Towards formalization of the information modeling dialog

Sander Bosman Theo van der Weide
sanderb@cs.ru.nl Th.P.vanderWeide@cs.ru.nl
Computing Science Institute, Radboud University Nijmegen
The Netherlands

Abstract

This paper proposes a formalization of the modeling dialog between domain expert and system analyst, in which domain knowledge is specified and refined until a formal conceptual model of a domain results. This formalization is then used to describe the task of the system analyst in two settings. The ultimate goal of this work is to enable computer support for the system analyst in the modeling process.

1 Introduction

1.1 Domain modeling

Domain modeling (also known as *information modeling* or *conceptual modeling*) is the process of creating a formal model of the concepts and relations that 'exist' in some domain of interest.

In general this is done by eliciting knowledge from a *domain expert*: some person or group having specific knowledge of this domain. This knowledge then is abstracted and represented in some (graphical) modeling language.

A model is always made with a specific *goal* in mind. The goal determines the required properties of the model: its form, type of information it contains, precision, etc. Therefore, a model can be characterized to be a specification in some modeling language that fulfills the modeling properties required by the goal (Falkenberg et al., 1998). Often required properties include:

Completeness The model must completely describe the relevant aspects of the domain of interest.

Consistency The model must be internally consistent, as well as consistent with the domain itself.

Precision The model must describe the domain at a certain abstraction level, containing enough precision but avoiding unnecessary detail.

Compactness A model usually is regarded as a 'generative device', that efficiently describes (can generate) all relevant information in the domain. In other words, a model usually should be *intentional* instead of *extensional*.

Formality The model, being a representation, should have a clear syntax and semantics.

Strictly speaking, a model is a representation of a *mental model* of a domain, instead of the domain itself. The domain expert perceives the domain, and constructs (conceives) a mental model of that domain. Therefore, the information provided by a domain expert is always subjective and reflects only the structure of the domain *as seen by the domain expert* (Bosman and Weide, 2003). The subjective nature of models is shown in figure 1.

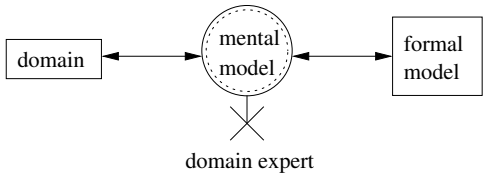

Figure 1: *Relation between domain, mental model and formal model.*

1.2 Dialogs

Creating a formal domain model requires specialized skills, that domain experts are not supposed to have. Roughly speaking, a domain expert can be characterized as someone with superior detail knowledge, but often minor abstraction skills (Frederiks and Weide, 2004). Furthermore, we may not assume a domain experts' mental model of the domain to be complete and consistent in each respect. The mental model may be *good enough* for the normal tasks the expert performs; however, formal domain models usually have stricter requirements on precision, consistency and completeness, and refinement of the mental model is needed.

As a result, an initial specification by a domain expert will not fulfill all required model properties. For this reason, the role of *system analyst* is introduced, who helps the domain expert with the modeling task. A system analyst has superior powers of abstraction, but minor detail knowledge.

The two actors engage in a *dialog* in order to create a formal model where both are satisfied with (see figure 2). We use the concept 'dialog' in the most general form, including any form of communication such as face-to-face communication, communication by e-mail, workshops, etc.

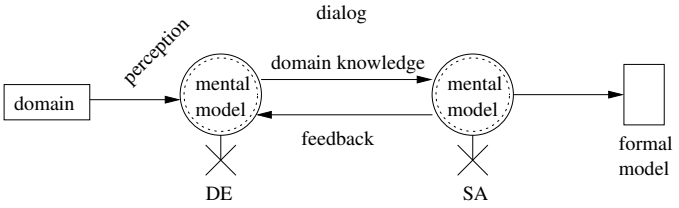

Figure 2: *Dialog between domain expert and system analyst.*

In the dialog, the domain expert is responsible for producing the domain knowledge. The system analyst analyzes the given knowledge, guarding consistency and completeness. Feedback is given about detected problems, such as inconsistencies and vague information. At the end of the dialog, both domain expert and system analyst should have a *shared mental model* of the domain. As the system analyst has ensured that the achieved domain knowledge satisfies the model requirements, he can now produce the formal representation in the desired modeling language.

1.3 Modeling as refining shared knowledge

Within the dialog, two basic processes can be recognized: *knowledge sharing* and *knowledge refinement*.

