



ELSEVIER

Knowledge-Based Systems 16 (2003) 77–89

Knowledge-Based
SYSTEMS

www.elsevier.com/locate/knosys

The BORM methodology: a third-generation fully object-oriented methodology

Roger Knott^{a,*}, Vojtech Merunka^b, Jiri Polak^c

^a*Department of Computer Science, 14 Haslegrave Building, Loughborough University, Loughborough LE11 3TU, UK*

^b*Department of Information Engineering, University of Agriculture, Prague, Czech Republic*

^c*Management Consultant, Deloitte and Touche, Prague, Czech Republic*

Received 22 January 1999; revised 29 May 2002; accepted 22 July 2002

Abstract

Business object relationship modelling (BORM) is a development methodology developed to capture Knowledge of typical business systems. It has been in development since 1993 and has proved an increasingly effective method which is popular with both users and developers. The effectiveness gained is largely as a result of a unified and simple method for presenting all aspects of the relevant model. This paper outlines BORM, its tools, methods and its differences from other similar development methodologies.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Business object relationship modelling; Prototype; Class refactoring

1. Introduction

The attitude of business towards information technology (IT) is constantly changing as more and more sophisticated systems and tools become available. Additionally, there is a constant exchange of ideas between the IT and the business communities arising out of the development of knowledge-based systems. One such example is the method presented here, which was originally developed to capture knowledge necessary for the development of IT systems but which has revealed increasing potential for more general knowledge based system development. The work described here, originally started in 1993 and was intended to provide seamless support for the building of object-oriented software systems based on pure object-oriented languages, such as ‘Smalltalk’, together with pure object databases, such as ‘Gemstone’. It is now realised that this method also has significant potential in capturing knowledge of business processes, business data and business issues.

Our experience throughout the 1990’s, when we worked on major projects in information systems was that analysts faced the problem that not all system requirements

were known at the commencement of the project. Moreover, the customer expects that discovery and refinement of requirements was an integral part of the project.

The problems of system development have additional complications as a consequence of the introduction of, or significant changes to, any major systems. Such changes have significant impact on all the organizational and management structure of the company or organization where the system is implemented—such as new or modified job positions, management changes, new positions, new departments, etc. Therefore it is desirable during the development of information systems to also address the change in these related structures.

Process models composed from business objects represent a proven and widely used method of analysis, design and implementation for organizational changes. Such models are developed with the active participation of the customers and often form the early stages in the development of information system.

The methods described here have been used to successfully develop a wide range of systems of diverse sizes.

In particular

- To identify business processes in Prague city hospitals as a prerequisite to further cost analysis;

* Corresponding author. Tel.: +44-1509222938; fax: +44-1509211568.

E-mail address: r.p.knott@lboro.ac.uk (R. Knott).

- To model all necessary properties of the general agricultural commodities wholesale sector in the Czech Republic;
- For business process re-engineering in the electricity supply industry;
- For telecommunication network management in the Czech Republic.

Experience with these practical projects suggests that the description of business environments necessary to prompt new activities involve significant knowledge based content.

The systems mentioned above range through all sizes of software development as can be seen from Table 1.

The method presented in this paper, have proved to be effective and beneficiary in the process of describing and subsequently understanding how real business systems evolve. Such knowledge is the key for the success of any business and is especially crucial for those employees whose responsibility is business development.

The business object relationship modelling (BORM) method provides a tool to capture knowledge and present it in a way that, the authors believe, is far more effective than other business processes, data, or functional modelling methods [2–7]. This increase in effectiveness is due largely to the use of a unified and simple method for presenting all aspects of the relevant model.

To fully understand how a business works requires significant detailed knowledge, which can be captured in a knowledge base. BORM provides an effective tool to facilitate knowledge elucidation and an effective graphical language for describing the structures and interactions within the knowledge so elicited.

One problem, common for both the designers of information systems and knowledge elucidation, is the ‘conceptual gap’ or ‘representational mismatch’, which

occurs when communicating with a user or knowledge expert.

Goldberg [8] uses the term ‘concept space’ to describe what the user/experts believe, assumes or knows to be the case. The ‘articulation space’ is what the expert/user communicates in response to the analyst’s questions. The analyst then constructs a model to feed back to the user/expert their mental model of the concept space, which they construct out of the information presented in the articulation space. This model is known as the ‘analyst space’. The difference between this analyst’s model and the user space is the concept gap.

To a certain extent, part of this gap is unbridgeable; we cannot easily reduce the gap between concept and articulation space, as these exist in the user/expert’s head. It is true, however, that the languages, natural and graphical, used by the analyst in representing this model are a vital component in the user/expert’s ability to validate this model against her own concept space. The subset of BORM methods introduced here is, we believe, a useful tool in closing this gap.

2. What are the problems with object-oriented design methodologies?

The first and we think the major problem with object-oriented methodologies arises in the initial stages of system development cycle [9–13]. The initial stage of any object-oriented design methodologies should be concerned with two tasks. The first is the specification of the requirements for the system. The second is the construction of an initial object model, often called an *essential object* or *conceptual model* built out of a set of domain specific objects known as essential objects. Both these tasks should be carried out with the active participation of the stakeholders, in order to

Table 1
Metrics for sample system developed with BORM methodology

Project	NSF	NS	NPD	NO	ANS	ANA
National agrarian chamber (analysis and design of software for fruit market public information system)	4	7	7	6	4	4
Hospital complex (BPR of organization structure)	6	12	12	8	10	12
TV and radio broadcasting company (BPR and company transformation for open market)	4	9	9	14	8	8
Regional electricity distribution company (customer information system analysis)	12	19	19	23	12	12
Regional electricity distribution company (failure handling information system analysis and prototype implementation)	19	31	34	27	13	14
Regional gas distribution company (BPR of all company)	28	81	97	210	11	12
Regional gas distribution company (BPR of all company)	23	60	63	120	12	12

NSF, number of system functions; NS, number of scenarios; NPD, number of process diagrams; NO, number of objects (participants) (only objects having activities, objects realizing data flows in processes are not included here); ANS, average number of states per object; ANA, average number of activities (in BPR projects, each activity includes about 4–6 additional business related entities (goal, job position, success factor, etc.) per object.

ensure that the correct system is being developed. Consequently, any tools or diagrams used at these early stages should be meaningful to the stakeholders, many of whom are not ‘computer system literate’.

The most common technique for requirements specification in current object-oriented methodologies is Use Case modelling. Indeed Use Cases are often the foundation of most object-oriented development methods [20]. Use Case modelling is concerned with the identification of actors, which are external entities, who interact with the system. This means that in order to employ Use Case modelling, it is necessary for developers to already know the system boundary and distinguish between entities, which are internal and external to that boundary. It is our experience that the correct identification of the system boundary is a non-trivial task and can only take place at the end of the requirements specification stage.

Use Case modelling is essentially a text-based system, any diagrams employed do not contain any significant information but only identify the actors involved in each Use Case. Neither is it an object-oriented process as the Use Cases determined could be subsequently developed in any programming paradigm. In Addition, Use Case Modelling is often insufficient by itself to fully support the depths required for initial system specification. Fowler [21] highlights some deficiencies in the Use Case approach, suggesting that Use Case diagrams if they are to convey all the necessary information need supplementation by Sequence and Activity diagrams. These modifications to Use Case Analysis would result in a number of different diagrams that are used initially to define the interaction between any proposed system and its users. There are many views on the effectiveness of Use Cases as a first stage in System Design. Simone and Graham [22] for example describe a situation where Use Case Modelling obscures the true business logic of a system.

