Object-Role Modeling as a Domain Modeling Approach

H.A. Proper[1], A.I. Bleeker[2], and S.J.B.A. Hoppenbrouwers[1]

[1] University of Nijmegen*, Sub-faculty of Informatics, IRIS Group, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU e.proper@acm.org, stijnh@cs.kun.nl

[2] Luminis, IJsselburcht 3, 6825 BS Arnhem, The Netherlands, EU araminte.bleeker@luminis.nl

Published as:

H.A. Proper, A.I. Bleeker, and S.J.B.A. Hoppenbrouwers. Object-role modelling as a domain modelling approach. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'04), held in conjunctiun with the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004)*, volume 3, pages 317–328, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology.

Abstract. This paper focuses on the potential role of the Object-Role Modeling (ORM) approach to information modeling for the task of domain modeling. Domain modeling concerns obtaining and modeling the language (concepts, terminologies, ontologies) used by stakeholders to talk about a domain. Achieving conceptual clarity and consensus among stakeholders is an important yet often neglected part of system development, and requirements engineering in particular.
This paper starts out with a brief discussion on the importance of domain modeling in system development. This is followed by an outline of the activities involved in proper domain modeling. We will then discuss why the ORM approach is, in principle, a good candidate for the tasks involved in domain modeling. This is further substantiated by a more detailed evaluation, both from a theoretical and a practical perspective.

1 Introduction

In today's business, software projects still often fail, and/or their costs tend to turn out to be higher than initially estimated [1]. Various different reasons may underlie this problem, be they political, organizational, economic (budgets) or process-oriented. We believe that one of the key factors contributing towards failure is a lack of clarity regarding the language and concepts used in the communication among the stakeholders involved in the development project. These stakeholders typically range from problem owners, contract authorities, perspective users, domain experts, to software engineers and system administrators.

Achieving conceptual clarity and consensus among stakeholders is an important yet often neglected part of modeling activities in system development. During system development, a myriad of models may be produced, ranging from high level sketches of the problem, via informal/formal requirements, to designs at several levels of technical detail. Underlying each of these models a set of domain specific concepts can be discerned, such as *client*, *order*, *stock*, etc. The process of obtaining and modeling these concepts, essentially modeling the language (concepts, terminologies; ontologies) used by stakeholders to talk about a domain, is what we refer to as *domain modeling*.

The absence of explicit and well managed domain models plays an important part in the failure of a significant number of system development projects [2]. Achieving clarity and consensus

* This paper results from the ArchiMate project (http://archimate.telin.nl), a research consortium that aims to provide concepts and techniques to support enterprise architects in the visualisation, communication and analysis of integrated architectures. The ArchiMate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Instituut, Centrum voor Wiskunde en Informatica, Katholieke Universiteit Nijmegen, and the Leiden Institute of Advanced Computer Science.

among stakeholders about the domain concepts used, boils down to answering the question: *what are we talking about?* The purpose of a domain model is therefore to uniformly define and scope the domain concepts in terms which the stakeholders understand and agree upon. For the stakeholders in the business environment, the domain model can provide a unified vocabulary and support understanding of the scope of the system; towards software architects and engineers it provides guidance in making implementation decisions (for example database design). For the project leader it can be an aid in planning and prioritization of the project.

Given that domain models are central in stakeholder communication, terminological aspects of modeling can best be tackled in direct relation to such models. This should ensure timely identification of terminology-related issues, yet keeps options open as to how such issues are to be solved or managed (that is, in which way, and in which development phase). The different models used during system development, and their underlying domain model(s), need to be communicated with different stakeholders. This puts an extra burden on the task of domain modeling, as a shared understanding must be reached of the concepts involved.

We are not alone in arguing the importance of domain modeling. For example, the Rational Unified Process [3] (RUP) explicitly underlines its importance as well. The RUP includes the notion of a *glossary*, which is used to define terminology specific to the business domain, explaining terms which may be unfamiliar to the reader of use-case descriptions or other project documents. Quite often, this document is used as an informal data dictionary, capturing the definitions of the terms. However, we believe that RUP's notion of a glossary, in particular when viewed as a "list of terms", is too light a mechanism for domain modelling.

