

CLOVE: A Framework to Design Ontology Views

Rosario Uceda-Sosa¹, Cindy X. Chen² and Kajal T. Claypool²

¹ IBM T. J. Watson Research Center, Hawthorne, NY 10532, U.S.A.
rosariou@us.ibm.com

² Department of Computer Science, University of Massachusetts, Lowell, MA 01854, U.S.A.
{cchen | kajal}@cs.uml.edu

1 Introduction

The management and exchange of knowledge in the Internet has become the cornerstone of technological and commercial progress. In this fast-paced environment, the competitive advantage belongs to those businesses and individuals that can leverage the unprecedented richness of web information to define business partnerships, to reach potential customers and to accommodate the needs of these customers promptly and flexibly. The Semantic Web vision is to provide a standard information infrastructure that will enable intelligent applications to automatically or semi-automatically carry out the publication, the searching, and the integration of information on the Web. This is to be accomplished by semantically annotating data and by using standard inferencing mechanisms on this data. This annotation would allow applications to understand, say, dates and time intervals regardless of their syntactic representation. For example, in the e-business context, an online catalog application could include the expected delivery date of a product based on the schedules of the supplier, the shipping times of the delivery company and the address of the customer. The infrastructure envisioned by the Semantic Web would guarantee that this can be done automatically by integrating the information of the online catalog, the supplier and the delivery company. No changes to the online catalog application would be necessary when suppliers and delivery companies change. No syntactic mapping of metadata will be necessary between the three data repositories.

To accomplish this, two things are necessary: (1) the data structures must be rich enough to represent the complex semantics of products and services and the various ways in which these can be organized; and (2) there must be flexible customization mechanisms that enable multiple customers to view and integrate these products and services with their own categories. Ontologies are the answer to the former, *ontology views* are the key to the latter.

We propose *ontology views* as a necessary mechanism to support the ubiquitous and collaborative utilization of ontologies. Different agents (human or computational) require different organization of data and different vocabularies to suit their information seeking needs, but the lack of flexible tools to customize and evolve ontologies makes it impossible to find and use the right nuggets of

information in such environments. When using an ontology, an agent should be able to introduce new classes using high level constraints, and define contexts to enable *efficient*, *effective* and *secure* information searching. In this paper we present a framework that enables users to design customized ontology views and show that the views are the right mechanism to enhance the usability of ontologies.

2 Ontology Views

Databases views and XML views [1–3, 5–7], have been used extensively to both tailor data to specific applications and to limit access to sensitive data. Much like traditional views, it is imperative for ontology views to provide a flexible model that meets the demands of different applications as well as different categories of users. For example, consider an online furniture retailer, OLIE, that wants to take advantage of ontology-based technologies and provide a flexible and extensible information model for its web-based applications. The retailer creates an ontology that describes the furniture inventory, manufacturers and customer transactions. Let us assume that two primary applications use this ontology. The first application, a *catalog browsing application*, allows customers to browse the furniture catalog and make online purchases, while the second application, a *pricing application*, allows marketing strategists to define sales promotions and pricing. The information needs of these two applications are very different. For example, customers should not be allowed to access the *wholesale price* of a furniture piece. Similarly, an analyst is only concerned with attributes of a furniture piece that describe it as a marketable entity, not those that refer to its dimensions, which are primarily of interest to customers.

The catalog browsing and the pricing applications need to take these restrictions into consideration when querying and displaying the ontology to their respective users. If the ontology changes, regardless of how powerful the inferring is, the applications will invariably need to change their queries. This hard-coded approach to accessing ontologies is costly in development time and error prone, and underlies the need for a flexible model for ontology views. In this case, it is desirable to be able to define the *MarketingView* and *CustomerView* as in the ontology fragment shown in Figure 1.

Despite their similarities with relational database views, ontology views have also differentiating characteristics. First, ontology views need to be first-class citizens in the model, with relations and properties just like regular ontology classes. For example, suppose that the pricing analyst wants to define the *PreferredCustomer* category, as a customer with a membership card that offers special prices for furniture and accessories. Now the catalog application needs a *PreferredCustomerView*, similar to the *CustomerView* defined in Figure 1, adding the promotional price for card holders. It would also be desirable to define the *PreferredCustomerView* as a subclass of *CustomerView*, so that whenever some information is added or removed to the *CustomerView*, the changes are automatically reflected in the *PreferredCustomerView*. Notice that, in this case, we

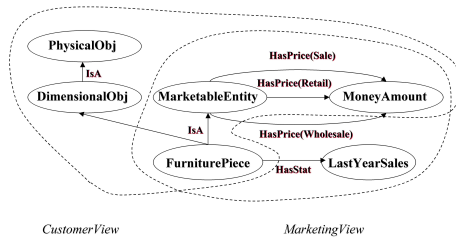


Fig. 1. Two Views of a Furniture Piece.