The BORM approach is based on the fundamental concept of process modelling [1]. This follows from the belief that it is necessary, for the deployment of a new system not to view that system in isolation, but to view it in the context of the companies total organizational environment. A new system, when introduced into an organisation will normally totally change the way that the organisation operates. In addition, a BORM process model is object oriented from the beginning and is defined in easy to understand graphical notation. From the process model scenarios can be developed. Scenarios were originally developed in Object Behaviour Analysis (OBA) to capture similar information to that presented in Use Cases. A Scenario however is an *instance* of a User Interaction whereas a Use Case is more like a procedural description of a *type* of user interaction. Our experiences on the projects listed above suggest that the process way of thinking is more natural to business employee. Consequently, stakeholders in the proposed system can more easily understand BORM models and consequently make a greater contribution to the correctness of the system design.

In BORM any initial diagram supports only problem domain specific concepts; any software-orientated concepts are left until later in the modelling process. In Addition, in the early stages BORM uses a single diagram that embodies the same information as the numerous diagrams used by other methodologies. This is an attempt to make it easier for the user to form a complete understanding of the interaction of the various system components.

In BORM concepts and their notation change as the development process proceeds. This is in sharp contrast with UML, which claims to be a universal system in that the same notation is used for analysis, design and documenting the implementation. Our reasons for changing notation is based on the observation that this universality of the UML’s notation hinders the design process. In this we are in broad agreement with the criticism of this aspect of UML expressed by Simone and Graham [22].

The second problem that we find with most development methodologies is that, during subsequent stages, they require a number of different diagrams to fully describe the system. Each diagram is used to model an independent aspect of the system. Thus, we have one diagram for the objects static structure and a second for the state changes of particular objects. One diagram showing the message passing between objects and a further diagram to model the activities the system must perform. The fundamental principle of object-oriented systems is one of encapsulation. This means that all an object’s data values are stored in the same place as the functions (methods) that manipulate them. The synergy created by this unification of data and functionality leads to greater understanding of the situation being modelled, and, to a correctly designed solution being developed.

Each diagram is a visual representation of an abstract model, which exists in the brain of the analyst. Developers use diagrams to communicate with customer and other designers. They are the basis of the *Analyst Space* defined earlier. If this model is object oriented in nature, its representation should reflect it and must not require the viewer to deduce this fact from a number of different diagrams, each of which reveals one particular aspect of the object nature of the model.

Finally, the modelling concepts used in most development methodologies are used throughout the system development cycle. Moreover these notations tend to be specifically designed to represent, at an abstract level, concepts from object-oriented programming languages.

For example, the models used in OMT [14,15] or UML [16,17] can use quantifiers, link classes, external relations, aggregations, etc. While many of these concepts, are necessary for software implementation in hybrid object-oriented programming languages such as Java, C++ or C#, they are too ‘computer-oriented’ to be useful in capturing primary system information. Such diagrams are often ‘conceptually rich’ and difficult for the customer and other ‘non-computer’ people to fully understand. There is of

course no compulsion to use these features, however, in our experience, software designers will often use all the facilities provided without any consideration as to their appropriateness. The use of complex concepts too early in the design process often compromises the requirements determined for the system, since users find themselves constricted by the programming nature of the models and consequently are unable to fully articulate their needs. Simone and Graham [22], speaking of UML state that “Developers often take the most concrete example of notational element in use and retrofit these interpretations higher up in the analysis process”.

If we compare standard Entity-Relation Diagram (ERD) [18,19] with ‘object-class diagram’ used in OMT or UML, we find that ERD only uses three basic, but powerful concepts. Object-class diagram, on the other hand, generally uses about twenty different concepts, spread over a number of different levels of abstraction.

In the analysis phase, we need to acquire the best possible knowledge of the problem formulation, and there the implementation details may cause trouble. On the other hand, in the design phase we need to focus on implementing the outputs from the analysis but we do not need to know some aspects of the reality modeled.

Underestimation of the model differences in the individual phases of development of an information system results—in some instances of ‘real programmers’—in such a simplification where analysis using UML is viewed as mere graphic representation of the future software code—typically in C++. Analytic models are then used not to specify the problem formulation with the potential users of the system who are also stressed by the complexity of the models that are presented to them. Projects in UML very often suffer from this problem.

3. The BORM approach

The BORM approach is based on each object having three independent attributes called *dimensions*; namely *data*, *behaviour* and *history* (the latter is a composition of states and transitions, i.e. the object lifecycle.)

Each object’s visible properties (car has label, type, garage membership,...) and its associations to other objects are shown in the data dimension. The activities of each object are revealed in the behaviour dimension and its states and transitions create the history dimension (Figs. 1–3).

Individual concepts are represent in BORM by a three-dimensional concept space. If one or two dimension for a number of concepts are fixed, a *view* is construct, which is equivalent to various diagrams used in other development methodologies, such as state transition or data flow diagrams (Fig. 4). However, in BORM, since each view is essentially a projection of the same underlying three-dimensional concept space, consistence is assured.

Because all BORM views use the same or compatible graphical notation and concepts, it is possible, and indeed recommended, to merge the various views of the related parts of a modelled system into a single diagram. This diagram then fully expresses the required detail and identifies any meaningful relations between the various concepts. A consequence of such an approach is the frequent uncovering of additional useful information. For example, we often discover ways to express the details of some object-oriented algorithm, or some way of expressing dependencies between different business processes, when working at the business object level. In object-oriented software implementation, most sophisticated object-oriented algorithms are based on cooperation of several objects in an appropriate data structure. In other development methodologies, such solutions must be drawn into separate diagrams describing separately the static data structure, the calculation mechanism, and the states and transitions of the objects—only by individual objects separately.

4. BORM basics

BORM, like other OOA&D methodologies is based on the spiral model for the development life cycle [7]. One loop of the object-oriented spiral model contains stages of strategic analysis, initial analysis, advance analysis, initial design, advanced design, implementation and testing. (Fig. 5).

The first three stages are collectively refereed to as the expansion stages. Expansion ends with the finalising of the detailed analysis conceptual model, which fully describes the solution to the problem from the requirements point of view [5].

The remaining stages are called the consolidation stages. They are concerned with the process of developing from ‘expanded ideas’ to a working application. During these stages the previously completed conceptual model is transformed, step-by-step, refined, reduced and finalised into a software design. In contrast to other methodologies, in BORM the object-oriented design stage is fluently connected to implementation without any sharp discontinuity, similar to the smooth transition between object-oriented analysis and object-oriented design. As a consequence, we can consider the program coding as at the last and the most detailed phase of the design process.

During the progress around the loop, the developer may undertake small excursions (little spirals out from the main spiral) into the implementation of smaller partial applications, used to develop, test and tune the program incrementally by using previously completed modules.