In [4] the importance of proper domain modeling is argued as well. The book spents an entire chapter on the issues involved in domain modeling. It, however, does not provide explicit guidance to modelers, not does it provide an underlying theory of domain modeling. We believe that such an underlying theory should be developed, and that more guidance should be provided to domain modelers in practice. This article aims to take some first steps into this direction, by evaluating ORM from the perspective of domain modelling.

The aim of this article is to provide a brief analysis, both from a theoretical and practical perspective, of the complexities involved in domain modeling and the potential use of ORM for the tasks involved. We will identify several ambition levels concerning the domain modeling process (section 2) [5]. Based on this theoretical analysis, we will (section 3) then continue by discussing, from a theoretical perspective, how the ORM (Object-Role Modeling) [6] approach can be employed for the task of domain modeling. We will argue that ORM's roots in natural-language analysis makes it highly suitable for this task. Some alternative modeling approaches, such as OOSA [7] and the UML [8] will be discussed briefly as well. Before concluding, section 4 focuses on some practical experiences with the use of ORM in software development projects.

2 Domain Modeling

This section provides a brief discussion of the activity of domain modeling. A more detailed discussion on domain modeling, and its role in the system development process, can be found in [2, 5]. In general, the goals for doing (business) domain modeling are [2]:

1. articulate clear and concise meanings of business domain concepts and
2. achieve a shared understanding of the concepts among relevant stakeholders.

Our starting point is not the domain model (a conceptual *representation*), but the actors (typically people) that are somehow –and relevantly– involved in *discoursing about a domain*. They operate in an *environment of discourse*[3].

Based on the results reported in [10], we consider domain modeling in the context of system development to revolve around three streams of (mutually influencing) activities:

[3] For an in-depth discussion of the notion of "environment of discourse" in relation with the classic notion of "universe of discourse", see [9].

Scoping environments of discourse: The aim of this stream of activities is to scope the environments of discourse that are relevant to the system being developed, and determine the set of actors associated to each of these environments.

Concept specification: For each of the identified environments of discourse, the relevant business domain concepts should be specified in terms of their:
- meaning,
- relationships to other concepts (and the constraints governing these relationships) and
- possible names used to refer to them.

Concept integration: The concepts as identified and defined in the different environments of discourse may quite well clash. As a part of this, homonyms and synonyms are likely to hold between different environments of discourse. The aim of this stream of activities is to determine how to deal with this, and act upon it.

Since these streams of activities can be expected to influence each other, it is not likely that they can be executed in a strict linear order.

In general, the processes that aim to arrive at a set of concepts together with their meaning and names, are referred to as a *conceptualization processes* [10]. When, as in the context of software development, conceptualization is performed deliberately, as a specific task and with a specific goal in mind, it is referred to as an *explicit conceptualization process*. The above mentioned stream of activities called *concept specification* is an *explicit conceptualization process*. In [10, 2] a reference model for explicit conceptualization processes is provided. This reference model distinguishes five streams of activities or *phases* (each being a sub-stream of the *concept specification* stream of activities):

Assess domain and acquire raw material: Domain modeling always begins with a brief scan or assessment of the domain to get a feeling for scope, diversity and complexity of the domain, as well as to identify the relevant stakeholders for the domain. In addition, the activity aims to bring together input documents of all sorts that provide a basic understanding of the environment of discourse that is relevant to the environment of discourse under consideration.

Scope the concept set: In this phase, formal decisions are to be made regarding the concepts that somehow play a role in the environment of discourse and how these concepts interrelate.

Select relevant concepts: The goal of this phase is to focus on those concepts in the environment of discourse that bear some relevance to the system to be developed. These are the concepts that should be defined and named formally in the next step.

Name and define concepts: All of the concepts selected in the previous phase should be named and defined. Defining the concepts may also include the identification of rules/laws/constraints governing instances of the defined concepts.

Quality checks: Final quality checks on the validity, consistency and completeness of the set of defined concepts.

