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

The advanced 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.

For the total understanding of our writing is absolute necessary to know our previous publication [Csiszár-Kókai 2004.1]. The mentioned article introduces 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. It contains the basic services of the framework (metadatabase, keys, unified handling of role oriented objects with the application specific extensions, transaction management, multithreading and concurrency).

In this article we go on reviewing the framework's further possibilities, including calculated fields handling, query support, GUI building, error handling and tracing.

Finally we summarise the possible future enhancements to this framework.

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, metadatabase

1. The calculated fields

1.1. Motivation

The calculated fields contain redundant values, which can be reproduced from the database following an adequate algorithm. The problem of calculated fields might be treated as a secondary goal but their introduction is almost inevitable and their calculation is an important theoretical problem.

The calculated fields are necessary for several reasons:

- The possibilities of the SQL queries are restricted. In the results of the queries the users want to see in a table cell not just field values but lists, embedded lists or other complex values.
- The other reason can be found inside the theoretical problems of the query-optimization in a relational database. The duration of the SQL query execution is rising exponentially with the number of table joins. Based on practical experience we can say that most of the databases produce acceptable results with queries containing a maximum of 6-7 joins.

Keeping the data in a redundant field inside an appropriate structure can solve the first problem. This way the performance can be improved too. The calculated fields, which are complex and require many joins, can be stored in these redundant fields. The address of a person for example must be generated from four tables (country, location, postal number, public place).

The other advantage of the calculated field is that the algorithm, which generates it, can be stored on class level. Stored procedures (PL/SQL functions) may hurt the theory of encapsulation and the portability between database systems.

If a calculated field is created with aggregation, several sub queries might be needed because one query permits only one grouping. The presence of several and probably embedded aggregations may cause the query to be much more complex, which leads to low performance.

1.2. Definition, implementation

A calculated field can be defined – like a query – using a partial graph of the metadata. The fields that must be calculated define the endpoints of the graph. The changes of these or the intermediate fields notify the connected objects and this way the calculating algorithms are executed. For the calculating one can implement arbitrary algorithms, which uses the pointed fields inside the partial graphs of the calculated fields.

Figure 1 shows the partial graph needed for address calculation (the example is highly simplified)

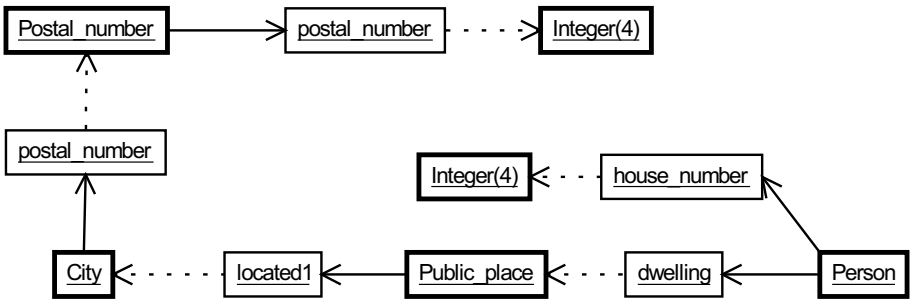

Figure 1: The graph needed for address calculation

If a city's postal number is changed the city's inhabitants (person) get notified. The same situation rises when a person is moving or a public place gets annexed to another city. The example shows that one calculated field listens to the notification of four separate objects.

By using several calculated fields the mechanism gets hardly embraceable, for this reason the notifications of the calculated fields are passed on automatically by the framework. The framework supports the recursion of the calculated fields and the dependence can be realized on recursive structures too.

If a calculated field is used as a key than one condition exists on it: the given dependencies on instance level must not form a cycle.

1.3. Calculated fields and transactions

Theoretically the field calculating can be done with two kinds of timing policies.

During modifications that affect only a few fields the immediate calculation gives sufficient performance.

Nevertheless during transactions, which involve huge amount of data the actual calculating of these fields is deferred, because some data could even be missing, because consistency checking happens at the end of the transaction. (As in the example above) Other issues rise from the fact that in these cases the immediate calculating would results an overwhelming amount of extra operations.

For these reasons the framework calculates the fields using the second more general and efficient approach. During transactions different modifications generate different messages. The target object registers the received message and delegates it maximum once. At the end of the transaction every target object tries to calculate its calculated fields. The calculations result in a topological order.