The behaviour of any prototype (in BORM we prefer the more accurate name ‘deliverable’) is also interesting. Every finalised application is also a deliverable and may be used as

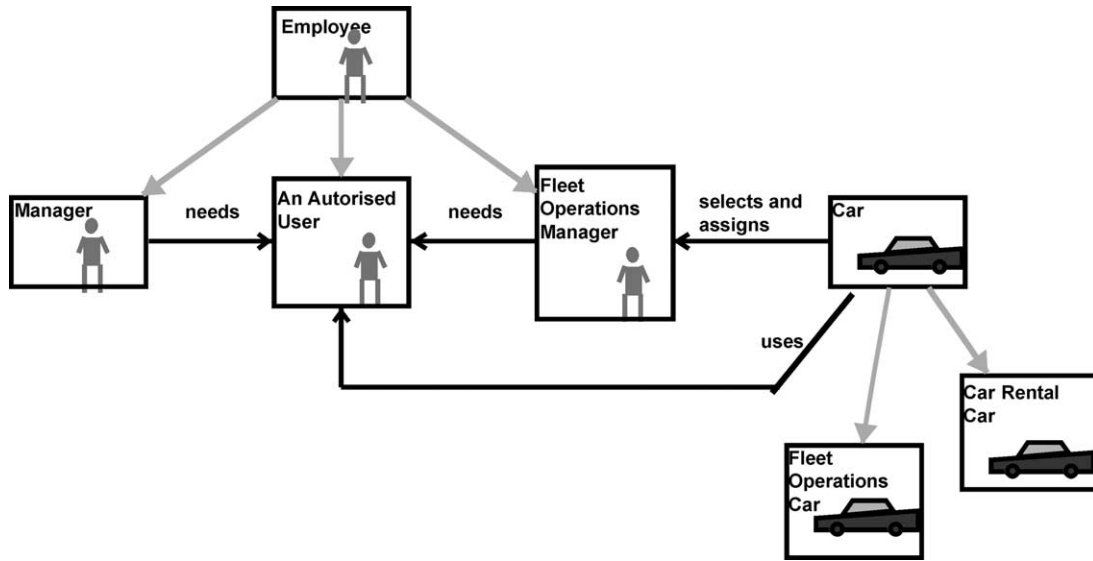


Fig. 1. Data dimension.

a basis for generating further requirements, which, in turn, leads to a new development loop.

BORM support development in pure object-oriented programming environments like Smalltalk [23–25], where it is possible to create software applications, which can be changed or updated at run time.

- The main concepts used in initial modelling are
 - Objects, and their behaviour,
 - Association (data links) between objects and
 - Communication (reports) between object behaviours.
- Every object is viewed as a machine with states and transitions dependent on the behaviour of other objects.

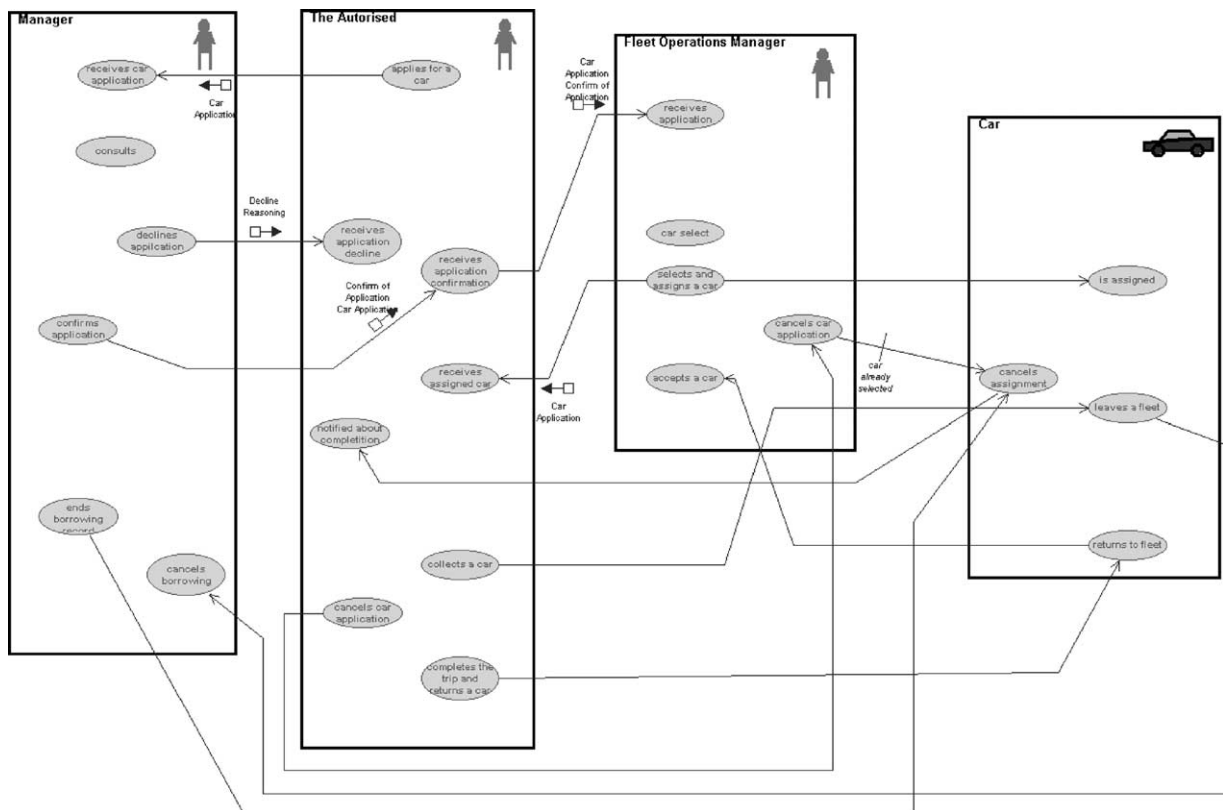


Fig. 2. Data and behaviour dimensions.

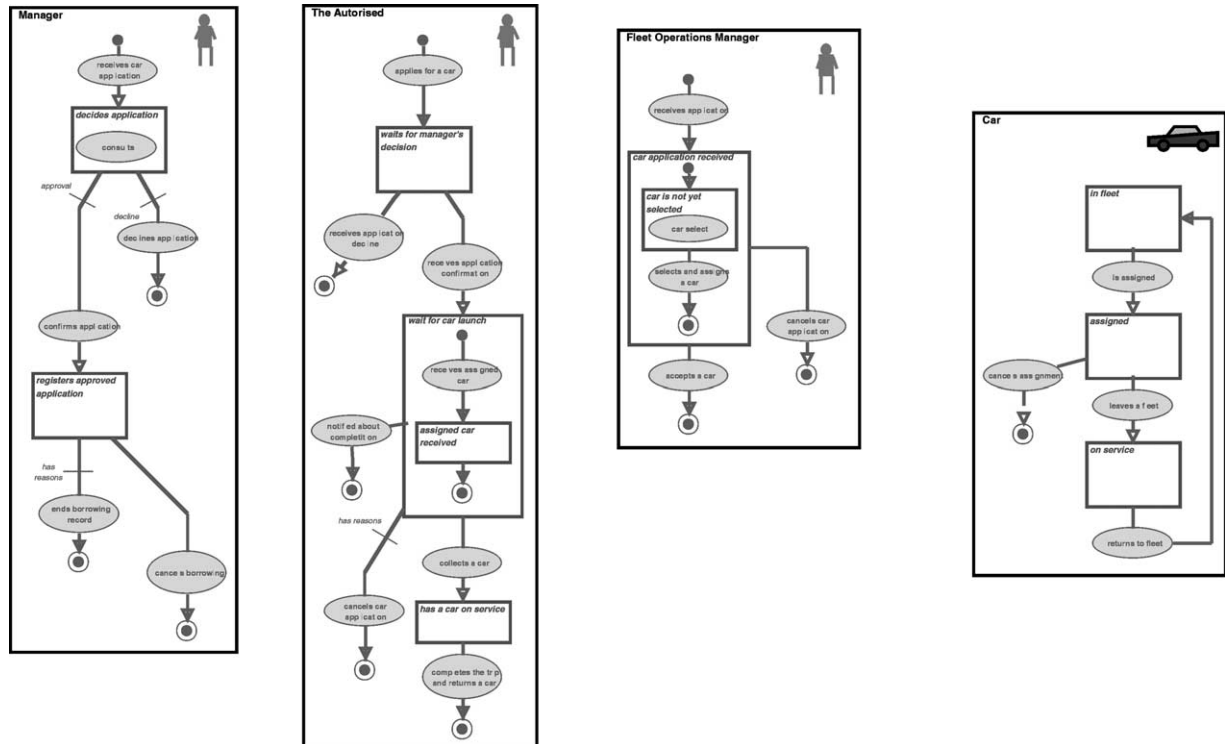


Fig. 3. Data and history dimensions.

