Pergamon

Information Systems Vol. 22, No. 6/7, pp. 349–385, 1994

0306-4379/94 $7.00 + 0.00

EXPLOITING FACT VERBALISATION IN CONCEPTUAL INFORMATION MODELLING

A.H.M. TER HOFSTEDE[1], H.A. PROPER[2], and TH.P. VAN DER WEIDE[3]

[1]Cooperative Information Systems Research Centre, Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane, 4001 Australia
[2]Origin Nederland B.V., POBox 7377, 1007 JJ Amsterdam, The Netherlands, E.Proper@acm.org
[3]Computing Science Institute, University of Nijmegen, Toernooiveld, 6525 ED Nijmegen, The Netherlands

Abstract — An increasing number of approaches to conceptual information modelling use verbalisation techniques as an aid to derive a model for a given universe of discourse (the problem domain). The underlying assumption is that by elaborate verbalisation of samples of facts, taken from the universe of discourse, one can elicit a complete overview of the relevant concepts and their inter-relationships. These verbalisations also provide a means to validate the resulting model in terms of expressions familiar to users. This approach can be found in modern ER variations, Object-Role Modelling variations, as well as different Object-Oriented Modelling techniques. After the modelling process has ended, the fact verbalisations are hardly put to any further use. As we belief this to be unfortunate, this article is concerned with the exploitation of fact verbalisations after finishing the actual information system. The verbalisations are exploited in four directions. We consider their use for a conceptual query language, the verbalisation of instances, the description of the contents of a database, and for the verbalisation of queries in a computer supported query environment. To put everything in perspective, we also provide an example session with an envisioned tool for end-user query formulation that exploits the verbalisations.

Key words: information modelling, ER, Object-Role Modelling, verbalisation, natural language queries, query formulation, information grammar, path expressions.

PUBLISHED AS:

A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Exploiting Fact Verbalisation in Conceptual Information Modelling. *Information Systems*, 22(6/7):349–385, September 1997.

1. INTRODUCTION

When developing an information system using a conceptual modelling approach, usually the first task is to derive a complete model of the underlying universe of discourse (also referred to as the application domain, or problem domain). Extracting this information from the universe of discourse is a hard problem, and is comparable to the problem of knowledge elicitation for the design of expert systems [WBC84, BS84, HRWL83]. A wide range of modern conceptual modelling techniques start out by verbalising sample forms, cases, etc. taken from the universe of discourse. This verbalisation process is usually conducted in close cooperation with a domain expert. Examples of such approaches are the ER variation discussed in [BCDT95], NIAM [NH89, Win90], Object-Role Modelling (ORM) [Hal95], OMT [RBP$^+$91], and the KISS method [Kri94]. The underlying assumption is that by elaborate verbalisation of the samples from the universe of discourse, a sufficient overview of the structure and rules in the universe of discourse can be obtained.

The verbalisations resulting from this initial step are then used as input for the actual design of the conceptual model. In this article we only focus on the information modelling side of conceptual modelling. So in our case verbalisations are used as input for the design of the so called *information model*. In the next section we argue that the information model essentially provides a crude grammar of the language spoken by the communication partners in the universe of discourse. It should already be noted though that this grammar does not have the richness and power (and inherent ambiguity) of a real natural language grammar.

As stated before, verbalisation of samples has been introduced as a means to obtain a complete overview of the universe of discourse. This in itself has proven to be useful in the many projects in which methods like

OMT, ORM, NIAM, and KISS have been used. Even more, some modern CASE-Tools like InfoModeler [Asy94] fully support these verbalisation techniques. During the modelling process, the fact verbalisations can also be employed for verbalisation purposes. By showing users some sample instances verbalised as natural language expressions, the users can validate the correctness of the information model [Dal92, HM92]. After the modelling process has been finished, these carefully crafted verbalisations are hardly put to use. In this article we are concerned with ways to exploit these verbalisations after the information system has been built.

The first area in which we will try to exploit verbalisations is to define a query language that is closer to the language users are used to. The idea for a more natural query language was born out of frustration with the formulation of queries in SQL. After all the trouble modellers and users have gone through in verbalising samples from the universe of discourse to arrive at a properly designed conceptual schema, it comes as an anti-climax if end-users (and database administrators) still have to use SQL to formulate queries, and even more, have to refer to the actual tables to access the information. This frustration has fueled the development of so-called *conceptual query languages* such as RIDL [Mee82] and LISA-D[†] [HPW93].

In this article we present a further refined version of LISA-D. What conceptual query languages have in common is that they allow for the formulation of queries in terms of the conceptual model, which is a gain in itself. Moreover, these languages use the verbalisations resulting from the samples to cater for query formulations that more closely resemble the language used by people in the universe of discourse. So-far, these languages have not received much attention due to the over exposure of commercial SQL systems. Nevertheless, the increasing number of commercially available tools that allow some form of natural language querying also support this development in the direction of query languages that are 'closer to the people'. A first commercial attempt in this direction can be found in the InfoAssistant tool, which is marketed by the same company as InfoModeler.

The second and third area in which we attempt to exploit fact verbalisations is concerned with the verbalisation of instances. Better verbalisation of instances is beneficial in two ways. The first way we can put these to use is in a further enrichment of the LISA-D language by improving on the way instances can be verbalised in this language. A second use is related to the integration of structured databases into the World-Wide-Web (WWW). At present, standards like Z39.50 [ANS95] receive a lot of attention, as they provide an avenue to access meta data about databases, i.e. what a database is *about*. The fact type verbalisations give us a good handle on the semantics of the stored information. Therefore, if we can use the fact type verbalisations as meta data and make it accessible through such protocols, then we expect the precision of selecting relevant databases to increase. If an instance of an object type in a conceptual schema can be verbalised explicitly, then this verbalisation can be used to advertise this database to search engines for the WWW (or internet in general). In [RS96], this idea is discussed in the context of federated databases without elaborating on the role of fact verbalisations. However, it should be clear that elaborate verbalisation of instances can aid in finding the underlying database. Verbalisations of instances carry more conceptual information than the table headers or the actual tuples in the database.

A fourth area in which we try to exploit verbalisations is support for verbalisation of queries in the context of computer supported query formulation (CSQF).

The central focus of this paper is both on a formalisation of the verbalisation of the samples provided during the analysis phase, as well as their exploitation for the above four purposes. The structure of the paper is as follows. In section 2, verbalisation as an approach for conceptual modelling is explained in more detail, with a special focus on Object-Role Modelling (ORM). There we will also justify our focus on ORM, although our results are most certainly not exclusive to ORM. Section 3 shows how the verbalisations on facts can be used to construct a conceptual query. In section 4 a verbalisation mechanism for instances is discussed, while some applications will be highlighted. The verbalisation of paths through conceptual schemas, needed for query verbalisation in a CSQF environment, is addressed in section 6. This is followed in section 6 by a brief discussion of a way to support users in query formulation processes. This sample session allows us to highlight the benefits of exploiting fact verbalisations.

2. INFORMATION STRUCTURES

[†]Language for Information Structures and Access Descriptions.

When using a verbalisation based approach to information modelling, the modelling technique used to express the resulting models should feature a rich set of modelling constructs. The reason for this is that the (syntactic) constructs used in examples of verbalisations should have direct pendants in the used information modelling technique. This does not only warrant a natural way of modelling, but also maintains the rich semantics of verbalisations in natural language as much as possible. Among the first to apply this principle to conceptual modelling was Nijssen [Nij81, TN85, LN88].

The information modelling technique we use throughout this paper is a modernised version of NIAM, called Object-Role Modelling (ORM). Before continuing we give a brief account of the history of this ORM version. In [BHW91] the Predicator Model was introduced as one of the first formalisations of NIAM [NH89, Win90, Hal95]. The Predicator Model was superseded by the first version of PSM [HW93, HPW93]. PSM was mainly designed to deal with complex objects occurring in complex domains such as CAD/CAM, hypermedia, office automation and meta-modelling. A proposal to refine the conceptual schema design procedure of ORM to effectively deal with complex objects during the modelling process was given in [CW93]. Meanwhile, new research has led to refinements of the original PSM model. In [HW94, HW97] a new class of constraints is proposed that allows for a reduction of the number of constructs. Research into the relationships between ORM and other information modelling techniques [BBMP95, HLF96, LH96, HP95, FHL97] has led to a better understanding of the underlying constructs and their mutual relationships. Finally, in [CP96] an Object-Role Modelling version was proposed, the Conceptual Data Modelling Kernel (CDM Kernel) which tries to put all these results into perspective. A key feature of the CDM Kernel is that it provides a generic data modelling technique that is general enough to include ORM models, (E)ER models, and the information models of some OO variations (at least the ones reported in [RBP+91, Kri94]).

The version of ORM (PSM) we employ in this article is setup such that it is most convenient for the purposes of this article. It is not complete with regards to the set of modelling constructs that exist in full ORM. From time to time we will briefly relate the formalisation used in this article to the one of the CDM Kernel. The CDM Kernel focuses on genericity and atomicity of the modelling constructs, whereas this article needs to focus on naturalness of modelling constructs from a verbalisation point of view. This latter requirement implies that some modelling constructs, though splittable from a formal point of view, should for pragmatic reasons not be split.

To warrant a proper understanding of the definitions given in the remainder of this article, we shall provide a summary of the formalisation of ORM. We only provide a summary, as ORM and its formalisations have now been widely published. Before we provide this summary, however, we first discuss the used modelling constructs from a verbalisation (grammatical) point of view. Some of the examples used in this paper are taken from a fragment of the so-called *Presidential Database*, dealing with the election process of presidents from the United States of America. This example served as a unified example in the special issue of Computing Surveys [FS76] and made its first appearance in [WBGW73]. This example was chosen here for its intuitive simplicity. As the schema does not contain any advanced concepts, also other, more complex, examples will be used.

2.1. Verbalisation based approach to information modelling

The point of departure of verbalisation based information modelling approaches is a description of the communication in the world to be modelled (universe of discourse) in terms of a set of sample sentences. The intention of the modelling process is then to derive a conceptual schema from this set.

An interesting question that comes to the fore, is to what extent the process of deriving a conceptual schema from a set of sample sentences can be supported by automatic tools [BRC93, BR95b, BR95c].

Another interesting issue that lingers beneath the surface, is to see to what extent these approaches depend on the language used to express the verbalisations. This actually involves two questions. First, one may wonder if this approach applies to all natural languages. The NIAM/ORM approach has been applied to English, Dutch, German, French, and even Japanese [HH93]. However, there is no guarantee that it can be applied successfully to all languages. Secondly, modelling a single domain in two different languages may lead to two different 'conceptual' models. This may be caused by the fact that different languages may distinguish different sets of atomic concepts.

In [Hal95] and [Kri94] the process of verbalising samples, refining the samples, and delivering up a

conceptual model in ORM or KISS respectively, is discussed in full detail. An important role in this process is played by the notion of an *elementary* fact. What is sought after are the most basic facts in a universe of discourse.

In our view, two of the key research challenges are now:

1. Build tools to aid humans in the verbalisation of facts from the universe of discourse.
2. Enhance the conceptual schema modelling language with constructs that allow for alternative, and more natural, verbalisations of facts to enrich query (and result) formulation.

This article is concerned with the second challenge. In [BRC93, BR95b, BR95a, BR95c, Bur96, Fre97] novel research efforts are described addressing the first challenge.

In the next section we provide a relatively simple mechanism to formalise the verbalisations provided with a conceptual schema. This formalisation provides a form of a grammar, the *information grammar*. The information grammar essentially provides a grammar for the sample fact verbalisations used during the modelling process. As such it can be seen as a (very primitive) grammar of the language spoken in the domain; the *expert language*.

2.2. Basic type classes

Object types characterise the type and their structure of things that may occur in the universe of discourse. They can be recognised quite easily in the sample sentences, since they tend to correspond to nouns. In fact verbalisations, verbs typically express how object types participate in *relationship types*. In the formalisation, *roles* will be used to capture the way in which object types participate in relationship types.

Figure 1 shows the membership relations between political parties and presidents as a binary relationship type. A president plays the *role* of *being* a member of a party, while a party plays the *role* of *having* members.

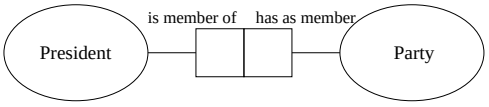

Fig. 1: A binary relationship type

It should be noted that the information diagram of figure 1 abstracts, not only from instances, but also from *labels* and *references*. Labels (or rather values) correspond to things that can be represented directly on a communication medium (for example strings as 'Eisenhower D.D.'), while entities correspond to things that *cannot* be represented directly on such a medium (for example the person referred to as 'Eisenhower D.D.'). In figure 1, the object types *Party* and *President* are examples of entity types. Instances of the entity types cannot be represented directly, but need identification via value types such as President Name and Party Name.

References are relations that connect between entities and values. Reference types [NH89] are also referred to as *bridge types* [Win90] as they bridge the gap between abstract and concrete object types. A typical manifestation of a reference type is the following sample sentence:

There is a party with name 'Republican'

This is an example of an *existence postulating fact*. The left hand side of figure 2 contains the resulting schema fragment. Usually this kind of schema fragment is abbreviated as shown in the right hand side of figure 2.

In figure 2 also examples of two important constraint types are shown. The black dot is the graphical representation of a *total role constraint*. The total role constraint on the role with name with expresses the fact that each instance of Party should have an associated name. The arrow tipped bars are examples of *uniqueness constraints*. The bar under the role with name with captures the fact that each instance of Party has at most one associated name, while the bar under the role with name of captures the fact that no two

different instances of Party have identical names. These three constraints are sufficient to guarantee that all instances from the entity type Party can be identified by instances from the value type Party Name. Note that value types are represented by dotted ellipses rather than solid ellipses. As this situation of a relatively simple identification occurs frequently, the abbreviation shown is used for these common cases.

Fig. 2: Example of a reference type

Identification can, unfortunately, be quite complex. An example is shown in figure 3. In this example entity type Address can be identified by the combination of Street and House Nr. Instances of Street can be identified by combinations of instances from Community and Street Name. Entity type Community is identified by value type Community Name.

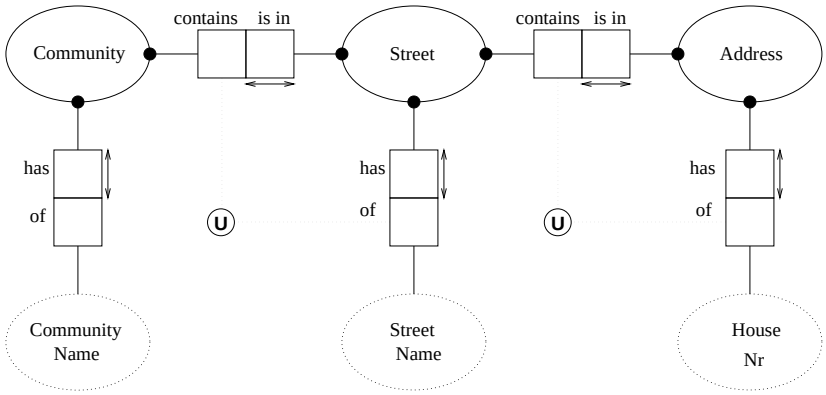

Fig. 3: An example of complex identification

Relationship types may be treated as object types, a process referred to as *relationship objectification*. As an example consider figure 4, which contains the complete schema of the presidential database. In this figure, relationship type Marriage is an objectified relationship type. Marriages may result in a number of children. Corresponding sample sentences could be:

President 'Washington G.' is spouse of person 'Custis M.D.'