In executing the domain modeling activities, three levels of ambition can be discerned with which a modeler may approach the task of modeling a domain. These levels can, incidently, also be regarded as the order in which a novice modeler may learn the art of domain modeling:

Singular: This level of ambition corresponds to the modeling approaches as described in e.g. NIAM [11] and ORM [6]. It involves the modeling of a single *environment of discourse* based on complete input; usually in terms of a *complete* verbalization of (*only*) the relevant parts of the domain.

Elusive: At this level of ambition, modelers need to cope with the unavoidable iterative nature of the modeling process. As a modeling and/or system development process proceeds, the insight into the domain may increase along the way. This replaces the idealized notion of completeness of input with one of incremental input. The increments in the model are not related to a changing domain, but rather to improved ways of conceptualizing it.

Pluriform: At this next level of ambition, we recognize the fact that when developing a realistic system, we do not simply deal with one single unified environment of discourse (and related terminologies and concepts), but rather with a number of interrelated environments of discourse [9].

One may actually also distinguish a fourth level of ambition. The domains themselves are not stable; they evolve over time. As a result, what may have started out as a correct model of a domain, may become obsolete due to changes in the domain. However, we consider this issue to be beyond this paper. A more detailed discussion of the impact of evolution can be found in [9].

3 ORM as a domain modeling approach

In this section, the ORM [6] modeling approach is presented as a candidate for the tasks involved in domain modeling. In doing so, we will use the above identified ambition levels to structure our discussion.

3.1 Motivation for ORM

Before positioning ORM with respect to the identified ambition levels, we will first provide our motivation for considering (and using in practice) ORM as a domain modeling approach.

ORM [6], and its many variations such as NIAM, PSM and NORM, have a rich theoretical foundation dating back to at least the 1980s and 1990s [12, 11, 13, 14, 15, 16, 17, 18, 19]. Even though the UML [8] is used intensively during the development of software systems, ORM still has an important role to play in the early stages of system development. An active community of ORM users exists (see e.g. www.orm.net and www.inconcept.com), while Microsoft's Visio for Enterprise Architects even provides advanced support for ORM diagrams [20] as well as mapping to different development platforms.

One of the important features of ORM, making it highly suitable for the tasks involved in domain modeling, is its foundation in natural language analysis. ORM views the world in terms of objects playing roles, and approaches a domain by verbalizing examples of such objects in natural language. These verbalizations are used as a starting point to further develop and refine the model.

Another important feature of ORM is its elaborated modeling procedure which guide modelers in their task. In [6] a lengthy discussion of this procedure can be found. However, the ORM modeling procedure as such is too rich for the purposes of domain modeling, as it is originally geared towards conceptual *design* of a database system rather than the *analysis* of concepts playing a role in a business domain. Nevertheless, the general structure of the procedure can still be applied. The trick is to be less rigid about some of the modeling details, such as the specification of domains for value types, identification mechanisms for objects and advanced constraints.

3.2 Modeling a singular domain

At this level of ambition we are only interested in the modeling of a single *universe of discourse* based on *complete input*. The ORM modelling procedure provides a (natural language driven) way of executing a domain modeling process at this ambition level. The modeling procedure as described in ORM [6] identifies the following steps:

Step 1 – Transform familiar examples into elementary facts This step involves the verbalization in natural language of samples taken from the domain.

Step 2 – Draw the fact types and apply a population check In this step, a first version of the schema is drawn. The plausibility of the schema is validated by adding a sample population to the schema.

Step 3 – Trim schema and note basic derivations In this step, the schema is checked to see if any of the identified concepts are basically the same, and should essentially be combined. Furthermore, derivable concepts (e.g. sales-price = cost-price + mark-up) are identified.

Step 4 – Add uniqueness constraints and check the arity of fact types At this point, it is determined how many times an instance of an identified concept can play specific roles. For example, is a *person* allowed to *own* more than one *car*?

Step 5 – Add mandatory role constraints and check for logical derivations This step completes the basic set of constraints on the relationships in the schema, by stating whether or not instances of a concept *should* play a role. For example, for each *car*, the *year of construction* should be specified.

Step 6 – Add value, set-comparison, and subtyping constraints The ORM diagramming technique provides a rich set of graphical constraints. This step is aimed at specifying these constraints.