Before knowledge can be 'used', it needs to be *shared* between the two actors. This involves *introducing* knowledge, by communicating it to the other party, and *understanding* what is com-

municated. If some piece of communication is not understood, meta-communication may take place about what is said, until understanding is reached (Hoppenbrouwers, 2003).

Only after information is understood, it can be analyzed and *refined* when necessary. Refinement has many forms, such as

- assessing the validity of knowledge, compared to the current mental model
- assessing validity, by checking consistency
- increasing the precision of knowledge
- withdrawing invalid knowledge
- assessing the completeness of knowledge
- obtaining specific types of knowledge
- etc.

Eventually, *agreement* has to be reached about each piece of knowledge. Only when both actors agree on a piece of knowledge, it will be part of the resulting model. This is the stop condition for the dialog (see figure 3).

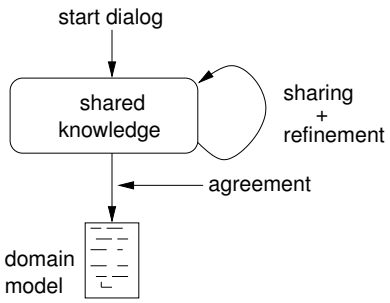

Figure 3: *Modeling as sharing and refining knowledge.*

As the two actors have different responsibilities, they also have different reasons to agree on information. The domain expert will agree when information is consistent with the 'real world'. On the other hand, the system analyst will agree on information that satisfies the desired model properties; this makes sure the system analyst can produce a formal description of the model.

2 Assistance for the system analyst

In this section we focus on the role of the system analyst. The tasks involved are elaborated on, and an existing method is described that assists the system analyst during modeling. The tasks and method to support this are used in the next sections, where computer support for the system analyst is introduced.

2.1 Communication based modeling

One way of viewing the system analysts' modeling task is known as *communication based modeling*. Methods based on this view include NIAM (Nijssen and Halpin, 1989) and ORM (Halpin, 2001).

In this view, the *communication* about the domain is analyzed in an explicit way and used to create the formal model. This assumes all domain knowledge is expressed in a common language as a set of sentences. Each sentence represents a *statement* about the domain; the collection of all

statements is the 'input' on which a formal model can be based. The resulting model is traceable and justified from the sentences.

The communication based modeling process is a specialization of the general modeling process using a dialog (see figure 4).

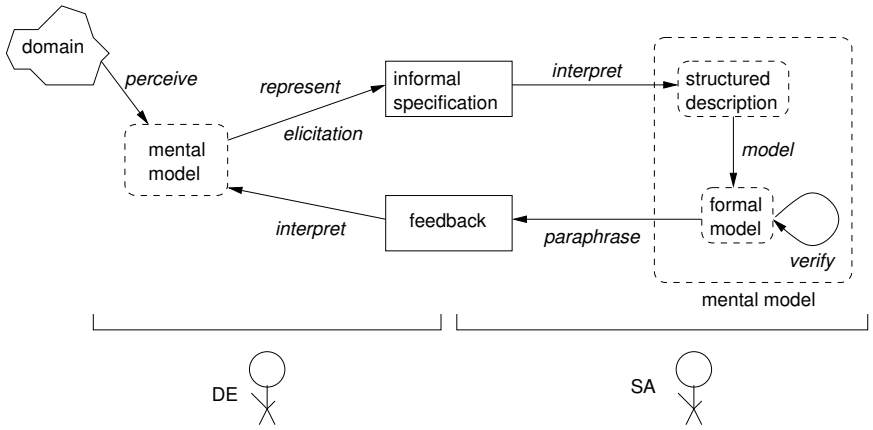

Figure 4: *Communication based modeling process.*

In the first step, domain knowledge is elicited from the domain expert. Although any format may initially be used to represent the knowledge (e.g., text, pictures, sketches), eventually the domain expert *verbalizes* the information resulting in the *informal specification*, written in a *common language*. This common language is constrained: it is concise, precise and its meaning is well defined such that subsequent analysis is not too complex.

The informal specification is then read, interpreted and 'understood' by the system analyst. This is done by grammatical analysis of the informal specification, resulting in a *structured description*. The structured description is a structure that captures the meaning of the informal specification in some way, i.e., by identifying concepts and their relations.

From the structured description, *abstraction* results in a formal model constructed with appropriate modeling concepts.

Analysis of the current model is split into *verification* and *validation*. Verification is an 'internal' check, most notably of internal consistency. Validation takes place by *paraphrasing* the constructed formal model, communicating it back to the domain expert. The domain expert, in turn, interprets the paraphrased model and compares it with the intended model. Problems with the paraphrased model will be reported by changing the informal representation.

