Validation of Object-Oriented Analysis Models using Informal Language

P.J.M. Frederiks, C.H.A. Koster, Th.P. van der Weide

Department of Information Systems, University of Nijmegen
Toernooiveld 1, NL-6525 ED Nijmegen, The Netherlands
{paulf,kees,tvdw}@cs.kun.nl

Published as: P.J.M. Frederiks, C.H.A. Koster, and Th.P. van der Weide. Validation of Object-Oriented Analysis Models using Informal Language. Technical Report CSI-R9609, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, May 1996.

Abstract

In this paper a conceptual model for object-oriented analysis is introduced. Three submodels are described which can be seen as milestones during the analysis phase. Each (sub)model has a corresponding paraphrasing mechanism which may be used (1) to provide a description of the structure of the model and (2) to generate sample instantiations. This paraphrasing mechanism is intended to enable the domain expert to validate the model by sentences in (semi-)natural language.

Keywords: object-oriented analysis, information grammar, paraphrasing, validation, PSM, Elisa-D, KISS, natural language

Classification: 68Q50 68Q55 *(AMS-1991)*, H.2.1 I.2.7 *(CR-1991)*

1 Introduction

During the late sixties and early seventies problem-oriented programming languages, like COBOL, ALGOL 68 ([WMP+76]) and PASCAL were introduced to solve the so-called *software crisis*. These languages provide abstraction mechanisms both on the algorithmic and data level, but initially lack abstraction mechanisms for modular programming.

In the eighties a number of fourth generation programming languages like SQL ([NF84]) were introduced. In combination with these languages, techniques were introduced for conceptual modeling, for example ER ([Che76]), IFO ([AH87]), NIAM ([NH89]), and PSM ([HW93]).

In the nineties a new phenomenon arose: the *object-oriented* approach, OO for short. An important argument that can be heard is that the basic philosophy of OO suits human problem solving methods better than conventional programming languages and methodologies. In a short time many new OO techniques and methods were developed, like OOA ([CY90]), OMT ([RBP+91]), Booch ([Boo91]), OOSE ([JCJO92]), and KISS ([Kri94]).

The work of this paper has started from the investigation of the fundamentals of the KISS-method. The reason for choosing this method is that it is based on natural language as a starting point for

analysis. We will, however, not adapt the terminology used is the KISS-method in order to have more freedom for extensions. We discuss how the *information grammar* ([NH89, FW96a]) can be seen as a basis for an informal language based approach[1]. In this paper we restrict ourselves to the description of a paraphrasing mechanism. This mechanism is useful to paraphrase a number of models such as introduced in the KISS-method. We also introduce the information architecture as a covering model for these models. However, the proposed architecture has also features which are not available in the KISS-method.

This paper is organized as follows. The analysis models are introduced in section 2. Section 3 shortly discusses the philosophy behind a informal language based way of working for object-oriented analysis. In this section also some related work is presented. The conclusions of this paper are presented in section 4.

2 Grammar Based Object-Oriented Modeling

After a short introduction to the analysis models in section 2.1, each analysis model will be discussed in the succeeding sections.

2.1 Introduction to the analysis models

As will be stated and detailed in section 3, the analysis starts from a set of sentences. During the modeling process these sentences are analyzed according to several grammatical perspectives, leading to a number of abstractions of the informal specification. Each abstraction provides a specific view on the nature of the application domain, and results in a corresponding model. The models together compose the conceptual model of the universe of discourse. We distinguish the following analysis models:

1. the *(object) action involvement model.* This model describes the types of objects and actions that have been determined from the informal specification, and states which *object types* are involved in what *action types*. For each action type it is indicated which object type is *responsible* for that action type. The external initiators of action types, the so-called *subject types*, may be considered as special object types. In this paper, however, they will not be integrated.

   Action types may come in several paraphrasing variants. This will lead to *generalizations* in the information architecture.

2. the *(object) property model.* This model deals with static aspects of the application domain, by modeling *properties* of object types. Usually the property is set during the *initialization* of the associated object type, and can be retrieved via *retrieval* action types. Strictly spoken, properties thus can also be modeled via retrieval action types. Properties are introduced to provide a more simple description mechanism.

3. the *(object) life model.* This model considers the application domain from a historical perspective, and provides for each object type the rules for the *course of life* of its instances.

   Each object type starts with its initialization action type. From the action involvement model it can be derived which other object types are involved in such action types. This leads to *life dependencies* between object types.

[1] The processing of natural language is not within the scope of this paper. In order to stress this, the terms natural language and informal language are used intertwined.

4. the *information architecture*. This model extends the three models mentioned above, and provides a complete description of the dynamics and statics of the intended system. Furthermore, the information architecture describes the *conceptual (object) type hierarchy*. This hierarchy introduces a classification based upon action types. For example, common asynchronous action types will lead to a *generalization*. Generalization thus is based on common potential behavior. *Specialization*, on the other hand, is based upon actual behavior of object instances.

   Each paraphrasing variant of an action type from the action involvement model returns in the information architecture as a relation type between the involved object types. Note that these relation types (also referred to as fact types) will be time-stamped in order to be able to reproduce the history of the universe of discourse as a logbook.

   Properties from the property model are represented as *attributes*. These attributes will be used for denotation of object types (and not for identification, as in classical data models). Finally, the object life model provides clues for the dynamic constraints of the information architecture.

Each analysis model represents a (partial) linguistic view on the information grammar. For feedback purposes (validation), these models also have an associated paraphrasing mechanism, which enables the system analyst to describe the model and its instantiations in a terminology understandable for informed users. This paraphrasing mechanism is both a syntactical and semantical abstraction of the informal specification. This mechanism effectively supports the reliability of the validation process. The validation process usually will lead to reconsiderations, extensions and reformulations of the informal specification.

The next sections discuss the four analysis models in more detail. All four models are explained in view of the sample informal specification as presented in appendix A. Note that this informal specification is already de-instantiated, i.e. on an abstract level.

2.2 Object action involvement model

The first step of the grammatical analysis processes the informal specification very roughly. Sentences are broken down into the following components: subject, predicate, direct object and preposition group. The predicates specify the type of action associated with the sentence. In actively phrased sentences, the subject of this sentence can be seen as the originator of the action. Each preposition group describes an object that is involved in the action. Special is that different types of sentences may have the same predicate. This step is comparable to what the method from [CY90] offers, and is in line with the construction of the *Initial Interaction Model* from the KISS method.

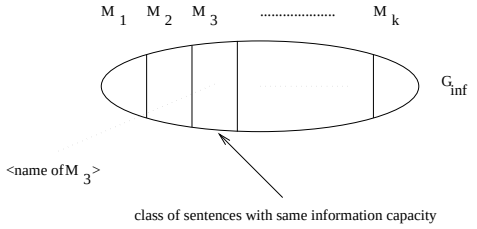

Figure 1: An abstract view on the information grammar

The sentences of the informal specification can be grouped together according to their *information capacity* (figure 1). Two sentences are said to have the same information capacity if they have the same deep structure. The class of sentences with the same information capacity is labeled with

the predicate of the associated sentence. Each such class corresponds to an action type, and has associated a grammar rule. The information grammar consists of all such grammar rules.

This syntactical analysis leads to the recognition of a set of object types, a set of action types, and responsibilities of object types.

Example 2.1

Consider the sentences:

the band 'Rolling Stones' records the song 'Paint it black'
the band 'Aerosmith' records the song 'Walk this way'

These sentences lead to the recognition of (1) the object types *Band* and *Song*, (2) the action type *to record* and (3) the association:

Band records *Song*

This results in the following action involvement model:

(name of) action type	(name of) object type			
	Band	Person	Recording	Song
to produce	r		i	
to produce		r	i	
to record	r			i

In this table, r codes responsibility, while i stands for other sorts of involvement. □

The simple view to sentences of the informal specification can be described by a context-free grammar.

