MCN Spectra. 24 (1), Spring 1999

Implementing the CIDOC CRM with a relational database

Introduction

The CIDOC Conceptual Reference Model (CRM) is an object oriented semantic reference model for cultural heritage information. Developed by the CIDOC Documentation Standards Group, it was launched at the CIDOC Melbourne conference in 1998. The CRM it is intended primarily for use in the development of mediation systems and other forms of data exchange, however, it can also be used as a basis for designing a production database. Although the CRM is object oriented it can be implemented quite easily using a relational database. Based on experience of Geneva's Musinfo project, this article is intended to offer a brief survey of the major issues involved in implementing the object-oriented CIDOC CRM using a standard relational database engine.

Reasons for using the CRM

The Musinfo project covers several institutions which are responsible for Geneva City's major cultural and scientific collections: the Ethnography museum, the Museum of natural history, the Conservatory and botanical gardens and the Museums of art and history. The inter disciplinary nature of the collections and the desire to ensure a coherent path for future development were the principle motivations for adopting the CIDOC CRM as the basis for the database schema.

- Unlike the CIMI Z39.50 profile, which is restricted to a limited subset common data elements, the CIDOC CRM adopts a *maximalist* approach which aims to encompass the widest possible range of information and to provide coherent and consistent ways of translating information between 'poorer' and 'richer' schema. The semantic depth of the CRM facilitates the integration of information from different sources and disciplines.
- Furthermore, the CRM is designed with evolution and extension in mind. The object oriented mechanism of inheritance provides a convenient mechanism whereby a local database can adapt and extend the model to encompass a greater or lesser degree of detail, while still maintaining compatibility.

Reasons for using a relational database

The database engine chosen for the Musinfo project is Oracle. A number of factors led us to choose a standard relational product rather than an object oriented database:

- Ready availability of technical expertise, both in house and externally
- Long term commercial stability of the software vendor
- Stability and reliability of the software products
- Wide range of third party development tools and accessory software available

Any large-scale IT project involves a number of risks, choosing a well established and familiar relational database product was an obvious way to reduce those associated with the choice of software. However, it did leave us with the problem of finding ways to implement the object oriented schema using relational technology.

Social work

In spite of all these imminently sensible reasons for adopting an 'object-relational' approach, convincing everyone involved still required some hard work and a lot of fast talking.

Users were won over by the application prototypes - the oo approach effectively removed a lot of confusing clutter from their overloaded screens. Having been used to interpreting ambiguous 'generic'

field names - intended to cover all options - and to ignoring inappropriate 'empty' fields - like the price of a donation - users were pleasantly surprised to find that the information supplied on the new screen forms adapted automatically to suit the class of information they were dealing with. Old habits die hard, however, and the tendency to try to group disparate pieces of information into a catch all generic field was difficult to overcome, both for users and analysts.

Developers were a much harder bunch to please, since the approach was unfamiliar and therefore *not the way we do things here*. In the end, the argument which made the most impact was the simplicity of the underlying oo data model. Initial attempts to design a multi-disciplinary relational database had resulted in a ER model that had a charming Jackson-Pollock-freeway-interchange-and-gasworks look to it.

Object Oriented terminology and concepts

Bridging the gap between the relational and the object oriented approach was not as hard as we had imagined. Fortunately, a lot of common ground exists between the object oriented world and the relational approach. Much of the initial difficulty stems from questions of terminology. Simply knowing how to translate from one jargon to the other can make things a lot easier. The list below, while riding rough-shod over the sensibilities of data model purists, provides some equivalents:

oo	*Relational*
class	entity or table
instance	row or record
attribute	field or column
method	stored procedure or function

Of course, not all oo concepts find direct equivalents in relational terminology, however, this is not because they cannot be implemented but because they do not form a central part of the relational approach. The most important of these for an understanding of the CRM are the notions of *class hierarchy* and *inheritance*.

Relational models sometimes incorporate *sub-typing*, whereby a relational entity is subdivided into several more specific entities: an entity describing *Documents*, for example, may be made up of sub-types *Books, Periodicals, Pamphlets,* etc. This is generally a fairly *ad hoc* process in relational modelling and there are no firm rules as to how sub-types are defined.

By contrast, sub-typing - *specialisation* in oo terminology - is fundamental to oo modelling**.** In the example just used, *Books, Periodicals and Pamphlets* would be defined as specialisations of the class *Documents*; inversely, *Documents* would be described as a *generalisation* or *abstraction* of the more specialised classes. Any class may be specialised into sub classes, allowing a complex ***class hierarchy*** to be constructed. This class hierarchy is often referred to as an *Isa hierarchy* - because a book *is a* document, just as a car *is a* vehicle. Isa structures often form complex, multi-level hierarchies, starting with extremely abstract and general classes, such as *physical objects* and *animals,* and descending through successive levels to highly specific classes such as *nitrogenous compounds* and *nematode worms*. This sort of multi-level hierarchy will be very familiar to anyone who has worked with a standard classification thesaurus such as AAT. An oo model effectively allows the classification hierarchy to be built-in to the data schema.