Step 7 – Add other constraints, and perform final checks Finally, there may be some constraints in the domain that cannot be expressed graphically. In this last step, these constraints can be specified.

In terms of our framework for domain modeling processes, this procedure constitutes a rather specific way of executing the *concept specification* stream of activities. The standard ORM modeling procedure is really geared towards the (conceptual) analysis of a domain in order to design a database, rather than a general analysis of concepts playing a role in a domain. The procedure presented above is not applicable to all situations and all modelers. During the *design* phase of a software system most of the seven identified steps are indeed needed. However, experienced modelers are also likely to merge steps 1-3, steps 4-5, as well as steps 6-7, into three big steps. The resulting three steps will generally be executed consecutively on a 'per fact' basis.

What the ORM procedure lacks is the proper documentation of a concept's definition. In ORM, concepts are essentially defined by the relationships they may have to other concepts and the constraints governing these relationships. However, what is not included in these definitions is a dictionary-like style definition/explenation of the concepts. This can easily be remedied by associating a such a dictionary-like style of definitions (not unlike RUP's notion of a glossary) to an ORM diagram. Note that some ORM tools, do allow modellers to add notes to the schema. This is, however, a rather ad-hoc mechanism. The ORM method does not provide any guidance in documenting the meaning of the concepts used in the models.

ORM is not the only modeling approach that is based on analysis in natural language. However, providing a full survey of such approaches is beyond the scope of this article. Nevertheless, one such approach is worth mentioning here. In [7] the Object-Oriented Systems Analysis (OOSA) method is presented. It uses a natural-language based approach to produce an Object-Relationship Model (accidently also abbreviated as ORM) that serves as a basis for further analysis. The *way of working* used is not unlike that of ORM. Its *way of modeling*, however, has a more sketchy nature and has been worked out to a lesser degree. A wide spectrum of modeling concepts are introduced (*way of modeling*) covering a wide range of diagraming techniques (not unlike the UML [8]).

3.3 Dealing with elusiveness

At this level of ambition we are still only interested in modeling a single and uniform *environment of discourse*. However, the assumption that we can base ourselves on complete input is dropped. In terms of the above framework for domain modeling, this ambition level assumes that:

- We only have to deal with one uniform environment of discourse, even though at the outset its scope may not yet be clear.
- Concept integration still only needs to take place within the given environment of discourse.

Dropping the assumption that we can base ourselves on complete input is quite realistic, as most real life domains can only be chartered out 'as we go along'. The past decades have seen a move away from linear software development to more iterative forms [3]. At the root of this development lies the observation that as the development process progresses, the insight into the domain, the requirements, the set of relevant stakeholders, technological (im)possibilities, etcetera, increases. This is supported by the observation that real life software development projects have a tendency to entail so-called wicked problems [21, 22]. A crucial property of wicked problems is that one does not fully understand a problem, until *a* possible solution has been developed. Developing such

a possible solution provides the necessary insights to enable developers to better understand the actual problem that needs solving [23].

Neither ORM nor OOSA provide mechanisms to deal appropriately with the iterative nature of software development. The UML may seem to provide this through its associated development process RUP. However, the UML modeling approach as such only provides a *way of modeling* and can hardly be seen to provide modelers with modeling guidelines in terms of a *way of working*. RUP, on the other hand, focuses on the software development process as a whole, and could just as well be combined with ORM or OOSA. In our opinion, further research into such an integration for the purpose of domain modeling is called for.

3.4 Dealing with pluriformity

Pluriformity concerns the existence of multiple terminologies within an environment of discourse, or the occurrence of similar concepts in different environments of discourse. The most obvious way in which pluriformity surfaces –and is dealt with– in domain modeling is in relation to *homonyms and synonyms*[4].

In [6, p74, p202], the occurrence of homonyms in stakeholder interaction is mentioned briefly, and is broadly approached as a problem that is to be solved: "you should get them to agree upon a *standard term*, and also note any *synonyms* that they might still want to use" (*ibid*, p74). This is the attitude towards homonyms and synonyms that, explicitly or not, appears to be embraced by most current domain modeling approaches. A similar attitude is voiced in [24, p194], where it is recommended that lists are kept of homonyms and synonyms from the domain under analysis. Generally, the existence of a homonym in a domain model is seen as ranging from mildly undesirable to damaging to the model; often, the very presence of a homonym in the model is considered a sign that the model is 'wrong'.