We will use the communication based modeling view as starting point for our formalization of the modeling dialog.

2.2 Handling vagueness in the informal specification

Within the modeling dialog, the informal specification evolves from an incomplete and 'vague' domain description to a formal and precise specification of the domain knowledge.

Identifying and removing 'vagueness' in the informal specification is a major part of the refinement task of the system analyst. Two sources of vagueness can be identified (Regan et al., 2002):

Epistemic uncertainty This uncertainty exists in the mind of the expert, and reflects the incomplete knowledge a domain expert has of the domain. The uncertainty is a result of limited mental resources and limited time to investigate the domain (Anderson, 1990).

Linguistic uncertainty This is uncertainty introduced in communication, occurring when an expression in common language has more than one possible interpretation. For flexible common languages, such as natural language, this may occur frequently. Very constrained languages, on the other hand, may prevent the occurrence of multiple interpretation, at the expense of limited expressive power.

At a general level, terms like 'uncertainty' and 'vagueness' indicate that some information is missing, leading to improper or incomplete understanding. Looking in more detail, various forms of missing or invalid information can be distinguished. A typical example is Smithson's taxonomy of ignorance (Smithson, 1989) (figure 5, 6). Although this taxonomy can be argued to be an 'arbitrary' one, it clearly indicates the existence of different types of ignorance, each having its own properties.

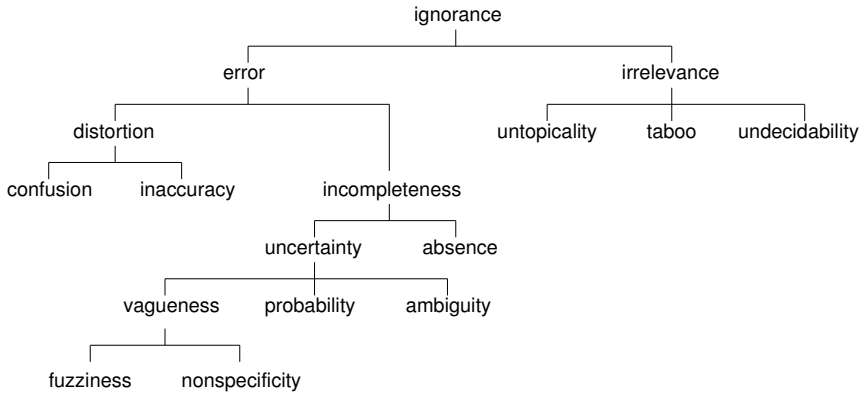

Figure 5: *Smithson's taxonomy of ignorance.*

Error	to be ignorant of
Distortion	qualitative impairment of knowledge; requires correction; misinformation, propaganda, lies
Confusion	distortion via wrongful substitution
Inaccuracy	distortion in degree
Incompleteness	quantitative insufficiency of knowledge
Uncertainty	information is present, but uncertain; reducing uncertainty is increasing likelihood
Absence	incompleteness in degree; when information isn't there
Vagueness	indescribability
Probability	risk; the most highly developed operationalization of uncertainty
Ambiguity	possibility of a double meaning; multiple alternative interpretations
Fuzziness	uncertainty of membership to sets
Nonspecificity	failure to indicate in sufficient detail to permit identification
Irrelevance	irrelevant information cannot be utilized
Untopicality	outside cognitive/intuitive domain; thought to be off-topic
Taboo	socially enforced ignorance
Undecidability	believed insoluble/not requiring verification; the 'so what?' category

Figure 6: *Description of Smithson's ignorance terms.*

When a system analyst encounters some form of uncertainty in a domain specification, he has

to handle it in some way. Although the precise method to handle uncertainty depends on the type of uncertainty, several general methods can be distinguished. (Lipshitz and Strauss, 1997) investigated how decision makers handled uncertainty; they found that four general ways to handle uncertainty were used:

Reduction of uncertainty Collect additional information, e.g. by asking.

Assumption based reasoning Fill gaps in knowledge by making plausible assumptions.

Weighing pros and cons of various alternatives.

Suppression ignore uncertainty, at least for a while.

3 A model of the modeling dialog

In this section we take an abstract view of the modeling dialog, formalizing the actions taken in this dialog. Using these dialog actions, we can formally describe dialog strategies, which in turn paves the way for computer support that helps the system analyst in modeling.