The marriage of president 'Washington G.' with person 'Custis M.D.' resulted in 0 children

Strictly spoken, objectification is not an elementary concept. In [CP96] it is shown how objectification can be regarded as a special flavour of abstraction. Due to our focus on verbalisation, however, objectification needs to be treated as a distinct modelling construct.

The running example of figure 4 also contains an example of specialisation. Object type President is a subtype of object type Politician, which on its turn is a subtype of object type Person. This specialisation hierarchy results from one of the last steps in the ORM design procedure in which types which may have instances in common are merged into subtype hierarchies.

2.3. Complex type classes

Objectification is not the only type construction mechanism. In this article we also discuss collection types (or power types) and schema types. Besides these two additional type constructs, one might decide to introduce: bag types, sequence types, list types, array types, etc. Objectification, collection types and schema types are addressed in this article as examples of *possible* (non-elementary) type construction mechanisms. However, as in the case of objectification, these type constructions are not elementary. Nevertheless, the additional type constructs are introduced for the same reason as objectification; they more directly support the natural verbalisation of phenomena in the universe of discourse.

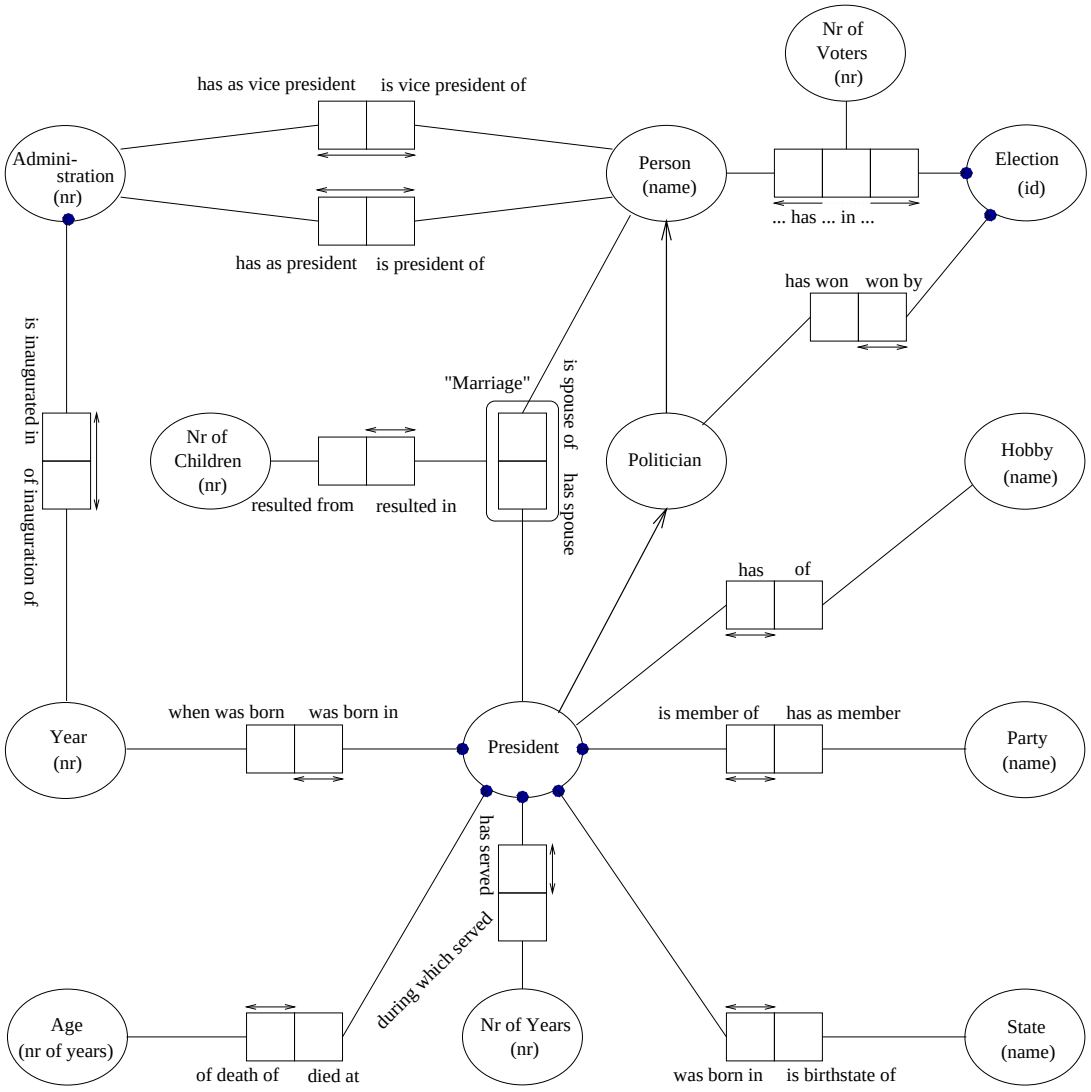

Fig. 4: Part of an information structure dealing with presidents of the United States of America

2.4. Inheritance between object types

Most advanced data modelling techniques offer constructs to define inheritance relations over types. Typical examples are specialisation, generalisation, and polymorphism. Is this article we will formally take the approach of only having one class of inheritance between types, independent of the question whether two different flavours of inheritance should indeed be distinguished. For the purposes of this article the existence of an inheritance hierarchy as such is simply sufficient.

2.5. Formal syntax

We have now arrived at the formal definition of PSM. For the intentions of this paper we will not discuss the constraints that can be defined on a conceptual schema. Rather, we focus on the underlying information structure of a conceptual schema as this structure provides the base upon which verbalisations will be build. The formalisation below is provided only to make the article self-contained.

A conceptual schema without constraints is referred to as an *information structure*. An information structure $\mathcal{IS}$ consists of the following components:

1. A nonempty finite set $\mathcal{OB}$ of *object types*.

2. A set $\mathcal{EN} \subseteq \mathcal{OB}$ of *entity types*.
3. A set $\mathcal{VL} \subseteq \mathcal{OB}$ of *value types*.
4. A finite set $\mathcal{RO}$ of *roles* that represent the ways in which object types may participate in relationship types.
5. A function $\mathsf{Player} : \mathcal{RO} \rightarrow \mathcal{OB}$. At the base of a role is an object type playing that role.
6. A partition $\mathcal{RL}$ of the set $\mathcal{RO}$. The elements of $\mathcal{RL}$ are called *relationship types*. Relationship types are also object types: $\mathcal{RL} \subseteq \mathcal{OB}$.

   The auxiliary function $\mathsf{Rel} : \mathcal{RO} \rightarrow \mathcal{RL}$ yields the relationship type in which a given role is contained, and is defined by: $\mathsf{Rel}(p) = r \iff p \in r$.
7. A set $\mathcal{CT} \subseteq \mathcal{OB}$ of *collection types*.
8. A set $\mathcal{SC} \subseteq \mathcal{OB}$ of *schema types*.
9. A function $\mathsf{Elt} : \mathcal{CT} \rightarrow \mathcal{OB}$ yielding the element type of collection types.
10. A relation $\prec \subseteq \mathcal{SC} \times \mathcal{OB}$ capturing the decomposition of schema types.
11. A binary relation $\mathsf{SubOf} \subseteq \mathcal{OB} \times \mathcal{OB}$ capturing the (unified) inheritance hierarchy.
12. A many-sorted algebra $D = \langle \mathcal{D}, F \rangle$, with $\mathcal{D}$ a set of concrete domains (e.g. string, natno) and F a set of operations (e.g. $+$).
13. A function $\mathsf{Dom} : \mathcal{VL} \rightarrow \mathcal{D}$. The instances of a value type originate from its associated domain.

A detailed discussion of well-formedness rules on information structures is not provided here. These axioms can be found in the aforementioned publications on ORM.

2.6. Verbalisation information

Given an information structure, a first simple verbalisation for this structure can be defined by providing names for the schema constructs present. The verbalisation $\mathcal{VB}$ of an information structure $\mathcal{IS}$ is composed of the tuple $\mathcal{VB} = \langle \mathsf{ONm}, \mathsf{PNm}, \mathsf{RNm}, \mathsf{MFix} \rangle$ of namings. During information analysis, all types receive a unique name, recorded by the function $\mathsf{ONm} : \mathcal{OB} \rightarrow \mathcal{NM}$, where $\mathcal{NM}$ is a set of names.

Roles also receive a name of their own. This name is referred to as the *participation* name, and are provided by the function $\mathsf{PNm} : \mathcal{RO} \rightarrow \mathcal{NM}$. Roles of different relationship types may have identical participation names. However, two roles of the same relationship type must have different participation names. Roles may also have a *reverse participation name*, $\mathsf{RNm} : \mathcal{RO} \rightarrow \mathcal{NM}$ which verbalises the decomposition of the relationship type into the components involved.

The participation names are usually omitted from schema diagrams. Schema diagrams usually only feature so-called *mix-fix* verbalisations. For instance, the fact verbalisation:

The president with name 'Eisenhower D.D.' is a member of the party with name 'Republican'

has as underlying mix-fix verbalisation: ... is a member of .... This is an example of a binary relationship type. A ternary example, from the presidential database, is: ... has nr of ... in .... Formally, these mix-fix verbalisations are captured by the relationship:

$$\mathsf{MFix} \subseteq \mathcal{NM}^+ \times \mathcal{RO}^+$$

If $\mathsf{MFix}([w_1, \ldots, w_m], [p_1, \ldots, p_n])$ then $m + 1 = n$, and all roles used must be from one relationship type: $\{p_0, \ldots, p_n\} \subseteq \mathsf{Rel}(p_0)$. Furthermore, the w_i's must all be non-empty names.

As an illustration, consider the ternary fact type in the presidential database example, and assume that the three involved roles are p, q and r for Person, Voters and Election respectively. We could now have the following possible verbalisations:

MFix(['has', 'in'], $[p, q, r]$)

MFix(['has in', 'as result'], $[p, r, q]$)
MFix(['chose for', 'in'], $[q, p, r]$)
MFix(['in', 'chose for'], $[q, r, p]$)
MFix(['in which', 'had'], $[r, p, q]$)
MFix(['had', 'for'], $[r, q, p]$)

MFix(['who was a candidate in'], $[p, r]$)
MFix(['that had as participant'], $[r, p]$)
MFix(['who was an election candidate'], $[p]$)

The first six verbalisations cover all possible orders in which a ternary fact type can be verbalised. The last three verbalisations are examples of *partial* verbalisations, which only deal with a selection of all roles of the fact type involved. In the next section we will see that these partial and total verbalisations, and in particular the binary ones, play a crucial role in the verbalisation of queries in LISA-D.

Please note that it is not a requirement that all possible orders of verbalisation be provided. All that is required is at least one complete verbalisation (covering all roles). However, the more verbalisations is provided the more flexible the resulting information grammar becomes.

We realise that when using a more advanced mechanism from linguistics, for example functional grammars [Dik89], a more flexible verbalisation mechanism would result. However, the aim of this article is simply to explore the possibilities provided when a repository of verbalisation information is available to us. The above definitions capture the minimal amount of verbalisation information that can be reasonably gleaned from a modelling process without burdening the actual process. Capturing verbalisations as functional grammar expressions would lead to a more intensive modelling process as each verbalised fact needs to be dissected into its linguistic components [Bur96].

3. CONCEPTUAL QUERY LANGUAGE

In this section the aim is to exploit fact verbalisations in the context of a conceptual query language. Simply allowing any question that can be formulated in the expert language as a query would introduce interpretation problems due to the inherent ambiguity of natural language. Therefore, we focus on a controlled way to do this without introducing ambiguity.

As explained in the previous section, the first step of the ORM design procedure results in a set of sample sentences describing the structure of the universe of discourse. The names of object types and the facts in which they can play a role originate from these sentences. Consequently, using these names in the formulation of queries and constraints will yield results very close to their original formulation in natural language. However, one should realise that with this mechanism one does by far not obtain the richness of a natural language. This apparent disadvantage, however, comes hand in hand with the advantage that the inherent ambiguity of natural language is absent.

Designers of query languages based on this idea necessarily have to perform a delicate balancing act between on the one side the requirement to define a mathematically based and unambiguous language, and on the other side the need to define a language that is close to natural language. The result is a language in which expressions have a clear intuitive meaning as they are close to natural language, but which handles 'sluggish' when formulating expressions. The word 'sluggish' here depends on one's background. From a pure natural language point of view, the resulting language is certainly not flexible and elegant. When looking from an SQL point of view though, the resulting language is highly flexible, orthogonal, and closer to one's intuition. Finally, by adding tool support helping users in the query formulation process, the 'sluggishness' of the language should be dramatically lessened.

The first language to pursue these ideas was the language RIDL (Reference and IDea Language [DMP84, Mee82]. RIDL, however, never received much acceptance as it did not have a formal syntax and semantics, and was based on a restricted binary version of ORM (see e.g. [VB82]). LISA-D (Language for Information Structure and Access Descriptions) can be considered as a redesign of RIDL, however, with a functionality

far exceeding it. The formal semantics of LISA-D is defined in [HPW93] by means of a translation to path expressions. Path expressions form the semantical base for information grammars. They also form the basis for visualisation of the disclosure mechanism (see section 6) used in the proposed computer supported query formulation system. Path expressions are an extension of relational algebra emphasising concatenation of relations. Appendix A presents an overview of path expressions.

The core of LISA-D is formed by the simple verbalisation $\mathcal{VB}$ of the information structure $\mathcal{IS}$ as we have discussed in the previous section. In subsection 3.1 the basic lexicon is introduced, which is filled by all simple names of the concepts in the information structure. In subsections 3.2, 3.3, 3.4, and 3.5 the general (application-independent) grammar rules are introduced. Subsection 3.6 concludes with a mechanism for specifying application specific grammar rules, and a discussion on the expressive power of LISA-D.

Finally, it should be noted that not the complete definition of path expressions and information descriptors is presented. For both, only a relevant subset of the existing constructions is presented, while the aspects that are new in the current LISA-D version do receive a more detailed discussion. For a complete definition of the 'old' version of LISA-D, refer to [HPW93, PW95].

3.1. Construction of the lexicon

The basis for LISA-D information descriptors is a lexicon, assigning a meaning to the words (names) that constitute the language. The meaning of names is administered by the relation

$$\mathsf{Lexicon} \subseteq \mathcal{NM} \times \mathcal{PE}$$

where $\mathcal{NM}$ is a set of names. The expression $\mathsf{Lexicon}(n, P)$ is used to indicate that n serves as a name for path expression P. We can now fill this lexicon with the predefined names and their meaning.

