6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

The basic services of an object-role oriented business framework

Tibor Csiszár[a], Tamás Kókai[b]

[a]Faculty of Informatics, ELTE Budapest, Hungary
INITON Ltd. Budapest, Hungary
e-mail: tibor.csiszar@initon.hu

[b]Faculty of Informatics, ELTE Budapest, Hungary
INITON Ltd. Budapest, Hungary
e-mail: kokai.tamas@initon.hu

Abstract

Nowadays application development processes that don't rely heavily on available software and tools are extremely rare to find. Modern enterprise systems are generally based on databases, object-relation mappings, and on a certain application server. The actual implementation is performed in one of the object-oriented programming languages.

The bibliography of the tools applied is overwhelming and their theoretical bases are massively researched. Nonetheless the matter of integration is really complicated. Several frameworks has been developed to solve the problem, which we think is not settled. Existing frameworks usually can't deal with every aspect of the integration they aren't formalised properly, so it is impossible to create an effective CASE tool based on them, that helps the design and maintenance activities.

A direct consequence is that most of the enterprise software developer companies must create a design methodology or system based on the existing tools, the company's own developments, and industry-imposed quasi-standards. The success of their developments relies heavily on their system's flexibility and efficiency. Furthermore debugging, testing and maintenance of the actual code, data and documentation are also crucial issues.

The framework presented by us is innovative from many aspects. It comprehends the whole spectrum of the development, from the object-role oriented modelling to the design of the user interface. The formal parts of the application under development are stored in a meta-database, which contains the datamodels, queries, user interface, and also all their relations. The non-formal parts or the exceptions from the general rules can be implemented by application specific extensions that use standard interfaces.

In this article we introduce a part of the whole subject, namely the short description of the object-role oriented modelling method applied by us, the brief view of the developed framework based on the concept covering some of the generic problems solved.

The article contains the following basic services of the framework: metadatabase, keys, unified handling of role oriented object with the application specific extensions, transaction management, multithreading and concurrency. Finally we summarise the possible future enhancements to this framework.
To all the specialists interested in this subject we recommend our publication [Csiszár-Kókai 2004.2] on the further possibilities of the framework, including calculated fields handling, query support, GUI building, error handling and tracing.

Categories and Subject Descriptors: D.2.11 [Software engineering]: Software architecture; H.1.1 [Information systems]: Systems and information theory

Key Words and Phrases: conceptual modelling, object-role modelling, framework, metadatbase

1. Introduction to object role modelling

Object role modelling could be regarded as a further improvement of object oriented modelling. It isn't a unified initiative several independent research-groups have published their results (`http://www.aosd.net/technology/index.php`, `http://www.research.ibm.com/hyperspace/`, `http://www.orm.net`), which are radically different. Still they have in common the separation of entities, or concerns into classes, which could be modified even at runtime.

For instance let's take a person class, which has two subclasses: doctor, patient. A certain person, at runtime could be classified in either one of the subclasses or even dynamically removed from them. A person may become a doctor or even get ill, and eventually get cured.

These kinds of classes are the so-called roles. It is also appropriate to say that an object "plays" a certain role. The roles of a class can be organised into disjoint and exclusive groups. An object instance of the class plays only one role from a group at a time.

The object's actual roles at a certain time are called configuration.

The subject of modelling (reality) can be described more accurately by role oriented modelling compared to relational or object oriented database design, because it unifies the advantages of the other two well known approaches. It has a richer expression set by which conceptual constraints can be declared more precisely, but an exhaustive description of that goes well beyond this article's scope. In the followings we present only the most crucial topics, which are essential in order to understand the underlying framework itself. (For the ones who are interested we warmly recommend these: [Csiszár-Kókai 2001], [Csiszár-Kókai 2002.1])

- A class and their roles form a tree. Multiple inheritance can be achieved by setting multiple roles at the same time. In the object role terminology the base class is also a role. Every non-root role could be referred as an optional one.
- The classes in a system are divided into two groups, based on the method, which is used to declare their components:
  - Base (or constant) classes: they are generated from a base set (character literals, integer or floating-point numbers) by forming a subset. The objects of these classes can be identified by their values. An important difference between this group and the second one is that the base classes cannot have more fields and optional roles. For a better differentiation we call these classes sets, and their objects elements.
  - Enumerated classes: The framework assigns a unique object identifier (OID) to every enumerated object (an object of an enumerated class). They can have multiple fields and optional roles. Enumerated classes are called roles and their objects are called instances.
- Every field can be regarded as a pointer, because it refers to another set's element. In case of a base class the field's value is the element itself, in other cases the referenced object's OID. Additionally we can differentiate two other cases. There is special field (the so-called calculated field) that is automatically calculated from other available data and treated like a base class. A role's flag field points to the actual active role, from that exclusive, optional group of roles. The difference is that it doesn't hold the active role's OID but its key.
- A list of the pointed OID-s is generated automatically for every pointer in every class that refers to another class, except for the flag fields.
- In addition to an object OID, every class should have a primary key, which is composed of its fields (and unlike the OID it is mutable), and with the key's aid the user can effectively identify the desired object.