3.1 Symmetric exchange of knowledge

In the previous sections, the modeling process seemed to be an asymmetric 'flow' of knowledge from domain expert to system analyst. In contrast, we can also view the dialog as a *symmetric* exchange of knowledge:

- DE transfers expert knowledge to SA
- SA transfers analysis knowledge to DE

Both domain expert and system analyst are experts in their own field, contributing knowledge to the common model. This situation is pictured in figure 7.

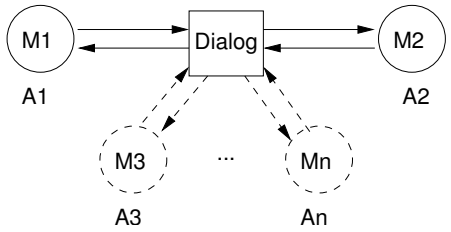

Figure 7: *Symmetric view of the modeling dialog, with two or more participating actors.*

The dialog consists of a set of *actors* A that participate in the dialog. Each actor has an internal model M. For human actors, this is the mental model.

3.2 Dialog actions

We assume the knowledge transfer between the actors in the modeling dialog is performed using the following *dialog actions*:

$\mathsf{Propose}(a, s)$	Actor a proposes statement s. It does not become part of the common model until every actor accepts it.
$\mathsf{Withdraw}(a, s)$	Actor a withdraws statement s. Withdraw is the opposite of a propose.
$\mathsf{Accept}(a, s)$	Actor a accepts statement s as a valid statement; it may eventually become part of its internal model M_a. A statement can only be accepted after it is proposed.
$\mathsf{Reject}(a, s)$	Actor a rejects statement s, because a finds s unacceptable even for further consideration. Reject is the counterpart of accept.
$\mathsf{Ask}(a, q)$	Actor a asks question q, to be answered by some actor. Queries can be withdrawn or answered.
$\mathsf{Answer}(a, q, s)$	Actor a answers question q with statement s; an answer functions as a special Propose.

We can now define a Dialog to be a sequence of dialog actions, reflecting communication between actors that is recorded. Initially, the Dialog starts empty. As the dialog progresses, all communication is captured in the Dialog by appending new dialog statements like in a transcript or logbook (Bommel et al., 1996). Without loss of generality, we assume statements have a textual form; the *principle of linguistic expressibility* states that any information can be expressed in a textual way (Adriaans, 1992).

3.3 Dialog state

As a dialog progresses, statements follow a particular life cycle after being proposed: they are accepted, rejected or even withdrawn by actors. To capture the current state of statements and questions in a dialog, we define a dialog state which we will use later in dialog strategies.

A *dialog state* DS is a structure that represents the state of the statements and questions in a dialog:

$$\mathsf{DS} = \langle \mathsf{A}, \mathsf{S}, \mathsf{Ac}, \mathsf{Q} \rangle$$

Here, A is the set of actors that participate in the dialog; S is the set of statements that is proposed and not withdrawn, i.e., the *active* statements; Q is the set of questions that have been asked but not yet answered. Ac is a total function which administrates the *acceptance state* for each combination of statement and actor:

$$\mathsf{Ac} : \mathsf{S} \times \mathsf{A} \rightarrow \{u, a, r\}$$

here, the acceptance state 'u' stands for undecided, 'a' for accepted and 'r' for rejected.

The dialog state DS can be derived from a Dialog in the following way:

$\mathsf{Propose}(x, s)$	$\mathsf{S} = \mathsf{S} \cup \{s\}; \mathsf{Ac} = \mathsf{Ac} \cup \{(s, y, u) \mid y \in \mathsf{A}\}$
$\mathsf{Withdraw}(x, s)$	$\mathsf{S} = \mathsf{S} \backslash \{s\}$
$\mathsf{Accept}(x, s)$	$\mathsf{Ac} = \mathsf{Ac} \cup \{(s, x, a)\}$
$\mathsf{Reject}(x, s)$	$\mathsf{Ac} = \mathsf{Ac} \backslash \{(s, x, r)\}$
$\mathsf{Ask}(x, q)$	$\mathsf{Q} = \mathsf{Q} \cup \{q\}$
$\mathsf{Answer}(x, q, s)$	$\mathsf{Q} = \mathsf{Q} \backslash \{q\}; \mathsf{Propose}(x, s)$

Statement states and transitions are shown in figure 8.

A statement which is proposed starts as a *proposed* statement, until actors have decided whether they will accept or reject it. If all actors have accepted the statement, it will change state from *proposed* to *accepted*. On the other hand, when at least one actor has rejected the statement, it will change state to *rejected*. Note that the dialog state does not record rejected statements.