3.1.1. Named concepts

The naming $\mathcal{VB} = \langle \mathsf{ONm}, \mathsf{PNm}, \mathsf{RNm}, \mathsf{MFix} \rangle$ is used to make an initial filling of the lexicon relation. For this initial filling we do not yet utilise the mix-fix verbalisations provided by MFix. These latter verbalisations will receive a separate treatment in subsection 3.3.

The name $\mathsf{ONm}(x)$ of object type x simply represents the path expression x: $\mathsf{Lexicon}(\mathsf{ONm}(x), x)$ providing, as it were, a path from itself to itself. As object types are usually nouns, we could also introduce the names: $\mathsf{Lexicon}(\mathsf{Article}(\mathsf{ONm}(x))\ \ \mathsf{ONm}(x), x)$, where Article is a function providing the article for a given noun.

If p is a role which has some fact participation name, then this name, $\mathsf{PNm}(p)$, describes a path from the player of p to its corresponding relationship type: $\mathsf{Lexicon}(\mathsf{PNm}(p), p)$. If p is a role which has a reverse participation name, then this name, $\mathsf{RNm}(p)$, describes a path from the relationship type corresponding to p to its player: $\mathsf{Lexicon}(\mathsf{RNm}(p), p^{\leftarrow})$.

3.1.2. Anonymous concepts and generic names

When using complex types like objectifications and collection types, or the graphical abbreviation for simple identification cases, some relationship types may remain unnamed. These relationship types are the so-called *implicit relationship types*. They are formally present in the conceptual schema, however, they are never drawn explicitly. For these, thus far anonymous, relationship types we can introduce default names. In figure 5 these default names are listed for all the implicit relationship types.

With these generic names, we will be able to formulate expressions like:

Nr of Children WHICH IS $<= 3$

President INVOLVED IN Marriage

Nr of Children that resulted from a Marriage OF President: 'J.F. Kennedy'

The phrase President: 'J.F. Kennedy' is a denotation mechanism for instances of object types. Throughout this section we will use the 'colon notation' for this purpose. In the next section we discuss instance denotation, and in particular its verbalisation, in more detail.

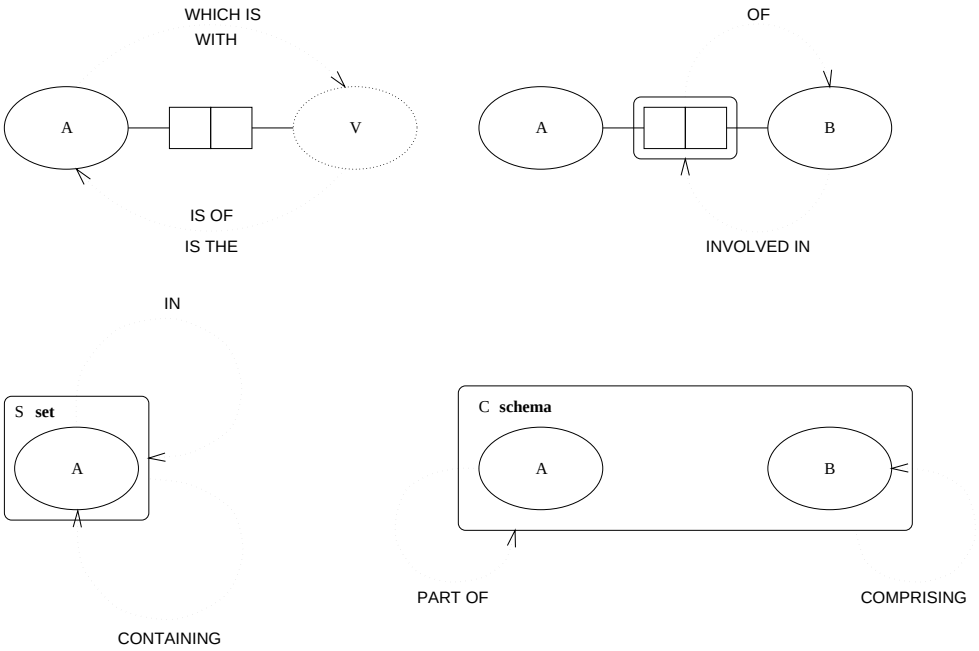

Fig. 5: Illustration of naming of anonymous concepts

3.2. Atomic information descriptors

The semantics of information descriptors are defined by means of the $\mathbb{D}$ function in terms of the path expressions as defined in appendix A. We define the semantics of LISA-D using the style of denotational semantics [Sto77, Wat91], using abstract syntax.

The basic building blocks for LISA-D information descriptors are the names from Lexicon and constants. The meaning of a name is obtained as the sum of all possible interpretations as recorded by this lexicon.

$$\mathbb{D}[\![n]\!] \quad = \quad \bigcup_{P:\mathsf{Lexicon}(n,P)} P$$

Examples of atomic information descriptors are names for object types (e.g. Year). Constants form the second group of atomic information descriptors. The string constant 'Ford G.R.' therefore is a valid information descriptor. Formally, if c is a constant, then:

$$\mathbb{D}[\![c]\!] \quad = \quad c$$

3.3. Concatenation of information descriptors

Atomic information descriptors as such are obviously rather limited. More useful information descriptors result when using concatenation. There are two ways to concatenate (atomic) information descriptors. The simplest concatenation operator for path expressions makes its appearance on the information descriptor level as a simple string concatenation:

$$\mathbb{D}[\![P_1\ P_2]\!] \quad = \quad \mathbb{D}[\![P_1]\!] \circ \mathbb{D}[\![P_2]\!]$$

Besides this basic form of concatenation, there is another form that utilises the mix-fix predicate verbalisations. This form of concatenation is able to concatenate n information descriptors, where n depends on the chosen mix-fix verbalisation.

Suppose $P_1, \ldots, P_{n-1}$ are existing information descriptors. Using a mix fix verbalisation like $w_1, \ldots, w_{n-1}$, we could concatenate these to become the information descriptor: $w_1\ [P_1]\ \ldots\ [P_{n-1}]\ w_{n-1}$. For $n = 1$ this simply leads to w_1. The resulting information descriptors can, obviously, be used in further concatenations.

For those cases where $n > 1$, the square brackets $[P_i]$ are needed to avoid ambiguities. In pure natural language, one would usually replace these brackets by commas. For example, the information descriptor we have seen before:

An administration which has as president a person who has a nr of voters [WHICH IS >= 1000000] in an election

would in natural language become:

> *An administration which has as president a person who has a nr of voters, that is larger or equal to 1000000, in an election*

Formally this class of information descriptors is introduced as:

$$\mathbb{D}[\![w_1\ [P_1]\ \ldots\ [P_{n-1}]\ w_n]\!] \quad = \bigcup_{\mathsf{MFix}([w_1,\ldots,w_n],[p_0,\ldots,p_n])} \left(p_0 \underset{0<i<n}{\circ} \mathsf{Fr}(p_i{}^{\leftarrow} \circ \mathbb{D}[\![P_i]\!]) \circ p_n{}^{\leftarrow} \right)$$

When $n = 1$ this simply becomes: $\mathbb{D}[\![w_1]\!] = p_0 \circ p_1{}^{\leftarrow}$. This latter case is a very important one. In most ORM schemas about 90% of the relationship types are binary, and for these binary relationship types we always have $n = 1$. As a result, this mechanism allows us to easily traverse a conceptual schema.

Incidently, one relatively easy, yet mechanical, way to derive partial mix-fix verbalisations from more complete ones is to define:

$$\mathsf{MFix}([w_1,\ldots,w_i,w_{i+1},\ldots,w_n],[p_0,\ldots,p_{i-1},p_i,p_{i+1},\ldots,p_n]) \ \vdash$$
$$\mathsf{MFix}([w_1,\ldots,w_i\ \mathsf{some}\ \mathsf{ONm}(\mathsf{Player}(p_i))\ w_{i+1},\ldots w_n],[p_0,\ldots,p_{i-1},p_{i+1},\ldots,p_n])$$

An example of this would be:

$$\mathsf{MFix}([\text{'chose for some person in'}],[q,r])$$

which can be derived from

$$\mathsf{MFix}([\text{'chose for'},\text{'in'}],[q,p,r])$$

as used in the earlier given list of example mix-fix verbalisations.

A crucial effect of concatenation is that it significantly reduces the number of possible interpretations. As can be seen from the definitions given above whenever a name, or a mix-fix verbalisation, has multiple path expressions as interpretation, we simply unite all possible results. Doing this is a very natural way to deal with the fact that in conceptual schemas role names, and in particular mix-fix verbalisations, are not unique. Please note, however, that this form of ambiguity is of a different magnitude than the ambiguities introduced by a full natural language.

3.4. Verbalising abstract operators

In the new version of LISA-D, information descriptors can also be combined by the use of the block, subset and superset operators from the path expression language (see appendix). The following definition shows the verbalisation of these operators:

verbalisation	operator
MISSING	$\not\forall$
ALL IN	$\Subset$
EVERY	$\forall$
AND ALSO	$\frown$
INTERSECTION	$\cap$
OR OTHERWISE	$\smile$
UNION	$\cup$
BUT NOT	$\not\cap$
MINUS	$-$

Note that the definitions of each of the above operators from the path expression language is given in the appendix.

An example of the use of one of these operators could be:

President who has EVERY Hobby

or

President who has EVERY Hobby of President: 'Johnson'

The ALL IN operator is employed in a complex query in subsection 3.6. As another example, to find the presidents that were born in California and served four years, one can formulate:

President (who was born in the State: 'California'
AND ALSO
who served the Nr of years: 4)

Functions and binary relations from the many-sorted algebra $D = \langle \mathcal{D}, F \rangle$ can be used in information descriptors. The expressions 45 + 25 and Nr < 35 are valid examples of information descriptors and can be translated straightforwardly to path expressions.

Finally, LISA-D has been extended with two *comprehension operators*, which serve as the verbalisations of the comprehension operators of the path expressions:

$$\begin{aligned} \mathbb{D}[\![v \text{ IN } P \text{ WHERE } C]\!] &= \{ v \in P \mid C \} \\ \mathbb{D}[\![P \text{ WHERE } v \text{ IN } Q]\!] &= \{ P \mid v \in Q \} \end{aligned}$$

An example of the use of the first comprehension operator can be found in the following information descriptor yielding the presidents having two hobbies:

x IN President WHERE NUMBER OF(Hobby of x) = 2

An example of the application of the second comprehension operator can be found in the information descriptor yielding all presidents who where inaugurated at an age younger than 45 years:

Person x who is president of an Administration which is inaugurated in a Year
≤ 45 + Year of birth of the President: x
WHERE x IN President

The above expression is an example of a correlation expression. The use of the second comprehension operator allows for the definition of complex correlation expressions.

3.5. Predicates

In LISA-D, predicates are treated as information descriptors as well. The basis for predicates is formed by the boolean values, which are introduced as special zero-adic operators:

expression	$\mathbb{D}[\![\text{expression}]\!]$
TRUE	$1_{\mathcal{PE}}$
FALSE	$\varnothing_{\mathcal{PE}}$

The conditional clause construction is introduced as follows:

expression	$\mathbb{D}[\![\text{expression}]\!]$
IF C THEN P ELSE Q	if $\mathbb{D}[\![C]\!]$ then $\mathbb{D}[\![P]\!]$ else $\mathbb{D}[\![Q]\!]$

The evaluation of this construct requires the evaluation of condition C, and, depending on the result of this evaluation, either P or Q is evaluated. The test whether an information descriptor has an empty result provides an illustration of the usage of the conditional clause:

SOME P IS IF P THEN TRUE ELSE FALSE
NO P IS IF P THEN FALSE ELSE TRUE

For the moment, the IS keyword is employed as an abbreviation definition mechanism. In the next subsection we introduce the macro mechanism, which allows us to properly introduce such abbreviations. Using the above operators, some further notational shorthands are defined. The traditional operations on predicates can be formed in the usual fashion using logical connectives and quantification:

expression	defined as
C_1 AND C_2	(SOME C_1) INTERSECTION (SOME C_2)
C_1 OR C_2	(SOME C_1) UNION (SOME C_2)
EXISTS P WHERE C	SOME (x IN P WHERE C)
ALL P FULFIL C	NOT SOME (x IN P WHERE NOT C)

3.6. Extending the Information Grammar

In the previous subsection, examples of the introduction of new language constructs were shown. In this section this idea is formalised by the definition of the concept of macro. The macro concept as introduced here is beyond the original LISA-D definition, which only had a simple rudimentary form of macro definitions at its disposal.

The purpose of macros is the enrichment of the retrieval language by assigning meaning to new language constructs. Each macro can be seen as a new grammar rule, extending the information language. The general format of a macro definition is:

$$\omega_0\ X_1\ \omega_1 \ldots X_n\ \omega_n\ \underline{\mathsf{IS}}\ E$$

where $X_1, \ldots, X_n$ are the names of the variables that are local to this rule. Each of the variables denotes a syntactical category that is to be instantiated by information descriptors upon application of the macro. The combination $\omega_0, \ldots, \omega_n$ forms the name of the macro. As a result of this definition, the expression

$$\omega_0\ P_1\ \omega_1 \ldots P_n\ \omega_n$$

is introduced as a valid information descriptor. To avoid ambiguity, the rule markers $\omega_0, \ldots, \omega_n$ have to satisfy certain rules. The first requirement is that rule markers which separate variables are not empty. Rule markers should also not already be defined by the lexicon. Furthermore, the rule markers $\omega_1, \ldots, \omega_n$ should not occur as starting markers of other macro definitions (one might say that the language should be LL(1), see e.g. [ASU86]). Finally, the rule markers receive the lowest priority of operators to avoid problems with the parsing of empty markers.

The evaluation of a macro call uses a fixpoint computation, as macros may be defined recursively in terms of each other. The idea of a fixpoint computation is to start from a bottom value (in our case the empty path expression) and to iterate the application of the definition of the macros involved until (hopefully!) some iteration does not produce new results. The existence of fixpoints is guaranteed if this computation has a monotonous nature. In that case, the evaluation schema yields the least fixpoint solution.

Let the i-th generation be the result of i iterations of the fixpoint computation. Formally, the semantics of an information descriptor is then defined as the union of all i-th generations:

$$\mathbb{D}[\![P]\!] = \bigcup_{i \geq 0} \mathbb{D}^i\ [\![P]\!]$$

The i-th generation $\mathbb{D}^i\ [\![P]\!]$ is inductively defined in terms of the structure of information descriptors. If P is a name from the lexicon or a constant, then simply

$$\mathbb{D}^i\ [\![P]\!] = \mathbb{D}[\![P]\!]$$

For compound operators the i-th generation is obtained by computing the associated operator, applied to the i-th generation of the arguments. For example:

$$\mathbb{D}^i\ [\![P \text{ UNION } Q]\!] = \mathbb{D}^i\ [\![P]\!] \cup \mathbb{D}^i\ [\![Q]\!]$$

The evaluation of macro call $\omega_0\ P_1\ \omega_1 \ldots P_n\ \omega_n$, with righthand side E, is defined recursively as follows:

$$\begin{aligned} \mathbb{D}^0\,[\![P]\!] &= \varnothing_{\mathcal{PE}} \\ \mathbb{D}^{i+1}\,[\![P]\!] &= \mathbb{D}^i\,[\![E\,[X_1 := P_1, \ldots, X_n := P_n]]\!] \end{aligned}$$

The expression $E\,[X_1 := P_1, \ldots, X_n := P_n]$ represents E where all formal parameters X_j are replaced by their actual counterparts P_j. Note that this substitution takes precedence over interpretation.

