Object-Oriented Design for 4GL
to smooth the Path to Business Information Systems

Elke Hochmüller, Claudia Kohl
Institut für Informatik
Universität Klagenfurt

Abstract

Modern 4GL systems tend to turn into really object-oriented environments which seem to be suited quite well to realize business information systems (BIS). BIS should support the accomplishment of daily enterprise work. Such systems are required to handle a huge volume of data, assure data integrity, cope with high input/output intensity, incorporate transaction management and provide enduser-oriented interfaces.

Existing object-oriented analysis methods fit rather well to the purpose of BIS development. Currently available object-oriented design methods, however, do not properly take into account the typical BIS characteristics. This paper proposes a powerful and flexible generic object-oriented process model for BIS development with 4GL. This process model particularly focuses on the adjustment of object-oriented design activities to the context of BIS development in connection with 4GL and on its incorporation into the overall object-oriented BIS life cycle. The applicability of the proposed process model will be demonstrated by some examples using CA-OpenROAD as underlying 4GL technology.

1. Motivation

In contrast to mere technical-scientific applications which focus on algorithmic complexity, business information systems (BIS) are rather characterized by the predominance of large date management, the importance of the graphical user interface, and decentralized user access. These characteristics typical to business information systems have to be paid attention to in the whole system development process and particularly during the design phase.

Today's market offers a variety of tools and environments using a fourth generation language (4GL) that enables high level programming together with integrated database access facilities. These tools enable the realization of user interfaces in a very simple, graphical manner. Most of these tools follow the principle of object-orientation. Hence, it is straightforward to embed the application development within an object-oriented system life cycle.

As most existing object-oriented design methods are very general and do not tackle BIS characteristics in particular, we propose an object-oriented process model for BIS development via 4GL which especially tries to overcome these design deficiencies. However, this process model should not be regarded as a new modeling technique but rather a set of guidelines of how to proceed during the development process of a BIS independently of the actual selected object modeling technique.

Section 2 investigates current object-oriented analysis and design approaches according to their applicability for BIS development and identifies BIS specific challenges. Section 3 gives an overview of 4GL systems in general and of CA-OpenROAD in particular. The object-oriented process model for BIS with 4GL will be presented in section 4. Some examples for the application possibilities of this process model will be outlined in section 5 by using CA-OpenROAD as 4GL environment.

2. Object-oriented BIS Development

Regarding the sequence and objectives of main development activities, the object-oriented life cycle in fact did evolve from the traditional software life cycle. Hence, the main levels of object-oriented development are:

- object-oriented analysis
- object-oriented design
- (object-oriented or conventional) implementation

The main difference between the traditional and the object-oriented software development life cycle consists in the fact that the latter focuses on the object as principal concept. All object-oriented life cycle activities share a common vocabulary as well as a common notation. Object-oriented development tasks include the identification, documentation, and implementation (often by means of prototyping) of objects and their abstractions by using the principles of encapsulation, aggregation, generalization, and inheritance.

Object-oriented Analysis

The development process of an object-oriented application system generally starts with the analysis of the relevant parts of the real world (the so-called problem domain) and results in a formal requirements model. During analysis, the user requirements regarding the target system are elicited, the structure and semantics of business-related objects, their relationships and their behavior are documented.

There are a lot of object-oriented analysis approaches (c.f. [CoYo91a], [RBP+91], [ShMe88], [ShMe92]) which use slightly different notations but propose very similar analysis tasks to be carried out to obtain the problem domain model (PDM). In general, object-oriented analysis activities can be attached to one of two mainstream task families which differ according to their way of tackling the problem domain: static analysis (which deals with the description of the static structure of objects and their relationships) and dynamic analysis (which is concerned with the behavior of the objects).

If we consider the Object Modeling Technique (OMT) methodology [RBP+91], the static analysis tasks include the definition of a so-called object model which is represented by object diagrams consisting of object classes and their relationships, and the dynamic analysis activities result in the definition of a dynamic model with state diagrams as main means of representation.

Object-oriented Design

Object-oriented design deals with the mapping of the problem domain model onto the target system. Existing object-oriented design methods, however, vary substantially in their proposed design tasks (c.f.[Booc91], [CoYo91b], [RBP+91]). Usually, the various object-oriented design approaches enrich the analyzed object model with additional classes and objects, but in a different manner and with varying accuracy. Such augmentations are necessary to realize application category specific tasks like enduser-oriented interfaces, extensive data storage, algorithmic complexity, and control of particular system activities.