2. The Framework's main parts

The framework is based on an SQL92 compatible RDMS of choice and Java language. Instead of the Java we could choose any other third generation language, but the rationale behind it was the platform independence, mainstream maturity and its relative simplicity. The bases of the framework's services are the persistent metadatabase, its content, and the special purpose algorithms. The metadatabase's task is to store all the formalisable elements of the system in a structured way and also to maintain its consistency. The source – that is functioning based on the data stored in the metadatabase – is enhancing the possibilities of the Java language with the following:

- **Emulation of role orientation**: The emulation covers the automatic switching of the roles, the administration of an object's lists, the maintain of consistency which is described in the metadatabase, and a high-level querying interface (including an object's fields and lists), which hides the underlying relational mapping. The roleoriented classes aren't necessary to be defined. They are provided by the framework based on the metadatabase.
- **Multithreading**: The framework is designed to be thread-safe, thus allowing multiple threads to access it.
- **Automatic persistency:** Every roleoriented object is stored in the relational database, which is hidden from the application programmer. The persistency includes transaction management, which is compulsory for every data update.
- **Advanced querying:** On the partial graphs of the metadatabase, one can easily compose highly complex queries. The results of these queries are role oriented objects, which usually form a list, but can also form a tree-like structure.
- **Support for creating User Interface**: The framework supports the building of user interfaces and their binding to objects and queries. The generalised user interaction templates can be reused.

3. The metadatabase

The main characteristic of the framework is formality that appears in every level of it. The application designers create the business data structure using the object role model, and then it's stored in the metadabase, which can describe even itself, and the constraints to secure the consistency of the whole. The necessary constraints and the actual optimal logical structure of the metadatabase are still researched. In this brief article we can only point out the main elements of it.

The metadatabase holds an inhomogeneous directed graph with roles and their fields as vertices (in figure 1 the role's bounding boxes have strong frames). The edges of the graph can convey three different meaning:

1. If both endpoints are roles the edge is represented by a broken line and the start point is an optional role of the end point. The line endings in the same point describe a disjoint, exclusive group of roles.
2. Black arrows point from the roles to their fields.
3. Dotted lines point from fields to the pointed values (which can be roles or sets).

Figure 1 models an aeroplane's state:

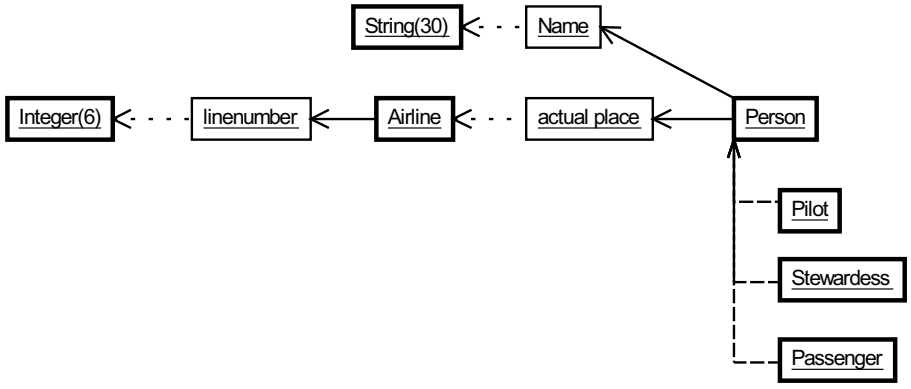

Figure 1: Representation of the metadatabase's content

The declaration of constraints, queries, and framework-managed fields (calculated fields) can be accomplished with the graph stored in this metadatabase.

4. Keys

In Object oriented database management the importance of keys isn't stressed. The only global rule is that every entity must have a unique identifier, which is managed by the system itself.

In practice – if the model describes a slice of the physical world – the keys cannot be put apart, because they are necessary to establish a bi-directional relation between the model and the real world, and additionally:

- The user interface doesn't display OIDs, so the user must identify the object with the help of their keys.
- It is regular that we encounter certain constraints in models like uniqueness of several fields. For instance an M:N relation descriptor object's pointers, through which accomplishes the relation are inherently unique. This is also applies to two or more classes' combination of keys.

The framework applies three constraints on keys:

- Either one of the keys mustn't be another key's subpart.
- The key must consist of not null (compulsory) fields.
- The keys' and the referenced classes' graph – where classes are the vertices and keys are the edges – must be acyclic with the following addition: if one of the keys' elements is self-referential than the structure defined by it must be acyclic. This way it is possible that in a tree the key of a vertex to be its parent and his name.