1.4. Deficiencies

The calculated field mechanism has many advantages, but we must also take into account its weak points:

- Abundant usage of these fields could radically extend the width of tables, thus slowing down queries.
- When modifying a certain field, on which several calculated ones depend, all of these values must be loaded and recalculated, which could result a slow response time.
- The locking of an object which, has calculated field should lock all connected object by following the dependencies of the calculated field. At this moment the framework locks objects and for efficiency reasons the reference locking mechanism isn't supported.

2. Querying

An application's key components are the queries. The popular SQL has several limitations and doesn't fit into the framework's concept.

- Even logically simple queries have long SQL representations. The main goals like clarity, simplicity and brevity haven't been accomplished.
- The parameterisation of an existing query is highly restricted, and cannot be obtained with present tools and query syntax.
- The framework hasn't got information about tables and fields affected by an SQL statement. (To achieve this goal we should implement a full-featured SQL pre-interpreter.)

The role-oriented queries are declared by drawing the metadatabase's partial graphs – not by language –, from which the framework automatically generate the necessary SQL statements. Even the role-oriented queries are stored in the metadatabase, so it's easy to list the used entities by individual queries and to automate certain modification tasks.

An example can better reveal the importance of this functionality. If the Partner table needs to be augmented with an additional field, and must be inserted in every query that involves this table, than it can be done in one step, avoiding dangerous manual inserts. This automatism can be additionally parameterised with other constraints (like only certain users must see the new field).

2.1. Command items

The generated SQL statements are capable to gather information scattered in the database, but sometimes the user demands more. In order to satisfy these special needs the framework provides certain automatically managed columns, which rely on the available business and system information. The classes behind these post-processing are called *command items.*

The command items aren't specific to a certain query, and can be attached to any query, by binding them (their input parameters) to columns.

For instance the command items compute the accessibility rights from the query's result using key column and data owner reference. (The information about columns can be found in the metadatabase.) Using this technique in a WEB based application the key column's value can be changed to a parameterised hyperlink, which refers to the appropriate object's editor page.

2.2. The parameterisation of the queries

Every query declared with a graph can be parameterized with the encapsulated constraints on the affected fields. The fields can have finite number of values, or quasi infinite.

The finite ones are every field that refer to an enumerated role, and flag fields whose possible values are the names of the roles from that exclusive, optional group. The constraint declaration on "finite fields" (so called filtering) can be done with the selection of one possible value.

The non-finite ones can be parameterized with a character literal and valid SQL functions, that's called searching.

The possible conditions can be combined using logical disjunction.

Applying the character literal condition to all columns, thus in the result will appear any record that matches the condition for any column.

2.3. Query prototypes and instances

The queries are stored on prototypical and instance level.

The prototypes describe the skeleton and basic configuration of a query, which consists of:

- The possibly displayable columns and their default order.
- The default viewable columns.
- Context dependent parameters (so called system filters), which localise the result of query. For example possible context dependent parameter is the user, who run the query.
- The default filter and search conditions and their parameters.
- The default sorting.

If a user after authentication requests a list, which uses a query, the framework takes the following actions:

- If the user has an instance of the query, but it is not used, then binds it to the current GUI element.
- If the user has an instance of the query and it is used by another GUI element then clones it and binds it to the current.
- If the user doesn't have an instance of this query, then its prototype's clone is generated and binds to the GUI element.

The users can save their personalised query configuration with names, and load them whenever it is needed.

3. User interface and support for interactivity

The framework in its present state supports only the creation of WEB based applications. Because of the limited possibilities of the browsers two kind of pages are available: list pages and editor pages.

To the creation of a list page:

- the adequate query must be given,
- the command items must be specified,
- the appropriate reference on the list page must be placed inside the menu system, which is stored in the meta-database.

The framework maintains all the other tasks.

Creating an object's editor pages is much more difficult. At first a JAVABean must be created and the get and set methods must be implemented together with the load, update and validate methods. After it the WEB GUI, which is based on the JAVABean, can be created using JSP or other technologies.

The framework supports the easier bean creation with the following services:

- Returns the needed CO's copy for editing and an ObjectHolder (OH) class, which can be aggregated by a bean. The difference between the CO and the OH is that the OH can store inconsistent data. The consistency errors can be asked in human readable format from the object.
- Generates the skeleton of the JAVABean, which contains the OH, the get, and set methods, which delegate the appropriate OH methods. The flag values also are stored as constants.
- Provides the several flag value consolidation into one virtual flag field. The application programmer can give the link with enumeration between virtual and real flag values.
- The ListHolder (LH) class returns the needed list from the object that can be manipulated inside the JAVABean.
- Executes the data validation, and returns the errors from the appropriate OH and LH methods. By giving a list about the fields, which are to be checked, partial validation is also supported.
- The OH and LH provide automatic storing methods so their implementation isn't needed.