An oo class hierarchy is governed by the rules of ***inheritance***. A subclass automatically acquires, or inherits the attributes of the more general class to which it belongs. Thus, if a *Document* has the attribute *Title*, all *Books, Periodicals and Pamphlets* will also automatically have a *Title*. Subclasses will usually also have attributes of their own: a *Book* may have an *ISBN number*, a *Periodical* will have an *Issue*

number, and so on. Attributes associated with higher level classes are inherited down through the hierarchy so that successive specialisations become progressively more detailed.

Of course, the class hierarchy is not the end of the story. Classes on their own are just classification categories into which *instances* of actual objects can be placed. Instances are, in effect, concrete examples of classes: the *Mona Lisa* is an *instance* of the class *Painting.* A particular instance of a class will usually have values for all its attributes. (In other words, the *instances* of a class are very much like the *records* of a relational database.)

Methods and encapsulation

Methods and encapsulation are key concepts in the oo world. Methods are the processing aspects of objects, which define their behaviour, whilst encapsulation basically means 'hide-all-the-messy-details'. The CRM intentionally does not define methods for classes, since they tend to be application specific. However, any implementation *does* obviously need to define object methods.

Application logic for Musinfo was programmed using PL/SQL, Oracle's procedural extension to SQL. Like many modern programming languages, PL/SQL allows for the creation of 'packages', which are essentially libraries of procedures and functions. Each package has an 'interface' section which allows certain procedures and functions to be made visible whilst others are hidden from view - an important step towards object encapsulation. Musinfo takes advantage of this possibility to create an interface section for each branch of the class hierarchy. All major functions: creation, modification, destruction and recall, are implemented as PL/SQL procedures. Hence the main application does not need to access the underlying data tables directly. This approach effectively encapsulates data storage and manipulation and means that the underlying relational structure is hidden from view. The end result behaves very much like an object oriented database. Modifications to the relational tables need not have any repercussions on the application programming, and users accessing the database see and manipulate classes of objects rather than data tables.

Implementing the Isa hierarchy

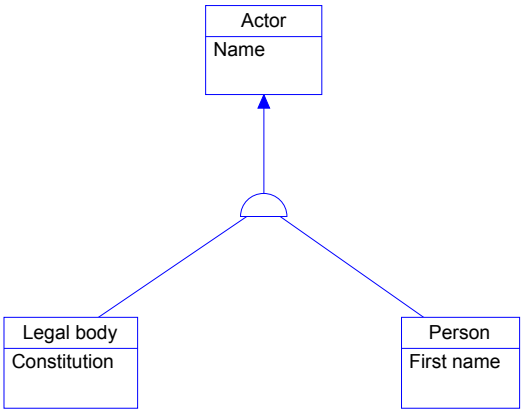
Implementation of the Isa hierachy represents a major challenge since relational databases have no comparable mechanism. However, the CRM does offer a way round this problem.

Apart from the actual class hierarchy, the CRM also specifies a parallel 'type' hierarchy. This type hierarchy can be considered as a thesaurus which duplicates the structure of the class hierarchy. The 'has type' attribute of each class allows instances to be associated with their corresponding type. If the type hierarchy did no more than this, it would be entirely redundant. However, the type hierarchy may go into finer levels of detail than are specified by the class hierarchy itself, thus allowing subtle distinctions to be made which do not affect the underlying attributes for the class. Different sub-species of dogs, for example might figure in the type hierarchy, but all be considered as instances of the general class of dogs. The type hierarchy is one of the CRM mechanisms which allows fine detail to be expressed without rendering the class hierarchy unnecessarily complicated.

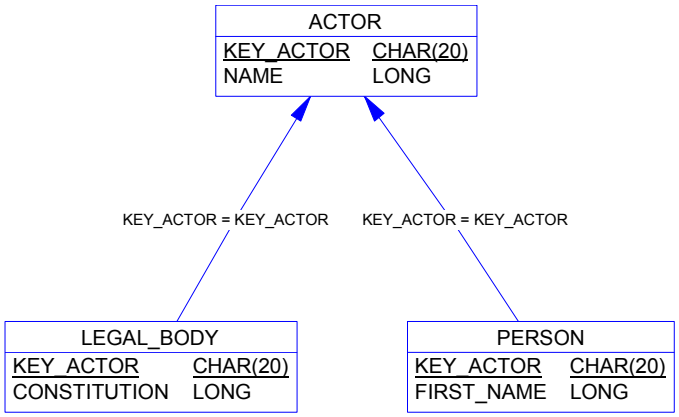
The CRM type hierarchy can be adapted to provide the basic mechanism for implementing the class hierarchy in a relational database. Musinfo uses a thesaurus structure like the CRM type hierarchy to maintain a hierarchy of classes. Each class is defined by a term in the thesaurus. The standard Narrow Term (NT) and Broader Term (BT) thesaurus relations are used to represent the oo concepts of specialisation and generalisation and the 'has type' field of Musinfo database records is used to associate each record with a particular class. Thesaurus scope notes provide a definition of each class and its intended use. The class hierarchy can be consulted and manipulated using standard thesaurus management software.

