Two-Hemisphere Model Driven Approach: Engineering Based Software Development

Oksana Nikiforova and Marite Kirikova

Institute of Applied Computer Systems, Riga Technical University
1 Kalku, Riga, LV-1658, Latvia
{ivasiuta,marite}@cs.rtu.lv

Abstract. Several model driven approaches are currently used and developed, namely, generic model driven approaches, agile model driven approaches, business process model driven approaches, etc. This paper proposes the model driven approach, which is based on a two-hemisphere model. The two-hemisphere model integrates application and problem domain issues. The model utilizes automatic model transformations, but in the same time allows room for input of tacit knowledge. It is a practice-oriented approach which ties together methods of business process modeling, object oriented, and model transformation approaches in order to support cognitive needs of requirements holders and object oriented software developers, and provide framework for explicit and transparent representation of mutually related business and software development knowledge. It utilizes tacit knowledge of stakeholders (including software designers), but in the same time reflects this knowledge in explicit and automatically reconfigurable models that form the basis for automatic code generation.

1 Introduction

"Agile" is one of the most popular words in current software development practice. Agile software development methods, agile modeling, etc are attracting more and more interest and attention. However the ultimate goal of the agility is not just software development, - it is business agility [1], [2] that is to be achieved by organizations to survive in a rapidly changing turbulent environment. The role of information technology and information systems in supporting business agility is well understood [1], [3], [4]. One of the most debated promises to support business agility is the Model Driven Architecture [4] that aims at automatic model transformation from a platform independent application domain model into platform specific design and implementation models [5]. The approach is developed by the Objects Modeling Group and is based on the UML [6] (object oriented) application domain model. However this approach does not address the question of how to develop such an UML platform independent model, which would meet business needs and would be ease adaptable to the changes of those needs. Therefore there is room for the claim that a sophisticated application domain model is not needed, i.e., that the agile model at this level is barely good enough [7]. This thesis is backed up by the practical as-

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 219–233, 2004.

sumption that tacit models that are close to the reality are better than sophisticated explicit models that are far from the reality. The claim reflects the main problem of contemporary object oriented approaches: an attempt to gather requirements on the bases of use case descriptions without automatically tracking relationships between use cases and without automatically analyzing their consistency. Automatic checking of correspondence between application model and problem domain model also is not supported.

Several model driven approaches are currently used and developed, namely, generic model driven approaches, agile model driven approaches, business process model driven approaches, etc. With respect to the model that drives the software development process we may distinguish between the art based model driven approaches (driven mainly by tacit or mental models) and the engineering based ones, which are driven by externalized explicit models. This paper proposes a model driven approach, which is based on an explicit two-hemisphere model. The purpose of the paper is to demonstrate that sophisticated models are not an obstacle in software development and that engineering based approaches can well support business agility. The two-hemisphere model integrates application and problem domain issues. The model utilizes automatic model transformations, but in the same time allows room for input of tacit knowledge. It is a practice-oriented approach which ties together methods of business process modeling, object oriented, and model transformation approaches in order to support cognitive needs of requirements holders and object oriented software developers, and provide framework for explicit and transparent representation of mutually related business and software development knowledge. It utilizes tacit knowledge of stakeholders (including software designers), but at the same time reflects this knowledge in explicit and automatically reconfigurable models that form the basis for automatic code generation.

The paper is structured as follows. In Section 2 we analyze several model driven software development approaches. Section 3 introduces the two-hemisphere model driven approach and discusses its applicability from business, software development, and cognitive perspectives. Section 4 briefly illustrates some model transformations utilized in the two-hemisphere model driven approach.

2 Software Development Driven by Particular Models

The notion Model Driven Approaches [8] has become popular only recently, however, all approaches are model driven. The question is only what type of model drives the approach. Is it a tacit mental model of the designer or a particular explicit model represented using particular formal notations that are supposed to be understood by all participants of software development team. In this section we discuss briefly the following software development approaches:

- Traditional object oriented approach (TOO)
- Generic model driven approach (GMD)
- Agile model driven approach (AMD)
- Business process model driven a approach (BPMD)
- Two-hemisphere model driven approach (2HMD)

Differences between approaches are graphically illustrated in Fig. 1. The fully explicit model here is depicted by filled rectangle, semi explicit model (some aspects of systems are represented by explicit representations, while other aspects are presented only in tacit mental models) are shown by non-filled rectangles, and fully or mainly tacit models (non essential proportion of explicit representations may be present) are represented by cloud like notation. Differences between approaches are analyzed from the point of view of model transformations, "the heart and the soul" of model driven approaches [5]. Formal (automatic transformations) between models at different levels of abstraction are denoted by continuous line arrows, but mental and manual transformations by dotted line arrows.