To help the user interaction the framework provides more possibilities:

- The framework can return the selectable values of a "finite field" to consider the existing constraints.
- Provides information about field necessity, and mutability.
- An object's JAVABean can store the data from another referenced object. When such data is modified all the dependent fields are reloaded.
- Using the provided tag library the application programmer can easily place the data on the editor page.

4. Error handling

During development even the most precise application programmer makes mistakes. The fast correction of the appeared errors is a critical success factor.

At the time of framework design the error handling and tracing an was emphasised task, which had been solved with the use of sessions [Csiszár, Kókai 2004.1].

In the working system every process is running inside the sessions and all events can be logged. Currently the logs are written into different files separated by users and sessions, thus size of log files cannot be enormous. The log level is adjustable between the two extremes: all event logging, only error logging.

An event log contains the following data:

- The executing object of method.
- Method's name
- Event level (microfinelog, finelog, log, warning, error)

The event logs can be extended with additional objects (list, hash table). Every insert to the event log is performed by the appropriate CO's overridable method.

Application programmers needn't know the detailed log mechanism, but they can create their own application specific log used by that.

4.1. Philosophy of error tracing

Inside the framework every method called checks its parameterisation. If it finds an error during a writing method the opened transaction is rolled back immediately. The reasons of the error and the wrong parameters are written into the event log and actual session's adequate variables. Finally the method returns with null.

In contrast to the general practice after a method's call the returned values will be checked on the caller's side also. (In special situations only the caller's side can check the returned values. For example when a query's result should contain at least one record, than an empty result can be an error free answer from the view of called method.)

The built in exception handling mechanism is used only for those generated by the virtual machine.

Our experiences confirm that the extra works are maximally compensated for error separation. We can distinguish errors caused by users and caught by framework from the framework's own real exceptions.

5. 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 2004.1].

The permanent task is to enhance the efficiency and scalability of our framework.

The most important development is in connection with GUI. The framework should be able to support native interface and to store it in metadatabase. If the structures of GUI are available in metadatabase, then they are connectable with queries.

At the moment the queries defined by graphs don't contain aggregate functions because that complicates the SQL translations. In longer period we'd like to reach the usage of these as calculated fields without storing.

The development of the logging into database will also be advantageous and only one appropriate method should be modified.

References

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

[Casanave] Casanave, Cory Requirement for Roles Revision 1.0 OMG Object & Reference Model Sub-Committee Green Paper

[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 szereportientált modellezés alapjai" V. Országos Objektum Orientált Konferencia Dobogókő2002.

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

[Csiszár, Kókai 2004.1] Csiszár, Tibor - Kókai, Tamás The basic services of an object-role oriented business framework Proc. of the 6^{th} International Conference on Applied Informatics Eger, Hungary Jan. 27-31, 2004

[Fábián 2002] Fábián, Gergely Szereporientált futtatókörnyezet, Diplomamunka ELTE TTK 2002

[Fowler 1997] Fowler, Martin Dealing with Roles Proc.of the 4th Annual Conference on the Pattern Languages of Programs, Monticello, Illinois, USA, Sept. 2-5, 1997 (`www.martinfowler.com/apsupp/roles.pdf`)

[Fowler 2000] Fowler, Martin UML Distilled Second Edition, Addison-Wesley, 2000

[Graham, Simons 1998] Graham, Ian, Simons, Anthony J H 37 Things that Don't Work in Object-Oriented Modelling with UML ECOOP'98 WS pp.209-232

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

[Hornyik 2002] Hornyik, Katalin Szereporientált elemzés és tervezés, Diplomamunka ELTE TTK 2002

[Mili 1998] Mili, F. On the Formalization of Business Rules ECOOP'98 WS pp.122-129

[Wieringa 1998] Wieringa, Roel A Survey of Structured and Object-Oriented Software Specification Methods and Techniques ACM Computing Surveys, Vol. 30, No.4, pp.459-527

Postal addresses

Tibor Csiszár
INITON Ltd.
26/a, Rahó utca, 1118 Budapest
Hungary

Tamás Kókai
INITON Ltd.
26/a, Rahó utca, 1118 Budapest
Hungary