A common problem with these methods consists in their effort to stay universally applicable, irrespectively of the actual application category. Thus, proposals for design tasks usually remain at a rather high level of abstraction which is of little help in detailed design activities where technical factors (like the kind of operating system, programming language, graphical user interface, and database management system) need to be obeyed.

Another problem of object-oriented analysis and design lies in the usage of a common notation leading to a rather blurred transition from analysis and design which - at a first glance - appears as a solution to the structural clashes on the boundary between these two phases in traditional software development. However, we should also take into account that this notationally caused blurred transition calls for a sophisticated process model describing development tasks and attaching them to the stages of software development in order to prevent from being lost in the 'analysis-design object-space'.

Object-oriented Analysis and Design for BIS

Existing object-oriented analysis methods fit rather well to the purpose of BIS development. This might mainly be due to the facts that object-oriented analysis focuses on modeling real-world objects (what is also true for analyzing the problem domain in case of business information systems) and that the static part of most object-oriented analysis methods evolved from conceptual data modeling. Thus, the object model can be regarded as the object-oriented counterpart of the conceptual data model.

The object-oriented design methods proposed so far, however, are mainly inadequate for carrying out the design activities necessary for BIS development. As already mentioned, most methods try to be applicable in all development situations and often ignore the fact that design activities have to take into account the requirements which are particular to the actual application category (e.g. be it a business information system with huge volumes of data, or be it a mere technical system with special algorithmic challenges).

Object-oriented design activities for business information systems should particularly address object management on the one hand, and the (preferably graphical) user interface on the other hand. Design from the object management perspective should deal with the planning of object visibility, multiple user access, data persistence, transaction management, integrity constraints, and external interface requirements. Graphical user interface design should deal with presentation and interaction mechanisms (i.e. static and dynamic layout requirements) by obeying user requirements (according their types and profiles). There exist some proposals for graphical user interface design in the literature (c.f. [Coll95], [CoWa93],[Shne92]) following an object-oriented approach, but some of them suffer from being too short-sighted by largely neglecting the connection to object management design whereas others are too tightly coupled to the selected object management repository.

3. BIS realization via 4GL-Systems

Fourth generation language (4GL) systems cannot be commonly defined. There exists a multitude of languages and systems, focusing on very special application categories. They span a wide area from enduser-tools like graphical database query tools and enduser query-tools to database application development environments and even database application generators [Balz96][Wegs97]. In general, these systems offer a very high level language and thus increase the productivity of programmers.

In the content of this paper with the aim of developing BIS we will make a restriction to 4GL systems that aim to build business information systems as data-intensive, interactive applications in an object-oriented manner. This 4GL application architecture family generally disposes of 4GL languages which are less procedural than 3GL languages, have DDL and DML constructs integrated in the 4GL language, offer a simple realization of the human interface (mostly in form of so-called frames) and have language constructs for frequently used tasks. As an representative of this 4GL application architecture family we will consider CA-OpenROAD and its components to build a BIS [CAOR94].

CA-OpenROAD was constructed to be used for the development of database applications. It is a window-oriented tool with a visual programming environment for editing, compiling and testing database-oriented applications.

An application appears as a sequence or hierarchy of frames, which are to be filled with data and/or present data. Through these frames a user can trigger different kinds of transactions in the system regarding the respective problem domain. CA-OpenROAD realizes a WYSIWYG interactive user interface development environment. It offers WINDOWS-4GL as a procedural event based language, which is type-equivalent with SQL. SQL statements can therefore be easily integrated.

CA-OpenROAD supports object-oriented concepts, too. There exist about 170 predefined *system classes* related in a certain inheritance hierarchy, which can mainly be used for the implementation of the user interface (e.g. a frame is a predefined system class) or for error-handling. System classes cannot be changed by the programmer.