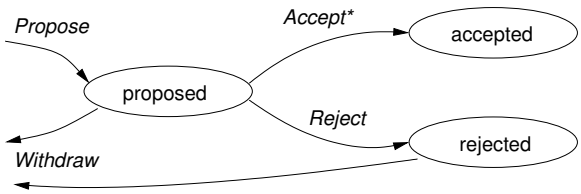

Figure 8: *State transitions for a single statement.*

The three states are formally defined by the following functions:

$$\mathsf{IsAccepted}(s) \triangleq \forall_{x \in \mathsf{A}}[\mathsf{Ac}(s,x) = a]$$
$$\mathsf{IsRejected}(s) \triangleq \exists_{x \in \mathsf{A}}[\mathsf{Ac}(s,x) = r]$$
$$\mathsf{IsProposed}(s) \triangleq s \in \mathsf{S} \wedge \neg\, \mathsf{IsAccepted}(s) \wedge \neg\, \mathsf{IsRejected}(s)$$

Let also the following sets be defined:

$$\mathsf{ASet} \triangleq \{s |\, \mathsf{IsAccepted}(s)\}$$
$$\mathsf{RSet} \triangleq \{s |\, \mathsf{IsRejected}(s)\}$$
$$\mathsf{PSet} \triangleq \{s |\, \mathsf{IsProposed}(s)\}$$

Note that all three sets contain sentences from a controlled language L_C: $\mathsf{ASet} \subseteq L_C, \mathsf{RSet} \subseteq L_C, \mathsf{PSet} \subseteq L_C$.

The modeling dialog is complete when:

1. No open questions exist: $\mathsf{Q} = \emptyset$
2. All statements are accepted by each actor: $\mathsf{ASet} = \mathsf{S}$
3. All actors are ready: Dialog will not grow anymore.

3.4 Obtaining a formal model

When the modeling dialog is completed, the set ASet essentially *is* the formal model: its statements are in a controlled format that is understood by all participants. Controlled language thus may be seen as an intermediate modeling technique. As it is also understandable for a system analyst, this controlled language has a sufficiently sound formal basis.

Let Model be a function that produces a model in a formal language from a set of statements in a controlled language format:

$$\mathsf{Model} : \wp\mathcal{O}(L_C) \rightarrow \wp\mathcal{O}(L_F) \cup \bot$$

The special outcome $\bot$ indicates *inconsistency*, meaning it is not possible to obtain a correct formal model from the input.

The Model function is *maximal* in the sense that it tries to make a formal model that incorporates all information available in the input set. If this is not possible, Model yields $\bot$.

Maximality can be expressed assuming the existence of a *paraphrase* function Ph, which paraphrases a formal model in terms of controlled language:

$$\mathsf{Ph} : \wp\mathcal{O}(L_F) \rightarrow \wp\mathcal{O}(L_C)$$

Now, maximality of Model can be expressed using the following axiom:

$$\forall_{R \subseteq L_C}[\mathsf{Model}(R) = \mathsf{Model}(\mathsf{Ph}(\mathsf{Model}(R)))]$$

4 Basic modeling dialog process

This section discusses the construction of a 'basic' modeling dialog process. This process is described in such detail that it enables computer support for the system analyst. In the basic process, uncertainty with respect to the interpretation of statements is not allowed. The next section will discuss an extension to the basic process, allowing a certain type of uncertainty and thereby increasing the flexibility for the domain expert.

4.1 Rules of communication

A dialog process is defined by the *rules of communication* that are agreed upon by all participating actors before the dialog starts. The rules determine all relevant aspects that have to be agreed on before a useful dialog can take place, such as:

- which dialog actions can be used in which situation
- what is the agreed common language, i.e., the form (syntax) in which statements may be expressed
- what are the assumptions on validity of statements
- what types of uncertainty are allowed
- etc.

The communication rules for the basic dialog process are based on (a strict interpretation of) the ORM method (Halpin, 2001). The process poses strict constraints on the communication between and tasks of domain expert and system analyst:

1. the controlled language has a strict format; there is no uncertainty about how to interpret statements in this controlled language
   (a) the syntax is well-defined
   (b) the *basic semantics* of sentences is well defined. This indicates the straightforward relation of syntactical elements to 'concepts' (rather than the 'real meaning' as only the domain expert can see).
   (c) *no distortion*: expressed statements are (required to be) valid
   (d) *no irrelevance*: statements are relevant for resulting model
   (e) *specific*: at all stages of the dialog, all information needed for a correct formal model is available.
   (f) *unambiguous*: only a single interpretation possible (results from well-defined syntax and basic semantics).
   (g) only *absence* is allowed during the dialog (missing statements); the validation feedback-cycle works towards completeness of the model itself.
   (h) no other incompleteness is allowed
2. the domain expert is responsible for
   (a) providing a complete and valid mental model of the domain
   (b) checking a verbalized model for validity and correctness
3. the system analyst is responsible for the mapping of a set of statements onto the underlying modeling concepts (i.e., performing the Model function)

As the task of the system analyst is limited to translation into a formal model, this basic dialog allows full computer assistance for the system analyst.