Example 2.2

The set of production rules of the sample sentences, capturing the structure of the example UoD, is:

$$\begin{aligned}
\langle \text{to produce} \rangle &\longrightarrow \langle \text{Band} \rangle \text{ produces } \langle \text{Recording} \rangle \\
&\mid \langle \text{Person} \rangle \text{ produces } \langle \text{Recording} \rangle \\
\langle \text{to record} \rangle &\longrightarrow \langle \text{Band} \rangle \text{ records } \langle \text{Song} \rangle \\
\langle \text{Band} \rangle &\longrightarrow \langle \mathsf{ObjectId} \rangle \\
\langle \text{Song} \rangle &\longrightarrow \langle \mathsf{ObjectId} \rangle \\
\langle \text{Recording} \rangle &\longrightarrow \langle \text{to record} \rangle \\
\langle \text{Person} \rangle &\longrightarrow \langle \mathsf{ObjectId} \rangle
\end{aligned}$$

The set OIDs will be the collection of all object identifiers, and thus is the target for syntactical class ⟨ObjectId⟩. □

The start symbol of the information grammar is in general (see figure 1) the following rule:

$$\langle \text{InfoGrammarStart} \rangle \longrightarrow \langle \text{name of } M_1 \rangle \cdots \langle \text{name of } M_k \rangle$$

The grammar rule, associated with the start symbol in our sample grammar, has as right-hand side ⟨to produce⟩ ⟨to record⟩. Each action type has associated a unique rule in the grammar reflecting the structure of the corresponding sentence type. For each object type X there is a corresponding nonterminal symbol ⟨name of X⟩. Action types that are also used as object types, have associated a grammar rule rewriting it conforming to the rules of the corresponding action

type. In our example, action type named *to record* is an object type under the name *Recording*, leading to the rule:

$$\langle \text{Recording} \rangle \longrightarrow \langle \text{to record} \rangle$$

Object types that are not actified are concretized by the rule:

$$\langle \text{name of } X \rangle \longrightarrow \langle \mathsf{ObjectId} \rangle$$

Usually, domain experts use more than one style to express the same fact. For instance, a fact can be expressed by either an active or a passive sentence. This is useful to avoid bad constructions. An example of such a bad construction is the following sentence:

$$\langle \text{Band} \rangle \text{ produces } \langle \text{Band} \rangle \text{ records } \langle \text{Song} \rangle$$

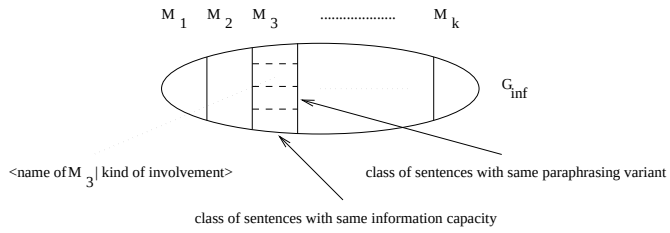

Figure 2: Paraphrasing variants of the information grammar

This can be solved by extending the grammar with extra paraphrasing variants for action types. These variants may already be recognized from the sample sentences by allowing classes of sentences with the same deep structure (see figure 1) to have more surface structures. This is reflected in figure 2. These variants are recognized in the grammar by adding a dedicated variant nonterminal for each paraphrasing variant. For example, action type ⟨to record⟩ will have three variants. The rewrite rule for this action type thus becomes:

$$\langle \text{to record} \rangle \longrightarrow \langle \text{to record} \mid r \rangle \mid \langle \text{to record} \mid i \rangle \mid \langle \text{to record} \mid \textit{objectified} \rangle$$

For the variants we have:

$$\begin{aligned} \langle \text{to record} \mid r \rangle &\longrightarrow \langle \text{Band} \rangle \text{ records } \langle \text{Song} \rangle \\ \langle \text{to record} \mid i \rangle &\longrightarrow \langle \text{Song} \rangle \text{ is recorded by } \langle \text{Band} \rangle \\ \langle \text{to record} \mid \textit{objectified} \rangle &\longrightarrow \text{the recording of } \langle \text{Song} \rangle \text{ recorded by } \langle \text{Band} \rangle \end{aligned}$$

The better formulation now is enforced by the following rule to paraphrase recordings:

$$\langle \text{Recording} \rangle \longrightarrow \langle \text{to record} \mid \textit{objectified} \rangle$$

In this new grammar, the above construct will be paraphrased as:

$$\langle \text{Band} \rangle \text{ produces the recording of } \langle \text{Song} \rangle \text{ recorded by } \langle \text{Band} \rangle$$

2.3 Object property model

Reconsider the final grammar presented in section 2.2. Obviously object types have properties, for example, a band has a name (*B-name*). Properties bridge the gap between *concrete* object types, also referred to as *label types*, and *abstract* objects types. A property may be retrieved via a retrieval action. In a retrieval action type label types and abstract object types are involved. This leads to the following extension of the object action involvement model as presented in example 2.1:

(name of) action type	(name of) object type				(name of) label type					
	Band	Person	Recording	Song	B-name	P-name	B-date	R-date	Tape-nr	S-name
to produce	r		i							
to produce		r	i							
to record	r			i						
to have	r				i					
to have		r				i				
to have		r					i			
to have			r					i		
to have			r						i	
to have				r						i

Note that properties can be recognized in sentences with predicates such as *to have* and *to be*. Typical in such sentences is that they have only one (abstract) object type involved. Properties may also have other appearances in sentences, for example as an adjective. A complete treatment of the recognition of properties from the informal specification falls outside the scope of this paper. For more information see [Gra94] or [Kri94].

The extended object action involvement model may be very large. The property model gives a more convenient description of this extension, by relating object types to properties. This can be represented in tabular form. In our example we have:

object type	*property*
Band	B-name
Person	P-name
Person	B-date
Recording	R-date
Recording	Tape-nr
Song	S-name

Paraphrasing the object property model is founded on the following grammar extension:

$$
\begin{aligned}
\langle \text{to have} \rangle &\longrightarrow \langle \text{Band} \rangle \text{ has band name } \langle \text{B-name} \rangle . \\
&\mid \langle \text{Person} \rangle \text{ has person name } \langle \text{P-name} \rangle . \\
&\mid \langle \text{Person} \rangle \text{ has birth date } \langle \text{B-date} \rangle . \\
&\mid \langle \text{Recording} \rangle \text{ has recording date } \langle \text{R-date} \rangle . \\
&\mid \langle \text{Recording} \rangle \text{ has tape number } \langle \text{Tape-nr} \rangle . \\
&\mid \langle \text{Song} \rangle \text{ has song name } \langle \text{S-name} \rangle . \\
\langle \text{B-name} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle \\
\langle \text{P-name} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle \\
\langle \text{B-date} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle \\
\langle \text{R-date} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle \\
\langle \text{S-name} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle \\
\langle \text{Tape-nr} \rangle &\longrightarrow \langle \mathsf{LabelId} \rangle
\end{aligned}
$$

where LabelId corresponds with a concrete label. The population of the problem domain can now be achieved collecting all currently available label types.

The start rules of the extended grammar are:

$$\langle \text{InfoGrammarStart} \rangle \longrightarrow \langle \text{to have} \rangle \ \langle \text{to produce} \rangle \ \langle \text{to record} \rangle$$

2.4 Object life model

The object life model has its roots in the *KISS-models* of the KISS method ([Kri94]). However, the KISS models lack a formal paraphrasing mechanism. An advantage of having this paraphrasing mechanism is that a reasonably well readable summary of the course of life of each object type can be automatically generated (see [DFW96]).

2.4.1 Object histories

In the object life models we consider order relations for action types from the action involvement model. This order can be derived from relations between sample sentences. For example, the following fragment of the informal description:

> *After a band has been set up, persons may join this band.*

During the previous step, this sentence has been split into the following:

> *Person sets up band.*
> *Person joins band.*

The original sentence adds that the action of joining a band can only occur after the action of setting up a band. Considering the course of life of a band, we thus have a precedence relation for two of its actions.