Additionally, so-called *user classes* enable the possibility to realize user-defined classes. These user classes are hooked into the system class hierarchy at a certain node. Regarding the degree of object-orientation, CA-OpenROAD's classes are restricted in the sense as they are not treated as objects. This is also the reason why CA-OpenROAD classes only can specify instance methods but no class methods. If class methods are needed to administrate all instances of a class, a special class-user class has to be specified and implemented. Also the inheritance and rewriting of attributes and methods works in a strange manner. Multiple inheritance is not supported.

4. Object-oriented Process Model for BIS with 4GL

Before we introduce our proposal for an object-oriented process model to develop a BIS by means of a 4GL environment, we honestly concede that we would promise too much if we pretend to have found a fully fledged model which takes into account all the particular facilities provided by every single development tool on the market. By way of contrast, our proposed process model has to be considered as a meta model to help as a general guideline in its first place. Nevertheless, it is powerful and flexible enough to be refined and instanciated according to a given development situation concerning the object-oriented analysis method to be applied, the actual presentation strategy, the selected 4GL architecture as well as the general type of the underlying database system, and subsequently the finally used 4GL environment and database system, respectively.

As can be seen from Fig.1, the process model follows the already introduced object-oriented life cycle and consists of three main (object-oriented) process levels. The primary goal of this process model is to continuously sustain the encapsulation principle between the different process levels to fully exploit one of the main advantages of object-oriented development. Thus, a major independence between the three levels can be achieved leading to increased degrees of maintainability as well as portability.

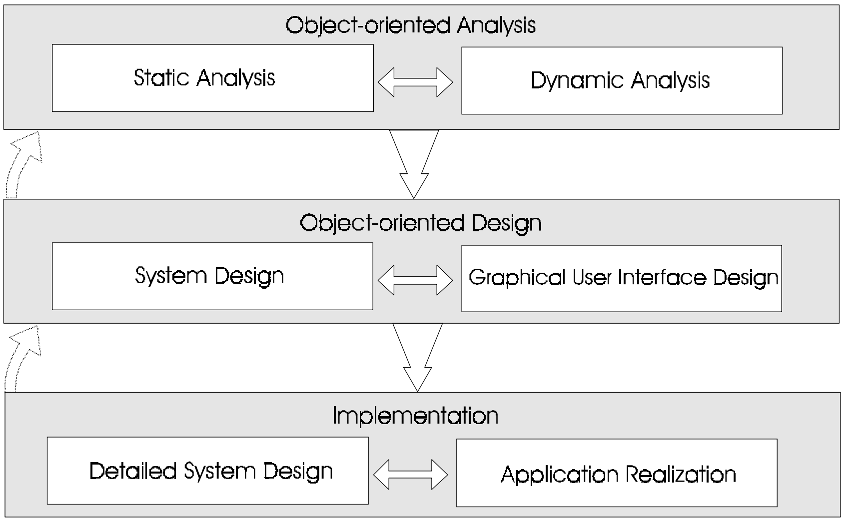

Fig.1: Object-oriented Process Model for BIS

In the sequel, the major phases of the three process levels will be further described using a general outline by initially stating the overall goal of each phase which is followed by a list of activities and their consumed input and produced output.

The individual phases can not be considered to be independent of each other. In Fig.1, the actual connections and the respective directions of the information flows are represented as arcs. In general, phase products will be used as input on subsequent process levels. Additionally, the arcs between phases located at the same level symbolize the iterative nature of those phases. Hence, preliminary phase product parts will be passed over to the sibling phase, and vice versa. Thus, the activities of each phase can be carried out by concentrating on the phase specific tasks without loosing contact to the parallel phase. In such a way the ongoing work can be supervised to stay within the boundaries common to both phases on the same process level.

4.1. Object-oriented Analysis

The topmost process level deals with the analysis of the problem domain. For this purpose we propose an object-oriented strategy like OMT. According to the state-of-the-practice in object-oriented analysis, we account for the two major viewpoints in analyzing the problem domain. Thus, we distinguish between two phases, one of which is concerned with static analysis of the problem domain, the other one focuses on the analysis of the problem domain dynamics.

Although both phases will be separately introduced in the sequel, we should always keep in mind, that especially these two phases cannot (and should not) be regarded in isolation. Moreover, results of one phase can influence the outcomings of the other one, and vice versa. In fact, both phases have the problem domain in their center of interest. They are just tackling the same problem domain from a different point of view. Hence, the two phases do not compete each other but, on the contrary, should be used to control overall consistency and completeness.