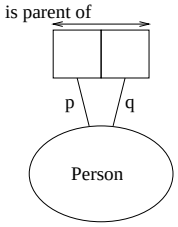

Fig. 6: Simple PSM schema

As an example of the use of recursive macros consider figure 6, which records the parental relation between persons. The ancestor relation is defined by the following macro:

Ancestor IS a Person who is parent of a Person UNION a Person who is parent of an Ancestor

This assigns the following meaning to the information descriptor Ancestor:

$$\mathbb{D}[\![\mathsf{Ancestor}]\!] = \bigcup_{i \geq 0} (p \circ q^{\leftarrow})^i$$

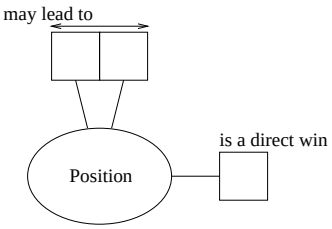

Fig. 7: Schema of a simple game

As a more complex example of the use of recursive macros, consider the set of positions in a 2-person game (see figure 7). In this domain the relationship type may lead to describes how positions can be reached from one another. The unary relationship type is a direct win gives all winning positions for the first player. The question now is to yield all positions from which the first player can win [Cha88]. This is captured by the following macro:

Winning Positions IS a Position which is a direct win UNION
a Position which may lead to a Position ALL IN
Position which may lead to Winning Positions

The above query is an example of a query that cannot be expressed as a so-called *stratified query* (see figure 8, taken from [Cha88]). Stratified queries can express all the first order queries and negation is allowed between the so-called *strata*. It has been shown however that stratified queries do not express all fixpoint queries, in particular, they have difficulty taking fixpoints over universal quantifiers, as needed in the above query (see [Kol91, Dah87]).

The macro mechanism of LISA-D allows for the specification of arbitrary fixpoint queries. There are, however, some relatively simple queries which cannot be expressed as fixpoint queries (due to lack of

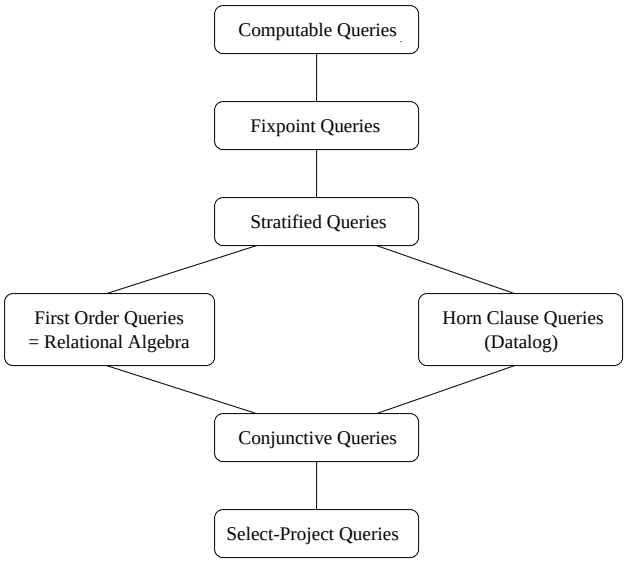

Fig. 8: Query language expressiveness hierarchy

arithmetic operations). An example is the query *Even*, which determines whether the number of instances in a certain relation is even [Cha88]. LISA-D provides the necessary basic arithmetic operators and this query can therefore be straightforwardly expressed:

EVEN R IS IF NUMBER OF(R) / 2 = 0 THEN TRUE ELSE FALSE

Whether every computable query can be expressed in LISA-D remains an issue for further research.

4. VERBALISATION AND DENOTATION OF INSTANCES

As we have seen in the earlier sections, during information analysis the information structure $\mathcal{IS}$ and its verbalisation $\mathcal{VB}$ are derived from a set of sample sentences. From these sample sentences, the verbalisation and denotation of instances can be derived as well (see subsection 2.1). The resulting verbose representation of instances can be used by the information system in user dialogues (e.g. part of LISA-D), as well as to verbosely describe the contents of a database to a search engine [RS96].

Each object type in a conceptual schema has a number of ways to uniquely identify its instances, called identification schemes. In general we will require there to be at least one such identification scheme. In practice, most object types will only have a single identification scheme. For each of these identification schemes, verbalisation rules can be formulated to denote the instances in a richer format. As an illustration, consider the address example shown in figure 3. An Address is identified by the combination of a community name, street name and house nr. As a result, one could denote an address as: Address: $\langle$Nijmegen, Toernooiveld, 1$\rangle$. This might, with some caution, be abbreviated to: Address: Nijmegen, Toernooiveld, 1. This is reasonably acceptable. It is, however, not very scalable to other situations. As an example of this weakness, consider the denotation of a relationship type instance. The marriage between president Washington and Custis would be denoted in this way as: Marriage: 'Washington G.', 'Custis M.D.', which is rather mechanical to say the least. A more human oriented verbalisation of this fact would be: the marriage between 'Washington G.' and 'Custis M.D. For a final example of a verbalisation consider Nr of Voters. Denoting an instance of this object type would e.g. lead to: Nr of Voters: 10000. A more elegant verbalisation would be to simple state: 10000 voters.

This section deals with the introduction of a scalable verbalisation mechanism that allows us to define more elegant and situation specific verbalisations of instances. The verbalisations are provided by a set of verbalisation rules. In the remainder of this section we first (subsection 4.1) discuss the underlying verbalisation mechanism in general. This is followed in subsection 4.2 by a discussion on how to utilise the verbalisation mechanism in the context of LISA-D queries.

4.1. The verbalisation mechanism

The verbalisation mechanism used here is rather simple. The basic idea is to associate a set of *meta* production rules to each object type. These rules will govern the verbalisation of its instances. Given a set of those meta rules and a set of instances, a set of concrete production rules can be derived from the meta rules. In the verbalisation mechanism used in this article, the resulting set of concrete production rules will correspond to a traditional context free grammar. However, when a more ambitious approach is required, the same principle could be applied to more advanced forms of grammars (e.g. functional grammars).

Each of the (meta) verbalisation rules is associated to an object type. Doing this allows us to derive instance specific verbalisation rules while using the structural information that is locked in the underlying information structure. For example, inheritance between types will have a great influence on this process as we will see below.

Each of the rules associated to an object type is numbered by a natural number. These numbers are used to identify the verbalisation rules. This provides us with more explicit control during the generation process of verbalisations. As such, the number does not provide any ordering; we could have chosen any domain of labels. The general format of a verbalisation rule is:

$$\langle O : x \,|\, i \rangle \rightarrow \omega_0 \; V_1 \; \omega_1 \ldots \omega_{n-1} \; V_n \; \omega_n$$

for $n \geq 0$, where each ω_j ($0 \leq j \leq n$) is a (possibly empty) string, each V_j ($1 \leq j \leq n$) an *element verbalisation* and i the rule number. The rule describes a verbalisation of instances of the object type named O. In this rule, x is used as a reference to the instance that is to be denoted. Note that typing of x is necessary as a value can be an instance of more than one object type. For example, if O is a subtype of another object type, then x is also an instance of that supertype.

Each element verbalisation yields the verbalisation of a specific property of object x. An element verbalisation for object type named O has the form:

$$\langle O : D \; x \parallel L \rangle$$

where attribute rule D is a LISA-D information descriptor and L a rule preference list. The latter list gives the preferred order in which verbalisation rules for O should be tried. The information descriptor $D \; x$ provides the property to be verbalised. This is only useful if the evaluation of $D \; x$ yields a single object related to x.

The 0-th verbalisation has a special status. It is the *main verbalisation rule*. A sophisticated verbalisation system typically features a mixture of verbalisation rules for object types, based on alternative ways to identify instances and partial verbalisations. Therefore, the main verbalisation rule usually has the format:

$$\langle O : x \,|\, 0 \rangle \rightarrow \langle O : x \parallel L \rangle$$

Each of the alternatives listed in L must be a complete verbalisation of O in line with the identification schemes of O. We usually abbreviate $\langle O : x \,|\, 0 \rangle$ to $\langle O : x \rangle$, both in the header of the rules as well as in the bodies.

Element verbalisations usually omit explicit verbalisation of typing information. For example, we would use 'J.F. Kennedy' in the presidential database example if only referring to 'J.F. Kennedy' as a person. When referring to this person as a president, this would become president 'J.F. Kennedy'. This way of treating typing information, and using it only when needed, is a convention used in natural language that keeps sentences readable. However, when we integrate the element verbalisations in the LISA-D language, special care must be taken not to introduce ambiguities this way. In some situations explicit typing must simply be added.

As an example, we consider the formula example (see figure 9). Instances of object type Formula are verbalised by the verbalisation rule of the root (morph) object type from which they originate. In addition, formulas *may* have a special name, in which case this name is preferred as the verbalisation. This leads to the following verbalisations:

$$\langle \mathsf{Formula} : x \rangle \quad \rightarrow \quad \langle \mathsf{Formula} : x \parallel 3, 2, 1 \rangle$$

$$\langle \mathsf{Formula} : x \,|\, 3 \rangle \quad \rightarrow \quad \langle \mathsf{Formula\ name} : \mathsf{is\ of}\ x \rangle$$

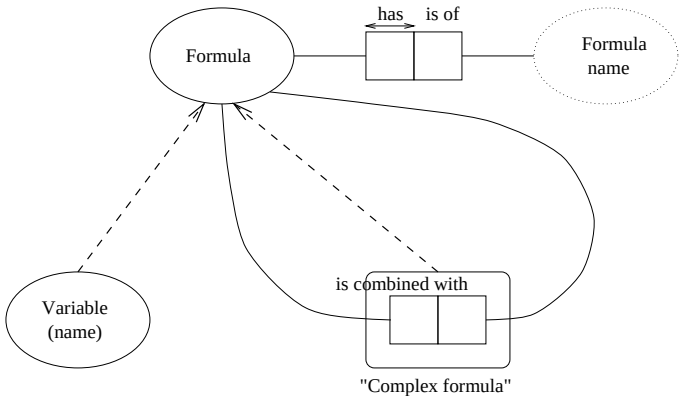

Fig. 9: Registration of formulas

$$\langle \mathsf{Formula}:x\,|\,2\rangle \quad\rightarrow\quad \langle \mathsf{Variable}:x\rangle$$
$$\langle \mathsf{Formula}:x\,|\,1\rangle \quad\rightarrow\quad \langle \mathsf{Complex\ formula}:x\rangle$$

The examples used here are complete verbalisations. Below we also discuss examples of partial verbalisations.

Instances of relationship type Complex formula are denoted as expressions of the form **(**$x \diamond y$**)**. However, the arguments x and y should not be represented by their name (formula name or variable name), but rather in terms of their construction. This is laid down in the following rule:

$$\langle \mathsf{Complex\ formula}:x\rangle \quad\rightarrow\quad \langle \mathsf{Complex\ formula}:x \parallel 2,1\rangle$$
$$\langle \mathsf{Complex\ formula}:x\,|\,2\rangle \quad\rightarrow\quad \mathbf{(}\,\langle \mathsf{Formula}:\mathsf{is\ left\ arg\ of}\ x \parallel 2\rangle\ \diamond\ \langle \mathsf{Formula}:\mathsf{is\ right\ arg\ of}\ x \parallel 2\rangle\,\mathbf{)}$$
$$\langle \mathsf{Complex\ formula}:x\,|\,1\rangle \quad\rightarrow\quad \langle \mathsf{Formula}:\mathsf{is\ left\ arg\ of}\ x\rangle\,,\langle \mathsf{Formula}:\mathsf{is\ right\ arg\ of}\ x\rangle$$

where is left arg of and is right arg of are presumed to be role participation names. Some examples of valid formulas are: u, **(**$u\ \diamond\ v$**)** and **(**$u\ \diamond$ **(**$v\ \diamond\ w$**))**, where u, v and w are variables.

4.1.1. Deriving the actual grammar

The (meta) verbalisation rules are treated as meta-rules in the sense of two-level grammars (see e.g. [WMP+76]). They are instantiated by a population of the information structure. As a result, with any population Pop, a concrete verbalisation grammar can be associated. Note that this instantiation mechanism is independent of the actual class of grammar used. In this case we are using context free grammars, but it could be applied to more advanced grammars as well.

The nonterminals of the resulting grammar are:

$$\left\{\langle O:v\,|\,i\rangle\ \middle|\ v \in \mathsf{Pop}(O) \wedge i \text{ a verbalisation rule number for } O\right\}$$

The rules of the verbalisation grammar originate from meta rule instantiation, a process consisting of two steps. Consider meta rule:

$$\langle O:x\,|\,i\rangle \rightarrow \omega_0\ \langle A_1:D_1\ x \parallel L_1\rangle\ \omega_1 \ldots \omega_{n-1}\ \langle A_n:D_n\ x \parallel L_n\rangle\ \omega_n$$

The first step instantiates element verbalisations, while the second step processes the rule preference list for each element verbalisation.

1. Let v be an instance of O, or, $v \in \mathsf{Pop}(O)$, and a_j $(1 \leq j \leq n)$ an (the) instance which is associated with v via information descriptor $D_j\ v$. Then the resulting intermediate rule is of the form:

$$\langle O:v\,|\,i\rangle \rightarrow \omega_0\ \langle\!\langle A_1:a_1 \parallel L_1\rangle\!\rangle\ \omega_1 \ldots \omega_{n-1}\ \langle\!\langle A_n:a_n \parallel L_n\rangle\!\rangle\ \omega_n$$

2. In this step the rule preference lists are processed. This list provides the order in which verbalisations are to be applied (if possible!). In case of an empty list, the element $\langle\langle A_j : a_j \parallel \rangle\rangle$ is replaced by $\langle A_j : a_j \rangle$. If the list L_j is not empty, then let r_j be the first rule number in L_j for which $\langle A_j : a_j \,|\, r_j \rangle$ appears as the left hand side of some intermediate rule. Then the element $\langle\langle A_j : a_j \parallel L_j \rangle\rangle$ is replaced by $\langle A_j : a_j \,|\, r_j \rangle$.