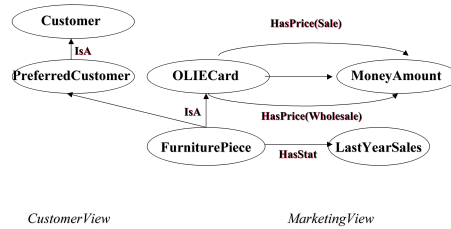


Fig. 2. Inheritance Hierarchy in Ontology Views.

have an inheritance hierarchy within the views, that is PreferredCustomerView *IsA* CustomerView, as shown in Figure 2.

Second, views need to be used as contexts to interpret further queries. For example, suppose that the marketing analyst defines the class SeasonalItems as a set of furniture pieces or accessories that have unusually high volume of sales in a given shopping season, based on previous years sales statistics. The analyst also defines ChristmasItems, SummerItems and FallItems as refinements of SeasonalItems. When a customer queries for information on large oval tablecloths in Christmas, items in the ChristmasItems view should be selected, and the information on each item should be filtered through either the CustomerView or the PreferredCustomerView, depending on the type of customer.

It is easy to see that views need to represent structures of the ontology (like CustomerView) as well as new classes defined through constraints, much like OWL [4] class operators. In fact, the views proposed here are, extensions to OWL classes and expressions, as discussed in Section 2.1.

2.1 CLOVE - A View Definition Language for OWL

We focus on the systematic description and management of views as first-class objects in ontologies, as described in the scenarios above. To the best of our knowledge, this work is the first of its kind in defining ontology views as first-class objects. In particular, we extend OWL [4], a recently proposed standard of W3C, to describe ontologies and their views. OWL allows the definition of classes from other classes through set operations, thereby providing the basic infrastructure support for defining simple views, like the SeasonalItems category described above. However, it has limitations. First, even though ontology views can be considered as classes that are derived from the underlying ontology, they can also refer to subnetworks or structures of classes and relations (like in the case of the CustomerView), underscoring the need for a language rich enough to define both types of views. Second, we need to define a set of standard rules that govern the creation and management of these views, as well as their scope and visibility. While the later is still an open problem, there are some simple mechanisms that allow adequate view definitions. In this paper, we present an overview of a high level constraint language – CLOVE (*Constraint Language*

for *Ontology View Environments*) that extends OWL constraints. We employ CLOVE as the underlying mechanism to support the creation of OWL views.

A view in CLOVE is defined by a set of (1) *subject* clauses; (2) *object* clauses; and (3) variable definitions. The subject clauses describe the constraints under which the view is valid, as well as the range of instances for which the view is applicable. The subject clauses are used to check whether the view (if declared active) should be used in the current query. CLOVE does not restrict the number of subjects of a view. For example, the *CustomerView* defines as subjects all types of customers. It is also possible to not specify the subject of a view by using the keyword *ANY*, in which case, the CLOVE runtime system uses the view to filter all queries when the view is active.

The objects are expressions that describe the content of a view, and have the form:

```
{INCLUDE|EXCLUDE} NavigationExpression ConstraintExpression
```

where the keywords *INCLUDE* and *EXCLUDE* indicate whether the classes or instances satisfying the clause are included or excluded from the view. The *NavigationExpression* is a Boolean expression of relations or properties that are navigated from the set of currently evaluated classes and instances or from a variable or name included in the expression. For example, *?Object SUBSUMES IS-A* are valid navigation expressions.

The *ConstraintExpression* is an extension of an OWL expression. In its simplest form is just the name of a class or instance, but it can also describe the content of its data (the *WITH CONTENT* in Figure 3) or the data type of the properties of a class or instance (with *WITH TYPE*) among others. In the example below, *Customer* and *MarketableEntity* are valid and very simple- constraint expressions.

CLOVE also defines variables that can be directly used in clauses, as well as it allows users to define their own variables. A variable in CLOVE is preceded by the question mark. In Figure 3, the variable *?object* refers to the currently evaluated content of the view. There is also a pre-defined variable, *?subject* that refers to all the currently evaluated subjects of the view. User-defined variables can be used to define scripts or procedures to calculate data from the existing data, like *LastYearXmSales* in Figure 3, which is evaluated from existing properties of *LastYearSales*, the November and December sales.

The full specification of CLOVE is beyond the scope of this paper but Figure 3 gives a brief example of the creation of some of the OWL views of the scenarios above using CLOVE.