It is common practice to initially start analysis with just one of the phases depending on the actually applied object-oriented analysis approach (e.g. static analysis in case of OMT). On the availability of the first useful results, the other phase will be entered and the iteration will take its course.

Both phases proceed from a common externally originated input which constitutes the real world in the problem domain. This input can be obtained either by analyzing existing information or by eliciting additional information in a rather dynamic manner through different kinds of communication.

The output of both phases together is the problem domain model which will consist of a static and a dynamic part.

a) Static Analysis

The main goal of static analysis is to get a full understanding and a detailed description of problem domain statics.

- *input:*
  - dynamically obtained from users (domain experts, clients):
    - interviews (structured, open-ended)
  - static analysis of available documents:
    - reports
    - existing system descriptions
    - environment descriptions
    - task descriptions
- *tasks:*
  - identification of inherent and transaction classes and objects
  - application of abstraction mechanisms (aggregation, generalization)
  - identification of relationships between problem domain classes
  - identification of attributes
  - identification of methods
- *output:*
  - problem domain model - static view:
    - object schema
    - description of problem domain classes

b) Dynamic Analysis

Dynamic analysis aims at the identification of potential users (or user classes) and at the achievement of a full understanding and a detailed description of problem domain dynamics.

- *input:*
  - dynamically obtained from users (domain experts, clients):

- interviews (structured, open-ended)
- observations (Video, on-site observation)
- prototyping

static analysis of available documents:

- reports
- existing system descriptions
- environment descriptions
- task descriptions

• *tasks:*

identification of potential users
identification of different user profiles
identification of core task families
identification of tasks within each task family
relationship of tasks to problem domain classes
identification of transaction classes
structuring of tasks into atomic subcomponents

• *output:*

problem domain model - dynamic view:

- user types and profiles
- scenarios
- event traces
- state transition diagrams per problem domain class
- frame sketches

4.2. Object-oriented Design

The second process level is concerned with object-oriented design. In proposing a phase dealing with the internal system architecture and a phase focusing on the design of the graphical user interface, we follow the commonly accepted distinction between external and internal design [Coll95].

Input of both phases are not only the products received by the preceding analysis activities, but also additional information from the users, about interface requirements to other systems or about the architecture of the target system itself.

The main product of the object-oriented design level as a whole is the so-called application model which consists of the (probably revised) problem domain model, the system model (as a result of the system design phase) and the presentation model (delivered by graphical user interface design).

In order to be able to decide about the overall presentation strategy for the external design, the kind of representation should be definitely selected when starting with graphical user interface design. We distinguish between three different kinds of presentation strategies:

• data-oriented: table-driven presentation
• task-oriented: function-driven presentation
• object-oriented: object-oriented presentation (data-method combinations)

For internal design, however, the target database architecture (e.g. relational, object-oriented) needs to be captured as it will affect the system model. Nevertheless, the system design itself will be carried out in an object-oriented manner. Hence, additional control classes (i.e. classes with no attributes but only methods which are bunched together in a toolbox-like fashion [CoYo91a]) will be defined; especially in case of an underlying relational database architecture, so-called database classes will have to be established to maintain the encapsulation principle.

a) System Design

The goal of the system design phase is to explore and get an appropriate model of the internal system architecture.

- *input:*

  problem domain model
  external:
  - interface requirements to other (existing) systems
  - persistency requirements to be elicited from users
  - quality constraints to be elicited from users
  - integrity constraints (e.g. primary & foreign keys, additional constraints)

  internal:
  - target database architecture (e.g. OODBMS, RDBMS)
  - general technical environment constraints

- *tasks:*

  identification of persistent and transient problem domain classes
  development of logical data model
  - according to target database architecture
  - obeying both quality and integrity constraints

  identification of system classes (database and control classes) and their dynamics

- *output:*

  system architecture:
  - revised problem domain model
  - logical data model
  - system model (including system class definitions)

b) Graphical User Interface Design

Graphical user interface design aims at the exploration and establishment of an appropriate user interface model.