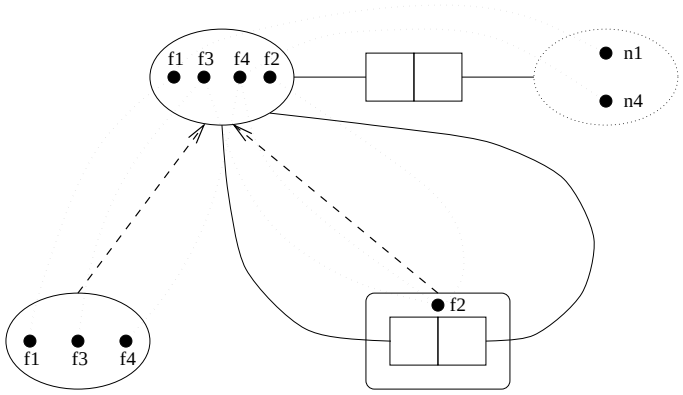

Fig. 10: Sample population

This two step construction mechanism is illustrated by showing its effect for the formula example, when populated as in figure 10. The formula instance f_1 has an associated formula name n_1, whereas formula instance f_2 has no such name. Note that the rule instantiation mechanism also looks after the generation of the proper concrete verbalisation rules for instances of polymorph types. The rule instantiation step results in:

$$
\begin{aligned}
\langle \textsf{Formula} : f_1 \,|\, 3 \rangle &\rightarrow \langle\langle \textsf{Formula name} : n_1 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_1 \,|\, 2 \rangle &\rightarrow \langle\langle \textsf{Variable} : f_1 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_1 \rangle &\rightarrow \langle\langle \textsf{Formula} : f_1 \parallel 3, 2, 1 \rangle\rangle \\
\\
\langle \textsf{Formula} : f_2 \,|\, 1 \rangle &\rightarrow \langle\langle \textsf{Complex formula} : f_2 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_2 \rangle &\rightarrow \langle\langle \textsf{Formula} : f_2 \parallel 3, 2, 1 \rangle\rangle \\
\\
\langle \textsf{Complex formula} : f_2 \,|\, 2 \rangle &\rightarrow \textbf{(} \langle\langle \textsf{Formula} : f_3 \parallel 2 \rangle\rangle \;\diamond\; \langle\langle \textsf{Formula} : f_4 \parallel 2 \rangle\rangle \textbf{)} \\
\\
\langle \textsf{Complex formula} : f_2 \,|\, 1 \rangle &\rightarrow \langle\langle \textsf{Formula} : f_3 \parallel 0 \rangle\rangle \,, \langle\langle \textsf{Formula} : f_4 \parallel 0 \rangle\rangle \\
\langle \textsf{Complex formula} : f_2 \rangle &\rightarrow \langle\langle \textsf{Complex formula} : f_2 \parallel 2, 1 \rangle\rangle \\
\\
\langle \textsf{Formula} : f_3 \,|\, 2 \rangle &\rightarrow \langle\langle \textsf{Variable} : f_3 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_3 \rangle &\rightarrow \langle\langle \textsf{Formula} : f_3 \parallel 3, 2, 1 \rangle\rangle \\
\\
\langle \textsf{Formula} : f_4 \,|\, 3 \rangle &\rightarrow \langle\langle \textsf{Formula name} : n_4 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_4 \,|\, 2 \rangle &\rightarrow \langle\langle \textsf{Variable} : f_4 \parallel 0 \rangle\rangle \\
\langle \textsf{Formula} : f_4 \rangle &\rightarrow \langle\langle \textsf{Formula} : f_4 \parallel 3, 2, 1 \rangle\rangle
\end{aligned}
$$

In the second step, the rule preference lists are processed. This leads to:

$$
\begin{aligned}
\langle \textsf{Formula} : f_1 \,|\, 3 \rangle &\rightarrow \langle \textsf{Formula name} : n_1 \rangle \\
\langle \textsf{Formula} : f_1 \,|\, 2 \rangle &\rightarrow \langle \textsf{Variable} : f_1 \rangle \\
\langle \textsf{Formula} : f_1 \rangle &\rightarrow \langle \textsf{Formula} : f_1 \,|\, 3 \rangle
\end{aligned}
$$

$$
\begin{array}{rcl}
\langle \mathsf{Formula}: f_2 \,|\, 1\rangle & \rightarrow & \langle \mathsf{Complex\ formula}: f_2\rangle \\
\langle \mathsf{Formula}: f_2 \,|\, 0\rangle & \rightarrow & \langle \mathsf{Formula}: f_2\rangle \\
\langle \mathsf{Complex\ formula}: f_2\rangle & \rightarrow & \langle \mathsf{Complex\ formula}: f_2 \,|\, 2\rangle \\
\langle \mathsf{Complex\ formula}: f_2 \,|\, 2\rangle & \rightarrow & \mathbf{(}\, \langle \mathsf{Formula}: f_3 \,|\, 2\rangle \ \diamond\ \langle \mathsf{Formula}: f_4 \,|\, 2\rangle \,\mathbf{)} \\
\\
\langle \mathsf{Complex\ formula}: f_2 \,|\, 1\rangle & \rightarrow & \langle \mathsf{Formula}: f_3\rangle\, , \langle \mathsf{Formula}: f_4\rangle \\
\langle \mathsf{Formula}: f_2\rangle & \rightarrow & \langle \mathsf{Formula}: f_2 \,|\, 1\rangle \\
\\
\langle \mathsf{Formula}: f_3 \,|\, 2\rangle & \rightarrow & \langle \mathsf{Variable}: f_3\rangle \\
\langle \mathsf{Formula}: f_3\rangle & \rightarrow & \langle \mathsf{Formula}: f_3 \,|\, 2\rangle \\
\\
\langle \mathsf{Formula}: f_4 \,|\, 3\rangle & \rightarrow & \langle \mathsf{Formula\ name}: n_4\rangle \\
\langle \mathsf{Formula}: f_4 \,|\, 2\rangle & \rightarrow & \langle \mathsf{Variable}: f_4\rangle \\
\langle \mathsf{Formula}: f_4\rangle & \rightarrow & \langle \mathsf{Formula}: f_4 \,|\, 3\rangle
\end{array}
$$

As this grammar is only invoked from the symbols $\langle O : x\rangle$ for all object types O and $x \in \mathsf{Pop}(O)$, some rules of this grammar might be superfluous in that they will never be invoked.

4.1.2. Example verbalisation rules

Now we have discussed the general principle, we can discuss some specific examples in more detail. We do this briefly for every class of object types.

Value types

For value types, it is sometimes useful to deviate from the standard rule, for example when dimensions are to be involved. Therefore, we could for example add the following rule for a value type Length.

$$\langle \mathsf{Length}: x \,|\, 2\rangle \ \rightarrow\ \langle \mathsf{Length}: x \parallel 1\rangle\ \mathsf{yards}$$

Entity types

For the example given in figure 3, we have the following default verbalisation:

$$
\begin{array}{rcl}
\langle \mathsf{Community}: x\rangle & \rightarrow & \langle \mathsf{Community\ name}: \mathsf{Community\ name\ of\ Community}\ x\rangle \\
\langle \mathsf{Street}: x\rangle & \rightarrow & \langle \mathsf{Street\ name}: \mathsf{Street\ name\ of\ Street}\ x\rangle\, , \langle \mathsf{Community}: \mathsf{Community\ contains\ Street}\ x\rangle \\
\langle \mathsf{Address}: x\rangle & \rightarrow & \langle \mathsf{House\ Nr}: \mathsf{House\ Nr\ of\ Address}\ x\rangle\, \langle \mathsf{Street}: \mathsf{Street\ of\ Address}\ x\rangle
\end{array}
$$

This leads to a denotation of addresses in the following format:

1 Toernooiveld, Nijmegen

In some countries, this is not the accepted style to write addresses. For example, in some countries the preferred style to write this address is:

Toernooiveld 1, Nijmegen

What makes this case special is that while a street is identified by a street name (Toernooiveld) and the community name (Nijmegen), the house number needs to be placed in between. To obtain this kind of verbalisation we need to introducing two partial verbalisation rules for Street.

$$
\begin{array}{rcl}
\langle \mathsf{Street}: x \,|\, 2\rangle & \rightarrow & \langle \mathsf{Street\ name}: \mathsf{Street\ name\ of\ Street}\ x\rangle \\
\langle \mathsf{Street}: x \,|\, 3\rangle & \rightarrow & \langle \mathsf{Community}: \mathsf{Community\ contains\ Street}\ x\rangle
\end{array}
$$

If we then change the complete verbalisations of Address to:

$$\begin{aligned}
\langle \mathsf{Address}:x\rangle &\rightarrow \langle \mathsf{Street}:x \parallel 2,1\rangle\\
\langle \mathsf{Address}:x \,|\, 2\rangle &\rightarrow \langle \mathsf{Street}:\mathsf{Street\ contains\ Address}\ x \parallel 2\rangle\ \langle \mathsf{House\ nr}:\mathsf{House\ nr\ of\ Address}\ x\rangle\,,\\
&\quad\ \langle \mathsf{Street}:\mathsf{Street\ contains\ Address}\ x \parallel 3\rangle\\
\langle \mathsf{Address}:x \,|\, 2\rangle &\rightarrow \langle \mathsf{House\ nr}:\mathsf{House\ nr\ of\ Address}\ x\rangle\ \langle \mathsf{Street}:\mathsf{Street\ of\ Address}\ x\rangle
\end{aligned}$$

the desired verbalisation style is obtained.

Relationship types

Verbalisation of relations can benefit a lot from our verbalisation mechanism. The earlier used example of the marriage between 'Washington G.' and 'Custis M.D.' as opposed to the marriage: 'Washington G.', 'Custis M.D.', is an illustrative example.

In particular for relationship types it is also useful to introduce partial verbalisations. A partial verbalisation, as its name suggests, only verbalises part of a fact instance. For example, the participation of a person in an election, irrespective of the received nr of votes, may be verbalised as:

$$\begin{aligned}
\langle \mathsf{Election\ results}:x \,|\, 2\rangle \rightarrow\ & \mathsf{the\ participation\ of}\ \langle \mathsf{Person}:\mathsf{has\ Election\ results}\ x\rangle\\
& \mathsf{in}\ \langle \mathsf{Election}:\mathsf{with\ Election\ results}\ x\rangle
\end{aligned}$$

Another typical example is the verbalisation of bridge types in the case of simple identifications. For example, the bridge type Registration that relates a Student to his/her Student number. The verbalisation of this relationship type will mention the student and the student number. As, however, students are denoted by their student number, this verbalisation would verbalise the same instance twice leading to: the registration of student 862424 with student nr 862424. A more natural verbalisation would be the registration of student 862424, which can be obtained by the additional rule:

$$\langle \mathsf{Registration}:x \,|\, 2\rangle \rightarrow \mathsf{the\ registration\ of\ student}\ \langle \mathsf{Student\ number}:\mathsf{Student\ number\ of\ Registration}\ x\rangle$$

Although this latter verbalisation is a partial verbalisation, one could argue that it is complete from an identification point of view. The reason is that instances of Registration are necessarily functional dependent on the student number. This implies that each instance of Registration can be uniquely identified by a student number. In particular for bridge types it therefore makes sense to allow these semi partial verbalisations as complete verbalisations as they prevent redundant verbalisations.

Inheritance hierarchy

Inheritance of verbalisations really illustrates the advantages of using meta verbalisation rules. Depending on the actual type of an instance, the (preferred) verbalisation will be adopted based on its inherited verbalisation.

In the inheritance hierarchy, the default is that verbalisations are inherited downward. However, the inherited verbalisations may be overriden. In case of the running example, the verbalisation of instances of President differs from the verbalisation of instances of Person. The following example shows how this can be handled.

$$\begin{aligned}
\langle \mathsf{Person}:x\rangle &\rightarrow \langle \mathsf{Politician}:x\rangle\\
\langle \mathsf{Politician}:x\rangle &\rightarrow \langle \mathsf{Politician}:x \parallel 2,1\rangle\\
\langle \mathsf{Politician}:x \,|\, 2\rangle &\rightarrow \langle \mathsf{President}:x\rangle\\
\langle \mathsf{Politician}:x \,|\, 1\rangle &\rightarrow \mathsf{politician}\ \langle \mathsf{Person}:x\rangle\\
\langle \mathsf{President}:x\rangle &\rightarrow \mathsf{president}\ \langle \mathsf{Person}:x\rangle
\end{aligned}$$

Complex types

So far, the verbalisation of special object types (such as collection types and schema types) has not been discussed explicitly. If the verbalisation of a collection type or schema type does not employ the underlying element type(s), there are no special problems. If it does, however, a special notation needs to be introduced to indicate that the information descriptor $D\,x$ may yield more than one instance. Consider for example the following possible verbalisation rule for the object type Convoy of the schema in figure 11:

$$\langle \mathsf{Convoy}:x\rangle \;\rightarrow\; \text{the convoy consisting of } \langle \mathsf{Ship}:\mathsf{IN}\ x\rangle^{+}$$

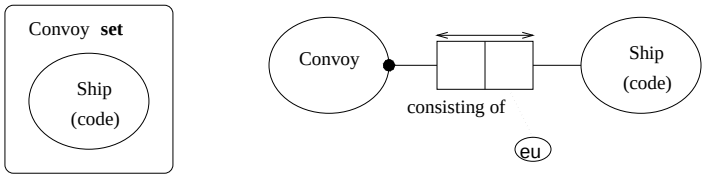

Fig. 11: Example collection type

In this case Ship IN x may yield several instances for a concrete instantiation of x, hence the plus. To capture this case, the instantiation of the element verbalisation is treated differently. In general, a verbalisation of the form $\langle O:D\,x \parallel L\rangle^{+}$ yields:

$$\langle\!\langle O:a_1 \parallel L\rangle\!\rangle\,,\langle\!\langle O:a_2 \parallel L\rangle\!\rangle\,,\ldots \text{and}\ \langle\!\langle O:a_m \parallel L\rangle\!\rangle$$

if $a_1,\ldots,a_m$ ($m>1$) result from the evaluation of $D\,v$ for some $v\in \mathsf{Pop}(O)$. In case $m=1$, the element verbalisation yields:

$$\langle\!\langle O:a_1 \parallel L\rangle\!\rangle$$

Note that the rules for populating collection types exclude the possibility $m=0$ (see [HPW93]). Of course, the verbalisation of power types may employ some external identification. As an example, suppose that convoys *do* have convoy codes:

$$\langle \mathsf{Convoy}:x\,|\,2\rangle \;\rightarrow\; \text{the convoy } \langle \mathsf{Convoy\ code}:\text{is name of } x\rangle$$

4.2. Object denotations

Verbalisation rules specify how object instances are represented in the universe of discourse. As this is the preferred way to denote instances as used by the parties involved in the universe of discourse, it makes sense to ensure that in the communication with the information system these denotation styles can be used as well.

To use the verbalisation rules in the context of LISA-D, a different treatment of the rules is required. For the verbalisation of instances, a *backward chaining* approach was followed in that the verbalisation rules allowed us to re-write the instances of abstract types (entity types, collection types, etc.) in terms of instances of other object types, and eventually value type instances. For LISA-D, this process needs to be reversed. What is needed is a *forward chaining* approach. In a LISA-D information descriptor, instances of abstract types are denoted using values from the underlying value types. In processing an information descriptor the actual abstract instance is required. Therefore, a mechanism is required that derives the abstract instance from the concrete value type instances. Such a mechanism should yield a path expression that performs this forward chaining task.