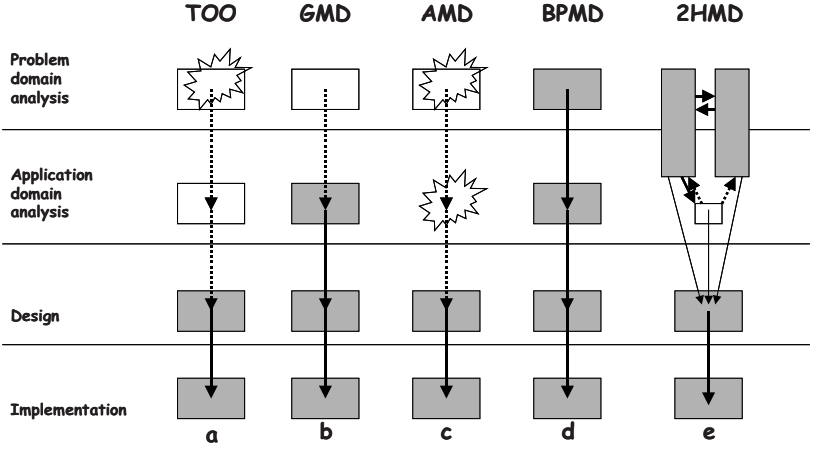

Fig. 1. Level of exploration in model driven approaches

2.1 Traditional Object Oriented (TOO) Approach

The TOO approaches [10-16] the problem domain is considered as a black box by describing a number of aspects of the system [17]. Primarily, designers' tacit knowledge acquired during application domain analysis drives the traditional object-oriented approach (Fig. 1. a). Thus it is an approach, which is based on art rather than engineering, despite sophisticated modeling techniques used in lower levels of abstraction.

Modeling efforts in TOO usually start with the identification of *use-cases* (Fig.2.) for the software to be developed. A use-case reflects interactions between the system to be built and the actor (an outside object in a particular role) that has a particular purpose of using the system. Each interaction starts with an event directed from the actor to the system and proceeds through a series of events between the actor, the system, and possibly other actors, until the interaction initiated by the original event reaches its logical conclusion. The sequence of interactions can be specified in words or by one or more prototypical scenarios, which then are to be translated into the elements of an *interaction diagram*. The interaction diagrams are created for each

use-case and show the sequence of message passing during certain use-case realization. The c*lass diagram* shows an overall structure of the software system and encapsulates the responsibility of each class. The c*omponent diagram* represents the realization of classes into a particular programming language. Therefore during the design stage the target software system is organized into components, based on the knowledge gained in the analysis stage. As a result the *design model* is developed that further *may be automatically translated* into a particular programming language, and thus serve as a basis for software system's implementation [17].

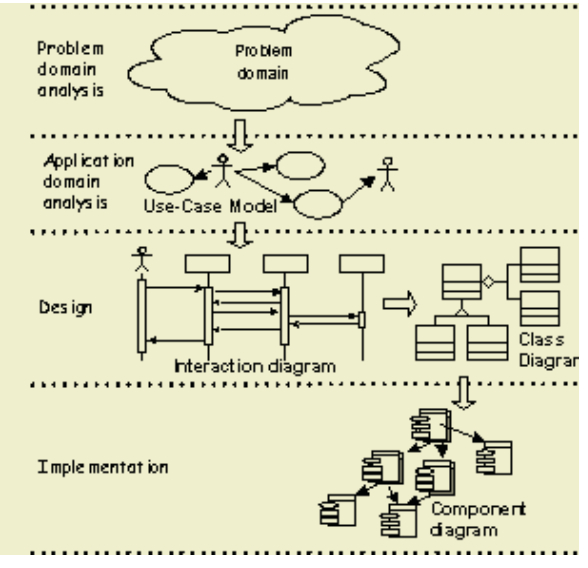

Fig. 2. UML diagrams to be built during traditional object-oriented software development

The TOO approach is usually based on a quite rigid requirements specification, which is developed on the basis of use-cases and problem domain analysis. Knowledge in higher levels of abstraction is documented, however, the form of documentation – use-cases, does not permit one to check consistency of requirements and does not show their relationship to the problem domain explicitly. This leads to major problems in change management of traditional object oriented projects.

2.2 Generative Model Driven (GMD) Approach

The GMD approach (Fig. 1. b) "is based on the idea that people will use very sophisticated modeling tools to create very sophisticated models that they can automatically transform with the tools to reflect the realities of various deployment platforms" [7]. One of such approaches is Object Management Group's Model Driven Architecture [8]. Formal transformation here starts from platform independent application domain model represented in UML. This model is transformed into platform specific design models, and further the code is generated from the platform specific model [18]. The

main gain here is higher flexibility that can be obtained by shorter time needed for software design and implementation, because automatic transformation is possible not only from the design level models into the implementation, but already from application domain models into the design level models. Therefore the GMD approach indirectly addresses business agility better than the TOO approach [4].