Each state is defined by its semantic rule over object data associations and each transition is defined by its behaviour, necessary to transform the object from its initial to its terminal state. Consequently BORM objects have the characteristics of Mealy type automaton [9].

- Business object diagram accents the *mutual relationships* (communications and associations) of states and transitions of objects in the modelled system.

In BORM (see Fig. 6), it is possible for each concept to have some of the following:

- (1) A *Set of predecessor concepts* from which it could be derived by an appropriate technique and a *Set of successor concepts*, which could be derived from it by an appropriate technique. For example a conceptual object composition from a business object association.
- (2) A *validity Range*—the phases (of the development process) where it is appropriate. State-Transition diagrams for example are used extensively used in business conceptual modelling but are not supported by any current programming language.
- (3) A *Set of techniques and rules*, which guide the step-by-step transformation and the concept revisions between the system development phases, These are the following:
 - (a) *OBA*; which is a technique for transforming the initial informal problem description into the first object-oriented representation.

- (b) *Behavioural Constraints*; which is a set of determining rules which describes the set of possible transformation of the initial model into more detailed form with precisely specified object hierarchies like inheritance, dependency, aggregation, etc.
- (c) *Pattern Application* which helps to synthesise the analysis object model by the inclusion of object patterns [26].
- (d) *Set of Structural Transformations* (Class Refactoring, Hierarchies Conversions and Substitutions,

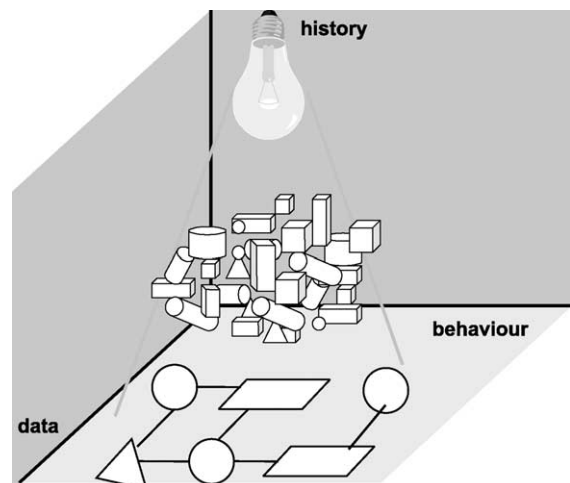


Fig. 4. Views.

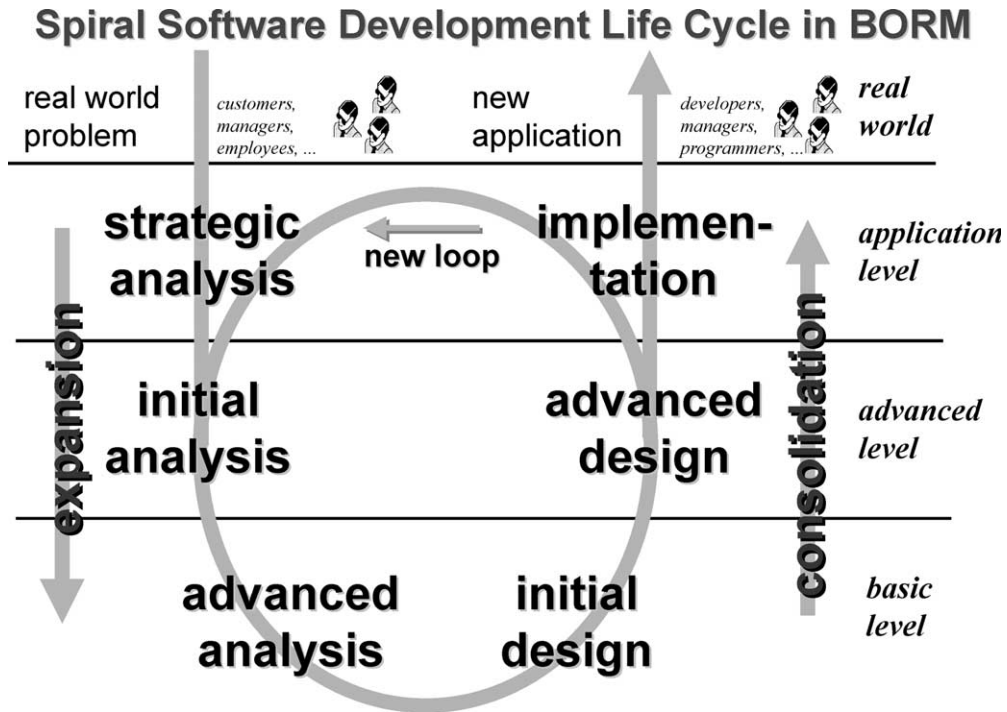


Fig. 5. BORM stages.

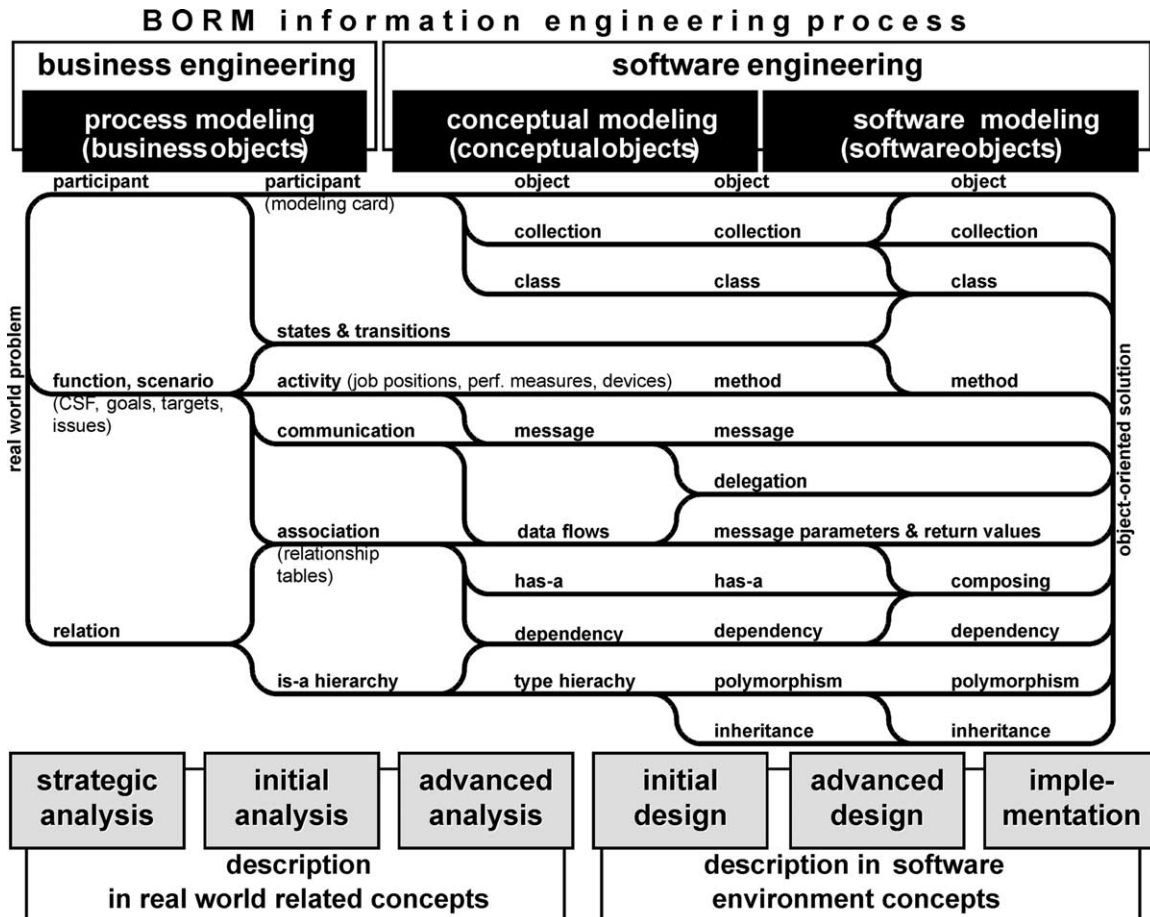


Fig. 6. BORM evolution of concepts.