Some ORM tools allow modellers to deal with homonyms at a syntactical level, by allowing them to associate the name (form) of concepts with the same name to different name spaces. Ideally, however, these name-spaces should correspond to environments of discourse. Furthermore, the modelling procedure should also more explicitly provide guidelines on dealing with homonyms.

A somewhat more nuanced treatment of the phenomenon is provided by OOSA [7, p206, p208]. Here it is made explicit that under their approach, "We may [...] choose not to resolve homonym conflicts. Our [...] model does not assume, as many models do, that a reoccurrence of a name for the same construct makes the construct the same. For example, we may have *name* of *person* and *name* of *preferred customer group*. Here, *name* and *name* are homonyms and we may choose to leave them as they are. When we leave them the same, we have to disambiguate them by their context in the same way we disambiguate homonyms in everyday life". The option of relating disambiguation to context is thus modestly taken aboard.

We believe that ORM, as well as other modeling approaches that could be used for domain modeling, requires refinements in its modeling procedure to enable it to deal with pluriformity. Essentially denying pluriformity by stating "get them to agree upon a standard term" is not realistic in the face of large-scale practical application.

4 Domain modeling experiences with ORM

This section aims to illustrate the discussion of ORM provided above, with some practical experiences with the use of ORM as a domain modeling approach. It will do so from three different points of view:

- Modeling perspective (*way of modeling*);
- Process perspective (*way of working*);
- Communication & documentation perspective (*way of communicating*).

[4] Concepts are homonyms if they have a similar form and different meaning. Concepts are synonyms if they have a different form but a similar meaning.

The discussion below should in essence be regarded as the outcome of a brown-paper session focussing on the practical experiences of one of the co-authors of this article. Based on these experiences, combined with the theoretical perspective as discussed in the previous section, we are currently working on an adaptation of ORM's CSDP to a "Domain Modeling Procedure' (DMP). This adaptation will be developed hand-in-hand with its use in practical situations, using a research paradigm called *action research* [25].

4.1 Modeling perspective

Verbalization is key – Verbalization of aspects of a domain is at the very heart of the modeling process. Verbalizations are constructed with the aid of domain experts who provide input about the domain. Obtaining the correct input from domain experts is absolutely paramount.

Step one in ORM's CSDP dictates elaborated verbalizations. In practice, however, one will usually jump straight to drawing an initial diagram (step two). One will therefore hardly ever explicitly document all verbalizations as such.

Populating the schema – Though not necessarily written physically in the schema, populating the schema can be a good means for validation with domain experts. Domain experts will find concrete instances of the domain easier to understand which will help them to better understand the boundaries of the domain. Consequently, it is more likely that modeling errors and/or incorrect limitations of the domain surface quickly.

Restricted repertoire of constraints – ORM offers a wide variety of constraints. In general, uniqueness and total role constraints are used most often. ORM also caters for advanced constraints. In domain modeling practice, however, the use of these advanced constraints is likely to be limited for several reasons:

- Constraints make the boundaries of a domain very explicit which sometimes causes a feeling of unease with domain experts. What one finds in practice, is that constraints tend to apply "in principle"; there are often exceptions while domain experts like to keep their options open.
- The domain is not known well enough (e.g. because a future situation is being modeled), so domain experts cannot really be conclusive about constraints.
- Complex constraints are exceptional; many domains do not require the use of these constraints or can be modeled in another way, thus avoiding such constraints.

Identification – In the standard ORM modeling procedure, identification is an important issue for database design reasons. During domain modeling, however, the only relevant question is how concepts are uniquely identified in the domain by the stakeholder, if at all. There is not necessarily a one-to-one mapping from such a domain "identification scheme" to the identification scheme as intended by ORM. In fact, during domain modeling we are not yet certain if (parts of the) domain are to be mapped to a relational database or an object-oriented database. Only when there is a need to eventually map to a relational database, the need occurs to add elaborate identification schemes to the diagrams. For example, object-oriented database approaches offer implicit identification of objects by means of their so-called object identifiers.

