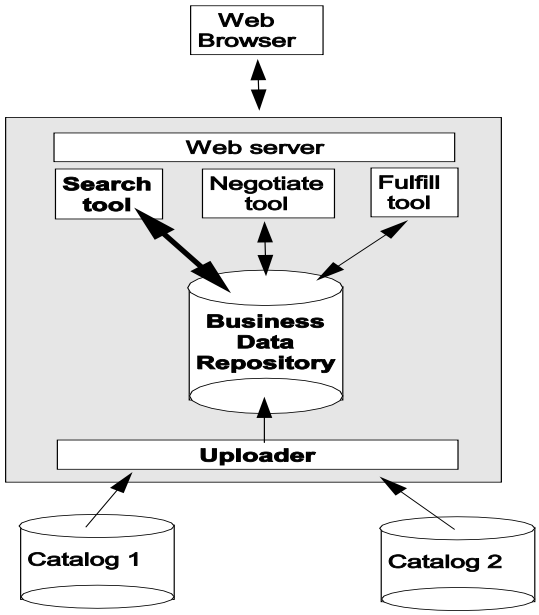

Fig. 7. Architecture of the MEMO system

Figure 7 displays the integration of product catalogs in the MEMO project. The relational uploader of ConceptBase integrates heterogeneuous catalogs into the central repository. The repository contains also (multiple) ontologies into which the catalog items are classified into by the rules. The search tool is invoked by a web browser. A user query (given by a combination of ontology concepts) is translated into a Datalog rule and the result is returned in HTML format.

5 Conclusion

We presented a method to classify product details from heterogeneous product catalogs into multiple, multi-language ontologies in order to support the search for products using the search terms of professional user groups. Multi-linguality is fully supported on the level of ontologies and hence the level of query formulation. Entries in the product catalogs are however not available in multiple languages.

The main innovation lies in the introduction of *attribute concepts* and the classification of product profile fields into them. Attribute concepts make users independent from the somewhat arbitrary data structures chosen by catalog providers. The reader should note that attribute concepts do not make any assumption about the type of the attribute values. If an attribute is

classified into an attribute concept, then we can just conclude that the attribute values is about the attribute concept. This relaxation on the data integration allows enourmous flexibility in the structure of product catalogs that are supported by our approach. The only strong assumption is that each product profile should contain a uniform product identifier, in our case the EAN code.

The price to be paid is that queries like "all tiles with size greater than 10x10" cannot be answered directly. The business data repository simply makes no assumptions about the attribute value type of the size attribute.

The method for product profile classification has been implemented using the ConceptBase system. In the current prototype, product catalogs are copied into the business data repository. A possible extension is to keep the catalogs at the providers site and only maintain the ClassifiedAs relation in the central repository. The classification method was demonstrated with the example of product catalogs. It can be applied to any collection of catalogs where some entities are classified (products, companies, customers) and where these entities have descriptive attributes.

Future work shall investigate attribute typing to support the comparison of attribute values. While not all attributes are subject to comparison, it should be possible to maintain a type lattice aside the ontology into which attribute values are classified into. A type lattice that just mirrors the relational schema would allow for supporting the same type of queries that are possible on the original catalogs. The challenge is to select types for attribute concepts that are not a one-to-one counterpart of the relational attribute and to design type conversion routines to transform values between types. Basic techniques for this are already known. Our ontology-based approach just allows to decide to which degree inter-operability is desired. The vanilla type used in our current implementation is good enough to generate human-readable answers for product searches over multiple catalogs.

Another extension is to integrate process information onto the ontology. For example, the architect knows about an activity 'lay tiles' which includes tiles (`sbk43_2`) and some kind of glue. A process-aware query would be to find fire resistance information about all products involved in the 'lay tiles' activity.

Acknowledgements. This work was supported in part by ESPRIT project 26895 (MEMO: Mediating and monitoring electronic commerce). I would like to thank my colleagues esp. Kees Leune, Mareike Schoop and Christoph Quix for fruitful discussions. Special thanks go to Peter Raadsheer who helped us make the contact with the Dutch construction industry.

References

[Callahan and Koenemann, 2000] Callahan, E. and Koenemann, J. (2000). A comparative usability evaluation of user interfaces for online product catalog. In *Proceedings of the 2nd ACM conference on Electronic commerce, October 17 - 20, 2000, Minneapolis, USA*, pages 197–206. ACM.

[Jarke et al., 1995] Jarke, M., Gallersdörfer, R., Jeusfeld, M., and Staudt, M. (1995). ConceptBase - a deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4(2):167–192.

[Mylopoulos et al., 1990] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. (1990). Telos: A Language for Representing Knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4):327–362.