Modeling Linguistically Complex Business Domains

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

[1] University of Nijmegen*, Sub-faculty of Informatics, IRIS Group, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU stijnh@cs.kun.nl, e.proper@acm.org
[2] Luminis, IJsselburcht 3, 6825 BS Arnhem, The Netherlands, EU araminte.bleeker@luminis.nl

Published as:

S.J.B.A. Hoppenbrouwers, A.I. Bleeker, and H.A. Proper. Modeling linguistically complex business domains. Technical Report NIII-R0405, Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU, 2004.

Abstract. The paper focuses on business domain modeling as part of requirements engineering in software development projects. 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 requirements engineering. Domain modeling can play a key role in supporting it. This does, however, require a nuanced approach to language aspects of domain modeling as well as ambition management concerning its goals, and the procedure followed. We provide an analysis of the linguistic complexities involved, as well as of various levels of ambition concerning the domain modeling process. On top of the "classic" approach to modeling singular, stable domains, we distinguish aspects like incremental modeling, modeling of multiple terminologies within a domain, and domain evolution; we will elaborate on the first two aspects.

1 Introduction

In today's business, software projects still often fail and/or cost much more than initially estimated [1]. Various different reasons underlie this problem: political, organizational, economic (budgets) and software development reasons. This article focuses on the latter.

Software developers tend to be good at thinking in terms of technology enabled solutions; they focus on sound engineering. In the ideal case, this leads to software systems that are properly engineered and that operate well technically. However, projects may still fail because it is unclear what their underlying business need is and what problem is actually being solved. Not only do we want to build the system right, but we also want to build *the right system* for the business environment it is built *for* [2]. This requires sound analysis of the business domain[3].

Indeed we believe that a main cause of failure of system development projects lies in the requirements part. People involved (i.e. stakeholders) rarely know precisely what they want in the initial phases of a project; in addition (and partly as a consequence), there usually is a rather high level

* 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.

[3] In [3] a distinction is made between *usage world*, *subject world*, *system world* and *development world*, when discoursing about information system development. What refer to as business domain is essentially the subject world.

of "vagueness" concerning the very concepts used to formulate the initial requirements [4]. Also, in any multi-stakeholder development project, the language used by stakeholders to formulate requirements can be expected to differ between stakeholders –at least initially. Not only will different stakeholders often use different terms to verbalize similar ideas, but –rather more treacherously– even if the same terms are used, their underlying meaning may differ to a considerable extent. Finding out what everyone means by the terms they use is an activity that may demand considerable effort, but it is a highly useful if not unavoidable part of the requirements engineering process. This line of thought is explored in the paper, in particular in relation to the activity of *domain modeling*.

In many contemporary projects, a chief item in the process of constructing, formulating and communicating about requirements is some sort of *business domain model*. The absence of explicit and well managed [5] business domain models[4] plays an important part in the failure of a significant number of system development projects. Given that business domain models are central in stakeholder communication, terminological aspects of modeling can best be tackled in direct relation to such models (i.e. rather than in relation to, for example, construction models; see section 2). 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).

It should be noted that the fields of conceptual modelling, requiremens specification, ontology engineering, etc, all provide means to define concepts. The aim of this article is, to investigate the relationship between the definition of concepts (in particular as part of requirements engineering) and the use of language by human beings. This requires a bridging between modeling as it occurs in system development and theories of a more linguistic origin.

The structure of this paper is as follows: first we further explore the relation between domain modeling, natural language, and requirements engineering (section 2). In section 3, we provide some theoretical background concerning our notion of 'linguistic complexity'. In section 4, we present an analysis of the domain modeling challenge as we envisage it, and its tight relation to matters of language and the process of conceptualization. We also introduce a distinction between four levels of ambition for domain modeling. Three of these are elaborated on in section 5; we focus on the modeling process, referring to some prominent existing natural language-based domain modeling methods. Section 6 provides a brief conclusion.