4.2 Process perspective

Incremental modeling – The modeling process is usually executed in an incremental way, meaning that there is no rigid order in which the key steps of the ORM CSDP are executed. For example, in an iterative modeling process, one will not necessarily wait with adding constraints until step one and two of the CSDP have been completed for the entire domain. Furthermore, if an initial domain model already exists, e.g. as produced in the definition phase in support of requirements engineering, it will have to be used as a starting point for completion.

Also, the level of ambition of the domain model is determined by the purpose and scope of the system, as well as the delivery strategy. If the system is to be delivered in multiple releases it is generally not desirable to try to model the entire domain at once. A better strategy is to

first obtain a general impression of the scope and complexity of the domain and then continue to model incrementally (and iteratively), in line with the planned releases.

For example, during the requirements engineering phase of the software development life-cycle, when the main goal is to support the requirements engineering activities, the seven steps as described above are likely to be overkill. In such a context, modelers are likely to skip steps six and seven. The modeling procedure as discussed in [6] also requires modelers to identify how concepts (such as car, co-workers, patient, etc.) are identified in a domain (e.g. by means of a registration number, employee number, patient number, etc.). During the definition phase, these identification mechanisms are not likely to be relevant (*yet*).

Ultimately, the perfect model only exists in a perfect world. Making the right decisions about what to model, what terms to use and when to stop, requires skills, experience and talent of the modeler, part of which is highly intuitive.

Two-pass strategy – When applying ORM in practice we use a two-pass strategy: First we produce a "pure" conceptual model of the domain from the perspective of the ideal world. Next, we go over the model again and assess it for technical feasibility. Some modeling choices may be expensive to implement and for practical reasons we may want to use an alternative that is semantically equivalent. For example, ternary and objectified relationships may be flattened/reduced to binary relationships. These transformations are essentially design optimizations "at the conceptual level", as reported in [26, 6]. Another example is the use of exclusion constraints and subtypes. In ORM a subtype hierarchy can in some situations also be expressed in terms of entities and roles. Exclusion constraints can be applied both in the subtype hierarchy and between the roles. In many object oriented programming languages (e.g. Java and C++), however, sub classes (the logical equivalent of ORM subtypes) are disjunct by definition, whereas an exclusion constraint needs to be programmed explicitly. The latter is more expensive, therefore we will generally decide to use a subtype hierarchy rather than entities and roles in this situation. A detailed discussion of mapping between ORM diagrams and UML class diagrams can be found in [6].

***What* to model is the question** – ORM, just as any other modeling methodology, explains *how* to model but not *what* to model. The latter question may very well be a bigger challenge than applying the methodology itself.

The first consideration in a practical situation is to be aware of *why* we model. Most of the time modeling will be initiated in the context of a system development project and the goal will be to come up with a model that serves as a basis for the construction of the system. However, modeling makes the domain explicit and that is where different interpretations of terms and concepts will surface. Depending on how big the differences are and how important overcoming them is (e.g. are we dealing with a strategically important domain?), obtaining consensus about such terms and concepts may become an equally important goal of the modeling process. The goal(s) of modeling influence(s) the outcome of the process [27]. If consensus is very important, the modeling process itself may be far more important than the eventual result in terms of schemas and documents.

The role of the domain experts – Domain experts provide information about the domain, on the basis of which the modeler produces a model. The kind of involvement of the domain experts is determined by the importance and complexity of the domain, but also by whether the *current* situation is modeled or some desired *future* situation. If the current situation is being modeled, the resulting model may not yield any new insights to the domain experts and the more skeptical ones might say the model is "trivial" or not really adding anything new. In a way this is actually a good sign because the model apparently succeeded in specifying the relevant concepts in a recognizable way. A model of the current situation structures domain knowledge and unifies terms but is not explicitly intended to bring new insights to the domain experts in view of their specialism.