For object denotations in LISA-D, we introduce an additional syntactic category. The bulk of LISA-D's definition was centered around the syntactic category *information descriptor*, but for the object denotations an extra category is required. This is the *object denotation* category, and its semantics (in terms of path expressions) will be provided by the function:

$$\mathbb{O} : \texttt{ObjectDenotation} \times \mathbb{N} \times \mathcal{OB} \rightarrow \mathcal{PE}$$

For each of the verbalisation rules of the form:

$$\langle \mathsf{ONm}(o) : x \,|\, i \rangle \rightarrow \omega_0 \, \langle \mathsf{ONm}(o_1) : D_1 \; x \parallel L_1 \rangle \, \omega_1 \ldots \omega_{n-1} \, \langle \mathsf{ONm}(o_n) : D_n \; x \parallel L_n \rangle \, \omega_n$$

we can define:

$$\mathbb{O}[\![\omega_0 \; [E_1] \; \omega_1 \ldots \omega_{n-1} \; [E_n] \; \omega_n]\!] \, (i, o) \quad = \quad \bigcap_{j=1}^{n} \mathsf{Fr}(\mathbb{D}[\![D_j]\!]^{\leftarrow} \circ \mathbb{O}[\![E_j]\!] \, (m_j, o_j))$$

where $E_1, \ldots, E_n$ are existing object denotations, and m_j is the first element from L_j such that a verbalisation rule labeled m_j for object type o_j indeed exists.

The square brackets in the object denotation are only required when their absence would introduce ambiguities, otherwise they can be omitted. The actual integration of object denotations into the category of information descriptors is done as follows. Let E be an object denotation, o an object type, such that $\mathbb{O}[\![E]\!] \, (0, o)$ is defined, then we have the following information descriptor:

$$\mathbb{D}[\![\mathsf{TQ}(o) : E]\!] \;=\; \mathbb{O}[\![E]\!] \, (i, o)$$

where $\mathsf{TQ}(o) = \mathsf{Article}(o) \;\; \mathsf{ONm}(o)$ provides the *type qualification* for object denotation E.

Where this does not lead to ambiguity, the type qualification can be reduced. For example, in the case of the marriage between 'Washington G.' and 'Custis M.D.', no type qualification is needed. On the other hand, omission of typing information is not a luxury that can be afforded in the formula example:

Formula: u
Formula: **(** $u \;\; \diamond \;\; v$ **)**
Formula: **(** $u \;\; \diamond \;\;$ **(** $v \;\; \diamond \;\; w$ **))**

5. QUERY VERBALISATION

In the next section, we discuss a sample query tool that should support users with the formulation of queries. The intention of this tool is to provide the user with a guidance system to interactively compose queries. Simple queries correspond to linear paths through the information structure, while complex queries result from applying operators to other queries (see subsection 3.4).

The proposed query tool needs the ability to verbalise a given linear path expression and yield a LISA-D information descriptor. The reason for this requirement is two-fold. Firstly, one of the key features of the envisioned query tool is to let users navigate through an information structure, and while doing so they build a linear path. This resulting path should eventually be presented to the user in the form of a (readable) LISA-D information descriptor. The second reason to require a verbalisation mechanism for paths is that if we build a natural language query tool that takes a query in natural language and then tries to interpret this, the 'interpretation' will lead to a number of possible path expressions. The resulting path expressions have to be shown to the users for perusal such that they can select the proper interpretation. Key to this is the verbalisation of these path expressions.

In this section we do not discuss the verbalisation of all complex operators that could be used in a path expression. The complex operations themselves usually have a direct translation, and do not pose a real challenge for verbalisation. More challenging is the verbalisation of linear path expressions, and verbalisations in the context of mix fix predicate verbalisations and instance verbalisations. In this section, we therefore focus on these aspects of path verbalisation.

The verbalisation function we seek should take a path expression P and find an information descriptor D such that $\mathbb{D}[\![D]\!]$ is equivalent to P. So in essence we are trying to define an 'inverse' of $\mathbb{D}$. For one P there are, however, usually multiple D's that would meet our needs. Therefore, we define a scoring system that expresses which verbalisation is preferred over another by using a demerit system. We define the verbalisation function in terms of a set of derivation rules that derive the verbalisation alternatives, together with associated penalty points. The predicate $\rho(P, n, x)$ is used to express the fact that path expression P can be verbalised as n, with penalty x. The verbalisation cost of a linear path is then defined as:

$$\mathsf{Cost}(P) \quad = \quad \mathsf{min} \left\{ \alpha \;\middle|\; \exists_n \left[\rho(P, n, \alpha) \right] \right\}$$

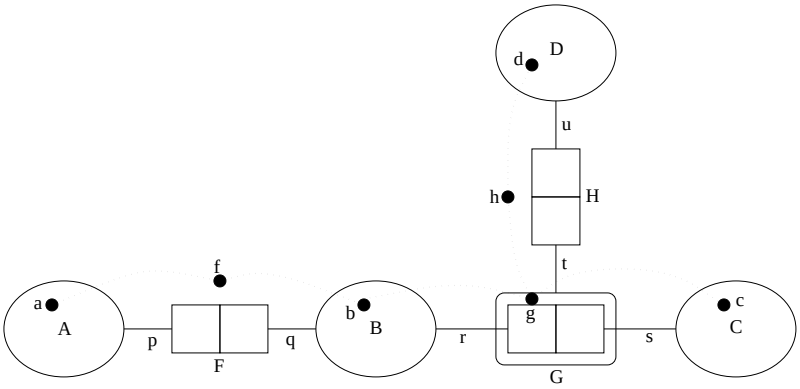

Fig. 12: A minuscule fragment of the presidential database

The chosen verbalisation $\rho(P)$ of path P should be a candidate with the least demerit points, so $\rho(P, \rho(P), \mathsf{Cost}(P))$.

As a running example in this section, we consider a small (abstract) fragment of the presidential database (see figure 12). Object types A, B, C and D correspond to Hobby, President, Person and Nr of children, while fact types F, G and H correspond to Recreation, Marriage and Offspring respectively. Furthermore, the fact participation name of role r is is partner in.

5.1. Simplification of verbalisations

The first verbalisation rule handles the simplification of paths:

[V1] (*simplification rule*) If path P_1 can be simplified to P_2, then:

$$\rho(P_2, n, \alpha) \vdash \rho(P_1, n, \alpha)$$

where path P_2 is simpler than P_1 if they have the same semantics, and P_2 is smaller than P_1 in terms of the number of operators. For example:

$$\begin{aligned} \rho(A \circ a \circ p \circ q^{\leftarrow} \circ B \circ b, n, \alpha) &\vdash \rho(A \circ a \circ p \circ \mathsf{Rel}(p) \circ f \circ q^{\leftarrow} \circ B \circ b, n, \alpha) \\ \rho(\mathsf{Rel}(p) \circ f, n, \alpha) &\vdash \rho(A \circ a \circ p \circ \mathsf{Rel}(p) \circ f \circ q^{\leftarrow} \circ B \circ b, n, \alpha) \end{aligned}$$

5.2. Elementary verbalisation

An empty path expression is only used as a starting point in the navigation tool. Therefore we verbalise it as follows:

[V2] (*empty path expression*) $\vdash \rho(\epsilon, \mathsf{Start}, 0)$

The verbalisation of typed instances is governed by the verbalisation rules for instance denotation as discussed in subsection 4.1.1. Let $\lambda(\langle \mathsf{ONm}(o) : x \rangle)$ be the verbalisation of instance x of object type o as follows from the grammar introduced in subsection 4.1.1. Now we can define the following verbalisation rule:

[V3] (*instantiation rule*) $\vdash \rho(o \circ x, \mathsf{TQ}(o)\ \lambda(\langle \mathsf{ONm}(O) : x \rangle), 0)$

where $\mathsf{TQ}(o)$ provides the type qualification for object type o if needed to prevent ambiguities. Note that there is no derivation rule for the verbalisation of isolated constants, as a constant may belong to several label types. Of course, abstract objects have no direct verbalisation, as such objects are verbalised in terms of associated concrete objects. As an example of the instantiation rule, we consider the example of figure 12. In this example, we assume each object type to be identified by a label type with an underlying domain of strings. Then we have:

$$\begin{aligned} &\rho(A \circ a, \mathsf{hobby}\ \text{'fishing'}, 0) \\ &\rho(B \circ b, \mathsf{president}\ \text{'Washington G.'}, 0) \end{aligned}$$

if a and b have the names 'fishing' and 'Washington G.' respectively.

5.3. Verbalisation of information structure components

Naming of information structure components enters the verbalisation system via the lexicon. Some names in the lexicon have been introduced as generic names. These generic names are used as a default verbalisation, but preference goes to the more specific names as specified by the user. Let therefore $g(n)$ have value 2 for all generic names n, and value 1 otherwise. The involvement of the lexicon, and the preference for non-generic names is then expressed by the following rule:

[V4] (*lexicographic verbalisation*) $\mathsf{Lexicon}(n, P) \vdash \rho(P, n, g(n))$

Some examples are:

$\rho(A, \text{a hobby}, 1)$ $\quad$ $\rho(B, \text{a president}, 1)$
$\rho(p, \text{is}, 1)$ $\quad$ $\rho(p, \text{INVOLVED IN}, 2)$
$\rho(q^{\leftarrow}, \text{of}, 1)$ $\quad$ $\rho(q^{\leftarrow}, \text{OF}, 1)$

5.4. Composing verbalisations

The main rule for composing verbalisations is juxtaposition of already verbalised parts. We take for each concatenation a penalty of 2 point into account:

[V5] (*concatenation of verbalisations*)

$$\rho(P_1, m_1, \alpha_1) \wedge \rho(P_2, m_2, \alpha_2) \vdash \rho(P_1 \circ P_2, m_1\ m_2, \alpha_1 + \alpha_2 + 2)$$

Mix-fix verbalisations also lead to concatenation of existing verbalisations.

[V6] (*mix-fix verbalisation*)

$$\mathsf{MFix}([w_1, \ldots, w_n], [p_0, \ldots, p_n]) \wedge \rho(P_1, m_1, \alpha_1) \wedge \ldots \wedge \rho(P_{n-1}, m_{n-1}, \alpha_{n-1}) \vdash$$
$$\rho(p_0 \underset{0<i<n}{\circ} \mathsf{Fr}(p_i{}^{\leftarrow} \circ P_i) \circ p_n{}^{\leftarrow}, w_1\ [m_1]\ \ldots\ [m_{n-1}]\ w_n\ , \alpha_1 + \ldots \alpha_{n-1})$$

The following example shows some verbalisation variants. The first results from verbalising each element, the second results from simplification to $A \circ p \circ q^{\leftarrow} \circ B$, and the use of a mix-fix predicate verbalisation. The third variant by using generic names for roles.

$\rho(A \circ p \circ f \circ q^{\leftarrow} \circ B, \text{a hobby which is a recreation pursued by a president}, 6)$
$\rho(A \circ p \circ f \circ q^{\leftarrow} \circ B, \text{a hobby of a president}, 1)$
$\rho(A \circ p \circ f \circ q^{\leftarrow} \circ B, \text{a person INVOLVED IN a recreation OF a hobby}, 8)$

The preferred verbalisation is therefore a hobby of a president.

6. COMPUTER SUPPORTED QUERY FORMULATION

This section discusses a case study in which the benefits of verbalisation for computer supported query formulation (CSQF) are highlighted. A more elaborated discussion of the underlying ideas can be found in [HPW96].

We start of with a discussion of query by navigation. This is followed by a discussion on how a natural language query tool could be used to provide a flexible query tool. The section concludes with a brief discussion of an integrated query workbench that combines the different query formulation styles using a syntax directed editor.

6.1. Query by navigation

From an end user's point of view, conceptual query languages, and natural language query approaches for that matter, are particularly useful for ad-hoc queries. Database administrators may have pre-defined some standard queries but a large number of queries still need to be formulated on an ad-hoc base, and preferably by the end users themselves.

End users, however, usually do not have a good overview of the information stored. In particular in the context of evolving information systems [HPW95], or federations of information systems, this problem

comes even more to the fore when the structure of the conceptual schema changes regularly. The result is that irrespective of the actual query language used, be it a SQL, a conceptual query language, or a natural language, users still have to deal with the fact that they *don't know what's out there*. Users may start to feel *lost in conceptual space*. This problem of feeling *lost in conceptual space*, was found to be similar to the problem of *lost in hyperspace* as it can be encountered in the world of information retrieval. In the context of information retrieval, this problem is approached by using query by navigation [GGP89, Luc90, CD90, ACG91, BW92]. The query by navigation interaction mechanism between a searcher and the system is well-known from the Information Retrieval field, and has proven to be useful. This has triggered us in applying this principle to query formulation on information systems [BPW94, HPW96, HPW95]. The effectiveness of query by navigation has been proven using empirical testing in [BBB91].

In line with the above discussion, a query formulation process (both for an information retrieval system, and an information system) can be said to roughly consist of four phases: an *explorative phase*, a *constructive phase*, a *feedback phase*, and a *presentation phase*. A more detailed discussion of these phases, and their inter-relationships can be found in [HPW96].

A query by navigation system for query formulation in information systems is aimed at the *explorative phase*. The reason to briefly discuss query by navigation in this article is that it provides a very natural application of the verbalisations provided with a conceptual schema. In a query by navigation session, a user essentially builds a linear path through the conceptual schema. This path is verbalised using the verbalisations provided with the conceptual schema. As these verbalisations are directly based on samples taken from the universe of discourse, the resulting paths are verbalised in a way that more directly relates to the user.

To illustrate the elegance of query by navigation, we offer the reader a 'test drive' with such a system. The example is in the context of the presidential database example, but let us presume the user of the system is not really aware of the information stored in the system. The first node shown to the user is depicted in figure 13. It simply lists all object types in the conceptual schema. Let us presume the user is interested in presidents who are married to someone, and the number of children that resulted from these marriages. In the starting node, the user may select 'president' as the first refinement of the information need. This leads to the example node as presented in figure 14. The associated node shows the direct environment of object type President. The set of possible refinements of the current focus is built as follows. For each n-ary relationship type in which the current focus (president) plays a role, we have $n-1$ possible refinements since there are $n-1$ possible ways to continue the path *through* this relationship type. The associative links are now derived from the subtyping hierarchy in the conceptual schema. In our example ORM schema these are the supertypes of the object type President, being Politician and Person. An obvious refinement of this interface would be to link it to an organisation specific dictionary in which terms like President and Politician are defined (possibly organised as a hypermedia document).

Let us presume the searcher selects the president who is involved in a marriage as the next focus. This action leads to the node depicted in figure 15. This node shows a second class of associative links. Besides associative links resulting from subtyping, we also distinguish associative links to the reversed formulation of the current focus, i.c. the marriage of a president. When navigating through the hyperindex, refinements to the current focus will take place on the tail of the current focus. By reversing the current focus, refinements can be made on the front as well.

The user decides to select the refinement with as spouse the person, leading to figure 16. The user considers this, for the moment, to be a proper description of the information need. To get an impression of the query result so far, the user selects the beam down option. This results in the node depicted in figure 17. This node is neither part of the hyperbase, nor is it part of the hyperindex. It is an ad-hoc node representing the result of the focus of figure 16 interpreted as a query on the underlying database. The user can now select an instance for further navigation, which will then indeed take place in the hyperbase.