2 Domain Modeling and Natural Language in Requirements Engineering

Traditionally, the definition part of a software development process focuses on the requirements which should be met by the system:

1. What should the system do? (functional requirements)
2. How well should it do this? (non-functional requirements)

There is, however, a third aspect that needs to be considered when specifying *the right* system: *what are we talking about*, in other words, what are the concepts in the domain in terms of which the system to be developed will provide its functionality? For stakeholders to trust the language they speak among each other is a crucial aspect of building general confidence and consensus. The need to involve natural language in this specification process [6, 7], and the impact of terminology control for over-all quality management [8], should not be underestimated. If confusion and imprecision

[4] In the rest of this paper, if we use the term "domain", we refer to a "business domain" unless indicated otherwise.

concerning the use and definition of concepts enter a software development project from the start, unless it is dealt with appropriately they will continue to haunt all concept use in the project.

The language aspect of software development can be strongly improved upon by means of business domain modeling, i.e. *modeling the intrinsic structure and boundaries of the business domain in terms of the language spoken in it*. Its results constitute a *domain model*. Such a model *defines the relevant concepts in the business domain and the relationships between them. Its purpose is to clearly and explicitly define and scope the business domain in language that the stakeholders understand and agree upon*. Consequently, it is in terms of business domain concepts that the functional requirements and qualities of the planned system should be specified (e.g. in terms of use cases from the UML [9]) to assure that the (non)functional requirements are specified in a language that stakeholders understand and appreciate. We focus on *explicit* domain modeling, that is, domain modeling that results in explicit models that play an official role during system development.

Towards the stakeholders in the business environment, the domain model provides a *terminology* [10, p13-4] and an understanding of the scope of the system; towards software engineers it provides guidance in making design decisions (for example, database design). For project leaders it can be an aid in planning and prioritizing a project.

The question: *what are we talking about* is not only limited to the world of requirements. It equally well applies to system design. In fact, one might state that it is a question that forms an undercurrent in system development as a whole. It should be noted that during the initial stages of system development, one may chose not to provide a precise meaning for relevant concepts (yet). For example, using a metaphorical meaning of concepts pertaining to a future business/system, may actually stimulate innovative thinking. Focussing on a precise definition to early may stifle innovation.

The software development life cycle generally consists of four essential stages: definition, design, construction and deployment. The definition stage focuses on *what* system is needed, and *why* it is needed. The design stage concerns *how* the system will be constructed. The construction stage deals with the actual building of the system, i.e. putting all the components together. In the deployment stage, the system is actually brought to life in its organizational context.

There is no single mandatory order in which to execute the work involved in the above mentioned stages. Several strategies exist, for example, linear, incremental and iterative. (Business) domain modeling cannot be very strictly placed in any specific stage; however, most of the work is done during the definition stage. In this stage, the relevant stakeholders, sub-domains and terminologies are identified, and the boundaries of the domain are explored.

In the design stage the domain model may be further refined, but as during the construction stage, the domain model will guide developers. During the design stage the domain is transformed into a *construction* (domain) model. The construction model extends the business domain model with concepts that are introduced for sound engineering reasons only. The business stakeholders do not necessarily recognize any of these concepts, nor do they need to.

Note that the construction model should never introduce new business concepts by itself. It should "inherit" all business concepts from the domain model. If during construction new concepts appear, then it should be carefully assessed whether they are really new and need to be incorporated. If so, their place is in the business domain model. Conversely, if the planned system precisely covers the modeled domain then each business domain concept must be present in the construction model. If not, then there apparently are irrelevant concepts after all. Obviously, a planned system does not always have to precisely cover a modeled domain: the planned system may be delivered in releases, or the domain model may contain a part that is not realized in a system, for example manual business processes.

3 Linguistic complexity in domain modeling

We do not refer to syntactic or lexical complexity of the domain as such. With "linguistic complexity of domains" we primarily refer to the possible existence of *multiple terminologies* within a designated domain, related to particular stakeholders (either individuals or groups) involved in that domain. Also, it may be relevant to consider other, related domains and how their terminologies/concepts influence (or are influenced by) the chief domain focused on. This type of complexity will be later on referred to as *pluriformity* of domains. In addition (and secondarily), we refer to the linguistic complexity caused by *different stages in refinement of and agreement about* a domain model. This type of complexity will later on be referred to in relation to the *elusiveness* of domains.

To ensure a precise and concise definition of the first sort of linguistic complexity, we will now indulge in some basic defining using some standard set theory, loosely based on [11, 79-82].