The object life model describes for each object type all sequences in which actions can be invoked subsequently. Such a sequence is referred to as a *(process) trace* or *history* and describes the behavior of individual objects of that type. The object life model can also be seen as a dynamic state space for each object type, as it presents a framework for the life (the sequence of actions) of each object within the system under consideration.

Process algebra ([BW90]) may be used to provide a short description of sets of traces. In order to communicate about expressions of process algebra, a visualization mechanism for object life models is introduced. An object life model is a special graph with a node for each object type, and nodes for *action type roles* and *structure roles*. For structure roles the following symbols are used: *repetition role*, *choice role*, *parallel role* and *repetition-choice role*. Nodes for action type roles and structure roles are also referred to as *components* (or *general roles*). The general roles of the object life model are connected by three types of arrows: *role sequencing arrow*, *role decomposition arrow* and *object type initialization arrow*. The structure roles and the *role-sequencing arrow* form the *control structures* for the object life models. The associated symbols are shown in appendix C. Note that the symbol repetition-choice role is short for a repetition role symbol together with a choice role symbol.

Each object starts its course of life at a particular point in time, by a particular action (called its initialization). At this point some properties of the object are set. For example, the name of a band is assigned during its birth. In our example, the life of object type *Band* is as follows:

> *The life of an object of type Band starts whenever it is set up. A band can record, produce, be joined by or left by a person zero or more times. Finally, the life of a band is (explicitly) ended whenever it is disbanded.*

From this description order relations for action types involving a band, can be derived. Furthermore, at any point in the life of an object a retrieval action may occur. In the above example there is a single retrieval action *to have*. Figure 3 shows the life cycle of object type *Band*.

An object life model may become rather complex and hard to understand if all retrieval actions are included. As retrieval actions refer to properties of the object type, i.e. *static* aspects, these actions do not affect the course of life of an object type. Therefore, retrieval action types will be omitted in the sequel. The simplified object life model of object type *Band* is shown in figure 4. The course of life of the other object types of our sample informal specification are shown in appendix D.

Paraphrasing of the object life model is not straightforward as it requires not a single sentence. In the rest of this section we give some rules of the thumb. Paraphrasing of object life models is

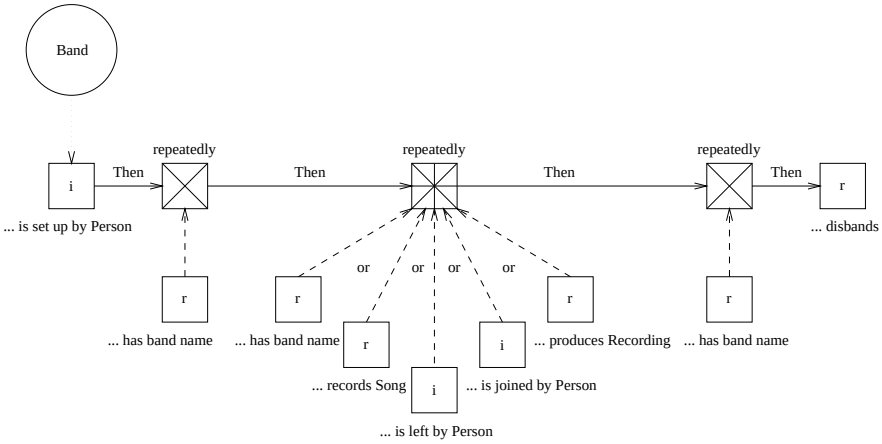

Figure 3: An example of an object life model including retrieval actions

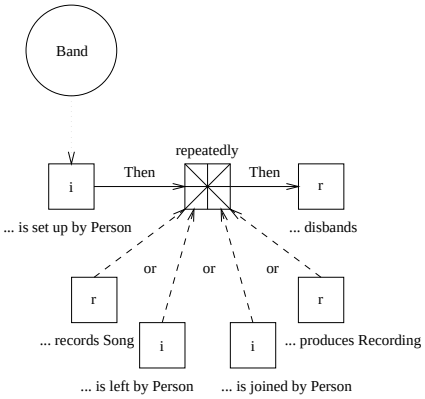

Figure 4: An example of an object life model

still a topic for research. The paraphrasing of an object life model results in a description of all behavior rules, grouped by object type. The paraphrasing of the course of life of an object type is governed by a number of translation rules. The life of an object type starts with its initialization, followed by a sequence of components. Suppose object type X starts its life with component c. This leads to the following grammar rule:

$$\langle \text{life of } X \rangle \quad \longrightarrow \quad \langle \text{seq } c \rangle$$

A sequence of components is elaborated as follows:

$$\langle \text{seq } c \rangle \quad \longrightarrow \quad \begin{cases} \langle c \rangle \text{ . Then } \langle \text{seq } d \rangle & \text{if } d \text{ follows } c \\ \langle c \rangle \text{ .} & \text{otherwise} \end{cases}$$

For each component c that corresponds to an action type we add the rule:

$$\langle c \rangle \longrightarrow \langle \text{name of } c \rangle$$

The rule for control structure b is found in the following table:

component	**Decomposition**	**Grammar rule**
repetition	c	$\langle b \rangle \longrightarrow$ repeatedly $\langle$frag $c\rangle$
choice	$c_1, \ldots, c_k$	$\langle b \rangle \longrightarrow \langle$frag $c_1\rangle$ or . . . or $\langle$frag $c_k\rangle$
parallelism	$c_1, \ldots, c_k$	$\langle b \rangle \longrightarrow$ at the same time $\langle$frag $c_1\rangle$ and . . . and $\langle$frag $c_k\rangle$

The nonterminal $\langle$frag $c\rangle$ leads to the description of the behavior starting from components c. If the component c is not followed by another component, then no special care is required. Otherwise, the description leads to a text fragment within some other sentence. Therefore, this is marked explicitly with brackets:

$$\langle \text{frag } c \rangle \quad \longrightarrow \quad \begin{cases} [\ \langle \text{seq } c \rangle\ ] & \text{if } c \text{ has a successor} \\ \langle c \rangle & \text{otherwise} \end{cases}$$

Applying these rules to course of life of a *Band*, we get:

$$\langle \text{life of Band} \rangle \quad \longrightarrow \quad \langle \text{seq to set up} \rangle$$

The sequence of actions that starts from the component *to set up* is embedded in the following rules:

$$\begin{aligned} \langle \text{seq to set up} \rangle \quad &\longrightarrow \quad \langle \text{to set up} \rangle \text{ . Then } \langle \text{seq } c \rangle \\ \langle \text{seq } c \rangle \quad &\longrightarrow \quad \langle c \rangle \text{ . Then } \langle \text{seq to disband} \rangle \\ \langle \text{seq to disband} \rangle \quad &\longrightarrow \quad \langle \text{to disband} \rangle \text{ .} \end{aligned}$$

where c corresponds to the repetition-choice symbol. The substructure for component c is described by:

$$\langle c \rangle \quad \longrightarrow \quad \text{repeatedly } \langle \text{frag to record} \rangle \text{ or } \langle \text{frag to join} \rangle \text{ or } \langle \text{frag to leave} \rangle \text{ or } \langle \text{frag to produce} \rangle$$

In this case, all sub-fragments are simple:

$$\begin{aligned} \langle \text{frag to record} \rangle \quad &\longrightarrow \quad \langle \text{to record} \rangle \\ \langle \text{frag to join} \rangle \quad &\longrightarrow \quad \langle \text{to join} \rangle \\ \langle \text{frag to leave} \rangle \quad &\longrightarrow \quad \langle \text{to leave} \rangle \\ \langle \text{frag to produce} \rangle \quad &\longrightarrow \quad \langle \text{to produce} \rangle \end{aligned}$$

Paraphrasing variants of action types can be used to obtain better readable sentences. In this case, a more convenient paraphrasing would be:

$$\langle \text{frag to record} \rangle \quad \longrightarrow \quad \langle \text{to record} \mid r \rangle$$

A paraphrasing of the object life model of object type *Band* of figure 4 would be:

⟨Band⟩ is set up by ⟨Person⟩. Then repeatedly ⟨Band⟩ records ⟨Song⟩ or ⟨Band⟩ is joined by ⟨Person⟩ or ⟨Band⟩ is left by ⟨Person⟩ or ⟨Band⟩ produces ⟨Song⟩. Then ⟨Band⟩ disbands.