- *input:*

  (revised) problem domain model
  user types and profiles
  frame sketches
  external:
  - interface requirements as elicited from users

  internal:

- system dependent style guides

- *tasks:*
  - decision about presentation strategy (data-, task- or object-oriented)
  - definition of common look and feel (general frame style)
  - definition of frame hierarchy
  - refinement of frame sketches according to actual presentation strategy
  - prototyping and evaluation of frame layout
  - definition of frame layout according to
    - user types and profiles
    - actual presentation strategy
  - definition of frame dynamics (interaction mechanisms):
    - state transition diagrams
    - transaction concept

- *output:*
  - static presentation model:
    - general frame style
    - frame hierarchy
    - frame layout
  - dynamic presentation model:
    - a state transition diagram per frame
    - transaction concept (transaction boundaries within frames)

4.3. Implementation

The third and final process level concentrates on the realization of the final system which consists of a database holding persistent objects and one or more application systems (depending on the amount of different user types and the diversification of their tasks). This is the level, where a definite decision about the target database system and the actual 4GL environment has to take place.

By distinguishing between system design at the OOD level and detailed system design at the implementation level, we can maintain the object-oriented manner of design until implementation. Thus, the exchange of actual database systems with the same underlying architecture will only affect the detailed system design phase achieving a major independence between design and implementation.

In case of an RDBMS, we have to provide a mapping of the object-oriented system design model onto the relational database model. This will be realized within the code of database class methods through which encapsulated database access will be provided.

The products of the application realization phase will be constructed on basis of the already developed presentation model and the development possibilities provided by the selected 4GL environment.

a) Detailed System Design

In case of a business information system as target system, detailed system design mainly comprises database design activities. Its main goal is to produce the database definitions for the target database system.

- *input:*
  - logical data model
  - system class definitions
  - external:
    - expected quantity of data
    - performance requirements
  - internal:
    - characteristics of actual target database system
    - specific technical environment constraints

- *tasks:*
  - definition of database tables according to database classes
  - definition of primary keys, mandatory attributes, and indexes
  - definition of constraints (foreign keys, integrity, usage)
  - physical database design
  - definition of user privileges
  - implementation of system class methods

- *output:*
  - database definitions for target database system
  - system class methods

b) Application Realization

The main objective of application realization is the implementation of application components in order to obtain the final application system(s).

- *input:*
  - (revised) problem domain classes
  - system class definitions
  - static and dynamic presentation model
- *tasks:*
  - mapping design constructs onto the actual 4GL system:
    - frame realization (final graphical representation, properties, biases, ...)
    - frame scripts
    - additional 4GL system specific components (e.g. user classes)
    - code optimization
  - assembling frames into application system(s)
- *output:*
  - 4GL components (frames, scripts, 4GL procedures, ...)
  - complete application system(s)

5. Selected examples from a particular instance model

As shown in the previous section, the final output of the application of the proposed object-oriented process model for BIS is dependent on the object oriented analysis method, the presentation strategy, the target database architecture, the target database system, the 4GL architecture as well as the actual 4GL environment - factors which are influencing the process products on different process levels. In this section some small examples of development outputs are presented.

In order to demonstrate the applicability of the proposed process model, we assume the following specializations:

- object-oriented analysis method: OMT
- presentation strategy: object-oriented
- database architecture: relational
- database system: Ingres
- 4GL architecture: object-oriented
- 4GL environment: CA-OpenROAD

The example does not cover the entire process but at least one possible output per development phase.

5.1. Static Analysis

The considered problem domain concerns the administration of employees and their projects within a fictive enterprise. Fig.2 represents an OMT object schema as part of an actually more comprehensive problem domain model.