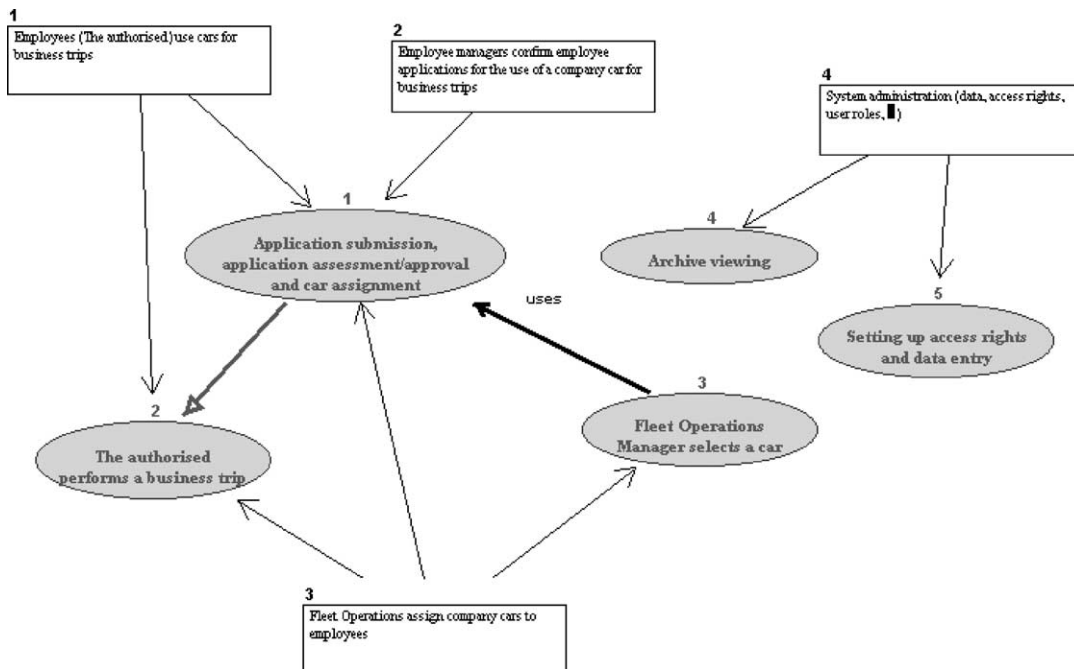


Fig. 7. System functionality.

solving Legacy problems solving programming environment constraints.) which are aimed at the final transformation of the detailed design model into a form acceptable in the implementation environment (programming language, database, user interface, operating system, etc.).

In Fig. 7 the required system functions are shown as rectangles, while ovals are used to show scenarios. Black, light arrows are used to link scenarios to system functions.

5. System scenarios

We can describe each scenario by a short textual description, which includes details of its association to functions and participants. This is stored as part of the model and all scenarios can be converted to html document. Some sample scenarios are shown in Table 2.

Scenarios play a similar role for BORM to that played by Use Cases in those development methodologies supported by UML. However, our diagrams are different to the equivalent Use Case diagrams as the system functions and scenarios are shown on the same diagram, as their associations. In such a diagram, light bold arrows are used to show transition between scenarios. Finally, bold black arrows between scenarios show the existence of a 'uses' relationship. (Our definition of this type of relationship is identical to the <<uses>> relationship in UML 1.3)

At the same time we are developing the system function diagram we also develop the business objects view. This is our initial object diagram in BORM. In this diagram we do not distinguish whether a participant represents a class, instance or collection of objects. This diagram is Fig. 8.

In such a diagram, we consider only the following two kinds of relationships:

1. *Associations*, which are represented by black arrows and
2. *is-a* hierarchy represented by grey arrows.

It is important to note that in this phase of development; an is-a hierarchy is not the same as software object inheritance based on a conceptual object type hierarchy. The conditions for two business objects to be in an is-a hierarchy are based on their membership of domains and sub domains, where a domain is a set of real world objects.

We next develop the process diagram, which in many ways, is the key diagram at this stage in the process. This diagram shows the same association between the business objects as the previous one but adds time and behavioural dimensions. Thus we show the states, activities transitions and operations for the business objects. This is a very powerful diagram. It conveys information that in the UML would require at least two diagrams (State and sequence diagrams). Yet despite conveying large amounts of information, the BORM group have found that it is clearly understood by stakeholders in the system development.

The next diagram marks the transition to the advanced analysis phase of development. Here we develop

Table 2
Sample Scenarios

Scenario No.1—associated with function(s): 1, 3, 2		Continues in scenario No. 2 uses scenario No. 3	
Initiation	Action	Participants	Result
Employee needs a car for a business trip	Application submission, application assessment/approval and car assignment	Car, An Authorised User, Manager, Fleet Operations Manager	The employee is either assigned a car or he must cancel the trip
Scenario No.2—associated with function(s): 1, 3		Follows scenario No. 1	
Initiation	Action	Participants	Result
An Authorised user has a car assigned	An Authorised user makes a business trip	Car, An Authorised User, Fleet Operations Manager	After the completion of the business trip, the car is returned to Fleet Operations. Alternatively, the car was not used and is returned unused

the conceptual object relation diagram. We are concerned here with conceptual objects and the various associations that exist between them. Grey arrows are used to show type hierarchies where polymorphism is the determining criteria. Note this is different to hierarchies in the previous diagram. In this diagram we identify object classes, which are denoted by rectangles and collection of classes (multi-objects in the UML) that are denoted by double border

hexagons. Both these are derived from entities in the previous diagram (Fig. 9).

Note the conceptual class ‘Car’ which is associated with two different multi-objects, company cars and rental cars. For this association we do not need to have class hierarchy structure, even though we will have two different collections of Car objects. The criterion we use is to create a new class only if its behaviour is different from its subclass.