In this paraphrasing variant the verbs *to set up*, *to join* and *to leave* are used in a passive form as the corresponding action types are *not* the responsibility of object type *Band* but of the responsibility of object type *Person*. This information is available in the object action involvement model.

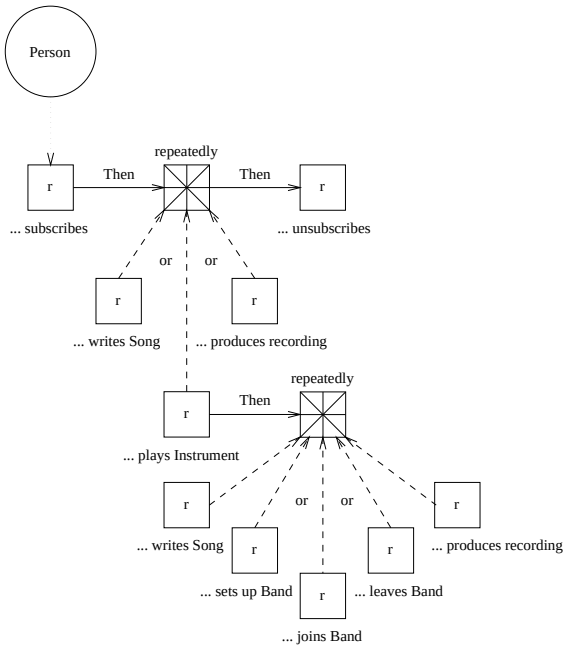

Figure 5: The course of life of object type Person

There are lots of possibilities to improve the readability of the generated text. As noted before, this topic needs more research. For example, nested decompositions will lead to complex sentences. Consider the course of life of a person, as described in 5. Paraphrasing leads to the following text:

⟨Person⟩ subscribes. Then repeatedly ⟨Person⟩ writes ⟨Song⟩ or [⟨seq to play⟩] or ⟨Person⟩ produces ⟨Recording⟩. Then ⟨Person⟩ unsubscribes.

which subsequently leads to:

⟨Person⟩ subscribes. Then repeatedly ⟨Person⟩ writes ⟨Song⟩ or [⟨Person⟩ plays ⟨Instrument⟩. Then repeatedly ⟨Person⟩ writes ⟨Song⟩ or ⟨Person⟩ sets up ⟨Band⟩ or ⟨Person⟩ joins ⟨Band⟩ or ⟨Person⟩ leaves ⟨Band⟩ or ⟨Person⟩ produces ⟨Recording⟩.] or ⟨Person⟩ produces ⟨Recording⟩. Then ⟨Person⟩ unsubscribes.

Note that this description suggests the introduction of a name (meaningful in the application domain!) for the behavior addressed by ⟨seq to play⟩. In this case, a suitable name would be ⟨Person acts as Musician⟩, leading to:

⟨Person⟩ subscribes. Then repeatedly ⟨Person⟩ writes ⟨Song⟩ or ⟨Person acts as Musician⟩ or ⟨Person⟩ produces ⟨Recording⟩. Then ⟨Person⟩ unsubscribes.

⟨Person acts as Musician⟩ $\longrightarrow$ [⟨Person⟩ plays ⟨Instrument⟩. Then repeatedly ⟨Person⟩ writes ⟨Song⟩ or ⟨Person⟩ sets up ⟨Band⟩ or ⟨Person⟩ leaves ⟨Band⟩ or ⟨Person⟩ joins ⟨Band⟩ or ⟨Person⟩ produces ⟨Recording⟩.]

Note that this corresponds to the introduction of a subtype *Musician* of object type *Person*.

Including retrieval action types in the paraphrasing of the object life model leads to fuzzy sentences. For example:

> ⟨Band⟩ is set up by ⟨Person⟩. Then repeatedly ⟨Band⟩ has band name ⟨B-name⟩. ...

is a part of the paraphrasing of figure 3. Clearly, these sentences are not desirable. Since retrieval action types are already paraphrased in the property model, these action types are left out in the paraphrasing of the object life model.

2.4.2 Further restrictions on object histories

The object life model provides a framework for histories of instances of each object type. However, this demarcation may be too wide. Further restrictions, also called *history constraints* or *dynamic constraints*, can be described in terms of the properties of objects. Such restrictions may involve objects of several types. Restrictions on histories can be defined with so-called *attribute functions*.

Attribute functions assign values to histories. These values can be taken from a concrete domain, or from the domain for abstract objects (OIDs). The value $\perp$ (undefined) is used to make attribute functions total functions.

Example 2.3

Reconsider the course of life of a *Band* (see figure 4). The following histories for *Band* instance b and *Person* instances p and q obviously are not valid:

to set up(b,p)*; to leave*(b,q)
to set up(b,p)*; to join*(b,p)*; to join*(b,p)

The problem with these histories is that a person can only leave a band after joining this band. This will be referred to as BM1. Furthermore, only new members, or previously left members, can join a band. This will be referred to as BM2. In order to express these constraints, we introduce the attribute function

$$\mathsf{BandMembers} : \text{histories of } b \rightarrow 2^{\text{Person}}$$

as follows:

$$\begin{aligned}
\mathsf{BandMembers}(\varepsilon) &= \varnothing \\
\mathsf{BandMembers}(\textit{to set up}(b,p)) &= \{p\} \\
\mathsf{BandMembers}(h; \textit{to join}(b,p)) &= \begin{cases} \perp & \text{if } p \in \mathsf{BandMembers}(h) \\ \mathsf{BandMembers}(h) \cup \{p\} & \text{otherwise} \end{cases} \\
\mathsf{BandMembers}(h; \textit{to leave}(b,p)) &= \begin{cases} \perp & \text{if } p \notin \mathsf{BandMembers}(h) \\ \mathsf{BandMembers}(h) - \{p\} & \text{otherwise} \end{cases} \\
\mathsf{BandMembers}(h; \textit{to disband}(b)) &= \varnothing
\end{aligned}$$

where ε denotes the empty history, and h stands for the history of *Band* b. This leads to the following formulation for both BM1 and BM2:

$$\mathsf{BandMembers}(h) \neq \perp$$

□

The paraphrasing of such constraints requires a language to formulate constraints. In the next section the information architecture is introduced, together with the language Elisa-D ([PW95]). This language will be used to paraphrase constraints.

2.5 Information architecture

The information architecture describes the complete conceptual structure of the intended information system. This model is based on the previously discussed analysis models and it describes a conceptual object type hierarchy.

2.5.1 The conceptual modeling technique

For the denotation of information architectures we use an extension of the conceptual data modeling technique PSM ([HW93]). This extension is called PSM^2. We have chosen PSM as a basis as this technique has a formal definition of both its syntax and semantics. Furthermore, PSM is graphically-oriented.

Besides the well-known notions of *object type* and *fact type*, PSM also contains more complex constructs like:

- *set types.*
  Set types (in PSM called *power types*) form the data modeling pendant of power sets in conventional set theory.
- *sequence types.*
  Sequence types are comparable with power types with as difference that the ordering is important and that elements may occur more than once.
- *schema types.*
  Schema types allow for schema decomposition. They can be used to modularize the information architecture during the design phase.

New object types can also be introduced by *generalization* and *specialization*. Generalization may also have a recursive nature. Besides the traditional integrity constraints (*total role* and *uniqueness*), the PSM modeling technique provides some extra graphical constraints like the *occurrence frequency constraint*, the *subset constraint*, the *equality constraint* and the *exclusion constraint*. Not all constraints can be expressed graphically in a PSM schema. For such constraints the query and constraint language Lisa-D ([HPW93]), or, if time is involved, Elisa-D ([PW95]) will be used.