Opponents of the Model Driven Architecture call GMD "a great theory - as was the idea that the world is flat" [7]. And, indeed, the application domain class model [5], [19], which should conform to all problem domain requirements and incorporate all details necessary for platform specific design model generation, is extremely complicated and may be understood only by experts in object oriented software development. This is a weakness of the approach, because there is no possibility to prove the application domain level model's conformance to user requirements neither formally nor mentally.

2.3 Agile Model Driven (AMD) Approach

The AMD approach [20] uses formal model transformation from design level into implementation level, like the TOO approach, but it relies on simpler formalized models and highly elaborated tacit models in upper levels of abstraction. Some arguments as to why the AMD approach is viewed as more effective than the generative model driven approach are as follows [7]: (1) every software system has both a user interface on the front end and the database on the back end, yet UML still does not address these issues; (2) in many cases people do not have the modeling skills in UML and (3) the tool support is still not sufficient for proper handling of UML models, i.e., vendors claim support for a standard, but then implement their own version of it for competitive reasons.

The main claim of users of the AMD approach against the TOO approach is that it is more reasonable to spend time for acquiring proper tacit knowledge about user requirements than spend time for developing specifications and models that are formal but hard to understand and change. The AMD approach is based on the art (tacit knowledge) of highly qualified systems developers in upper, problem and application, domains and may utilize engineering for translation of design level models into implementation (Fig. 1. c).

2.4 Business Process Model Driven (BPMD) Approach

Engineering can be applied already at the problem domain level. One way how this can be achieved is through the use of appropriate business process modeling methods and tools. If a detailed enough business process model is developed, it automatically may be translated into application level UML model [21]. We call such approach a business process model driven approach (Fig. 1. d). Theoretically, such an approach directly supports business agility, because the only thing that is to be changed to obtain new software code is the business model. However the approach requires high business process modeling skills with respect to both problem and application domain

levels, because the problem domain business process model should include the details necessary for application domain model generation. Thus business process model becomes a complicated multilevel system where automated processes are clearly identified and their expected behavior represented. The BPMD approach has high potential feasibility because business process redesign and improvement is one of the methods used by many organizations to achieve or preserve their competitive advantages [18], [22], [23].

2.5 Two-Hemisphere Model Driven (2HMD) Approach

The 2HMD approach [9] may be considered as a version of business process model driven approach. The approach addresses the following issues currently relevant in software development:

- Business process models usually are developed in a comparatively high level of abstraction and rarely pin down all details needed for software development
- Diagrams preferred by business team differs from those preferred by software developers

Therefore the 2HMD approach utilizes the problem domain conceptual model and the application level use-case diagram in addition to the business process diagram for driving the software development process. It is based on sophisticated models, but it enables the generation of simpler models from the sophisticated ones in order to support development of stakeholders' tacit knowledge. Hypothetically, the transformation from two-hemisphere model (or just from one of its constituents – the business process model) into platform independent application level model is possible (Fig. 1e). However, this paper describes a softer version of 2HMD approach where stakeholders' tacit knowledge, if needed, may be added down to the design level. The approach is applicable for software development teams that possess conventional business process modeling and UML tools. The 2HMD approach is described in detail in Section 3.

3 The 2HMD Approach in Detail

Cognitive psychology [24] proposes that the human brain consists of two hemispheres, one of which is responsible for logic and another one for concepts. Harmonic interrelated functioning of both hemispheres is a precondition of an adequate human behavior. A metaphor of two hemispheres may be applied to software development process because this process is based on investigation of two fundamental things: business and application domain logic (processes) and business and application domain concepts.

Some recent surveys show that about 83% of companies are engaged in business process improvement and redesign [18]. This implies that many companies are common with business process modeling techniques [18] or at least they employ particular business process description frameworks [22]. On the other hand practice of soft-

ware development shows that functional requirements can be derived from problem domain task descriptions even about 7 times faster than if trying to elicit them directly from users [25]. Both facts mentioned above and existence of many commercial business modeling tools (such as GRADE [26], ARIS [27], etc.) are a strong motivation to base software development on the business process model rather than on any other soft or hard models.

However, business process diagrams developed by business analysts rarely show all details necessary for software development, as well as in many cases, do not reflect the "to be" business situation. Therefore formal transformation of business process model into the application model, design model, and implementation is not possible, and software developers shall step in and try to acquire software requirements. They usually interview business managers and then create UML diagrams, typically beginning with use-case and class diagrams. Business managers may be forced to review those diagrams, that can be frustrating for them, because use-case and class diagrams do not reflect the business perspective very well [18].

The 2HMD approach (Fig. 3) addresses this problem by use of two interrelated models at problem domain level, namely, the business process model and the conceptual model, which are related to the use-case model at the application domain level.