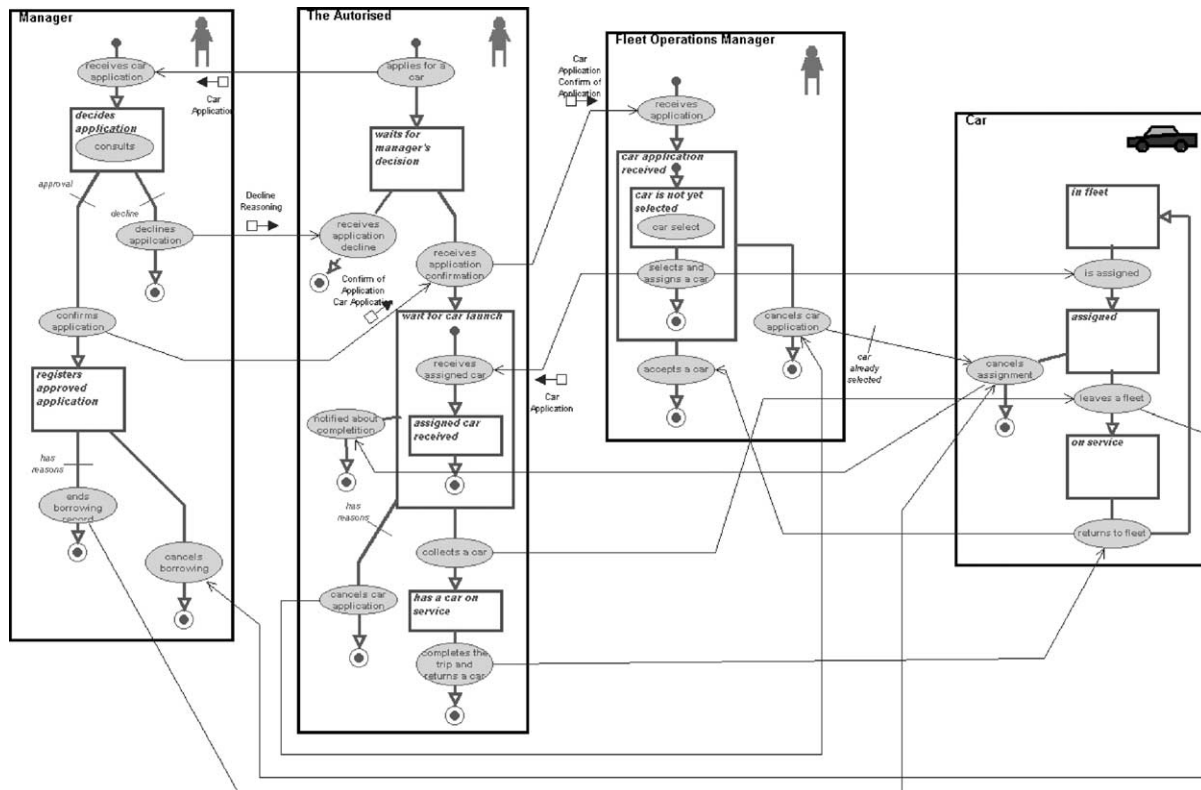


Fig. 8. Process model.

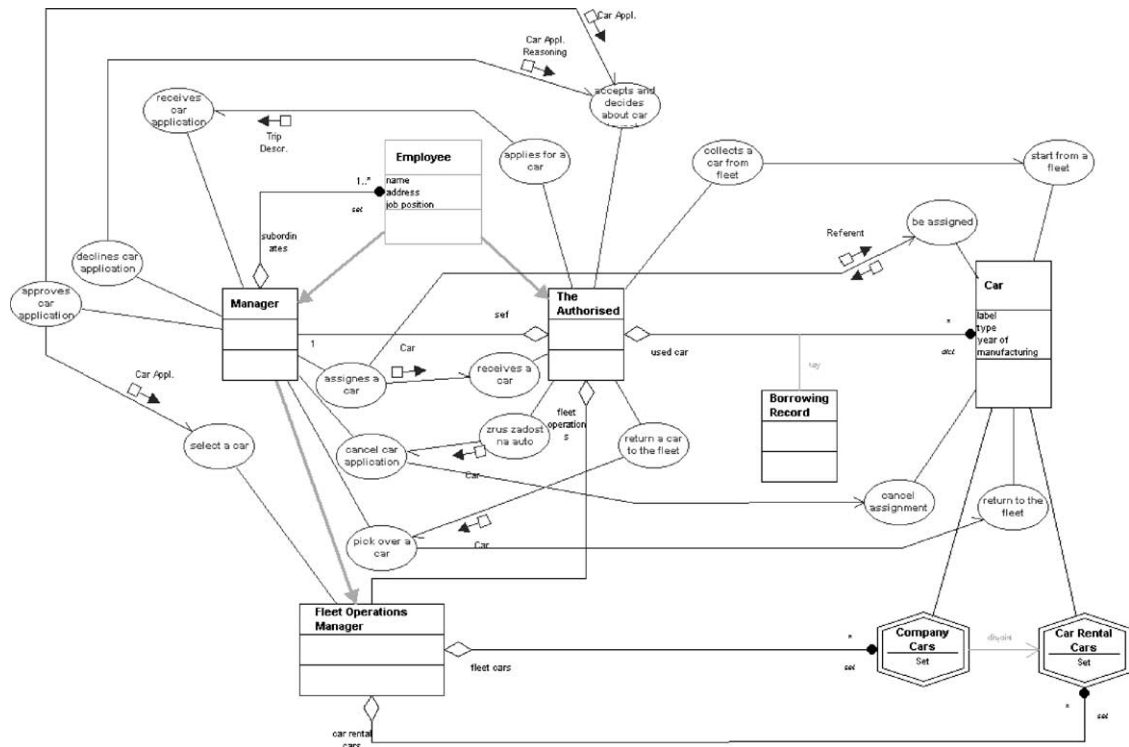


Fig. 9. Conceptual diagram.

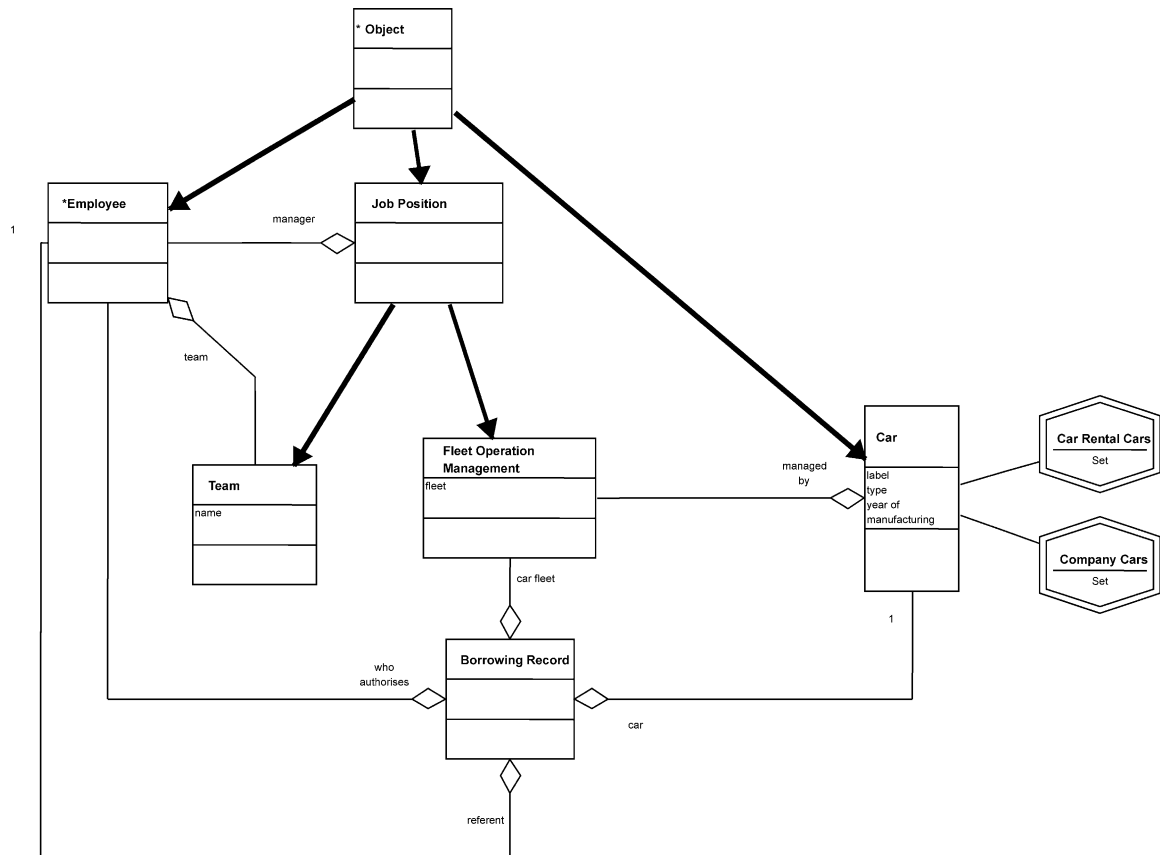


Fig. 10. Client-side software diagram.