The semantics of a PSM schema is the set of all its possible populations. In order to model dynamics the standard PSM semantics are extended with the notion of time. In [Pro94] the extension of populations of PSM schemata with time stamps is treated and formalized.

2.5.2 Construction of the information architecture

The intention of the information architecture is to provide a general framework for the intended information system, that is capable to represent each system trace. The main structure of this architecture is captured by the object action involvement model. Each object type of this model is also an object type in the information architecture. Action types are mapped onto fact types. If an object type is involved in an action type then this object type plays a role in the corresponding fact type. Each involvement of an object type in an action type thus is represented as a *role* (*predicator*) in the associated fact type. The responsibilities and other sorts of involvement are labeled in the PSM schema adding an r and i to the roles respectively. The PSM schema of the object action involvement model of the music example is depicted in figure 6.

Usually, data modeling techniques make a clear distinction between abstract and concrete objects. In NIAM ([NH89]) this gap is bridged by so-called *bridge types*, i.e. special fact types that relate an abstract object type to a concrete object type (also referred to as *label type*). The object

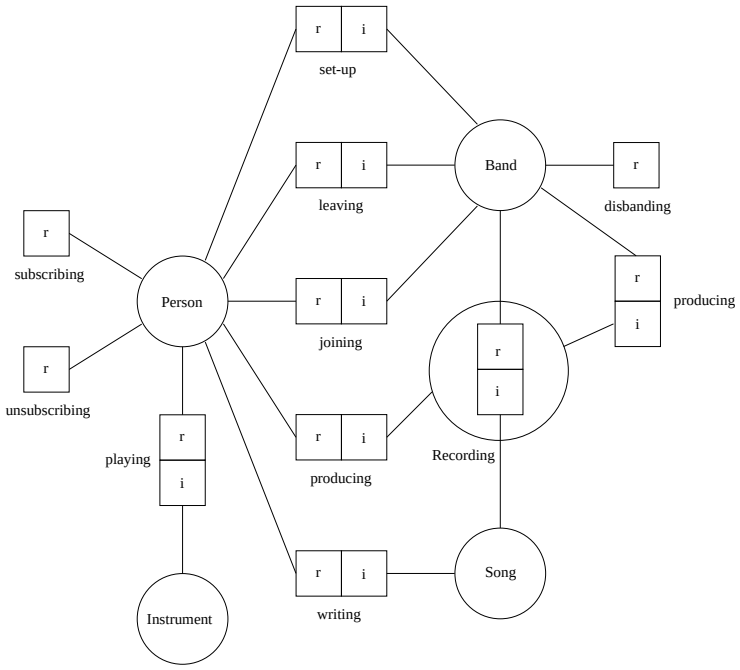

Figure 6: A PSM schema of the Object action involvement model

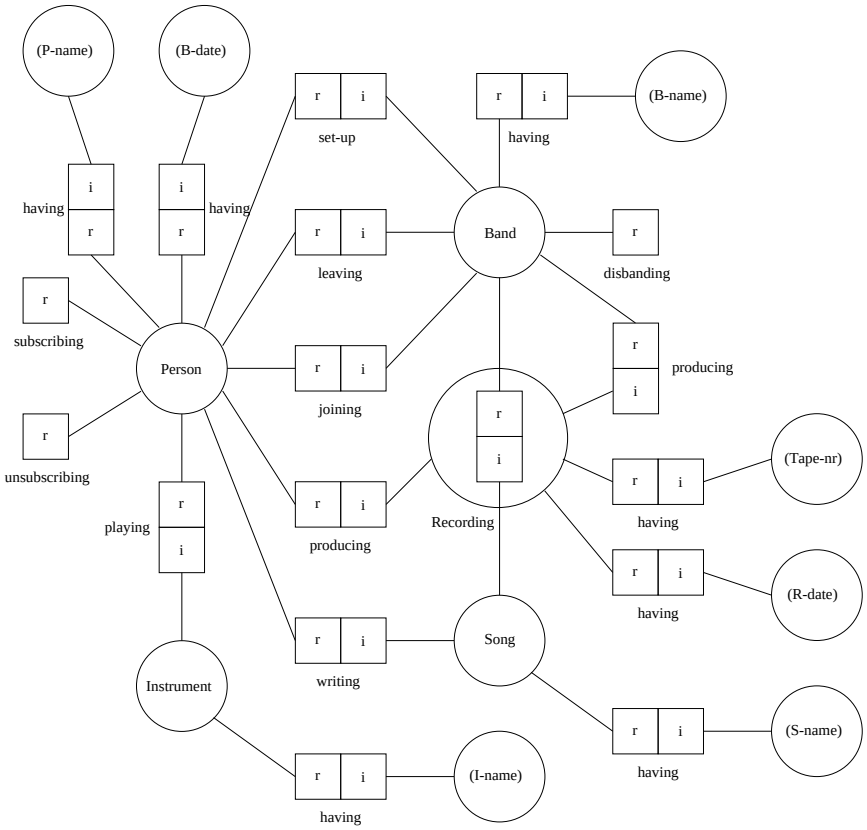

Figure 7: The Object action involvement model extended with properties

property model is reflected in the information architecture by the label types and their bridges. These label types are not used for *identification*, as required in PSM, but for *denotation*. As the object-oriented paradigm states that each object has a unique identification, the availability of an implicit label type is assumed, which has associated the domain for abstract objects (OIDs). From each object type, there is an implicit bridge to this implicit label type, assigning each instance a unique object identifier. Figure 7 shows the object action involvement model extended with the object property model.

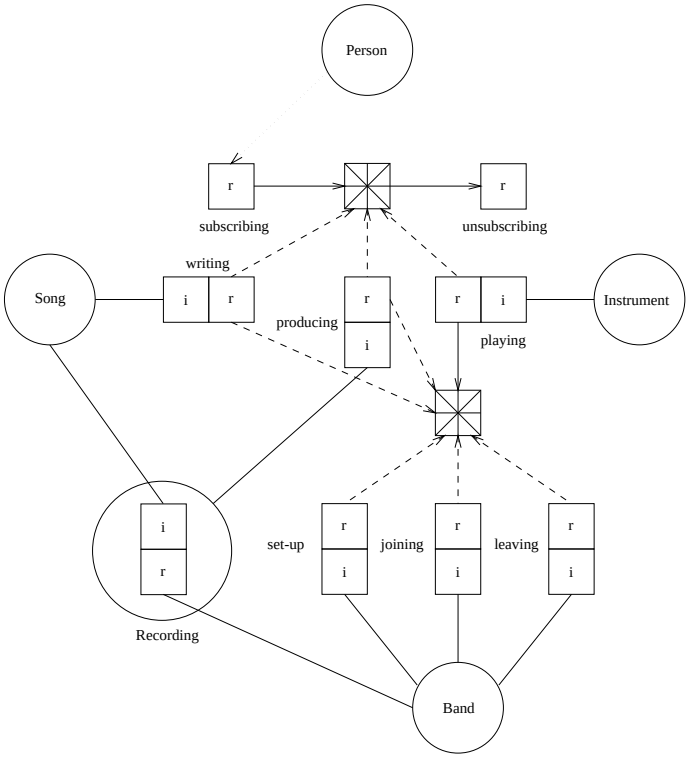

Figure 8: Incorporation of object life model into part of information architecture

The incorporation of the object life model in the information architecture is based on the observation that the action type roles of the object life model correspond to predicators in the information architecture. The structure roles of this model are introduced as so-called *structural predicators*. The term *generalized predicator* is used as a generic term for both predicator and structural predicator. The arrows in the object life model are translated as follows:

1. the object type initialization arrow is integrated as a relation between object types and generalized predicators.
2. the role sequencing arrow is modeled as a successor relation for generalized predicators.
3. the role decomposition arrow decomposes a structural predicator into its associated generalized predicators.

As an example of this incorporation, see figure 8. We have only extended the PSM schema with the course of life of object type *Person* as the addition of object life model to the PSM schema becomes rather complex. In this figure, the lines that connect a predicator to its base (in this case *Person*) are omitted for convenience. Furthermore, only fact types which include an involvement of object type *Person* are depicted.