4.2 Dialog grammar

Which dialog actions can be taken in what situation can be formally described by a *dialog grammar*. A dialog grammar specifies a set of possible Dialogs, i.e., the set of possible sequences of actions in that dialog.

The grammar for base dialogs will be described using L-attributed grammars (Knuth, 1968; Bochmann, 1976). Such a grammar is a context-sensitive grammar, in which help-symbols may have parameters that 'carry' the context information.

The main dialog loop is described by:

```
dialog(DS) ->
  completed?(DS) |
  (de-action(DS) | sa-action(DS)) dialog(DS)
```

Here all symbols are help symbols; the current dialog state DS is provided to each symbol as parameter. The special symbol 'completed?' is a *conditional symbol* which only matches if the dialog is completed.

The domain expert only proposes statements, and accepts these directly as the DE 'knows' they are valid statements:

```
de-action(DS) ->
  propose(DS,de,S)
  accept(DS,de,S)
```

The system analyst simply accepts any statement that is proposed. This is possible because of the strict rules of communication that have been agreed on; according to these rules, the domain expert will only express statements which are acceptable to the system analyst.

```
sa-action(DS) ->
  element?(S,PSet(DS)) accept(DS,sa,S)
```

The basic dialog actions are formulated in the L-attributed grammar style as follows:

```
propose(DS,A,S) ->
  "propose(" A "," S ")"                          {DS.S += S; DS.Ac(S,A)=p}

accept(DS,A,S) ->
  "accept(" A "," S ")" element?(S,PSet(DS))   {DS.S -= s; DS.Ac(S,A)=a}
```

Here, symbols within double quotes are literals; *grammar actions*, written within curly braces, specify change in parameters.
Testing whether the dialog is completed is specified by introducing the special 'completed?' grammar production. The dialog is complete when all statements have been accepted or rejected, no open questions exist, and the domain expert believes the current specification is valid and complete:

```
ready(DS) ->
  equal?(ASet(DS),DS.S) empty?(Q) de-ready?
```

4.3 Responsibilities of system analyst

Because of the constraints on the dialog in this basic approach, the system analyst has no other responsibility than creating a formal model from the statements in ASet. This responsibility is easily formalized.

The modeling step can be seen as a straightforward statement-by-statement translation:

$$\mathsf{Model}(\mathsf{ASet}) = \bigcup_{s \in \mathsf{ASet}} \mathsf{Translate}(s)$$

where Translate is a function that translates a statement in the controlled language L_C to a set of statements in the modeling language L_F.

$$\mathsf{Translate} : L_C \rightarrow \wp(L_F)$$

4.4 Example

As example, we describe the setting where the constrained language is so-called NNF, and the resulting formal model is to be represented in ORM.

4.4.1 Common language: NNF

We define NNF as a controlled language, based on (Collignon and van der Weide, 1993). It has the following context-free grammar (also written in the L-attributed grammar style):

```
S -> E a E           # sentence construct
E -> S | SN          # entity specifier
SN -> ET p LT L      # standard name
```

Here, `a` = verb, `p` = pro noun, `ET` = (ORM) entity-type, `LT` = (ORM) label-type, `L` = (ORM) label. Text after the `#` symbol is comment.

Each statement expressed in NNF is specific enough for creating a ORM model, as entities are fully qualified (Halpin, 2001; Nijssen and Halpin, 1989). The basic semantics of NNF statements are:

- *Sentence construct* represents a proposition
- a represents an action
- *entity specifiers* (E) are actors participating in an action
- *standard name* represents an entity in the domain
- *ET* (entity type) is the type of an entity
- Every entity has exactly one type

Example 1 The following statements are valid NNF statements:

```
person with name John lives in city with name Nijmegen
person with name John has age of years 40
```

□

4.4.2 Formal language

As formal language, a subset of ORM (Halpin, 2001; Hofstede and Weide, 1993) is used. Although ORM is presented as a graphical modeling language, each ORM model can also be expressed as a set of statements from the following list:

Statement	Description
orm-entity-type(x)	x is an entity type
orm-label-type(x)	x is a label type
orm-relation-type(x)	x is a relation type
orm-role-type(r, n, t)	role n of relation r has type t.
orm-entity(x)	x is an entity
orm-label(x)	x is a label
orm-relation(x)	x is a relation
orm-role(r, n, e)	role n of relation r is entity e.
orm-type(x, y)	instance x has type y.