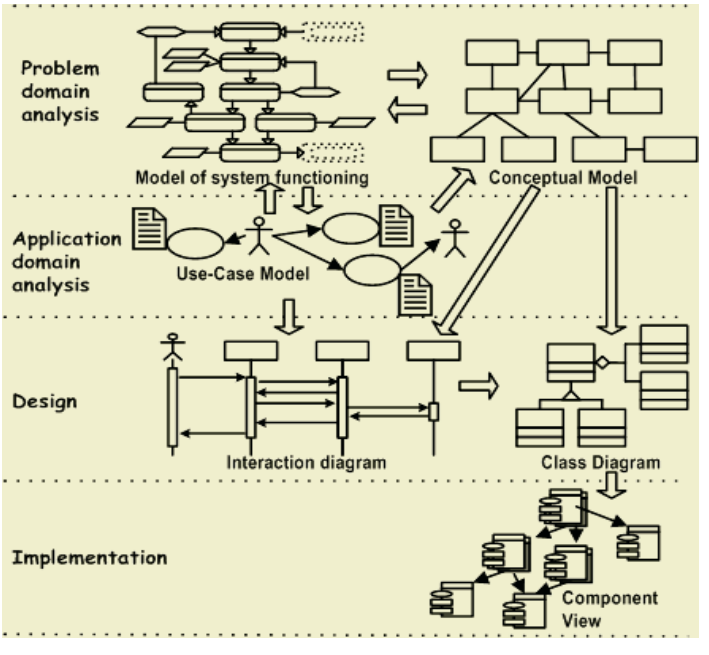

Fig. 3. Framework of the 2HMD approach

A notation of the business process model, which reflects functional perspectives of the problem and application domains, is optional, however, it must reflect the following components of business processes [28]: external entities (processes); subprocesses (the number of levels of decomposition is not restricted); performers; in-

formation flows; triggering conditions, and information (data) stores. Use-cases are tied to the business process model and can be derived from it. The conceptual model is used in parallel with business process model to cross-examine software developers understanding of problem and application domain models. Use-cases are always either generated from the business process model or reflected in the business process models, i.e., they "depart" from the business process model for discussions with respect to software development details, prototyping, etc., and, when details are known, manually return back to the business process model together with the details [29]. Current functional requirements always are present in the business process model, that helps to maintain their consistency [29]. As a result sophisticated models are used without disturbing software developers' and business managers' natural ways of thinking [9].

Relevance of particular models of 2HMD framework with respect to business modeling, object oriented software development, and model transformation is shown in Table 1.

Table 1. Use of problem domain and application domain models in 2HMD approach

Perspective	Business process model	Conceptual model	Use case model
Business modeling	Knowledge organized in business oriented terms Requirements can be derived faster from task descriptions than if asked directly from users Appropriate business modeling tools exist	May be used for checking adequacy of developers knowledge	Convenient for discussing requirements in detail
Object oriented software development	Enables consistency check of use cases	Developers usually build tacit and explicit conceptual models that reflect their current knowledge	Main tool for requirements gathering and understanding
Model transformation	At a particular stage the process model may be automatically transformed into implementation (see Section 2.4)	May be (at least partly) derived from business process model If organized as class diagram may (hypothetically) mirror business process model	May be automatically generated from business process model

Initial version of the 2HMD approach was proposed in [17], where the general framework for object oriented software development had been discussed and it's application for driving school's software development had been demonstrated. The current version of the approach supports semi-formal model transformation from problem and application domain into design and implementation. By semi-formal, we mean a transformation of part of elements of one model into the subset of elements of

another model. Transformation is fully formal if all elements of the target model can be obtained from the source model. The 2HMD approach utilizes two formal (automatic) transformations: (1) from a business process model into an use-case diagram, and (2) from a design level model into implementation. All semiformal and formal transformations are illustrated in Fig.4.