Modeling the future situation, however, puts other demands on domain experts, as they are now asked to think about how they believe their business is going to develop and what is important in future (software) systems. Especially when modeling an existing domain, modelers should always be very cautious about concepts that the domain experts consider to be "trivial". In this case the purpose of modeling is not to teach domain experts anything new, but leaving out concepts they find "trivial" eventually yields an incomplete or maybe even incorrect model.

Modeling and requirements engineering – Domain modeling is not a stand-alone activity. In projects it is generally carried out in close relationship with requirements engineering. In fact, the domain model defines the terms in which requirements (e.g. in terms of use cases) are being specified. This correlation between domain model and requirements provides a means to control quality and consistency in the development process. For example, if during requirements specification concepts surface that have no place yet in the domain model one should start wondering whether it concerns a new concept or a synonym of an already identified concept.

4.3 Communication & documentation perspective

Verbalization is compacted – Verbalizations are constructed with the aid of domain experts who provide input about the domain. In practice, one will usually not explicitly document all verbalizations in their traditional form, but choose a more natural way to communicate the essence of the domain; for example, a document in natural langauge that defines terms and explains relationships and constraints. This does require advanced writing skills, however, since natural language can be harmfully ambiguous.

Selective use of diagrams – An ORM diagram is an aid for the modeler but it can often not be shared as such with the domain experts since they usually do not "talk ORM". Even though one could explain how an ORM schema should be read, it is rather unlikely that the person will really be able to grasp the intrinsic semantics of the model.

Note that even though UML class diagrams can be used to express domain models as well, they have to deal with similar challenges. Even more, UML diagrams have an implementation oriented co-notation from the perspective of the stakeholders in the business domain. This co-notation actually disqualifies UML class diagrams for the purpose of domain modeling as domain modeling requires intensive communication with the stakeholders about the structures of their domain.

Consise diagrams – For the same reasons as discussed above, constraints (other than the straightforward frequency occurence constraints, such as mandatory roles and uniqueness) may be documented separately, rather than graphically in the schema.

Focus on stakeholder concerns – The above also implies that the schema is not always drawn completely. It depends on the stakeholder community what the best way of documenting is. Often natural language is a good way; sometimes other languages may be used, for example some mathematical notation or indeed an ORM or UML diagram.

5 Conclusions & further research

In this paper we argued the need for proper domain modeling in system development, and evaluated the ORM modeling approach as a potential domain modeling approach. In this evaluation, we considered both a theoretical and a practical perspective on domain modeling.

We find ORM, in principle, to be useful. However, further work is needed in order to adapt ORM's CSDP to the needs of domain modeling, leading to a proper "domain modeling procedure". This procedure should *in particular* also cater for the elusiveness and pluriformity ambition levels to domain modeling. We will not only need to consider ORM as relevant input, but also other conceptual modeling approaches, such as [8, 7]. In addition, we will also need to consider results and approaches in the field of ontological engineering [28, 29, 30].

References

1. The Standish Group International: CHAOS: A recipe for succes. Research report, The Standish Group International, West Yarmouth, Massachusetts, USA (1999).
   `http://www.standishgroup.com/sample_research/PDFpages/chaos199%9.pdf`
2. Bleeker, A., Proper, H., Hoppenbrouwers, S.: The role of concept management in system development – a practical and a theoretical perspective. In Gravis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology (2004) 73–82.