Finally, the information architecture is extended with the history constraints. The language Elisa-D, which has a multiset based semantics, can be used to denote these constraints in a semi-natural language format. This is demonstrated by the history constraints for bands from example 2.3. These constraints can be described by first introducing the following special sentence:

```
LET is_member_of BE
  ALL Person joins Band EVER MINUS ALL Person leaves Band EVER
```

The construction `LET ... BE ...` is a mechanism to extend the information grammar with new rules for forming sentences. The construction `ALL P EVER` gathers all instances of information descriptor `P` from the past. The constraints now are expressed as:

```
CONSTRAINT
  BM1: NOT Person leaves Band BUT NOT is_member-of THAT Band
  BM2: NOT Person joins Band AND ALSO is_member-of THAT Band
```

The resulting information architecture is correct but rather complex. By using the full power of PSM and using the hidden abstractions of the information grammar the information architecture can be more structured. For example, a closer inspection of the sample information grammar in

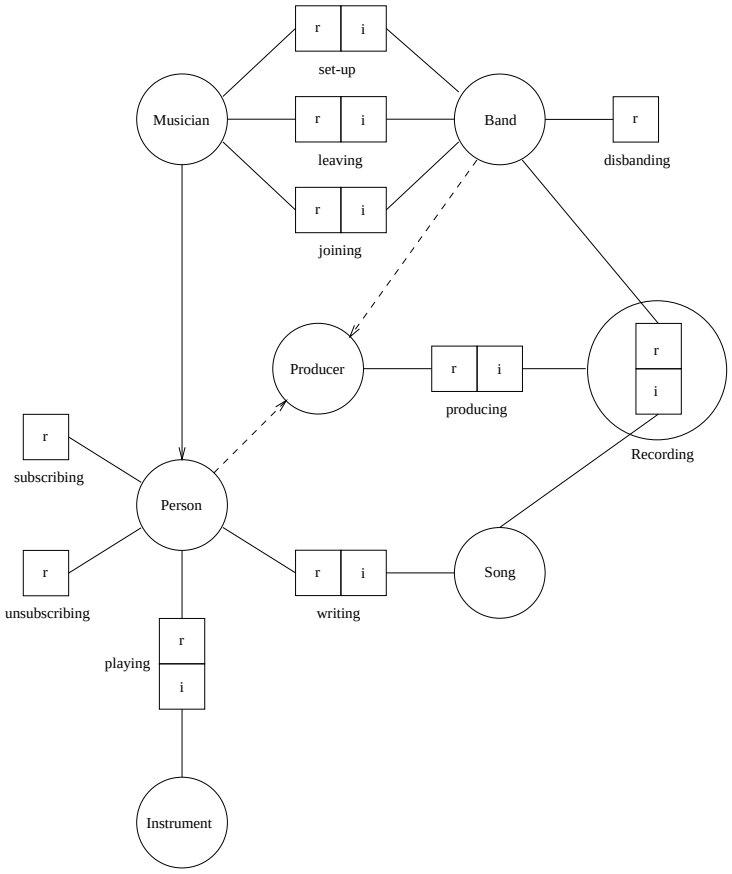

Figure 9: Using generalization and specialization as an abstraction mechanism

section 2.2 learns that action type *to produce* occurs in two rules of different information capacity. However, the result of both action types is the same: a recording is produced. This common

behavior can factored out introducing a generalized type *Producer*. A producer is either a band or a person. The information grammar is now changed in the following way:

$$\begin{aligned} \langle \text{to produce} \rangle &\longrightarrow \langle \text{Producer} \rangle \text{ produces } \langle \text{Recording} \rangle \\ \langle \text{Producer} \rangle &\longrightarrow \langle \text{Band} \rangle \\ &\mid \langle \text{Person} \rangle \end{aligned}$$

In section 2.4 abstraction was used to paraphrase the life of object type *Person* for the behavior addressed by ⟨seq to play⟩. In PSM terminology this leads to a specialization *Musician*. The *subtype defining rule*, i.e. the rule which determines whether a person is also a musician, is: ⟨Person⟩ plays ⟨Instrument⟩. All action types succeeding action type *to play* now require the involvement of *Musician* instead of *Person*. Figure 9 shows a part of the information architecture after applying generalization and specialization. This part contains a hierarchy for a number of object types which is embedded in the information grammar by:

$$\begin{aligned} \langle \text{to be} \rangle &\longrightarrow \langle \text{Band} \rangle \text{ is a producer.} \\ &\mid \langle \text{Person} \rangle \text{ is a producer.} \\ &\mid \langle \text{Musician} \rangle \text{ is a person.} \end{aligned}$$

3 The Role of Grammars in the Analysis Phase

In this section the motivation for using natural language as a base for information modeling is given together with references to some related work.

3.1 Philosophy

Almost all modeling techniques for object-orientation have as point of departure a description in natural language of the world to be modeled (the so-called *universe of discourse*, or UoD for short). This description is referred to as the *initial specification*, and is provided by so-called *informed users* (domain experts). Ideally, this initial specification is a precise specification from which system developers can derive the required information system.

Natural language has the potential to be a precise specification language *provided* it is used well. But there are not many people who can use natural language in a consistent (non-contradictory, expressed on a same level of abstraction), complete and unambiguous way. Still, natural language is the vehicle of our thoughts and their communication ([Qui60]). Since good communication between system analyst and domain expert is essential for obtaining the intended information system, the communication between these two partners should be in a common language: natural language. An additional advantage of using natural language for informal specification is that it saves a translation between the initial specification and informal specification as we assumed the initial specification to be written in natural language. Informal specifications may also give hints on the way in which a user wants to communicate with the information system. A formal specification can never capture the *pragmatics* of a system. Paradoxically, natural language informal specifications allow a higher degree of validation than formal specifications. As a final argument and described in this paper, the formal specification may very well be paraphrased in natural language which increases the possibility for domain experts to validate the formal specification. [Dal92] gives more arguments for paraphrasing a conceptual model to natural language.

3.2 Related work

Using natural language for problem specification is not new in the field of computer science. In the early seventies syntactic oriented programming methods, like step-wise refinement, were

introduced, see e.g. [Dij76, Mee78]. The step-wise refinement method paraphrases the problem top-down using simple control structures like `IF THEN ELSE FI`. Also a natural language approach to information modeling is not new. The conceptual data modeling techniques EER ([BCD+95]), NIAM ([DO90]) and PSM ([HPW94, CW93]) are based on such an approach. For these techniques the goal of the modeling process is to derive the grammar that governs the communication within the UoD, the so-called *information grammar*. This grammar can be depicted as an information structure diagram. The paraphrasing of the elements of the information structure diagram forms the base for the terminal symbols for this grammar. A major advantage of the linguistic approach is that (intermediate) results can be validated by the informed user readily, as each intermediate result corresponds to a (partial) grammar for sentences in the language spoken in the UoD.

In [RP92] for the generation of natural language out of conceptual schemata the *Chomsky theory* [Cho65] is used. The basic Chomsky assumption is the existence of an underlying structure, to any sentence in any human language. In addition, there is an infinite number of ways, namely the surface structures to represent the deep structure in different languages. The deep structure expresses semantics of a sentence by means of semantic elements and relationships among them. The proposed framework is supported by an expert design system, known as OICSI (French acronym for intelligent tool for information system design). Note that this approach is not particularly focused on object-oriented analysis. Furthermore, as a result of using the Chomsky theory this approach has a semantic nature.

A similar, and, quoting the authors ([MG94]), less sophisticated tool is LOLITA which supports a linguistic approach to developing object-oriented systems. A data dictionary is used to find semantic relations between the object types that are detected in the informal specification. This way, for example, the object types *teacher* and *professor* can be related. Furthermore, the alternative formulation *master* may be suggested by the system. Besides this feature, LOLITA is able to automatically identify ambiguities, inconsistencies, correcting misspellings and guess new words.