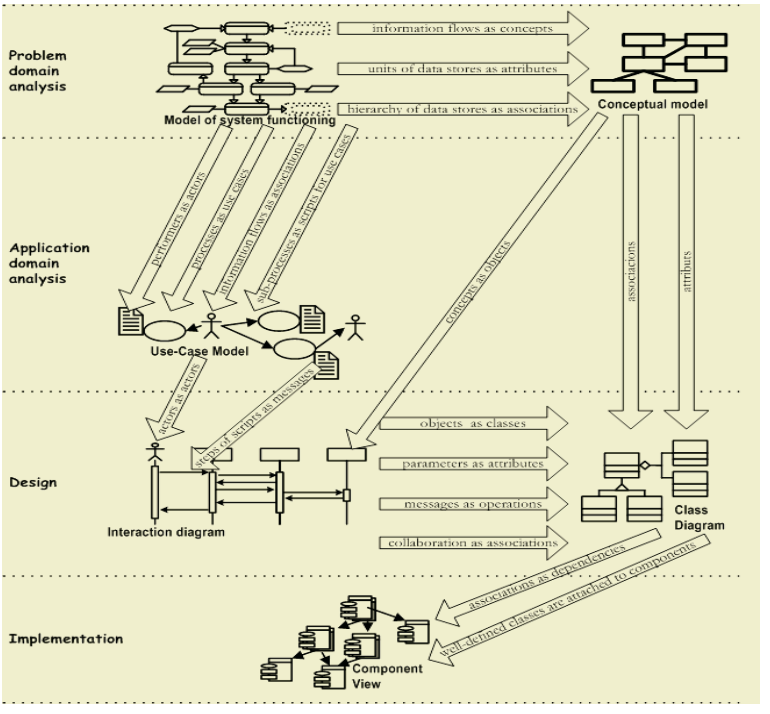

Fig. 4. Formal and semi-formal transformations in 2HMD approach

Semiformal transformation from the business process model into the problem domain conceptual model. As mentioned above, the conceptual model reflects tacit knowledge of software developers in an explicit diagram, i.e., software developers build it. However, part of a conceptual model can be generated automatically. Real-world classes relevant to the problem domain and their relationships are presented in the conceptual model. The conceptual model shows the things that exist in the problem domain and their relations to other things. The notational conventions of the business process diagram give a possibility to identify concepts by analyzing all the data stores in the diagram [17]. Data stores from the business process model can be transformed into the concepts of the conceptual model. The same refers to the units of data stores as well as information flows. Automatic transformation possibilities of other business process model elements into conceptual model are under the investigation. The automatically generated part of conceptual model may be compared to the manually constructed model to ensure consistency between the business process model and developers knowledge.

Formal transformation from the business process model into use-cases is possible, if processes to be performed by software system are identified in the business process model [29]. Processes to be performed by software system become use-cases in the use-case model, performers of related processes become actors in the use-case model, and scenarios for realization of use-cases may be defined by decompositions of business processes (sub-processes) corresponding to the use-cases [17].

The interaction diagram is developed by *semiformal transformation from the use-case model and the conceptual* model. Interaction diagram for each use-case is based on its realization scenario (or sequence of sub-processes). Appropriate interacting objects are extracted from the conceptual model. Alternatively, the transformation directly from the business process model could be provided, because the use-case model and part of the conceptual model are generated from the business process diagram. However, in the case where semi-formal transformations dominate over formal ones and human intelligence is involved at different levels of abstraction, simpler transparent transformations are more preferable than sophisticated ones.

The class diagram is based on the conceptual model and is formed according to information in the interaction diagram. It is obtained by *semiformal transformations from the interaction diagram and the conceptual model*. The class diagram here is already a structure of a software application and contains only those classes, whose objects interact during the use-case realization [17]. *Formal transformation from Class diagram into software code* may be utilized.

In overall, the 2HMD approach is engineering based, because only those use cases, which are automatically generated from the business process model are used for further transformations. This allows maintaining consistency between the requirements. On the other hand possibility to generate use cases automatically fastens software development process and support business agility. In the next section some of transformations mentioned in this section are described in more detail.

4 Application Case: An Administration of Driving School

Administration of the driving school [17] is used as a problem domain to illustrate how the 2HMD approach may be applied. This section shows main steps, which were made during software development for administration of the driving school.

Problem domain analysis: The simplified version of the business process for the driving school is reflected in Fig. 4. The driving school has several classrooms in several locations. The director of driving school assigns learning sessions for new groups based on a predefined schedule. The driving school already has a teaching staff, which consists of instructors having a car and teachers. When the applicant comes to driving school, the administrator of the school offers him a list of available groups for learning and helps to select the most appropriate group location and time schedule. After at least three applicants were assigned for learning in a particular group the start date of learning is defined, the teacher for the group is assigned and the instructor for every pupil in the group is attached. Each group is registered at the Road Traffic Safety Directorate (RTSD).