6.2. Natural language query interface

A language like LISA-D is better suited for non expert users to specify their own queries, then a language like SQL. However, due to its unavoidable formality, LISA-D as such may still be regarded as too limiting for end users. Therefore, it makes sense as a next step to develop a (more) natural language front end for LISA-D. Given a query in a full natural language, this query could be interpreted as a path expression

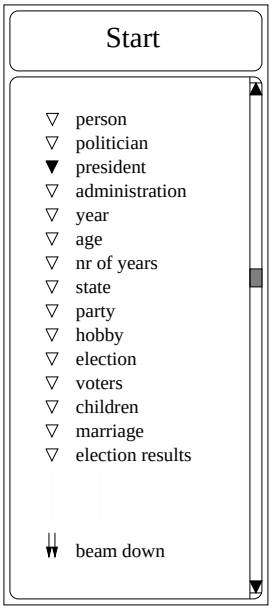

Fig. 13: The starting node of the hyperindex

using the verbalisations specified with the conceptual schema. This obviously requires more linguistic information than we currently specify at the moment [DR91].

In figure 18 an example natural language query is given. This query is ambiguous in that it is not clear to which person the who is the vice president of an administration refers to. The system would therefore be able to interpret this query in at least two ways. By translating each of these interpretations to a path expression, and re-verbalising these path expressions, we can build a feed back system that enables the system to have a natural dialogue with users. Figure 18 also shows the different interpretations.

This raises of course the question: *how realistic is this*? What is certainly possible with the limited verbalisation information we use in this article is to provide a mapping between natural language expressions expressing *linear* paths in a conceptual schema and a (partial) query. For example, a query like:

In which state was president 'Clinton' born?

Using existing techniques it should be possible to re-phrase this in the more structured format:

In which *state* was-born *president* 'Clinton'

which highlights those parts of the expression that correspond to nouns (hinting at object types). This expression can be interpreted as a small conceptual graph, which can be matched against the conceptual schema; which is essentially a conceptual graph as well.

In this process, the system can derive that the closest match according to the information available is:

State which is birthstate of President: 'Clinton'

6.3. Query by construction

The query by navigation interface on its own, does not allow users to build queries with all kinds of complex operators. This is not a flaw of query by navigation. The aim of query by navigation was to offer users a tool to explore the information stored in the information system, not to provide them with a complete query formulation tool.

Also a natural language query interface does not provide a means to formulate all possible queries in a convenient way. A natural language query tool will allow for the specification of a larger set of queries than a query by navigation tool does, but it does not support the formulation of arbitrarily complicated queries. One

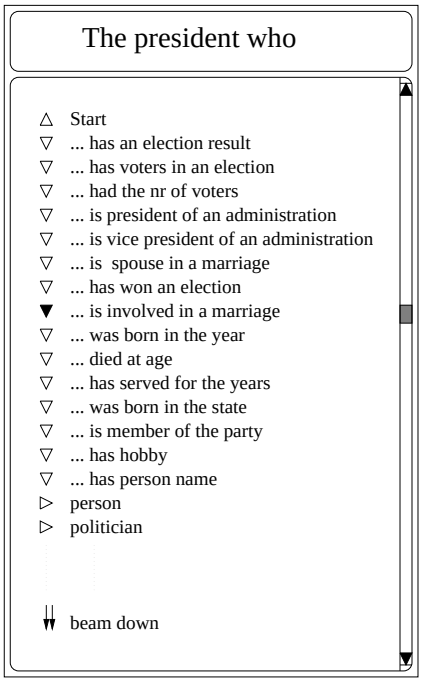

Fig. 14: The quest for a president who is married with a politician

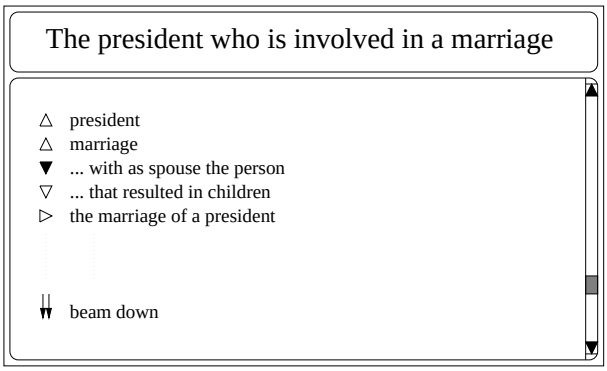

Fig. 15: Focus on marriage

could indeed specify a complex fix-point query involving different complex operations in natural language, but this would be rather tedious to do, and is bound to lead to a large number of ambiguities. Using natural language query formulation is suited for path expressions that do not contain too many complex operations.

To support the formulation of arbitrarily complex queries, we therefore need a tool that basically provides a syntax directed editor. An example of this is shown in figure 19. This *Query by Construction* tool basically provides a query workbench. Query 'snippets' that either result from a query by navigation session, or natural language query, can be dragged onto the query workbench where they can be combined into complex queries using an array of operations. The power of such a query workbench is obviously the combination of tools. The combination of tools should be rich enough to accommodate users with different levels of expertise. Finally, the roles that each of the query formulation tools discussed in this section could play in a query workbench, can be characterised by the following phrases:

Query by navigation I don't know what I'm looking for, but I'll know when I find it.

Natural language query I know roughly what I want, but I don't know how to say it formally.

Query by construction Now that I have all the bits and pieces, I want to formulate the proper complex

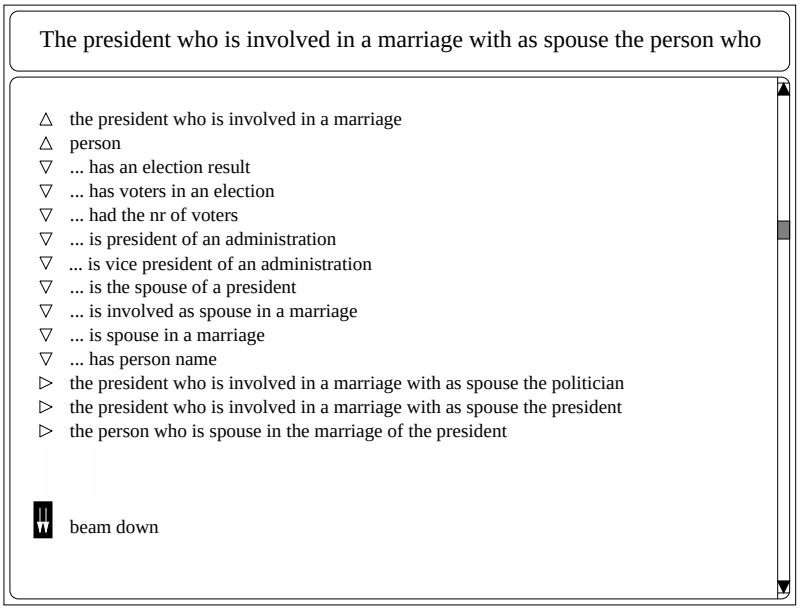

Fig. 16: Preliminary result in the hyperindex

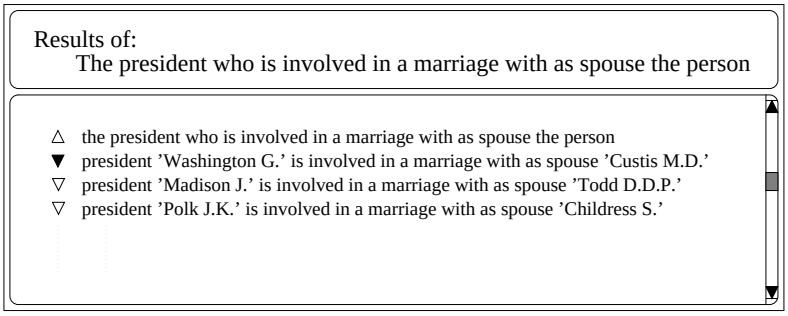

Fig. 17: Preliminary result of the query

query.

What remains to be done with regards to the sketched querying mechanism is empirical testing to verify that it is indeed an improvement over traditional query mechanisms. The effectiveness of query by navigation in the context of information retrieval [BBB91] gives us the hope that this mechanism will work well in the context of conceptual query formulation. Furthermore, the producers of the InfoModeler CASE Tool [Asy94] have recently launched a product featuring similar underlying principles.

7. CONCLUSIONS

In this article we have discussed four ways to exploit fact verbalisations. We did so in the context of a formal framework capturing our (modest) repository of verbalisation information. We have shown how an information system can communicate with users in a language that more closely resembles the language of the users. Query results can be presented in user-friendly formats and data can be conveniently presented to the system, which also facilitates the discovery of databases by search engines on the internet. Furthermore, the grammatical foundation provides a basis for sophisticated query formulation support, as outlined in section 6.

The verbalisation mechanisms described in this paper are very liberal. Subtypes may be given different verbalisations as their supertypes, complex types may be verbalised not only using their structure, but also

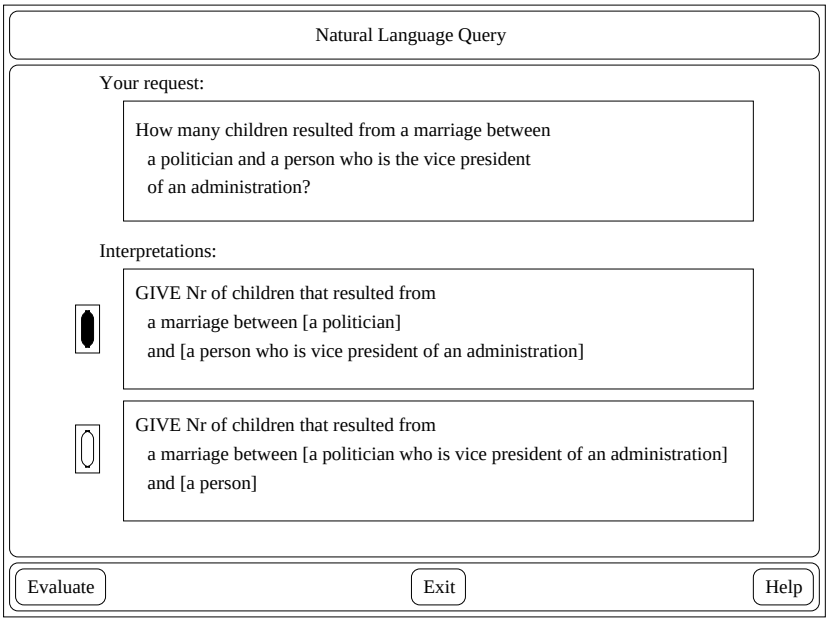

Fig. 18: Resolving ambiguities by re-verbalising interpretations

using external identifications, and verbalisations may employ context dependencies and be of a partial nature. In addition to that, a powerful mechanism has been presented for the specification of derived information: macros. This mechanism can also be straightforwardly supported by the described query formulation support system. Although emphasis has been on Object-Role Modelling, the results are sufficiently general to be applicable to other data modelling techniques.

Focus in this paper has been on (basic) grammatical aspects of conceptual data modelling, where we did not yet dare venture into the area of linguistics. The latter step will be part of future research in attempt to further improve the naturalness of LISA-D and instance verbalisations.

Acknowledgements — We would like to thank the anonymous referees for their comments and suggestions, which have lead to improvements of the original article.

REFERENCES

[ACG91] M. Agosti, R. Colotti, and G. Gradenigo. A two-level hypertext retrieval model for legal data. In A. Bookstein, Y. Chiaramella, G. Salton, and V.V. Raghavan, editors, *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 316–325, Chicago, Illinois, October 1991. ACM Press.

[ANS95] ANSI/NISO. Z39.50-1995 (Versions 2 and 3) Information Retrieval: Application Service Definition and Protocol Specification, 1995.

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, 1986.

[Asy94] Asymetrix. *InfoModeler User Manual*. Asymetrix Corporation, 110-110th Avenue NE, Suite 700, Bellevue, WA 98004, Washington, 1994.

[BBB91] R. Bosman, R. Bouwman, and P.D. Bruza. The Effectiveness of Navigable Information Disclosure Systems. In G.A.M. Kempen, editor, *Proceedings of the Informatiewetenschap 1991 conference*, Nijmegen, The Netherlands, 1991.

[BBMP95] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, and H.A. Proper. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3):213–235, 1995.

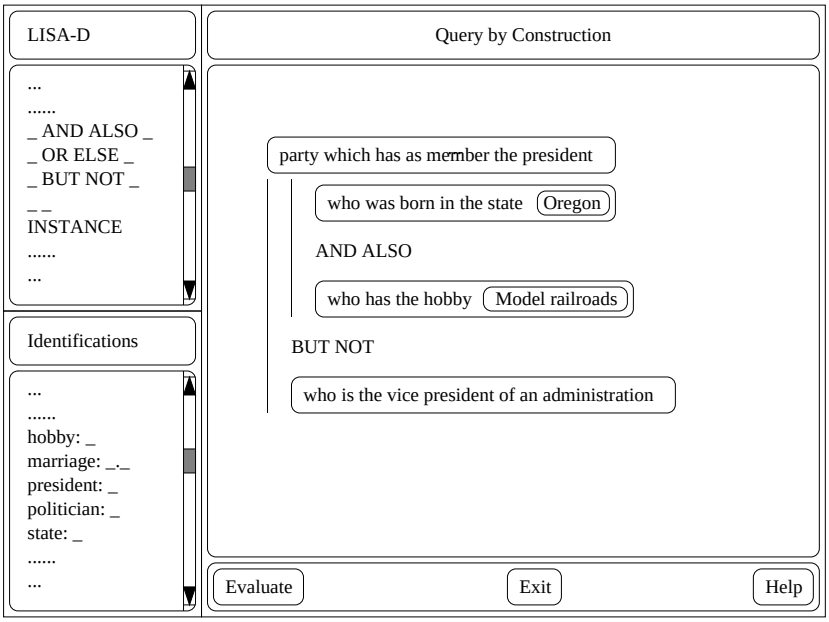

Fig. 19: Example query by construction session

[BCDT95] E. Buchholz, H. Cyriaks, H. Düsterhöft, A.and Mehlan, and B. Thalheim. Applying a Natural Language Dialogue Tool for Designing Databases. In *Proceedings of the First International Workshop on Applications of Natural Language to Databases (NLDB'95)*, pages 119–133, Versailles, France, June 1995.

[BHW91] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[BPW94] C.A.J. Burgers, H.A. Proper, and Th.P. van der Weide. An Information System organized as Stratified Hypermedia. In N. Prakash, editor, *CISMOD94, International Conference on Information Systems and Management of Data*, pages 159–183, Madras, India, October 1994.

[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.

[BR95c] J.F.M. Burg and R.P. van de Riet. The Impact of Linguistics on Conceptual Models: Consistency and Understandability. In *Proceedings of the First International Workshop on Applications of Natural Language to Databases (NLDB'95)*, pages 183–197, Versailles, France, June 1995.

[BRC93] J.F.M. Burg, R.P. van de Riet, and S.C. Chang. A data-dictionary as a lexicon: An application of linguistics in information systems. In Bhargava B., Finin T., and Yesha Y., editors, *Proceedings of the 2nd International Conference on Information and Knowledge Management*, 1993.