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*[5]. Let $\mathcal{DS}$ be a set of domains and let $\mathcal{ED}$ be a set of environments of discourse. An environment of discourse corresponds to *one* domain that is discoursed about:

$$\mathsf{Domain} : \mathcal{ED} \rightarrow \mathcal{DS}$$

Within each environment of discourse, we can identify a number of actors involved with the environment, and the complete set of concepts they use in discourse.

Let $\mathcal{AC}$ be the set of actors in the universe and let $\mathcal{CO}$ be the set of concepts. Following [13, 11], we define a concept as the combination of a *form* and a *meaning*. The community of actors involved in an environment of discourse is provided by:

$$\mathsf{Community} : \mathcal{ED} \rightarrow \wp(\mathcal{AC})$$

The set of concepts used in an environment of discourse is yielded by:

$$\mathsf{Concepts} : \mathcal{ED} \rightarrow \wp(\mathcal{CO})$$

Note that $\mathsf{Concepts}(d)$ is what is traditionally called a universe of discourse: the set of concepts used in the discourse about some domain.

Next, let $\mathcal{LA}$ be a set of languages. Thus, under our definition, which is strongly pragmatic in nature, a language is a language only if it is actually used in some environment of discourse. We model this as a function:

$$\mathsf{EoD} : \mathcal{LA} \rightarrow \mathcal{ED}$$

Consequently, a language is used by a community of actors:

$$\mathsf{Community} : \mathcal{LA} \rightarrow \wp(\mathcal{AC})$$

This community is a part of the community of the environment of discourse:

Axiom 1

$\forall_{l \in \mathcal{LA}} [\mathsf{Community}(l) \subseteq \mathsf{Community}(\mathsf{EoD}(l))]$

A language also has a set of concepts associated with it:

$$\mathsf{Concepts} : \mathcal{LA} \rightarrow \wp(\mathcal{CO})$$

This set of concepts is part of the concepts used in the environment of discourse:

[5] For an in-depth discussion of the notion of "environment of discourse" in relation with the classic notion of "universe of discourse", see [12].

Axiom 2

$\forall_{l \in \mathcal{LA}} [\mathsf{Concepts}(l) \subseteq \mathsf{Concepts}(\mathsf{EoD}(l))]$

Consequently, it is quite possible that different actors within one environment of discourse use different concepts in that discourse (implying possible hampering or even failing of communication between actors). Such linguistic complexity may or may not be acceptable or manageable, but at least in the initial stages of the domain modeling process it is normally a fact of modeling life. This complexity does, however, not prevent the environment of discourse to be related to a single domain: the domain is not identical to the domain *model*, it is more like a topic, or a focus. Therefore (in line with [14]), strictly speaking the domain cannot be charted objectively, only *constructively* through the various conceptualizations occurring in the environment of discourse.

So it is possible that within one environment of discourse, a number of sub-languages is distinguished, possibly related to sub-groups of actors within the language community belonging to the environment of discourse. It is also possible that various environments of discourse are distinguished that are related to one and the same domain, and even that a particular concept may be used by a group or individual active in various different environments of discourse. Finally, it is possible that a particular terminology (i.e. an explicitly agreed (sub)language) is *imported* or even *imposed* from outside an environment of discourse.

As for the secondary form of linguistic complexity, it can theoretically be seen simply as our primary complexity extended with the concept of time (i.e. different states/terminologies over time). However, the practical implications of this are more interesting. They center round properly managed awareness of and communication about the particular terminology appropriate for a particular conversation: with a particular stakeholder, at a particular time. Importantly, this requires some conceptual/linguistic *flexibility* from actors involved, or alternatively a rather advanced mechanisms for the administration of and translation between different terminologies within one environment of discourse. Cumbersome as this may seem, it reflects the reality of linguistic complexity in (business) domains; complexity that has to be dealt with one way or another.

4 The domain modeling challenge

In this section we turn to the question *how to model a domain*; a question to which there is no simple, one-size-fits-all answer.

4.1 Goal-bounded and communication-driven

Some modeling approaches, such as NIAM [15] and ORM [16], suggest or prescribe a detailed procedure. Practice shows, however, that experienced modelers frequently deviate from such procedures [17]:

> In most cases, [the information engineers] stated that they preferred to pay attention to a specific part of the problem domain, usually to fill clear lacunae in their insights in the problem domain. Their momentary needs strongly influenced the order in which the several modelling techniques were used. Modelling techniques were used as a means to increase insights or to communicate insights, be it in the problem domain itself or in a specific solution domain.