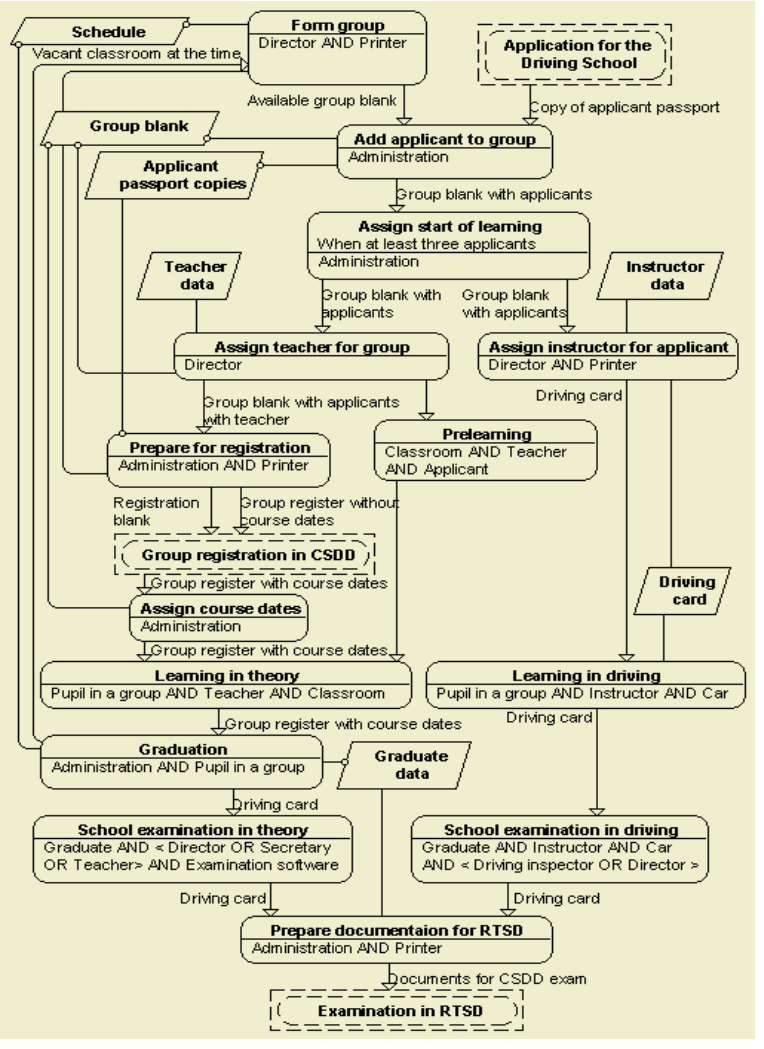

Fig. 5. A simplified business process diagram of the driving school

The diagram in Fig. 4 is a result of business process modelling done by the developer in straight collaboration with the user, and using a particular business-modelling tool – GRADE [26]. Identification of real-world classes relevant to the software system and their relationships is done during conceptual modelling. The conceptual model shows the things that exist in the driving school problem domain and their relations to other things. It is expressed in terms of classes. The notational conventions of the business process diagram give a possibility to identify concepts also by analysing all data stores in this diagram. Data stores are represented as concepts in the conceptual model Fig. 6.

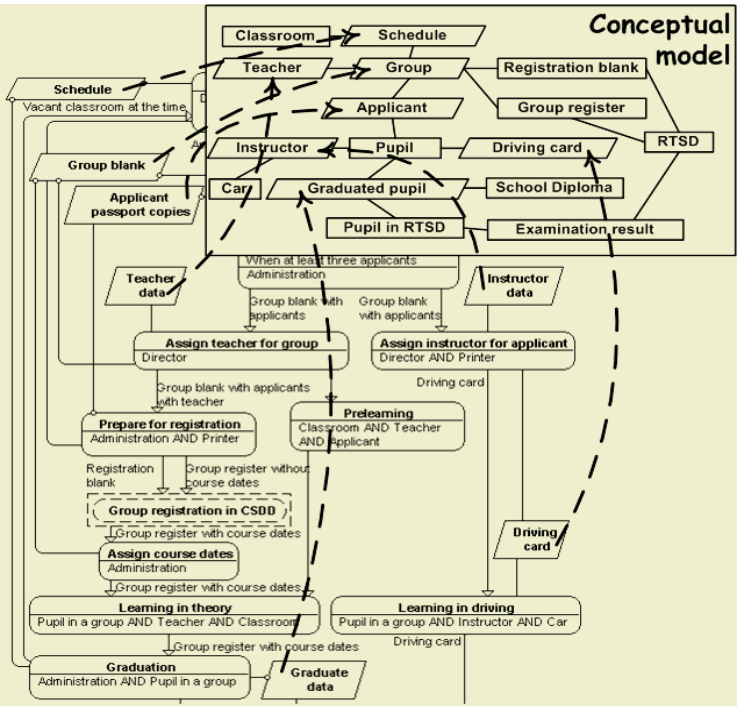

Fig. 6. Construction of a conceptual model for the driving school

Concepts coming from the business process diagram at the highest level of abstraction are indicated as parallelograms. Concepts identified by analysis of sub-process defined during business process modelling – as rectangles. The hierarchical structure of data stores in the business process model gives a possibility to detect potential relationships between system concepts. Data stores are characterized by a set of attributes, which are useful for definition of class structure.

Application Domain Analysis: Looking for processes in business process model (Fig. 5) that can be automated, and potential actors to implement use-cases is a basis for building the use-case diagram (Fig. 7). Analysis of the business process identifies the boundary of the software system and helps to decide, which processes refer to the software system. Those processes are presented as use-cases of software system required and their performers are presented as external actors that perform defined use-cases. The use-case diagram shows how driving school's actors use the software system.