The framework offers several services regarding keys:

- Checks the uniqueness of them.
- It makes possible to refer an object through its key.
- Restraints the mutability of the keys. A certain key (or its parts) can be constant after it has been set. To illustrate the two different modelling goals, here is an example with a two member key:
  - If a key is mutable then it only refers to the target. The main criteria in this case is, that only one object can refer – with their keys – to the object pairs from all class at the same time.
  - If the key can not be modified, then the objects not only refer to the object pairs, but they are the elements of the referenced classes' cross multiplication. To modify the parts of the key it is necessary to delete and recreate the same object, but the delete operation is not allowed throughout the framework.
- It handles correctly the invalid, but inactive objects' key.

5. The unified handling of role oriented objects

The framework offers access to every class that has been registered in the meta-database. The basis of this feature is not source- or byte-code generation, but a general-purpose class, namely CachedObject (in the following CO). Every CO instance knows its type, including its dynamic roles. The CO-s of one type are stored in their appropriate Repository.

The framework offers these services to help using COs:

- The construction of a CO is done with a framework's factory method.
- Its OID or key can refer to a CO in the framework.
- The CO's persistence is container managed, but can also be customised.
- The fields of a CO can be accessed easily through get(fieldname) and set(fieldname,value) methods. For better usage there are accessor methods with type-constrained parameters and also generic ones.
- The lists of the COs are accessed by getList(listname) method. The returned list is modifiable and the administration of the lists is completely automatic.
- The CO has a flag field for each exclusive role group whose value is always the name of the actual role in the group.

- The framework secures the consistency of the COs, because it applies every constraint that the metadatabase holds (type, not null, length, uniqueness etc....). For the temporary (inactive) CO-s the error list – which rises from missing or incorrect data or any other constraint – can be obtained. If the CO's status is active or cancelled (any modification is forbidden), than it is impossible to become inconsistent.

- Every modification of the CO-s is automatically logged. The log can be generated per class, globally, or even both. Due to this mechanism the user interface can display modification history for a desired period.

- The CO-s creator user and owner are also registered. The business logic regarding security aspects is based on this information.

- The CO-s are clonable. The cloning can be parameterised, so one can explicitly set which references of a CO are to be cloned, and how deeply. The newly generated CO-s are also consistent and the uniqueness constraints even for the keys are also kept, by deleting these fields (a direct consequence is that the resulting CO is inactive and the key and every unique field must be set).

Every object in the framework is a CO, even the metaobjects. Due to this comes a useful feature, that one can add new metaobjects at runtime (!) and modify the existing ones with valid transformations, providing that the system's consistency isn't broken. It is possible to add a new field or to modify its type or length. The framework can be freely and dynamically enlarged.

The CO-s default services and behaviour is general enough, still sometimes there is need for more. The system offers a standard way to extend the CO. The instantiation is done using the CO's name. Using the Reflection API the framework can discover existing extensions to a certain CO and act according to that.

There are other extensible aspects regarding fields, and lists. An example of list extension is the validation logic of the so-called decomposable product: the composing objects' percent sum must be one hundred, if not, the modification is invalid and is prohibited, returning an error list explaining the reasons of the unsuccessfulness.

6. Transaction management

6.1. Short transactions

The system executes every modification through the CO-s, in a transactional environment. The transaction marking is declarative, and during one only the previously locked objects can be modified. On unsuccessful locking it's necessary to provide competent error handling in order to avoid deadlocks. The required locks and their users can be accessed and displayed. In addition unused locks are released, or even manually deleted by the administrator.

Theoretically it is possible that a thread cannot acquire every lock that is necessary for further processing, in other words it starves. To avoid this unwanted scenario, we can introduce some kind of queuing policy, combined with the priority of threads and operations. This way we can achieve an optimal solution. Nonetheless practical results didn't suggest such measures to be taken. These results have provided valid proof for virtually every Web-based application we encountered.

Unless otherwise specified the transactions cover every statement regarding cached or persistent objects. The transactions are characterised with the well-known ACID properties. This gives enormous stability to the framework and the application above it. For over a year of continual uptime not even the application programmers were able to bring the system in an inconsistent state, or to abnormally halt it with a critical error.

Those transactions that are executed only in memory are called asynchronous transactions, and are more efficient than normal transactions involving the database, but one must be careful when using them, because they query the database, which holds perhaps different state than the memory. Consequentially there are certain rules that are kept:

- A class that was modified during another asynchronous transaction cannot be queried.
- The modifiable classes' every component must be recursively loaded in order to provide structural consistency.

During the test process it is useful to disallow transactions that involve database modifications. Another fine example of asynchronous transactions is the handling of queries, which are represented as complex and parameterised data structures that only modify their parameters. For efficiency reasons the framework reads from the database the actual configuration at the user's login and writes it at the end of a session.