[BS84] B.G. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Massachusetts, 1984.

[Bur96] J.F.M. Burg. *Linguistic Instruments in Requirements Engineering*. PhD thesis, Free University, Amsterdam, The Netherlands, 1996.

[BW92] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.

[CD90] W.B. Croft and R. Das. Experiments with Query Acquisition and Use in Document Retrieval Systems. In J. Vidick, editor, *Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 349–368. ACM Press, 1990.

[Cha88] A.K. Chandra. Theory of Database Queries. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*, pages 1–9, Austin, Texas, March 1988.

[CP96] P.N. Creasy and H.A. Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering*, 20(2):119–162, 1996.

[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.

[Dah87] E. Dahlhaus. Skolem normal forms concerning the least fixpoint. In E. Borger, editor, *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science*, pages 101–106. Springer-Verlag, 1987.

[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.

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

[DMP84] O.M.F. De Troyer, R. Meersman, and F. Ponsaert. RIDL User Guide. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1984.

[DR91] F.P.M. Dignum and R.P. van de Riet. Knowledge base modeling based on linguistics and founded in logic. *Data & Knowledge Engineering*, 7:1–34, 1991.

[FHL97] P.J.M. Frederiks, A.H.M. ter Hofstede, and E. Lippe. A Unifying Framework for Conceptual Data Modelling Concepts. *Information and Software Technology*, 39(1):15–25, January 1997.

[Fre97] P.J.M. Frederiks. *Object-Oriented Modeling based on Information Grammars*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1997.

[FS76] J.P. Fry and E.H. Sibley. Evolution of Data-Base Management Systems. *Computing Surveys*, 8(1):7–42, 1976.

[GGP89] R. Godin, J. Gecsei, and C. Pichet. Design of a Browsing Interface for Information Retrieval. In N.J. Belkin and C.J. van Rijsbergen, editors, *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 32–37, Cambridge, Massachusetts, June 1989. ACM Press.

[Hal95] T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition, 1995.

[HH93] T.A. Halpin and J. Harding. Automated Support for Verbalization of Conceptual Schemas. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 151–161, Paris, France, June 1993.

[HLF96] A.H.M. ter Hofstede, E. Lippe, and P.J.M. Frederiks. Conceptual Data Modeling from a Categorical Perspective. *The Computer Journal*, 39(3):215–231, August 1996.

[HM92] T.A. Halpin and J.I. McCormack. Automated Validation of Conceptual Schema Constraints. 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 364–377, Manchester, United Kingdom, May 1992. Springer-Verlag.

[HP95] T.A. Halpin and H.A. Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[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.

[HPW95] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Computer Supported Query Formulation in an Evolving Context. In R. Sacks-Davis and J. Zobel, editors, *Proceedings of the Sixth Australasian Database Conference, ADC'95*, volume 17(2) of *Australian Computer Science Communications*, pages 188–202, Adelaide, Australia, January 1995.

[HPW96] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Query formulation as an information retrieval problem. *The Computer Journal*, 39(4):255–274, September 1996.

[HRWL83] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat. *Building Expert Systems*. Addison-Wesley, Reading, Massachusetts, 1983.

[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.

[HW94] A.H.M. ter Hofstede and Th.P. van der Weide. Fact Orientation in Complex Object Role Modelling Techniques. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 45–59, Townsville, Australia, July 1994.

[HW97] A.H.M. ter Hofstede and Th.P. van der Weide. Deriving Identity from Extensionality. *International Journal of Software Engineering and Knowledge Engineering*, 8(2):189–221, June 1997.

[Kol91] P.G. Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90(1):50–66, January 1991.

[Kri94] G. Kristen. *Object Orientation – The KISS Method, From Information Architecture to Information System*. Addison-Wesley, Reading, Massachusetts, USA, 1994. ISBN 0201422999

[LH96] E. Lippe and A.H.M. ter Hofstede. A Category Theory Approach to Conceptual Data Modeling. *RAIRO Theoretical Informatics and Applications*, 30(1):31–79, 1996.

[LN88] C.M.R. Leung and G.M. Nijssen. Relational database design using the NIAM conceptual schema. *Information Systems*, 13(2):219–227, 1988.

[Luc90] D. Lucarella. A Model for Hypertext-Based Information Retrieval. In *Proceedings of the European Conference on Hypertext - ECHT 90*, pages 81–94, Cambridge, United Kingdom, 1990. Cambridge University Press.

[Mee82] R. Meersman. The RIDL Conceptual Language. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1982.

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630

[Nij81] G.M. Nijssen. An approach for knowledge base systems. In *Proceedings of the SPOT-2 Conference*, 1981. Stockholm, Sweden.

[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.

[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.

[RS96] R. Richardson and A.F. Smeaton. An Information Retrieval Approach to Locating Information in Large Scale Federated Database Systems. In R.P. van de Riet, J.F.M. Burg, and A.J. van der Vos, editors, *Proceedings of the Second Workshop on Applications of Natural Language to Databases (NLDB'96)*, pages 52–64, Amsterdam, The Netherlands, June 1996.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, Cambridge, Massachusetts, 1977.

[TN85] S. Tehrani and G.M. Nijssen. UCL: A User-Friendly Conceptual Language. *The Australian Computer Journal*, 17(4):174–180, November 1985.

[VB82] G.M.A. Verheijen and J. van Bekkum. NIAM: an Information Analysis Method. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: A Comparative Review*, pages 537–590. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1982.

[Wat91] D.A. Watt. *Programming Language Syntax and Semantics*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[WBC84] D.C. Wilkins, B.G. Buchanan, and W.J. Clancey. Inferring an expert's reasoning by watching. In *Proceedings of the 1984 conference on Intelligent Systems and Machines*, 1984.

[WBGW73] S.E. Willner, A.E. Bandurski, W.C. Gorhan, and M.A. Wallace. COMRADE data management system. In *Proceedings of the AFIPS National Computer Conference*, pages 339–345, Montvale, New Jersey, 1973. AFIPS Press.

[Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.

[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.

A. THE SEMANTICAL BASE: PATH EXPRESSIONS

The set of all path expressions is referred to as $\mathcal{PE}$. Path expressions are constructed from elements of the information structure (roles, types) and by a number of operators. Path expressions are evaluated in the context of an environment. This environment consists of the current population and a partial function assigning values to actual variables. The meaning of a variable is its value in the current environment. In its elementary form, a path expression corresponds to a linear path through the information structure, starting and ending in a type. This path is interpreted as the description of a relation between these two ending points.

As a result of uniting linear path expressions with different begin and end points, path expressions, however, may be inhomogeneous. Such path expressions lead to inhomogeneous binary relations. Consequently, the semantics of a path expression is defined as a binary relation over (multiple) types.

First we introduce the atomic path expressions. The empty path expression ($\varnothing_{\mathcal{PE}}$) and the neutral path expression ($1_{\mathcal{PE}}$) are atomic path expressions. The empty path corresponds to the empty relation, while the neutral path is the identity relation (for all active values). The empty and neutral path form the zero- and

one-element respectively for the concatenation operator for path expressions (to be introduced later). They are also used as truth values (*False* and *True* respectively).

Constants are also atomic path expressions. The constant c is interpreted as a binary relation that only relates c with itself. Each type forms an atomic path expression, and is interpreted as the identity relation over that type (relating each of its instances only with that instance). Roles relate instances of their players to precisely those instances of the associated relationship type in which they play a role. For an overview, see table 1.

A number of operators and functions are available for the construction of composed path expressions. First we introduce the most important unary operators. They allow for the reversal of a path ($^{\leftarrow}$) and the isolation of the front elements of a path (Fr).

Paths can be extended in several ways. The product ($\times$) of two paths connects all front elements of the first path with all tail elements of the second path. Concatenation ($\circ$) is perhaps the most fundamental operator for extending paths. Instances are related via mutual intermediate instances. The inverse of concatenation is blocking ($\nleftrightarrow$). Instances from the front of the first path are connected to instances from the tail of the second path if they are *not* related via mutual intermediate instances. Consequently, $P \nleftrightarrow Q = P \times Q - P \circ Q$. Paths can also be extended via de subset and the superset operators. The subset operator $\not\trianglerighteq$ yields all front elements of the first path expression whose second component is part of the front of the second path expression, while the superset operator $\not\trianglerighteq$ yields all elements of the first path expression which are related to all elements in the front of the second path expression (and possibly more). Furthermore, the usual set operators ($\cap$, $\cup$ and $-$) are available, corresponding to intersection, union and set difference of the corresponding relations.

Using a combination of the operator selecting elements at the front of a path expression (Fr) and the set operations, we can also define:

$$\begin{aligned} P \mathbin{\not\cap} Q &\triangleq \mathsf{Fr}\, P \cap \mathsf{Fr}\, Q \\ P \mathbin{\not\cup} Q &\triangleq \mathsf{Fr}\, P \cup \mathsf{Fr}\, Q \\ P \mathbin{\not-} Q &\triangleq \mathsf{Fr}\, P - \mathsf{Fr}\, Q \end{aligned}$$

These modified versions of the set operators are quite useful in practice as we usually like to apply these operators to the front elements only.

Label types have an underlying domain. For this domain a number of functions and binary relations will be available. These functions and relations can also be used to form information descriptors. The expressions $45 + 25$ and $P < Q$ are therefore valid path expressions.

The confluence operator is used when different sorts of information are to be integrated. For example, name, day of birth, salary and address of each employee of some specific department. If $P_1, \ldots, P_n$ and Q are path expressions then $[p_1 : P_1, \ldots, p_n : P_n \mid Q]$ is a path expression, referred to as the confluence of $P_1, \ldots, P_n$ under Q via roles $p_1, \ldots, p_n$. This relation will contain a tuple $\{p_1 : x_1, \ldots, p_n : x_n\}$ iff for some value y path expression P_i relates x_i to y (for $1 \leq i \leq n$), while y occurs in the first component of the evaluation result of path expression Q. The restrictive effect of the condition can be neutralised by choosing $Q = 1_{\mathcal{PE}} = \mathit{True}$. As a shorthand, we define: $[p_1 : P_1, \ldots, p_n : P_n] = [p_1 : P_1, \ldots, p_n : P_n \mid \mathit{True}]$.

Two special operators are the comprehension operators. The first comprehension operator restricts a domain to some condition. In the expression $\{v \in P \mid C\}$ the domain to be restricted is P, v a variable running over this domain (via the environment in which path expressions are evaluated), while C is the restricting condition. The second comprehension operator yields all expressions which result by substituting a variable with values from a specified domain. In the expression $\{P \mid v \in Q\}$ P is a path expression containing one or more occurrences of variable v and Q is a path expression acting as the domain for v.

Finally, it is possible to specify conditional path expressions (if ... then ... else ...).

B. GRAPHICAL CONVENTIONS

This appendix contains a legenda with the graphical conventions used for ORM schemas in this article.

zero-adic		*monadic*		*dyadic*		*rest*	
name	expr	name	expr	name	expr	name	expr
empty path	$\varnothing_{\mathcal{PE}}$	reverse	$P^{\leftarrow}$	concatenate	$P \circ Q$	confluence	$[p_1{:}P_1, \ldots, p_k{:}P_k \mid Q]$
	True	front	$\mathsf{Fr}\ P$	blocking	$P \nleftrightarrow Q$		
neutral path	$1_{\mathcal{PE}}$			subset	$P \subseteq Q$	comprehension	$\{v \in P \mid C\}$
	False			superset	$P \supseteq Q$		$\{P \mid v \in Q\}$
constant c	c			intersection	$P \cap Q$	*predicates*	
type X	X			union	$P \cup Q$		
role p	p			minus	$P - Q$	choice	$\mathsf{if}\ C$
variable v	v			product	$P \times Q$		$\mathsf{then}\ P\ \mathsf{else}\ Q$

Table 1: Overview of path expressions

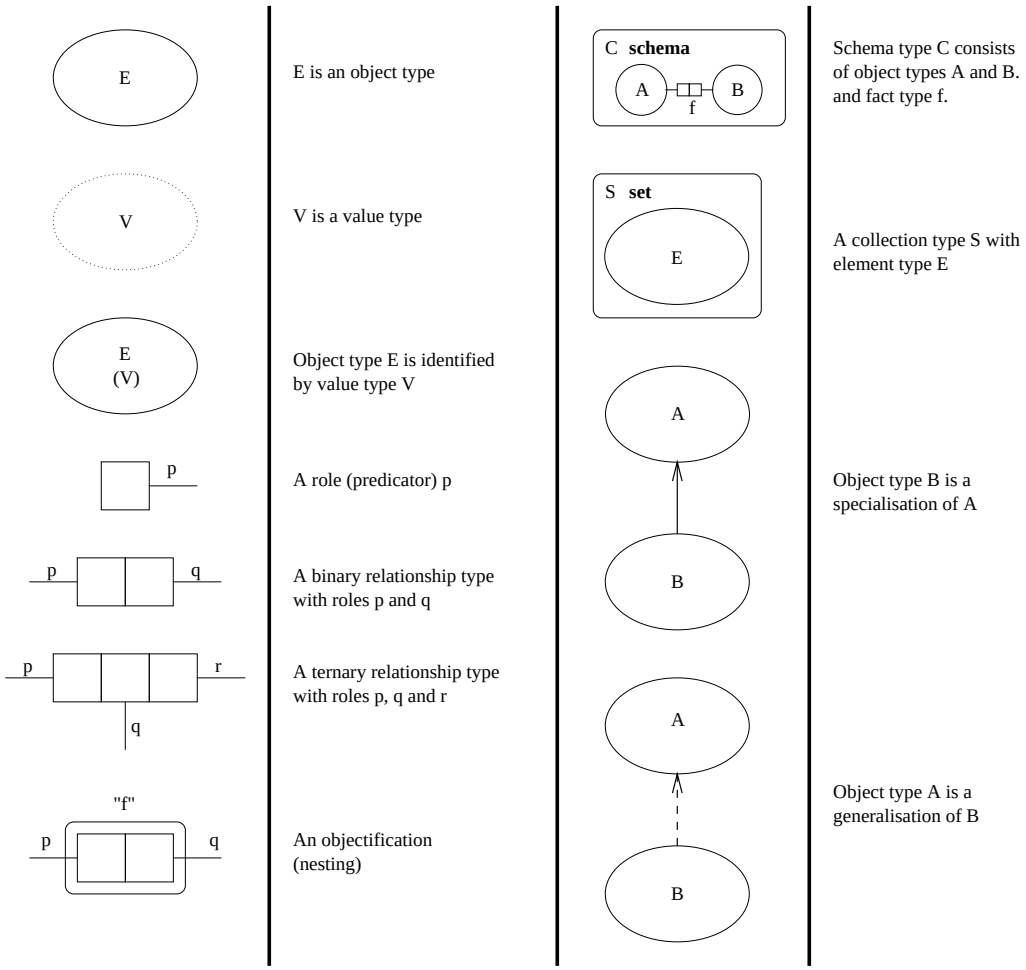

Fig. 20: Construction mechanisms used for ORM schemas