Yet deviating from a modeling procedure should be done with some caution. While a pre-defined modeling procedure should never become "an excuse to stop thinking", situational specificity should not become an excuse for taking an ad-hoc approach to the modeling effort. A more stable anchor is needed upon which modelers can base themselves when making decisions during the

modeling process. We believe that domain modeling requires a goal-bounded and communication-driven approach. With goal-bounded we hint at the fact that when modeling a domain, a modeler is confronted with a plethora of modeling decisions. These decisions range from the modeling approach used, the intended use of the results, to decisions pertaining to the model itself. For example:

- What parts of the domain should be considered relevant?
- What is the desired level of detail and formality?
- To what level should all stakeholders agree upon the model?
- Should the model be a representation of an actual situation (*system analysis*) or of a desired situation (*design*)?
- Should the model be a representation of *what* a system should do, or should it be a representation of *how* a system should do this?
- Should a certain phenomenon in the domain be modeled as a relationship, or is it an object on its own?

Modeling goals essentially provide the *means* to *bound* modeling space.

In most situations where a domain needs to be modeled, the modeler cannot merely passively observe the domain. Modelers will need to interact with representatives from the domain. These representatives then become *informers* (who are likely to also have a stake with regards to the system being developed). Therefore, modelers will need to communicate intensively with the informers in order to refine the model. What is more, numerous domain models that are produced during system development will need to be accepted and agreed upon –validated– by the informers (being stakeholders of the future system). The claim has often been voiced that in modeling practice, 'the process is just as important as the end result', suggesting that a correct end-result is not always a guarantee for success. A domain model should ideally be a product of a *shared understanding of a domain's stakeholders*. It requires a 'buy-in' by all stakeholders involved. A domain model that is correct from a theoretical or technical point of view but does not have the required support from the key stakeholders is arguably worse than a domain model with some flaws that does have such support.

A modeling process can thus be seen as a communication-driven process [18, 4]. The principles of natural language driven modeling approaches [15, 19, 20, 16] can be used as a basis for shaping the communication process between informer and modeler.

4.2 Aspects of a method

When considering a modeling approach or method, several aspects thereof can be discerned [21, 22]. An important distinction to be made is that between a product oriented perspective and a process oriented perspective. In terms of the framework presented in [21, 22] these are referred to as the *way of modeling* and *way of working*, respectively:

Way of modeling: The way of modelling provides an abstract description of the underlying modelling concepts together with their interrelationships and properties. It structures the models which can be used in the information system development, i.e. it provides an abstract language in which to express the models.

Way of working: The way of working structures the way in which an information system is developed. It defines the possible tasks, including sub-tasks and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

In the case of domain modeling, the *way of working* represents the process followed when modeling a domain. In the following sections, we will mainly elaborate on this aspect. The *way of modeling* used for domain modeling is likely to be prescribed by a diagramming technique such as ORM diagrams [16], ER diagrams [23] or UML class diagrams [9].

4.3 The process of domain modeling

In general, the goals underlying (business) domain modeling are [5]:

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

Based on the results reported in [11], we consider domain modeling in the context of system development to chiefly concern three streams of (mutually influencing) activities:

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)
- possible names used to refer to them

Concept integration: The concepts as identified and defined in the different environments of discourse may well clash. As a part of this, homonyms and synonyms are likely to hold between different terminologies. The aim of this stream of activities is to determine how to deal with this, and act upon it. In section 5.3 we will elaborate on this.

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 *conceptualization processes* [11]. 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 such an explicit conceptualization process. In [11, 5] a reference model for conceptualization processes is provided. This reference model distinguishes five streams of activities or *phases*:

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.[6]

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.

These streams should essentially be regarded as sub-streams of the concept specification stream.

[6] In an earlier version of this framework, this was referred to as *scoping the universe of discourse*.

4.4 Ambition levels for dealing with linguistic complexity

In the remainder of this article we discuss several aspects of the domain modeling process and the role of natural language therein. We have made a distinction between four levels of ambition at which a modeler may approach the task of modeling a domain. These levels can 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 [15] and ORM [16]. 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. Also see section 3.