Design and Implementation: The business process model developed, the use-case diagram generated, and the conceptual model built are used for the further system model refinement during the steps of design and implementation according to the framework described in the previous section.

The interaction diagram may be partly generated from the use-cases and the conceptual model, or, alternatively, obtained directly from the business process diagram

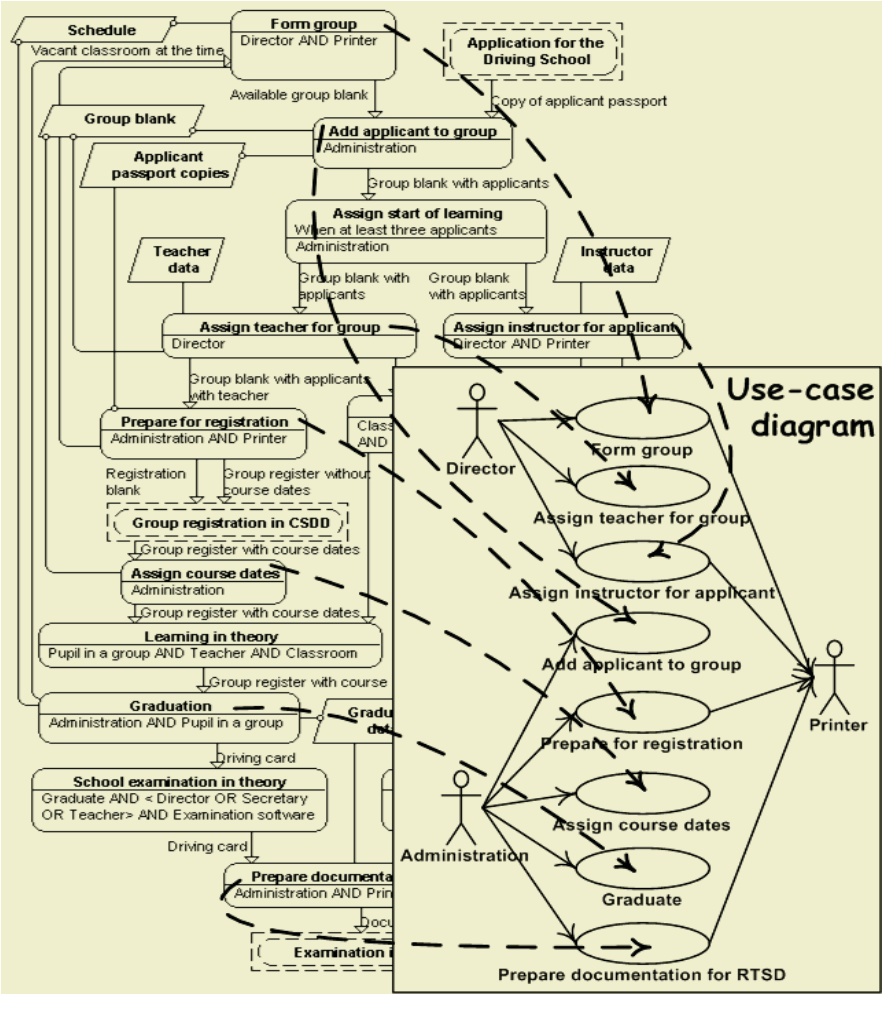

Fig. 7. Generation of an use-case diagram from business processes

as shown in Fig. 8 where generation of an interaction diagram for the use-case "Form group" is shown. As far as scenarios for realization of use-cases may be defined by decompositions of business processes (sub-processes) corresponding to the use-cases, sub-process diagrams serve for construction of object interaction. Information flows in sub-process diagrams help to find objects in message passing, and sub-processes are redefined as a messages passed between objects. The class diagram is constructed based on the information about object interaction and refines the structure of the conceptual model. Further implementation of the design model by components is based on traditional object-oriented approach.

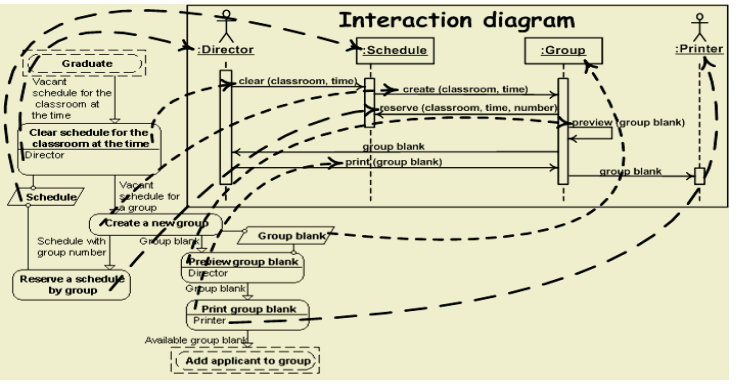

Fig. 8. Generation of a sequence diagram from a sub-process model

5 Conclusion

Today software systems should be built in a way they can support business agility. Therefore software development projects must deal with more complex and massive problem domain knowledge than years ago. This, in turn, requires processing problem domain knowledge more in the style of engineering than in the style of art. We analyzed several approaches of object oriented software development to identify the main differences in handling problem domain knowledge. Only the BPMD approach and the 2HMD approach use engineering at the problem domain level. The BPMD approach requires complete and consistent business process knowledge in the very beginning of the project. The 2HMD approach illustrates how sophisticated models and engineering based software development may be applied even in situations when complete business process knowledge is not provided at the beginning of the project and the most advanced experimental software development tools are not applied.