4.4.3 Example dialog

The following dialog is a valid example in the described basic setting:

```
DE> person with name John lives in city with name Nijmegen
  propose(de, person with name John lives in city with name Nijmegen)
  accept(de,  person with name John lives in city with name Nijmegen)
SA> accept
  accept(sa, person with name John lives in city with name Nijmegen)
```

From this dialog, the system analyst creates a formal ORM model, represented by the statements:

```
orm-entity-type(person)    orm-entity-type(city)
orm-label-type(name)       orm-label-type(name)
orm-entity(John)           orm-entity(Nijmegen)
orm-label(John)            orm-label(Nijmegen)
orm-type(John,name)        orm-type(Nijmegen,name)
orm-type(John,person)      orm-type(Nijmegen,city)

orm-relation-type(lives-in)
orm-role-type(lives-in, 1, person)
orm-role-type(lives-in, 2, city)
orm-relation(r1)
orm-type(r1, lives-in)
orm-role(r1, 1, John)
orm-role(r1, 2, Nijmegen)
```

5 Handling nonspecificity

In the basic dialog process, no uncertainty regarding the interpretation of statements is allowed of any form. This section discusses an extension of the basic dialog process, such that nonspecificity is allowed in statements.

5.1 Motivation

In the basic dialog process, the domain expert is responsible for expressing domain knowledge in a very strict format (such as NNF). As such a strict format is not natural for people to specify in, we would like to relax the constraints on the format, while preserving a formal dialog grammar that allows computer support for the system analyst.

A severe constraint is the *non-specificity* constraint: each statement has to provide enough information to derive a formal model. As seen with the NNF syntax, this makes expressing statements very non-natural and impractical for the domain expert.

Example 2 Compare the first specific sentence with the second, nonspecific sentence:

```
1) person with name John lives in city with name Nijmegen
2) John lives in Nijmegen
```

Clearly, the second statement is more natural and easier to specify. However, the types of the entities 'John' and 'Nijmegen' are not specified, making it impossible to incorporate statement 2 in a formal ORM model.
□

5.2 Detecting nonspecificity

Because of the given communication rules in the dialog, we can assume the only reason why $\mathsf{Model}(R)$ yields $\perp$ is when R is not specific. This leads to a definition of specificity:

Definition 5.1 *(Specific statement set)*
A set of statements R *is called specific in a dialog with state* DS *when*

$$\mathsf{IsSpecific}(\mathsf{DS}, \mathsf{R}) \triangleq \mathsf{Model}(\mathsf{ASet}(\mathsf{DS}) \cup \mathsf{R}) \neq \bot$$

□

Note that another definition is needed when rules of communication are weakened further, e.g., to allow inconsistencies. This would introduce other reasons for the Model function to return $\bot$.

The language L_{NNF} from the previous section is a special language, since any set of statements is specific:

Definition 5.2 *(Specific language)*
A language L is called specific *for a modeling function* Model *when*

$$\forall_{\mathsf{R} \subseteq L} [\mathsf{Model}(\mathsf{R}) \neq \bot]$$

□

5.3 Dialog grammar for handling nonspecific statements

As the domain expert is now permitted to express nonspecific statements, the system analyst needs extra skills to handle these statements.

1. *Detection* of non-specific statements
2. *Solving non-specificity by asking* domain expert for missing information
3. *Solving non-specificity by assuming* the missing information. This requires the sub skill of creating plausible information.

The new skills of the system analyst allow (and require) a more complex dialog. Specific statements are handled the same way as in the base dialog; however, non-specific statements need a special dialog strategy in order to be handled:

```
sa-action(DS) ->
  subset(R,PSet(DS)) (
    IsSpecific(DS,R) accept(DS,sa,R)  |
    solve-by-asking(DS,R)             |
    solve-by-assuming(DS,R)
  )

solve-by-asking(DS,S) ->
  ask(DS,sa,q)               {q = createQuestion(S)}

solve-by-assuming(DS,S) ->
  propose(DS,a)              {a = createAssumption(S)}
```

Note that system analysts now also must have the skill to formulate a question that, when answered, solves a nonspecificity. In addition, the system analyst must be able to create a plausible assumption on what information is missing from nonspecific statements.