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 [12]. Also see section 3.

Evolving: The final ambition level recognizes the fact that domains themselves are not stable; they evolve over time [12]. As a result, what may have started out as a correct model of a domain, may become obsolete due to changes in the domain. New concepts may be introduced, or existing ones may cease to be used. However, subtle changes may occur as well, such as minor changes in the meaning of concepts, or the forms used to represent them.

In the next section, we will discuss domain modeling at the *singular*, *elusive* and *pluriform* levels of ambition. The *evolving* level is omitted from this article, as it is applies generally, i.e. is not typically language-related.

5 Meeting the challenge

This section aims to discuss the domain modeling process with respect to three of the identified levels of ambition: *singular*, *elusive* and *pluriform*. We will structure our discussion by using the framework of activity streams for domain modeling as introduced in the previous section.

5.1 Modeling a singular domain

At this level of ambition we are only interested in the modeling of a single environment of discourse based on complete input. In terms of the above framework for domain modeling, this ambition level assumes that:

- No (further) scoping of the environment of discourse is needed
- The domain has been assessed and raw material is available
- Concept integration only needs to take place within the given environment of discourse

Natural language driven modeling approaches like NIAM [15] and ORM [16] concern elaborately described ways of executing a domain modeling process at this ambition level. For example, the modeling procedure as described in ORM [16] identifies the following steps:

Step 1 – Transform familiar examples into elementary facts
Step 2 – Draw the fact types and apply a population check
Step 3 – Trim schema and note basic derivations
Step 4 – Add uniqueness constraints and check the arity of fact types
Step 5 – Add mandatory role constraints and check for logical derivations
Step 6 – Add value, set-comparison, and subtyping constraints
Step 7 – Add other constraints, and perform final checks

In terms of our framework for domain modeling processes, this procedure constitutes a rather specific way of executing the *concept specification* stream of activities. It 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.

Even though the above order is very explicit, and therefore well suited for educational purposes, a goal-bounded approach to domain modeling requires a more refined view. The key question concerns the *goal* for which a domain is modeled. During the *definition* phase of the software development life-cycle, when the main goal is to support requirements engineering activities, the seven steps as described above are likely to be overkill. In such a context, modelers are likely to skip steps 6 and 7.

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' base. In other words:

1. For each fact type, execute 1-3
2. For each fact type, execute 4-5
3. For each fact type, execute 6-7

ORM is not the only modeling approach that is based on analysis of natural language. However, providing a full survey of such approaches is beyond the scope of this article. Nevertheless, two approaches are worth mentioning here. In [19] the Object-Oriented Systems Analysis 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. The KISS approach, as reported in [20], also uses natural language analysis as its basis. It provides some support in terms of a *way of working*, but does this in a rather prescriptive fashion that presumes some very particular (and limited) intermediary goals. A wide spectrum of modeling concepts are introduced (*way of modeling*) covering a wide range of diagraming techniques (not unlike the UML [9]).

Independent of the approach used, a modeling process always needs to be flanked by a continuous communication process with the stakeholders [4]. Communication brings along the aspect of documentation. Modeling itself can hardly do without face-to-face discussions; however, the (intermediate) results need to be recorded in such a way that they can be communicated effectively to the stakeholder community [6, 7]. In this respect we could argue that any modeling approach also needs a *way of communication/documenting*. Since documentation serves the purpose of communication, the documentation *language* should align with the accepted language concepts in the domain. In practice it turns out that graphical notations such as ORM or UML diagrams are not the most obvious way to communicate a model to stakeholders, since most domain stakeholders do not comprehend this kind of "IT language". Often, it is better to use more intuitively readable diagrams and natural language to communicate concepts and their relationships and constraints, while occasionally, a more mathematical or algorithmic style may be useful in certain expert domains.

5.2 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. This is quite realistic, as most real life domains can only be charted as we go along. The past decades have seen a move away from linear software development to more iterative forms [24]. 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, and technological (im)possibilities increases. This is supported by the observation that real life software development projects have a tendency to involve so-called wicked problems [25, 26]. A crucial property of wicked problems is that one does not understand the *real* 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 [27].