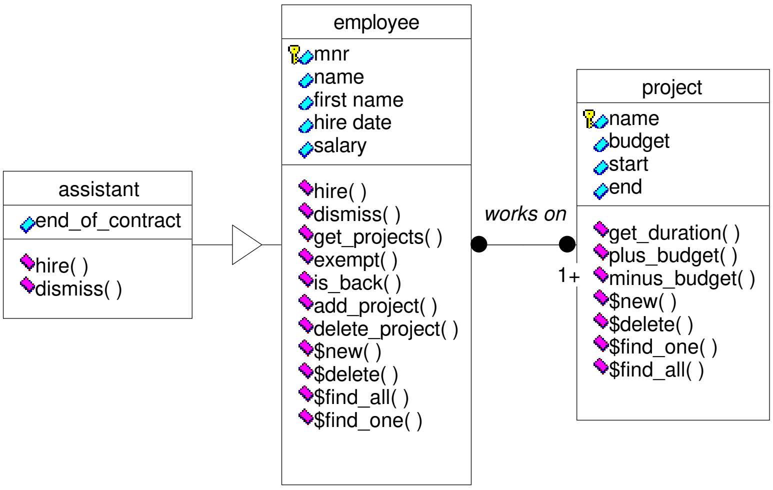

Fig.2 : Object schema of the problem domain

The object schema consists of the classes *project*, *employee* and its specialization *assistant*. A leading $ symbolizes class operations. Unique attributes are characterized by a key-symbol. Assistants are employees with a temporary employee contract. Thus, their operations *hire* and *dismiss* work in a different way.

5.2 Dynamic Analysis

The behavior of an individual class is part of the problem domain dynamics. Fig.3 shows the state transition diagram of class *employee*. The events triggering a state transition (e.g. hire, dismiss) can be found also as operations in the object schema of Fig.3 .

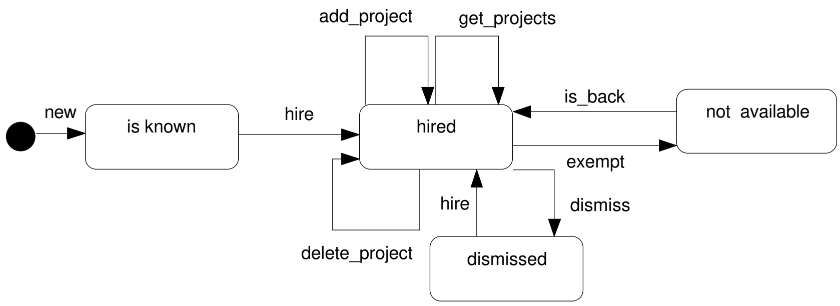

Fig.3: State transition diagram of class employee

5.3 Graphical User Interface Design

Corresponding to the problem domain statics and dynamics a frame hierarchy and individual frame layouts and dynamic designs have to be developed. Fig.4 presents the frame layout of the frame *administrate employees*. The state transition diagram of *administrate employees* can be seen in Fig.5 .