Implementing inheritance

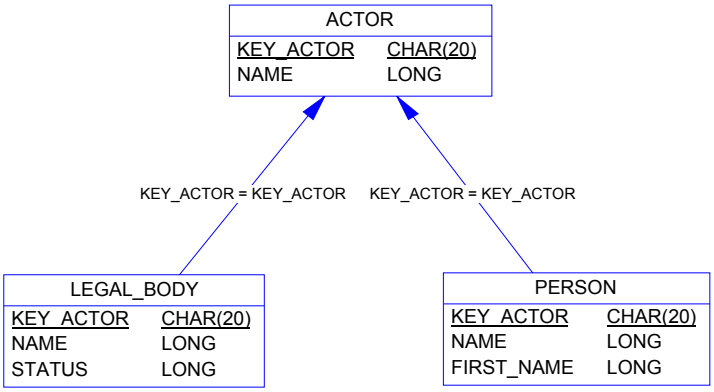
The second major difficulty involves implementing inheritance. This can be done in a number of ways. The following examples are based on the conceptual model below which shows a simplified Isa hierarchy from the CRM. An *Actor* is defined as any individual or a group which can be considered as an active participant in an event. e.g. a government, a museum, the Bauhaus, Monet, the Barbizon school. All actors have names, however, only an individual *Person* has a first-name, and only a *Legal body* may have a constitution.



Perhaps conceptually the most most satisfying approach and one which figures frequently in text books on the subject, is the creation of a primary super class table for all common attributes of a class hierarchy with additional tables used to define attributes needed for specialisation. The same primary key is used to identify and group together the records which are necessary to constitute the object in question. The NAME of an individual person, for example, would be stored in the actor table, whilst the first name would be stored in the PERSON table. This structure is certainly very flexible, since additional specialisations can be constructed simply by creating an appropriate table. However, join operations are necessary every time an instance is consulted or updated which may lead to poor performance.



A simple means of overcoming the performance handicap is to duplicate all necessary attributes within each class. In the example below, the name attribute figures in each table. This approach certainly accelerates reading from the database, but insertions and updating require as much processing power as before, since the value of the name attribute has to be inserted into more than one table. The redundant nature of the data also makes database management more complex.



The approach adopted by Musinfo uses a single table for each major branch of the CRM class hierarchy: Physical Entity, Actor, Event, etc . These tables contain all the attributes needed for all the classes within the hierarchy. A 'has type' field, containing a term from the type hierarchy thesaurus, is used to differentiate the different classes. This approach allows for direct reading, insertion and updating of data without the need for joins, and involves no redundant data. Administration of the database is also simplified thanks to the limited number of data tables.

ACTOR	
KEY_ACTOR	CHAR(20)
HAS_TYPE	CHAR(20)
NAME	LONG
CONSTITUTION	LONG
FIRST_NAME	LONG

One potential disadvantage of this approach is inefficient use of storage space. Instances (records) of 'Person' will not use the constitution attribute, similarly, Legal bodies will not have a first name. These null values inevitably take up unnecessary space. However, the amount of wasted space will depend on the database engine being used. With Oracle, the problem is insignificant since null value columns occupy only one byte of space in the data table. Indeed, trailing null columns (those at the end of a row) take up no space at all.

Of course, the model shown here does nothing to clarify the nature of the class hierarchy and conceals which attributes belong to which class. We have no way of reading this model to derive the existence of 'Legal body' and 'Person', nor the fact that Person has a first name and that Legal body does not. These aspects of the CRM have to be embodied in the thesaurus which contains the class hierarchy and in the application logic. It is the PL/SQL interface which 'knows' which attributes are appropriate for each class. It is important to remember, though, that this is an physical implementation model and not a conceptual schema. Semantic clarity is not a priority in this instance.

Conclusion

One major drawback to this approach is the 'hard wired' nature of class hierarchy logic which is built in to the PL/SQL interface. Although extensions and minor revisions can be accommodated relatively easily, any major overhaul of the class hierarchy would require the application logic to be entirely rewritten. We hope that that the CRM will not be subjected to major revisions in the near future! This does mean, however, that the approach adopted for Musinfo would be inappropriate in a context where conceptual model was unstable and subject to frequent mutations.

The approach described here has proved generally satisfactory. The end product works well and users are pleased with the results. Project management has benefited as well since the use of standard commercial software has ensured that we have had no difficulty in finding compatible development tools and competent outside partners able to undertake development work.

Nick Crofts
Documentation Standards Group chair,
Project Manager, Musinfo.
Direction des Systèmes d'Information (DSI)
Geneva, Switzerland

More detailed information about the CRM is available from the CIDOC Documentation Standards Group's web site: http://www.ville-ge.ch/musinfo/cidoc/oomodel.

The Musinfo project web site can be found at http://www.ville-ge.ch/musinfo.