The role of domain modeling in software development is to serve as a basis for the different stages (definition, design, construction and deployment) of the development process. Domain modeling should therefore closely follow the iterations of the development process as a whole. The ambition levels (i.e. the modeling *goal*) of each of the iterations of a domain model should be attuned to the ambition levels of the iterations of the entire development process. This entails that at times, a less than perfect and/or completed domain model needs to be settled for. For example, incremental software development [24] typically requires tangible results in an early stage. This requires domain modelers to be pragmatic about the quality of their initial models. Also, due to the wicked nature of real life development projects [26, 27], there is quite often no such thing as *the final* or *the best* domain model.

Elusiveness in domains also changes the nature of the stakeholder communication process. Assuming complete input, the modeler is mainly busy capturing and structuring what the stakeholders already know. The stakeholders mostly play an informing and validating role but probably do not gain many new insights in their domain. Yet elusiveness introduces uncertainty, so stakeholders may need to become more creative, and willing to develop new insights or change existing ones, together with the modeler. The role of the stakeholders shifts from mostly informing towards more constructive.

Neither of the three earlier reported modeling approaches ORM [16], OOSA [19] and KISS [20] provide mechanisms to deal appropriately with the iterative nature of software development. The UML [9] may seem to provide this through its associated development process RUP [24]. 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, OOSA or KISS.

5.3 Dealing with pluriformity

As discussed at some length in section 3, 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*[7].

In [16, 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 [stakeholders] 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, is embraced

[7] Concepts are homonyms if they have a similar form (label) and different meaning. Concepts are synonyms if they have a different form but a similar meaning.

by most current domain modeling approaches. A similar attitude is voiced in [20, 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'.

A somewhat more nuanced treatment of the phenomenon is provided by [19, 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 pluriformity to context is thus modestly taken aboard.

However, there is more to pluriformity than just the resolving of homonyms/synonyms. In particular in the case of homonyms, a wold of conceptual difference may be hidden behind what seems at first a mere matter of 'labeling'. [19, p206] hint at this when they say that "resolution of structural conflicts is much more difficult than resolutions of name conflicts". Indeed, the occurrence of homonyms may trigger further exploration of a part of the domain that perhaps was before considered unproblematic. In the authors' experience, amazingly fundamental differences between (and disagreements about) the conceptions of a domain may surface through the exploration of what at first seems to be a minor matter of terminological alignment. Consequently, from a process point of view there is much more to pluriformity than getting rid of form/meaning ambiguity; an extensive process of conceptual negotiation and gradual construction of a shared conceptual model is often required to achieve genuine mutual acceptation of and agreement about a domain model.

Unfortunately, such consensual modeling is often too much to ask for. If stakeholder are inflexible in their concept use (for whatever reason), it may be simply impossible to "get them to agree upon a standard term". In such a case, pluriformity may well have to be modeled explicitly, in relation to the particular context(s) (environments of discourse, communities) the respective conceptual variations are associated with. This may or may not lead to *design decisions* later on. It is quite possible that some pluriformity cannot be resolved, for example if some stakeholder is not willing or capable of the conceptual flexibility required to reach consensus, or if the system designed incorporates multiple 'use domains'. Ignoring this point is counterproductive.

6 Conclusion

We have presented an overview and discussion of language-related issues in business domain modeling, in particular as part of the requirements engineering phase of software development. We emphasized the central role of communication with various parties (stakeholders, modelers) in this process. We focused on linguistic complexity in domain modeling stemming from the variety of languages spoken in the environments of discourse related to a domain: its *pluriformity*. In addition, some linguistic complexity occurs because of what we have called the *elusiveness* of domain models.

We presented an analysis of process-oriented aspects of domain modeling (mostly in relation to *ways of working*) and how they are influenced by various language and communication related issues. This led us to emphasize the importance of a *goal bounded* approach to modeling.

We defined four levels of ambition concerning business domain modeling. In view of our subsequent discussion concerning ways to meet the challenges involved, we conclude only the first level of ambition (singular domain) is satisfactorily covered by current domain modeling methods. The second and third level (elusive and pluriform domains) call for more nuanced approaches and better guided modeling processes. The fourth level (evolution) has not been discussed at any length, and is left for future research.