The ovals are methods (Operations in UML), which the classes provide. The arrows between them show message passing, where the messages could be either synchronous (normal arrowhead) or asynchronous (half arrowhead).

Fig. 10 shows the client-side software objects. This model is obtained by a simple transformation from the type hierarchy into an inheritance hierarchy, in general this transformation is based on the need to fit conceptual object structure into concrete software environment structures, which are limited by:

- (a) target programming language syntax and paradigm and
- (b) reuse of existing objects from legacy components.

In the example, the implementation of the type hierarchy has to take into account the restriction that employee information is held in an existing legacy database. This has necessitated the construction of additional types in the database for job-position, team and manager. In such a diagram we are working with software objects, which have software relationships with other objects that need to reflect the concrete implementation environment. Black arrows are used to show inheritance. Note this is classical object-oriented inheritance and hence is not the same as our previous type hierarchy. This is because inheritance is not the only one way to implement new object types; we may implement new object types by composition. In our example three conceptual types, manager, authorised user and fleet operations manager will be implemented as instances of the legacy class employee composed with the new class job-position.

In this phase, it is also very important to know what classes are reused and what concepts are created as new ones or as reused artefacts. We denote the former by putting an asterisk before the name.

At this stage we also create a server-side software object diagram (Fig. 11).

The reason we need to differentiate between client and server side software models is because the system is often implemented in two different programming environments: pure object-oriented VisualWorks/Smalltalk for the client and relational ODBC/Oracle DBMS for the server.

Our final diagram is a hierarchy of Software components. Each component has its main (or interface) class and may be constructed out of other components or may serve as the link to some database table on the server (For example in VisualWork’s ObjectLens, special classes are created for implementing database access. If we work with instances of such this class, the system internally performs data manipulation operations on the server. Consequently, each instance of this special class is linked with one row in relation table). The above hierarchy diagram was directly used for to implement the system using this tool (Fig. 12).

6. The advantages of BORM

1. BORM follows the Process-oriented approach, which has proved to be beneficial in software development. Generally, the process-oriented approach lead to a faster

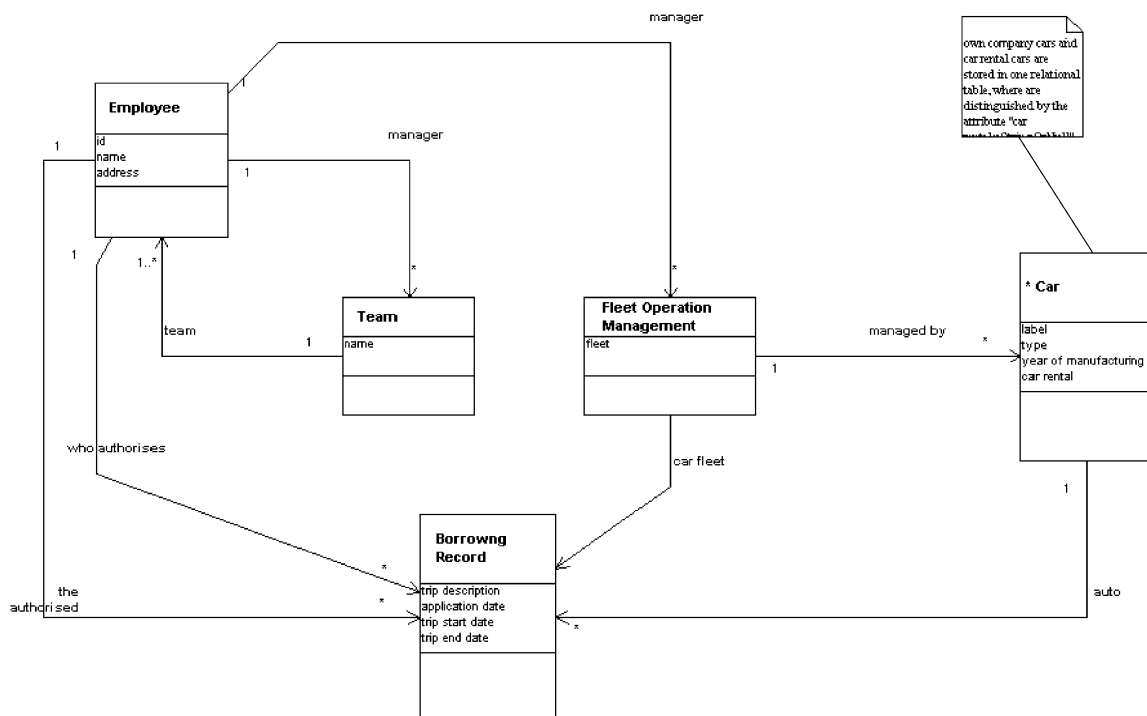


Fig. 11. Server-side software diagram.

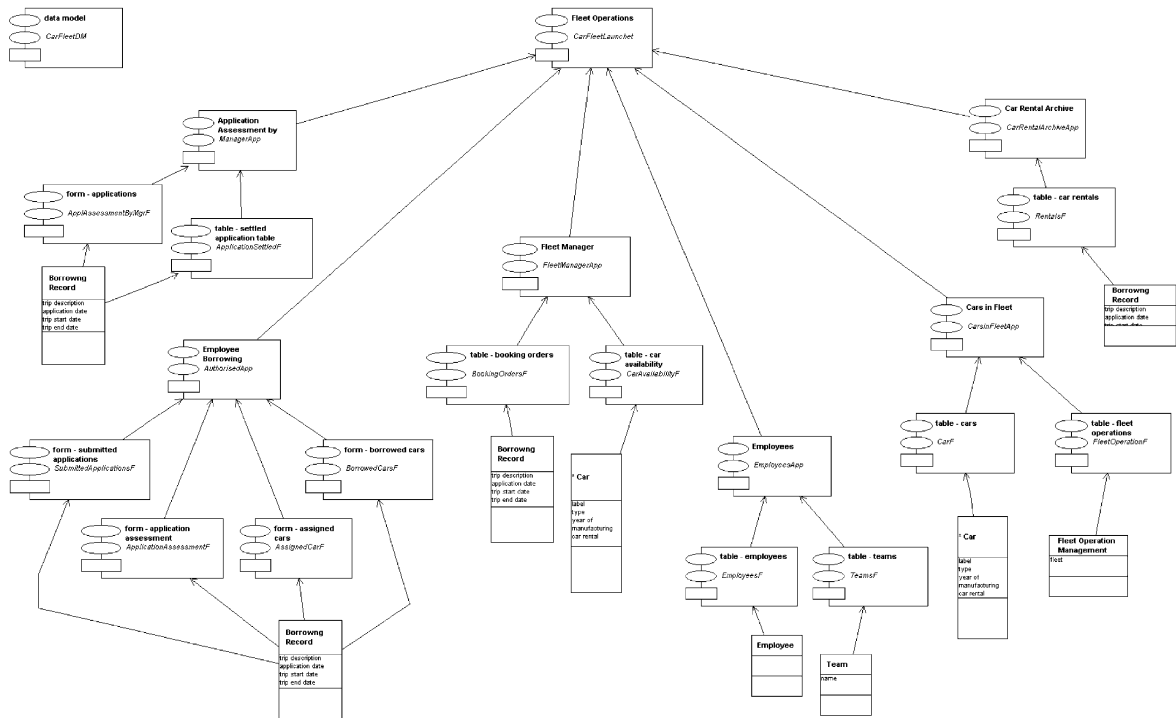


Fig. 12. Components.

and more comprehensive analysis of the problem being solved.