CLOVE allows arbitrary relations among views, in particular, inheritance (that is, *IsA*). CLOVE also allows the dynamic creation of classes to evaluate views (like the *LastYearXmSales* as a refinement of *LastYearSales* in Figure 3. After defining them, views can be activated or in-activated by their authors or users with administrative privileges. The runtime system requires that every query is tagged with information about the user, which is associated to a class in the ontology. Queries are evaluated with respect to the currently active views in

```

CustomerView{
  Subject:IS-A Customer
  Object: INCLUDE ATTRIBUTE-OF FurniturePiece
  Object: INCLUDE ATTRIBUTE-OF Accessory
  Object: EXCLUDE ?Object SUBSUMES MarketableEntity
  Object: INCLUDE ?Object HAS-PRICE Retail
  Object: INCLUDE ?Object HAS-PRICE Sale
}

PreferredCustomerView IS-A CustomerView{
  Subject: ANY
}

SeasonalItems {
  Subject: FurnitureItem
}

ChristmasItems IS-A SeasonalItems{
  Subject: ANY
  Define: LastYearXmSales IS-A LastYearSales WITH CONTENT
  LastYearSales.November+LastYearSales.December
  Object: Include ?object HAS-STAT ?LastYearXmSales >
  1.2 * (MarketableEntity HAS-STAT LastYearSales.AvgMonthlySales)
}

```

Fig. 3. Creating views with CLOVE

the order that they were defined. The result is that queries against the ontology are automatically filtered by one or more views, according to the current user context.

All users with access to the ontology should be able to create views. This is one of the most important design principles of CLOVE. However, not every view should be used to filter every query, that's why the CLOVE runtime system keeps track of view dependencies and who created them, with a simple access control system based on user IDs.

3 Conclusions

The Semantic Web brings forth the possibility of heterogeneous ontologies that are universally accessible to arbitrary agents through the Internet. These agents may not only access these ontologies, but also customize their organization and information with their own knowledge and communicate it in turn to their own users. Hence, the ability to create views and contexts on ontologies becomes as

crucial as the view mechanism in traditional database technologies, providing a scope and filtering of information necessary to modularize and evolve ontologies.

However, ontology views are not just straightforward extensions of database views. We have designed and implemented a framework that explores the issues of authoring and management of views and their underlying ontologies. Among them, we have focused on the dual nature of views as classes in the ontology and contexts to interpret new queries. As contextual elements, views are structures of classes and as classes they have relations to other views and even to other classes. We have also implemented a constraint language, CLOVE that takes into account this duality and allows users to both create and query views with an easy-to-use, natural interface.

References

1. S. Abiteboul, R. Hull, and V. V. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.
2. J. Gilbert. Supporting user views. *Computer Standards and Interfaces*, 13:293–296, 1991.
3. G. Gottlob, P. Paolini, and R. Zicari. Properties and update semantics of consistent views. *ACM Trans. on Database Systems*, vol.13(4):486–524, Dec. 1988.
4. OWL Web Ontology Language. <http://www.w3.org/TR/owl-guide/>.
5. A. Rosenthal and E. Sciore. First-Class Views: A Key to User-Centered Computing. *SIGMOD Record*, 28(3):29–36, May 1999.
6. P. V. S. Cluet and D. Vodislav. Views in a large scale xml repository. In *Proceedings of International Conference on Very Large Data Bases*, pages 271–280, 2001.
7. T. W. L. Y. B. Chen and M. L. Lee. Designing valid xml views. In *Proceedings of International Conference on Conceptual Modeling*, pages 463–478, 2002.

Dr. Rosario Uceda-Sosa is a researcher in intelligent information infrastructures, knowledge representation and usability at IBM T.J. Watson Research Center. She received a BS in Philosophy by the University of Seville (Spain), as well as a MS in Mathematics and a PhD in Computer Science by the University of Michigan. She is currently interested in the usability of ontologies and in ontology query languages.

Dr. Cindy Chen is an assistant professor at Department of Computer Science, University of Massachusetts at Lowell. She received B.S. degree in Space Physics from Peking University, China, M.S. and Ph.D. degrees in Computer Science from University of California, Los Angeles. Her current research interests include Spatio-Temporal Databases, XML, and Data Mining, etc.

Dr. Kajal T. Claypool is an Assistant Professor in the Department of Computer Science at University of Massachusetts - Lowell. She received her B.E. degree in Computer Engineering from Manipal Institute of Technology, India, and her Ph.D degree in Computer Science from Worcester Polytechnic Institute, Worcester MA. Her research interests are Data Integration focused on XML integration and Life Science Integration, Data Stream Engineering, and Software Engineering.