References

1. Sherrat M. Aligning costs with revenues, Financial Executive, October 2003, pp.59-62.
2. C.K. Prahalad, M.S. Krishnan, & Venkat Ramaswamy, The Essence Of Business Agility: Available at http://www.optimizemag.com/issue/011/leadership.htm
3. Meeting the agility challenge: Increasing business agility through adaptive IT infrastructure, Hewlett-Packard, 2002.
4. Witkop St. Driving business agility with Model Driven Architecture, EDS, 2003.
5. Sendall Sh. And Kozaczunski W. Model transformation: The heart and soul of Model Driven Software development, *IEEE Software*, September/October 2003, pp. 42-45.
6. "UML Specification. Ver.1.3", available at http://www.rational.com
7. Ambler Sc. W. Agile Model Driven development is good enough*, IEEE Software*, September/October 2003, pp.71-73.

8. MDA Guide Version 1.0.1, June 2003, available at http://www.omg.org/docs/omg/03-06-01.pdf
9. Nikiforova O., Kirikova M. *"Enabling Problem Domain Knowledge Transformation during Object Oriented Software Development"*, Conference of Information System Development, ISD'2003, Melbourne, Australia, 25-27 August 2003
10. Jacobson I., Booch G., Rumbaugh J. *The Unified Software Development Process*, Addison-Wesley, 1999.
11. Larman Cr. Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design, Prentice Hall PTR, 1998
12. Leffingwell D. & Widrig D. Managing Software Requirements: A Unified approach, Addison-Wesley, 2000
13. Martin J. & Odell J. *Object-oriented Methods: A Foundation*, Prentice Hall, 1995
14. Mathiassen L., Munk-Madsen A., Nielsen P. A. & Stage J. *Object-oriented Analysis & Design*, Marko Publishing House, 2000
15. Quatrany T. *Visual Modeling with Rational Rose 2000 and UML* (2nd ed.) Addison-Wesley, 2000
16. Rumbaugh J. Models Through the Development Process, *Journal of Object-oriented Programming*, May 1997, Vol. 10, No 2, pp. 5-8, 14.
17. Nikiforova O. "General Framework for Object-Oriented Software Development Process", Scientific Proceedings of Riga Technical University, Series – Computer Science. Applied Computer Systems, 13 vol., 2002.
18. Harmon P. *Business Process Change: A Manager's Guide to Improving, Redesigning, and Automating Processes, Morgan Kaufmann Publishers, 2003*
19. Atkinson C. and Kuhne Th. Model Driven Development: A Metamodelling foundation, in IEEE Software, September/October 2003.
20. The Impact of Agile Processes on Requirements Engineering, Advanced Development Methods, Inc. 2000, available at: http://www.agilealliance.com/
21. ArcStyler MDA-Business Transformer Modeling Style Guide for ARIS, Interactive Objects, 2002.
22. Bruce A. and Kutnick D. *Building Operational Excellence: IT People and Process Best Practices*, Intell Press, 2002.
23. *Organizing Business Knowledge:* The MIT *Process Handbook* (Th. W. Malone, K. Crowston, and G.A. Herman Eds.), MIT Press, 2003
24. Anderson, J.R.: *Cognitive Psychology and Its Implications*, W.H. Freeman and Company, New York, 1995.
25. Lausen S. Task descriptions as functional requirements, IEEE Software, March/April, 2003.
26. GRADE tools, GRADE Development Group, web-site -http://www.gradetools.com/
27. ARIS Toolset Available at: http://www.ids-scheer.com/
28. Kirikova M. "Modelling the boundaries of workspace: A business process perspective", *Information Modelling and Knowledge Bases XIII*, H.Kangassalo, H.Jaakkola, E. Kawaguchi, T. Welzer (eds.), IOS Press, Ohmsha, Amsterdam, Berlin, Oxford, Tokyo, Washington, DC, 2002, pp. 266-278.
29. Kirikova M. "Business Modelling and Use Cases in Requirements Engineering", *Information Modelling and Knowledge Bases XII*, H.Jaakkola, H.Kangassalo E. Kawaguchi (eds.), IOS Press, Ohmsha, Amsterdam, Berlin, Oxford, Tokyo, Washington, DC, 2001, pp. 410-420.