The domain expert also has new responsibilities:

1. Answering questions posed by the system analyst, and
2. Judging assumptions made by the system analyst

These tasks are reflected in the new dialog grammar:

```
de-action(DS) ->
  propose(DS,de,s)  |
  element?(q,Q) answer(DS,de,q,s) |
  element?(PS,s) (accept(DS,s) | reject(DS,s))
```

5.4 Example

As example, we relax the format of NNF into a language RNNF in the following way:

```
S  -> E a E           # sentence construct
E  -> S | SN | N      # entity specifier
SN -> ET p LT L       # standard name
N  -> L               # name
```

The basic semantics of this language is equal to the semantics of NNF, except that entities need not be fully qualified anymore: an entity may be represented by either a standard name or a simple name (a label). In the latter case, no type information about the entity is specified.

Note again that other communication rules remain in place. In particular, non-ambiguity of the RNNF language requires names to be unique.

5.4.1 Example scenarios

The following examples show the various ways to deal with nonspecific statements.

Example 3 A scenario which demonstrates asking for missing information:

```
...
DE> (s1) John lives in city with name Nijmegen
SA> qualify John?
DE> (s2) John is person with name John
  Accept(SA, {s1, s2})
...
```

□

Example 4 A scenario which demonstrates assuming information that is missing:

```
...
DE> (s1) John lives in city with name Nijmegen
  assume(sa, "(s2) John is a thing")
  accept(sa, {s1, s2})
...
```

□

Example 5 A scenario which demonstrates ignoring missing information for a while:

```
...
DE> (s1) John lives in city with name Nijmegen
DE> (s2) person with name John works in city with name Groningen
  accept(sa, {s1, s2})
...
```

□

6 Conclusion

This paper introduced a formalization of modeling dialogs, with the goal to describe various types of dialog in a precise way, thereby paving the road for computer support during the dialog. Several issues in the formalization were discussed: the need for a formal common language, strict communication rules, formalized dialog actions and the need to understand various types of uncertainty that may occur during the dialog. Two types of dialog were specified using the formalization: a basic dialog and a dialog in which specified information may initially be non-specific.

References

Adriaans, P. (1992). *Language Learning from a Categorial Perspective*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands.

Anderson, J. R. (1990). *Cognitive psychology and its implications*. W.H. Freeman and Company.

Bochmann, G. (1976). Semantic evaluation from left to right. *Communications of the ACM*, 19(2):55–62.

Bommel, P. v., Frederiks, P., and Weide, T. v. d. (1996). Object-Oriented Modeling based on Logbooks. *The Computer Journal*, 39(9):793–799.

Bosman, S. and Weide, T. v. d. (2003). A case for incorporating vague representations in formal information modeling. In *Conferentie Informatiewetenschap 2003*, TU Eindhoven, The Netherlands.

Collignon, M. and van der Weide, T. (1993). An information analysis method based on PSM. In Nijssen, G., editor, *Proceedings of NIAM-ISDM*. University of Utrecht.

Falkenberg, E., Hesse, W., Lindgreen, P., Nilsson, B., Oei, J., Rolland, C., Stamper, R., Van Assche, F., Verrijn-Stuart, A., and Voss, K., editors (1998). *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO.

Frederiks, P. and Weide, T. v. d. (2004). 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. Technical Report Radboud University Nijmegen NIII-R0324.

Halpin, T. (2001). *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman. ISBN 1-55860-672-6.

Hofstede, A. t. and Weide, T. v. d. (1993). Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100.

Hoppenbrouwers, S. (2003). *Freezing Language; Conceptualisation Processes across ICT-Supported Organisations*. PhD thesis, University of Nijmegen.

Knuth, D. (1968). Semantics of context-free languages. *Mathematical Systems*, 2.

Lipshitz, R. and Strauss, O. (1997). Coping with uncertainty: a naturalistic decision-making analysis. *Organizational Behaviour and Human Decision Processes*, 2(69):152–154.

Nijssen, G. and Halpin, T. (1989). *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia.

Regan, H., Hope, B., and Ferson, S. (2002). Analysis and portrayal of uncertainty in a food web exposure model. *Human and Ecological Risk Assessment*, 8(7):1757–1777.

Smithson, M. (1989). *Ignorance and uncertainty: emerging paradigms*. Springer Verlag.