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. Vliet, J.v.: Software Engineering – Principles and Practice. 2nd edn. John Wiley & Sons, New York, New York, USA (2000). ISBN 0471975087
3. Mylopoulos, J.: Techniques and languages for the description of information systems. In Bernus, P., Mertins, K., Schmidt, G., eds.: Handbook on Architectures of Information Systems. International Handbooks on Information Systems. Springer, Berlin, Germany, EU (1998). ISBN 3-540-64453-9
4. 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
5. 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.
6. Frederiks, P., Weide, T.v.d.: Deriving and paraphrasing information grammars using object-oriented analysis models. Acta Informatica **38** (2002) 437–88.
7. Frederiks, P.: OBject-Oriented Modeling based on Information Grammars. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands (1997). ISBN: 90-9010338-4
8. Wright, S.: Terminology and total quality management. In Wright, S., Budin, G., eds.: Handbook of Terminology Management – Volume 2: Application-Oriented Terminology Management. John Benjamins, Amsterdam, The Netherlands, EU (2001). ISBN 9027221553
9. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modelling Language User Guide. Addison-Wesley, Reading, Massachusetts, USA (1999). ISBN 0-201-57168-4
10. Wright, S., Budin, G., eds.: Handbook of Terminology Management – Volume 1: Basic Aspects of Terminology Management. John Benjamins, Amsterdam, The Netherlands, EU (1997). ISBN 9027221545
11. Hoppenbrouwers, S.: Freezing Language; Conceptualisation processes in ICT supported organisations. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (2003). ISBN 9090173188
12. 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.
13. Hoppenbrouwers, J.: Conceptual Modeling and the Lexicon. PhD thesis, Tilburg University, Tilburg, The Netherlands, EU (1997). ISBN 90-5668-027-7
14. Falkenberg, E., Hesse, W., Lindgreen, P., Nilsson, B., Oei, J., Rolland, C., Stamper, R., Van Assche, F., Verrijn-Stuart, A., Voss, K., eds.: A Framework of Information Systems Concepts. IFIP WG 8.1 Task Group FRISCO (1998). ISBN 3-901-88201-4
15. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design: a fact oriented approach. Prentice-Hall, Sydney, Australia (1989). ASIN 0131672630
16. 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
17. Verhoef, T.: Effective Information Modelling Support. PhD thesis, Delft University of Technology, Delft, The Netherlands, EU (1993). ISBN 9090061762
18. Frederiks, P., Weide, T.v.d.: Information modeling: the process and the required competencies of its participants. In: 9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004), Manchester, UK (2004) Technical Report Radboud University Nijmegen NIII-R0324.
19. Embley, D., Kurtz, B., Woodfield, S.: Object-Oriented Systems Analysis – A model-driven approach. Yourdon Press, Englewood Cliffs, New Jersey, USA (1992). ASIN 0136299733
20. Kristen, G.: Object Orientation – The KISS Method, From Information Architecture to Information System. Addison-Wesley, Reading, Massachusetts, USA (1994). ISBN 0201422999
21. Seligmann, P., Wijers, G., Sol, H.: Analyzing the structure of I.S. methodologies, an alternative approach. In Maes, R., ed.: Proceedings of the First Dutch Conference on Information Systems, Amersfoort, The Netherlands, EU (1989).

22. Wijers, G., Heijes, H.: Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In Steinholz, B., Solvberg, A., Bergman, L., eds.: Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering. Volume 436 of Lecture Notes in Computer Science., Stockholm, Sweden, EU, Springer-Verlag (1990) 88–108. ISBN 3540526250
23. Chen, P.: The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems **1** (1976) 9–36.
24. Kruchten, P.: The Rational Unified Process: An Introduction. 2nd edn. Addison-Wesley, Reading, Massachusetts, USA (2000). ISBN 0201707101
25. Rittel, H., Webber, M.: Dilemmas in a general theory of planning. Policy Sciences **4** (1973) 155–169.
26. Conklin, J.: Wicked problems and social complexity. White paper, CogNexus Institute, Edgewater, Maryland, USA (2003).
`http://cognexus.org`
27. Budgen, D.: Software Design. 2nd edn. Pearson Education, Harlow, United Kingdom, EU (2003). ISBN 0201722194