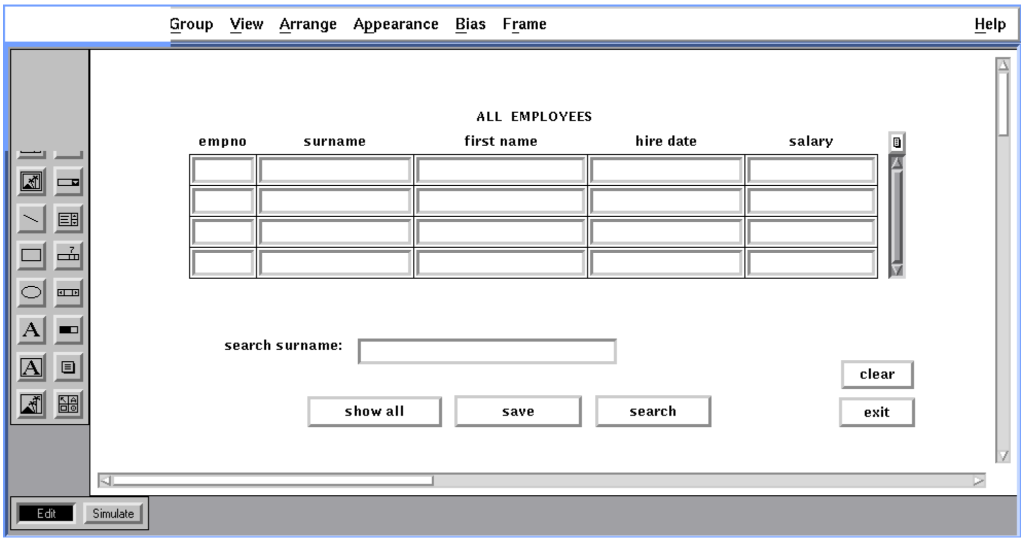

Fig.4: Frame layout *administrate employees*

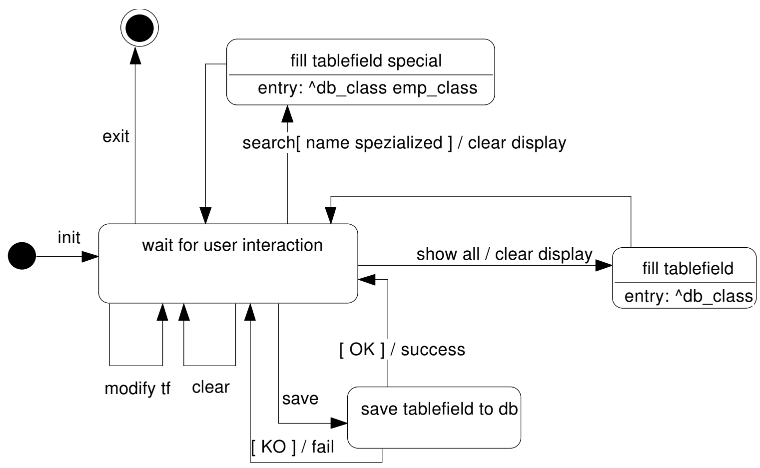

Fig.5: Dynamics of frame *administrate employees*

5.4 System Design

During this process level it becomes clear that all three PDM classes have to be made persistent. The attributes of the PDM classes can be stored in a relational scheme containing the tables *employee, works, project* and *assistant*. In order to achieve a loosely coupled access to the relational database, three database classes are identified: *emp_db* , *pro_db* and *assi_db,* which should realize an encapsulated relational database access. This system design assures that only the database classes have to be modified in case of a database exchange. Thus, the remaining system architecture will not be affected.

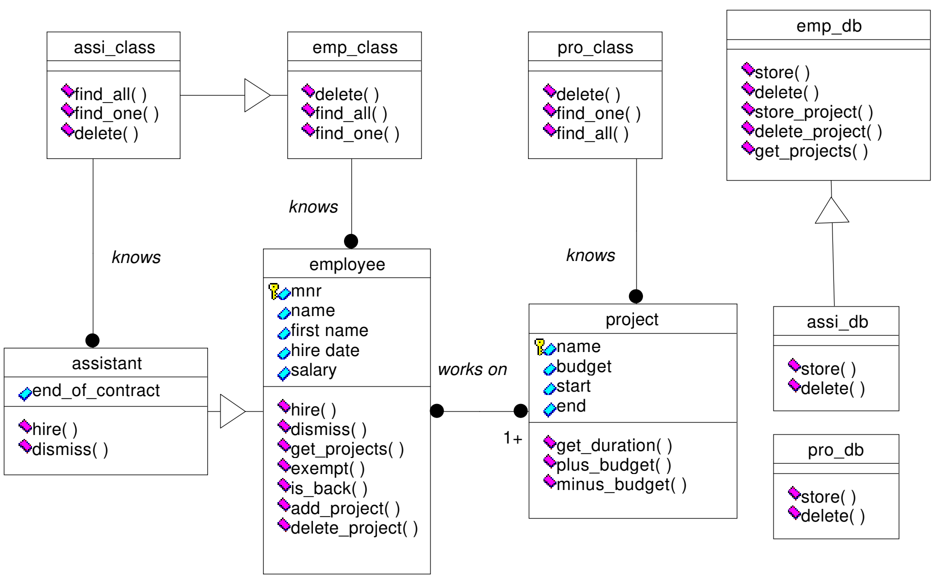

Fig.6 : Revised PDM

Since in the considered 4GL architecture classes are not treated as objects and therefore no class operations can be defined, it is necessary to revise the PDM. Classes as object repositories have to be modeled as classes themselves (e.g. *emp_class* is a collection of employees and offers respective operations that have been modeled as class operations in the original PDM before). Since this class class operations are all database access operations the class classes are treated as database classes, too. Fig.6 presents the revised PDM.