2. In our experience stakeholders from the problem domain are able to understand the BORM approach very quickly normally a one hour introduction at the start of analysis is enough.
3. In Deloitte and Touche (Prague office) a business consulting team have worked for the past three to four years using the BORM system as well as ARIS and Rational's Objectory/Unified method. They have found BORM to be on average 3–4 times faster in carrying out the analysis phase compared to other methods.
4. The methodology is easily acceptable to domain experts, analysis consultants and developers. Because BORM is based on a step-by-step transformation of the model and in each phase only a limited and consistent subset of BORM concepts are used.
5. BORM has been used enthusiastically by Smalltalk and Java programmers and by non-relational object database programmers. One feature of BORM they find attractive is the way it exploits collection concepts not just classes and the way that these collection classes are seamlessly integrated into the development environment. (Compare with multi-objects in UML).
6. BORM has more object hierarchies (polymorphism, is-a, dependency, etc.) than other methods which usually only provide concepts supported by programming languages. Usually only object inheritance is supported.
7. These last two features provide a much richer language with which to express modelling ideas.

7. Final comments

Today, when improved visual programming tools combined with the support of rapid application development environments are available, it would appear that the whole software development process is becoming easier. This statement is true, however, only for those cases where the complexity of the solution and of users' requirements is relatively simple. Business systems developed for real companies often have a much higher level of complexity which make development much more difficult. Consequently it is essential (from the software developer's viewpoint) to improve the initial phases of software development.

Until recently, it was correctly assumed that conceptual modelling tools and techniques were used through all stages of project development, from the initial phase to the eventual implementation. However, the position of conceptual modelling is currently being used solely in the implementation phase, as a result of the evolution of software development tools. The analysis is now being performed using newly developed techniques and 'business' objects modelling tools.

We believe that Object-oriented programming has changed not only system development but also all of computer science. In software development, any team must be well organised, with clear and common goals. Managers of such projects must be clear about the potential benefits as well as understanding the management Object-oriented development.

Object-oriented programming can help in the development of a large system by significantly reducing the developing and maintenance time. But the adoption of Object-oriented programming requires considerable developments not only in technical knowledge but also managerial and cultural realignment; such changes can only be achieved by suitable training combined with the use of well designed and easy to use software development tools like those described here.

8. Conclusion

Currently there is not a ‘standard solution’ to the problem of gathering and representing knowledge. Our approach, described here, developed out of business experience and enhanced by graphic models with clear connection towards system development seems to be a promising candidate for such a standard. The notation we propose may serve not only as a tool for formal representation of modelled information, but also as we have demonstrated as a useful tool for communicating with developers and experts from the problem domain (managers, employees, etc.). The key advantages of BORM are its graphic models of knowledge representation, which provides easy and effective feedback. There are also clear rules how to progress through the system development process using this knowledge representation.

References

- [1] R.P. Knott, V. Merunka, J. Polak, Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis, Proceedings of Fourth International Conference on Requirements Engineering ICRE, Chicago, IEEE Computer Society Press, New York, 2000, ISBN 0-7695-0565-1.
- [2] D.A. Taylor, Business Engineering with Object Technology, Wiley, New York, 1995, ISBN 0-471-04521-7.
- [3] G. Darnton, M. Darnton, Business Process Analysis, International Thomson Publishing, 1997, ISBN 1-861-52039-5.
- [4] C. Partridge, Business Objects—Reengineering for Reuse, Butterworth/Heinemann, London, 1996, ISBN 0-7506-2082-X.
- [5] E. Yourdon, Mainstream Objects—An Analysis and Design Approach for Business, Prentice-Hall, Englewood Cliffs, NJ, 1995, ISBN 0-13-209156-9.
- [6] A. Bahrami, Object Oriented System Development, McGraw-Hill, New York, 1999, ISBN 0-071-16090-6.
- [7] H.-K. Eriksson, M. Penker, Business Modeling with UML, Wiley, New York, 2000, ISBN: 0-471-29551-5.
- [8] A. Goldberg, R.S. Kenneth, Succeeding with Objects—Decision Frameworks for Project Management, Addison-Wesley, Reading, MA, 1995, ISBN 0-201-62878-3.
- [9] M. Cotterell, B. Hughes, Software Project Management, Thomson Computer Press, 1995, ISBN 1-850-32190-6.
- [10] M. Cantor, Object-Oriented Project Management with UML, Wiley, New York, 1998, ISBN 0-471-25303-0.
- [11] W. Royce, Software Project Management: A Unified Framework, Addison-Wesley, Reading, MA, 1998, ISBN 0-201-30958-0.
- [12] A. Davis, Software Requirements—Objects, Functions and States, Prentice-Hall, Englewood Cliffs, NJ, 1993, ISBN 0-13-562174-7.
- [13] G. Kotonya, I. Sommerville, Requirements Engineering: Processes and Techniques, Wiley, New York, 1999.
- [14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-Oriented Modelling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991, ISBN 0-13-630054-5.
- [15] K.W. Derr, Applying OMT—A Practical Guide to Using the Object Modelling Technique, Sigs Books 1995, Prentice-Hall, Englewood Cliffs, NJ, 1995, ISBN 1-884842-10-0, ISBN 0-13-231390-1.
- [16] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1998, ISBN 0-201-57168-4.
- [17] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, MA, 1999, ISBN 0-201-30998-X.
- [18] C. Carteret, R. Vidgen, Data Modelling for Information Systems, Pitman Publishing, London, 1995, ISBN 0-273-60262-4.
- [19] C.J. Date, An Introduction to Database Systems, Sixth ed., Addison-Wesley, Reading, MA, 1995, ISBN 0-201-82458-2.
- [20] I. Jacobson, Object-Oriented Software Engineering—A Use Case Driven Approach, Addison-Wesley, Reading, MA, 1992, ISBN 0-201-54435-0.
- [21] M. Fowler, K. Scott, UML Distilled, Second ed., Addison-Wesley, Reading, MA, 1999, ISBN 0-201-65783-X.
- [22] A.J.H. Simone, I. Graham, 30 Things that go wrong in Object Modelling with UML 1.3, in: H. Kilov, B. Rumpe, I. Simmonds (Eds.), Behavioral Specifications of Businesses and Systems, Kluwer, Dordrecht, The Netherlands, 1999, pp. 237–257, Chapter 17.
- [23] J. Hunt, Smalltalk and Object Orientation, Springer, Berlin, 1997, ISBN 3540761152.
- [24] T. Hopkins, B. Horan, Smalltalk—An Introduction to Application Development Using VisualWorks, Prentice-Hall, Englewood Cliffs, NJ, 1995, ISBN 0-13-318387-4.
- [25] K. Beck, Smalltalk Best Practice Patterns, Prentice-Hall, Englewood Cliffs, NJ, 1997, ISBN 0-13-476904-X.
- [26] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns—Elements of Reusable Object Oriented Software, Addison-Wesley, Reading, MA, 1996, ISBN 0-201-63361-2.