3. Kruchten, P.: The Rational Unified Process: An Introduction. 2nd edn. Addison-Wesley, Reading, Massachusetts, USA (2000). ISBN 0201707101
4. Rosenberg, D., Scott, K.: Use Case Driven Object Modeling with UML – A Practical Approach. Addison-Wesley, Reading, Massachusetts, USA (1999). ISBN 0-20143-289-7
5. Hoppenbrouwers, S., Bleeker, A., Proper, H.: Modeling linguistically complex business domains. Technical Report NIII-R0405, Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU (2004).
6. Halpin, T.: Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design. Morgan Kaufman, San Mateo, California, USA (2001). ISBN 1-55860-672-6
7. Embley, D., Kurtz, B., Woodfield, S.: Object-Oriented Systems Analysis – A model-driven approach. Yourdon Press, Englewood Cliffs, New Jersey, USA (1992). ASIN 0136299733
8. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modelling Language User Guide. Addison-Wesley, Reading, Massachusetts, USA (1999). ISBN 0-201-57168-4
9. Proper, H., Hoppenbrouwers, S.: Concept evolution in information system evolution. In Gravis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology (2004) 63–72.
10. Hoppenbrouwers, S.: Freezing Language; Conceptualisation processes in ICT supported organisations. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (2003). ISBN 9090173188
11. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design: a fact oriented approach. Prentice-Hall, Sydney, Australia (1989). ASIN 0131672630
12. Verheijen, G., Bekkum, J.v.: NIAM: an Information Analysis Method. In Olle, T., Sol, H., Verrijn-Stuart, A., eds.: Information Systems Design Methodologies: A Comparative Review. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1982) 537–590.
13. Wintraecken, J.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer, The Netherlands, EU (1990).
14. Bommel, P.v., Hofstede, A.t., Weide, T.v.d.: Semantics and verification of object-role models. Information Systems **16** (1991) 471–495.
15. Halpin, T., Orlowska, M.: Fact-oriented modelling for data analysis. Journal of Information Systems **2** (1992) 97–119.
16. Hofstede, A.t., Weide, T.v.d.: Expressiveness in conceptual data modelling. Data & Knowledge Engineering **10** (1993) 65–100.
17. Shoval, P., Zohn, S.: Binary-Relationship integration methodology. Data & Knowledge Engineering **6** (1991) 225–250.
18. De Troyer, O.: The OO-Binary Relationship Model: A Truly Object Oriented Conceptual Model. In Andersen, R., Bubenko, J., Sølvberg, A., eds.: Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering. Volume 498 of Lecture Notes in Computer Science., Trondheim, Norway, Springer-Verlag (1991) 561–578.
19. Habrias, H.: Normalized Object Oriented Method. In: Encyclopedia of Microcomputers. Volume 12. Marcel Dekker, New York, New York, USA (1993) 271–285.
20. Halpin, T., Evans, K., Hallock, P., Maclean, B.: Database Modeling with Microsoft's Visio for Enterprise Architects. Morgan Kaufmann, San Mateo, California (2003). ISBN 1558609199
21. Rittel, H., Webber, M.: Dilemmas in a general theory of planning. Policy Sciences **4** (1973) 155–169.
22. Conklin, J.: Wicked problems and social complexity. White paper, CogNexus Institute, Edgewater, Maryland, USA (2003).
`http://cognexus.org`
23. Budgen, D.: Software Design. 2nd edn. Pearson Education, Harlow, United Kingdom, EU (2003). ISBN 0201722194
24. Kristen, G.: Object Orientation – The KISS Method, From Information Architecture to Information System. Addison-Wesley, Reading, Massachusetts, USA (1994). ISBN 0201422999
25. Avison, D., Lau, F., Meyers, M., Nielsen, P.: Action research. Communications of the ACM **42** (1999) 94–97.
26. Halpin, T., Proper, H.: Database schema transformation and optimization. In Papazoglou, M., ed.: Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference. Volume 1021 of Lecture Notes in Computer Science., Gold Coast, Australia, Springer Verlag, Berlin, Germany, EU (1995) 191–203. ISBN 3540606726
27. Veldhuijzen van Zanten, G., Hoppenbrouwers, S., Proper, H.: System Development as a Rational Communicative Process. In Callaos, N., Farsi, D., Eshagian-Wilner, M., Hanratty, T., Rish, N., eds.: Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics. Volume XVI., Orlando, Florida, USA (2003) 126–130. ISBN 9806560019

28. Gruber, T.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N., Poli, R., eds.: Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands, Kluwer Academic Publishers (1993).
29. Wand, Y., Weber, R.: An Ontological Analysis of some fundamental Information Systems Concepts. In: Proceedings of the Ninth International Conference on Information Systems, Minesota, Mineapolis (1988) 213–226.
30. Kishore, R., Zhang, H., Ramesh, R.: A helix-spindle model for ontological engineering. Communications of the ACM **47** (2004) 69–75.