The LIKE project (Linguistic Instruments in Knowledge Engineering) has resulted in a method called COLOR-X which is an abbreviation for COnceptual Linguistically based Object-oriented Representation language for Information and Communication Systems (where ICS is abbreviated to X). The static and dynamic part of object-oriented modeling is described in [BR95b] and [BR95a], respectively. The expert language is captured in graphical models which can be translated to an intermediate language called Conceptual Prototyping Language (CPL). CPL ([Dig89]) is a formal modeling technique, based on *functional grammars* ([Dik89]), which can be used for specification as close as possible to the informal specification. As a result the paraphrasing of the graphical models is based on CPL.

The main difference between the above mentioned paraphrasing approaches and ours is that they are semantically based whilst ours is syntactically based. Another difference is that our approach models the *communication* between user and information system instead of modeling the information system. Finally, it can be remarked that the paraphrasing mechanism, as described in this paper, can be used for paraphrasing both instances and rules (structure) of the application domain.

4 Conclusions and Future Research

The focus in this paper has been on an informal language based way of object-oriented modeling. Although a complete system specification can be given in PSM^2, this approach also allows system analysts and domain experts to isolate specific views of the informal specification. For the model of each such view a paraphrasing mechanism has been described.

Many aspects of this paper can be used as a topic for future research. Action types form abstractions of a number of small and simple operations. Some action types require the involvement

of subjects (= users). These action types should be examined in more detail as they form the interface with the user. Work in the area of cognitive science can be used for closer inspection of these user action types, and can be used to obtain more suitable systems for end users (see e.g. [HVV91]).

The models of the analysis phase describe the application domain on a high level of abstraction. For implementation of these models more detailed models are necessary. In [FW96b] a closer inspection of the analysis models will be described. Models for specifying attributes and operations will be presented. Furthermore, the information architecture is explored to obtain type hierarchies.

Finally, more research has to be done for the information architecture. The information architecture which has been introduced has its roots in the formal conceptual data modeling technique PSM and the query and constraint language Elisa-D. More elegant information architectures can be obtained exploring the full power of both PSM and Elisa-D. Nonetheless PSM is a *data* modeling technique with semantics based on set theory. Since PSM is used in this paper to describe dynamics a change of semantics might be useful. There exists already some experiences with changing semantics of data models. In [FHL94] a general category theoretical framework is presented for the study of changing semantics of data models.

Acknowledgment

Several people have influenced this paper by constructive criticism and suggestions, most notably Patrick van Bommel and Erik van de Ven. Furthermore, Caspar Derksen assisted us with the examples involving the grammar workbench and parser generator. Their contributions are herewith gratefully acknowledged.

A The Music Example

> *This example deals with persons subscribing for the academy of music. The names and birth dates of persons are administrated. A person who can play an instrument (vocals are also an instrument) can set up a band. Both bands and instruments have names. After a band has been set up, persons may join this band. Persons can also leave a band. A song is written by one or more persons. Each song has associated a song name. Several bands have the opportunity to record some of these songs. Some of these bands will produce these recordings themselves or with help of one or more persons. For each recording a date and a tape number is assigned. A band does not exist any more when it is disbanded. Persons who unsubscribe the academy are not considered anymore.*

B Demonstrating the Information Grammar

In this appendix we demonstrate a first approach to an implementation of information grammars within the AGFL formalism ([Kos91]). Consider the following concretized part of the information grammar as presented in appendix D.

$$
\begin{aligned}
\langle \text{to join} \rangle &\longrightarrow \langle \text{to join} \mid r \rangle \mid \langle \text{to join} \mid i \rangle \\
\langle \text{to join} \mid r \rangle &\longrightarrow \langle \text{Musician} \rangle \text{ joins } \langle \text{Band} \rangle \\
\langle \text{to join} \mid i \rangle &\longrightarrow \langle \text{Band} \rangle \text{ is joined by } \langle \text{Musician} \rangle \\
\langle \text{Band} \rangle &\longrightarrow \text{The Rolling Stones} \mid \text{The Beatles} \\
\langle \text{Musician} \rangle &\longrightarrow \text{Keith Richards} \mid \text{John Lennon}
\end{aligned}
$$

This grammar is translated into a two level grammar in the AGFL formalism as follows:

```
inv :: RESP ; INV.

start: TO JOIN (inv).

TO JOIN (RESP): MUSICIAN, "joins", BAND.

TO JOIN (INV): BAND, "is joined by", MUSICIAN.

BAND: "The Rolling Stones" ; "The Beatles".

MUSICIAN: "Keith Richards" ; "John Lennon".
```

The Grammar Workbench ([DKNZ92]) can be used to generate sample sentences randomly according to this grammar. GWB in this case may generate randomly the following sample sentences.

```
Keith Richards joins The Rolling Stones

John Lennon joins The Beatles

The Rolling Stones is joined by Keith Richards

John Lennon joins The Rolling Stones
```

Note that this set is not complete, i.e. it does not contain all possible sample sentences. As syntax rules might be recursive, GWB is equipped in such way that it avoids coming into an infinite loop.

It is also possible to generate a parser for the information grammar using the parser generator (GEN). In order to accept new person names, the rule for `MUSICIAN` is modified as follows:

```
MUSICIAN: PROPER NAME.

PROPER NAME: $MATCH("[A-Z][a-z]*").
```

Applying GEN to this simple grammar leads to a parser that allows sentences like:

```
Paul joins The Rolling Stones
```

This parser will generate the following parse tree, augmented with some statistics:

```
parsing 1       (0.002 sec.)

start
  TO JOIN(RESP)
    MUSICIAN
      PROPER NAME
        "Paul"
    "joins"
    BAND
      "The Rolling Stones"

printing time was:      (0.010 sec.)

all:     1      (prescan: 0.011 s., parsing: 0.003 s., printing: 0.010 s. )
```

C Symbols of an Object Life Model

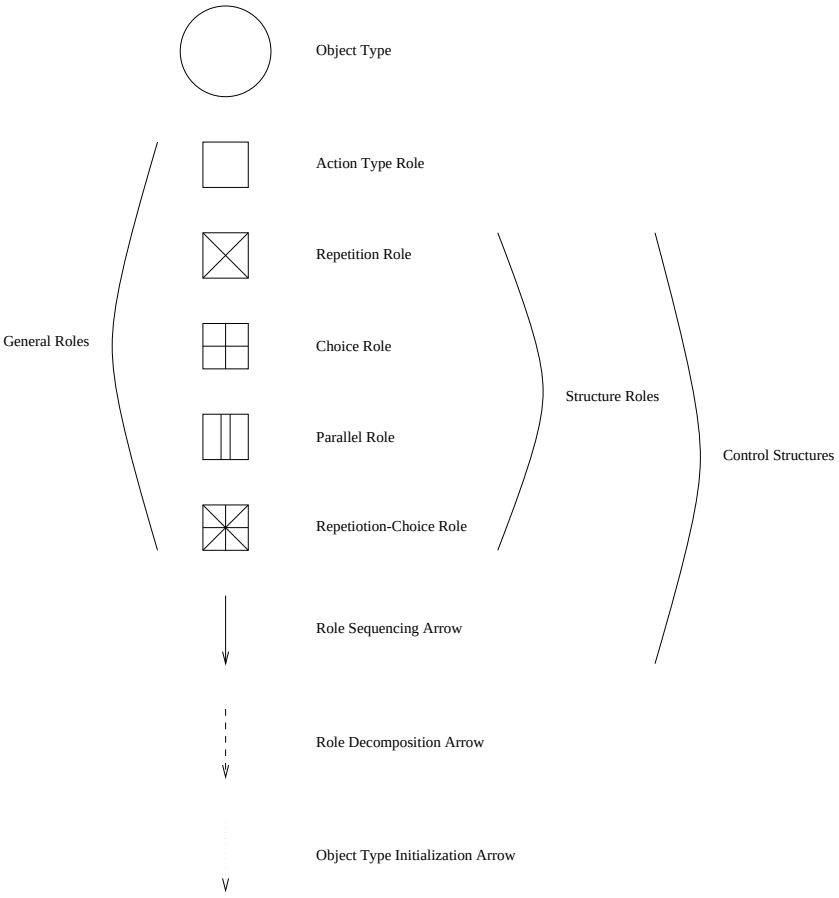

Figure 10:

D Some Models of the Music Example

The following matrix is the complete action involvement model of the music example.