6.2. The consistency during transactions

In a relational database system the problem of the consistency rises during transactions. Figure 2 shows a simple example where enterprises have several places of business and one of them is compulsory to be its headquarters.

The cyclic references make it impossible to populate the mentioned structure during separate transactions, and even during one, the constraints must be deferred. Naturally at the end of transactions the system has to be consistent.

6.3. Isolation of transactions

During writes the separate threads are isolated with locks. At reads there isn't such constraint, but for an object instance the parallel number of writes and reads can be reduced to one. Basically serializable transaction level is provided for clients.

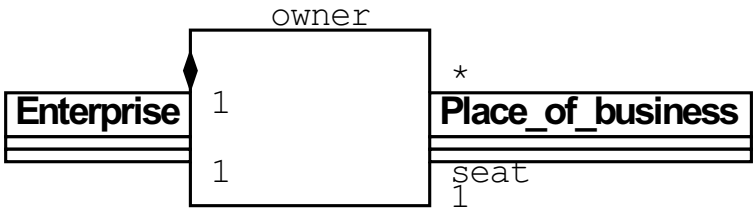

Figure 2: A structure that is impossible to populate with relational DMS

6.4. Long transactions

The framework supports this kind of transactions during creation of objects. It happens often that an object has many optional roles or fields, or references even, and their correct filling is tedious and takes time through a user interface. To ensure this kind of process the status field was added, which can be temporary, active or banned. The valid status transformations are shown in figure 3.

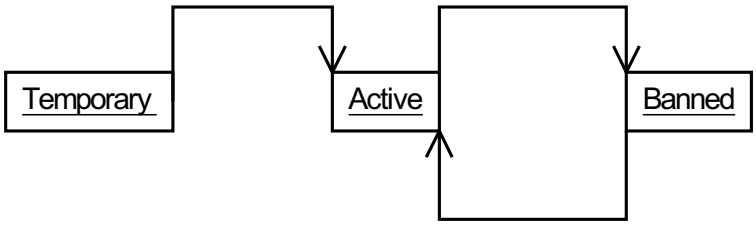

Figure 3: Valid status transformations

The temporary status means that the framework should allow partial filling of an object, though the active ones mustn't even refer to a temporary object. The banned state is a form of deletion. The main difference is that during physical deletion the modification history and log is broken. For efficiency reasons the temporary object at deletion are actually removed from the database.

7. Multithreading and concurrency

An expected feature that every framework should offer is thread-safe parallel access. After authentication every user receives a session ID. As far as operations are concerned every session is isolated from the others. The only constraint is that the framework allows one open transaction per session. All of this is accomplished by CO level locks and synchronised methods.

8. Future works

The framework, besides being successfully deployed in an enterprise environment and fully functional for more than a year, can be enhanced in many ways. In this chapter we only summarise the future additions and changes in a priority order, for further reading we recommend [Csiszár-Kókai 2004.2].

The main task is to provide a unified tool, for creating and editing models and queries. Besides this it is crucial to develop the necessary modules that realise this. For the time being the Dia drawing program is hard to use and the modification on the visual representation isn't automatically reflected on the database schema and vice-versa.

With the introduction of outer keys an object's key can contain any of the object's derived roles' mandatory fields. The proof of this need can be found throughout the vast bibliography available at: `http://www.orm.net` portal.

The transformation of the metadatabase constitutes further research, from simple cases like adding a field, to complex structural changes. The formalisation of these transformation could drastically cut-down the maintenance expenses.

References

[Andersen] Andersen, Egil P. Using Roles and Role Models for the Conceptual Modelling of Objects `www.ifi.uio.no/~trygver/documents/index.html`

[Csiszár, Kókai 2001] Csiszár, Tibor - Kókai, Tamás An approach of complex information system's modelling Lecture of „Fourth Joint Conference on Mathematics and Computer Science" Baile Felix, Romania 2001.

[Csiszár, Kókai 2002.1] Csiszár, Tibor - Kókai, Tamás Előadás és publikáció „A szereporientált modellezés alapjai" V. Országos Objektum Orientált Konferencia Dobogókő2002.

[Csiszár, Kókai 2002.2] Csiszár, Tibor - Kókai, Tamás „Szereporientált szoftverfejlesztés a gyakorlatban" V. Országos Objektum Orientált Konferencia Dobogókő2002.

[Csiszár, Kókai 2004.2] Csiszár, Tibor - Kókai, Tamás The advanced services of an object-role oriented business framework Proc. of the 6^{th} International Conference on Applied Informatics Eger, Hungary Jan. 27-31, 2004

[Halpin 2001] Object Role Modelling: An overview `http://www.orm.net` 2001.