5.5 Detailed System Design

One task of the detailed system design is to implement the database classes in CA-OpenROAD. The mapping of *emp_class* to CA-OpenROAD results in a user class *emp_class* with one attribute *emps*, holding a reference to all employees as a repository. Fig.7 shows the 4GL implementation through CA-OpenROAD of the operation *find_all* of the user class *emp_class*. *Find_all* has to access the Ingres database and loads data about all existing employees.

```
method find_all()=
DECLARE
  i = integer NOT NULL;
ENDDECLARE
BEGIN
  CurObject.emps.Clear();    /* CurObject references the Object itself ( like self) */
  i=1;
  SELECT empno AS :CurObject.emps[i].empno,  name AS :CurObject.emps[i].surname,
            fname AS :CurObject.emps[i].firstname , hire_dat AS :CurObject.emps[i].hire_dat ,
            salary AS :CurObject.emps[i].salary
  FROM emps
  BEGIN
    i=i+1;
  END;
  IF iirowcount=0 THEN Message 'no data available'; ENDIF;
  RETURN CurObject.emps;
END;
```

Fig.7: method implementation of *find_all*

5.6 Application Realization

As an example for the realization of a frame Fig.8 shows the essential parts of the CA-OpenROAD 4GL frame script for the frame *administrate employees*.

```
INITIALIZE ()=
DECLARE
          empclass = emp_class;          /* emp_class is a CA-OpenROAD user class realizing the employee
class*/
          err = integer NOT NULL;
ENDDECLARE
BEGIN
END;

ON CLICK show =
BEGIN
          employee_tf.clear();            /* employee.tf is the name of the tablefield in the frame layout */
          employee_tf=empclass.find_all(); /* call method find_all and show results in the tablefield */
END;

ON CLICK save =
BEGIN
         err = empclass.save_all();   /* call method to save possible changes in the tablefield onto the database*/
         IF err !=0 THEN ROLLBACK; */ close transaction */
         ELSE COMMIT;
         ENDIF;
```

```
END;

ON CLICK search =                    /* display only employees with a certain name in the tablefield */
BEGIN
        employee_tf.clear();
        employee_tf=empclass.get_one(name = namefield);
END;
```

Fig.8: implementation of frame *administrate employees*

6. Conclusion

Object-oriented design cannot be carried out without paying adequate attention to the kind of application category of the actual system to be developed. This paper proposes an object-oriented process model with special focus on design activities that should serve as guidelines in case of one particular application category, namely that of business information systems. Special emphasis is paid to the two major parts in object-oriented design, the phases of system design and graphical user interface design. The activities on the object-oriented design level are not considered in an isolated manner, but put into relation to the preceding level of object-oriented analysis (producing some of its inputs) and the subsequent level of implementation (consuming and refining its outputs). The presented approach should support the BIS development team in handling design problem patterns typical to business information systems in a systematic way dependent on the target 4 GL architecture.

References

[Balz96] H. Balzert: “From OOA to GUIs: The Janus System“, JOOP, Feb 1996, pp 43-47

[Booc91] G. Booch: "Object-Oriented Design with Applications", Benjamin/Cummings, 1991

[CAOR94] CA-OpenROAD Programming Guide, Language Reference Manual, Application Editors User’s Guide Release 3.0, Computer Associates International Inc., 1994

[Coll95] D. Collins: "Designing Object-Oriented User Interfaces, Benjamin/Cummings, 1995

[CoWa93] K. Cox, D. Walker: "User-Interface Design", Prentice Hall, 2nd ed., 1993

[CoYo91a] P. Coad, E. Yourdon: "Object-Oriented Analysis", Prentice Hall, 2nd ed., 1991

[CoYo91b] P. Coad, E. Yourdon: "Object-Oriented Design", Yourdon Press, 1991

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: "Object-Orented Modeling and Design", Prentice Hall, 1991

[ShMe88] S. Shlaer, S.J. Mellor: "Object-oriented Systems Analysis: Modeling the World in Data", Prent. Hall, 1988

[ShMe92] S. Shlaer, S.J. Mellor: "Object Lifecycles: Modeling the World in States", Prentice Hall, 1992

[Shne92] B. Shneiderman: "Designing the User Interface - Strategies for Effective Human-Computer Interaction", Addison Wesley, 2nd ed., 1992

[Wegs97] E. Wegscheider: "Toward Code-Free Business Application Development", IEEE Computer, Vol. 30, No. 3, 1997, pp. 35-43

[Your94] E. Yourdon: "Object-oriented Systems Design - An Integrated Approach", Prentice Hall, 1994