(name of) action type	(name of) object type				
	Band	Instrument	Person	Recording	Song
to disband	r				
to join	i		r		
to leave	i		r		
to play		i	r		
to produce	r			i	
to produce			r	i	
to record	r			i	i
to set up	i		r		
to subscribe			r		
to unsubscribe			r		
to write			r		i

The object property model of the music example is:

object type	*property*
Band	B-name
Instrument	I-name
Person	B-date
Person	P-name
Recording	R-date
Recording	Tape-nr
Song	S-name

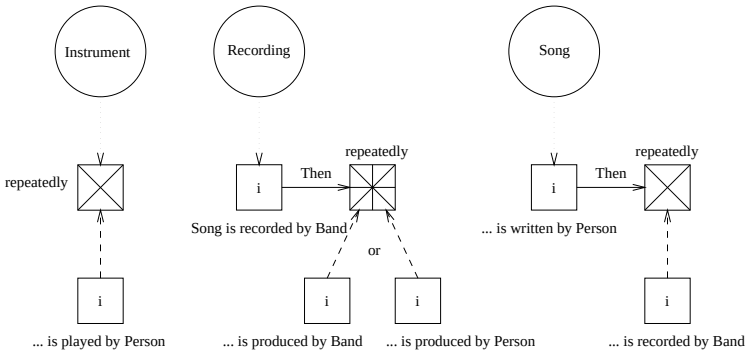

Figure 11: The remaining courses of life

Figure 11 shows all remaining courses of life of the object life model of our music example.

References

[AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[BCD+95] E. Buchholz, H. Cyriaks, A. Düsterhöft, H. Mehlan, and B. Thalheim. Applying a Natural Language Dialogue Tool for Designing Databases. In *Proceedings of the First Workshop on Applications of Natural Language to Databases (NLDB'95)*, pages 119–133, Versailles, France, June 1995.

[Boo91] G. Booch. *Object-Oriented Design with Applications.* Benjamin Cummings, Redwood City, California, 1991.

[BR95a] J.F.M. Burg and R.P. van de Riet. COLOR-X: Linguistically-based Event Modeling: A General Approach to Dynamic Modeling. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *The Proceedings of the Seventh International Conference on Advanced Information System Engineering*, Lecture Notes in Computer Science, pages 26–39, Jyvaskyla, Finland, 1995. Springer-Verlag.

[BR95b] J.F.M. Burg and R.P. van de Riet. COLOR-X: Object Modeling profits from Linguistics. In *Proceedings of the KB&KS'95, the Second International Conference on Building and Sharing of Very Large-Scale Knowledge Bases*, Enschede, The Netherlands, 1995.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge University Press, Cambridge, United Kingdom, 1990.

[Che76] P.P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Cho65] N. Chomsky. *Aspects of the theory of syntax.* MIT Press, Cambridge, Massachusetts, 1965.

[CW93] M.A. Collignon and Th.P. van der Weide. An Information Analysis Method Based on PSM. In G.M. Nijssen, editor, *Proceedings of NIAM-ISDM.* NIAM-GUIDE, September 1993. pp. 22.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Yourdon Press, New York, New York, 1990.

[Dal92] H. Dalianis. A method for validating a conceptual model by natural language discourse generation. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 425–444, Manchester, United Kingdom, 1992. Springer-Verlag.

[DFW96] C.F. Derksen, P.J.M. Frederiks, and Th.P. van der Weide. Paraphrasing as a Technique to Support Object-Oriented Analysis. Technical Report CSI-R9603, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, January 1996. (accepted for Natural Language to Databases '96).

[Dig89] F.P.M. Dignum. *A Language for Modelling Knowledge Bases.* PhD thesis, Free University, Amsterdam, The Netherlands, 1989.

[Dij76] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[Dik89] S.C. Dik. *The Theory of Functional Grammar. Part I: The Structure of the Clause.* Floris Publications, Dordrecht, The Netherlands, 1989.

[DKNZ92] C. Dekkers, C.H.A. Koster, M.-J. Nederhof, and A. van Zwol. The Grammar Workbench: A First Step towards Lingware Engineering. In *Proceedings of the second Twente Workshop on Language Technology, Memoranda Informatica 92-29*, pages 103–115, Enschede, The Netherlands, April 1992. University of Twente.

[DO90] L. Dunn and M. Orlowska. A Natural Language Interpreter for the Construction of Conceptual Schemas. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 175–194, Stockholm, Sweden, 1990. Springer-Verlag.

[FHL94] P.J.M. Frederiks, A.H.M. ter Hofstede, and E. Lippe. A Unifying Framework for Conceptual Data Modelling Concepts. Technical Report CSI-R9410, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, September 1994. Accepted for publication in Information and Software Technology.

[FW96a] P.J.M. Frederiks and Th.P. van der Weide. From a File-Oriented View to an Object-Oriented View. Technical Report CSI-R9601, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, January 1996.

[FW96b] P.J.M. Frederiks and Th.P. van der Weide. Properties and Design of Information Architectures. Technical Report, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, September 1996. (in preparation).

[Gra94] I. Graham. *Object-oriented Methods.* Addison-Wesley, Reading, Massachusetts, 1994.

[HPW93] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[HPW94] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Grammar Based Information Modelling. Technical Report CSI-R9414, Submitted for publication, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, October 1994.

[HVV91] G. de Haan, G.C. van der Veer, and J.C. van Vliet. Formal modelling techniques in human-computer interaction. *Acta Psychologica*, 78:27–67, 1991.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[JCJO92] I. Jacobson, M. Christerson, M. Jonsson, and P. van Overgaard. *OO Software Engineering, A Use Case Driven Approach.* Addison-Wesley, Reading, Massachusets, 1992.

[Kos91] C.H.A. Koster. Affix Grammars for natural languages. In *Attribute Grammars, Applications and Systems, International Summer School SAGA*, volume 545 of *Lecture Notes in Computer Science*, pages 469–484. Springer-Verlag, Berlin, Germany, June 1991.

[Kri94] G. Kristen. *Object Orientation, the KISS Method: From Information Architecture to Information System.* Addison-Wesley, Reading, Massachusetts, 1994.

[Mee78] L.G.L.T. Meertens. Program text and program structure. In P.G. Hibbard, S.A. Schuman, editor, *Constructing quality software*, pages 271–283, Amsterdam, May 1978. North-Holland. IFIP WG 2.1/WG 2.4 Working Conference, Novosibirsk.

[MG94] L. Mich and R. Garigliano. A Linguistic Approach to the Development of Object Oriented Systems using the NL System LOLITA. In E. Bertino and S. Urban, editors, *Proceedings of the International Symposium, ISOOMS '94: Object-Oriented Methodologies and Systems*, volume 858 of *Lecture Notes in Computer Science*, pages 371–386, Palermo, Italy, September 1994. Springer-Verlag.

[NF84] G.M. Nijssen and E.D. Falkenberg. *Introduction to IBM SQL, Release 2.* Nijssen Data Bases Pty. Ltd., Chapel Hill, Queensland, Australia, 1984.

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach.* Prentice-Hall, Sydney, Australia, 1989.

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1994.

[PW95] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Qui60] W. Quine. *Word and object – Studies in communication.* The Technology Press of the Massachusetts Institute of Technology, Cambridge, Massachusetts, 1960.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[RP92] C. Rolland and C. Proix. A Natural Language Approach For Requirements Engineering. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 257–277, Manchester, United Kingdom, 1992. Springer-Verlag.

[WMP+76] A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.T. Meertens, and R.G. Fisker. *Revised Report on the Algorithmic Language ALGOL 68.* Springer-Verlag, Berlin, Germany, 1976.