# A Conceptual Language for the Description and Manipulation of Complex Information Models

A.H.M. ter Hofstede, H.A. Proper and Th.P. van der Weide
Department of Information Systems, University of Nijmegen
Toernooiveld, 6525 ED Nijmegen, The Netherlands
E.Proper@acm.org

PUBLISHED AS:

A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Conceptual Language for the Description and Manipulation of Complex Information Models. In G. Gupta, editor, *Seventeenth Annual Computer Science Conference*, volume 16 of *Australian Computer Science Communications*, pages 157–167, Christchurch, New Zealand, January 1994. University of Canterbury. ISBN 047302313

## Abstract

Conceptual data modelling techniques aim at the representation of data (structures) at a high level of abstraction. This implies that conceptual data modelling techniques should not only be capable of representing complex structures in a natural way, but also the rules (constraints) that must hold for these structures. Contemporary data modelling techniques, however, do not provide a language which, on the one hand, has a formal semantics and, on the other hand, leads to natural looking expressions. In this paper, an informal introduction to such a language (LISA-D) for the data modelling technique (PSM), is presented. PSM is a generalisation of object-role models (such as ER, NIAM and FORM).

## 1 Introduction

Currently, many conceptual data modelling techniques exist. The *Conceptualisation Principle* ([ISO87]) states that a conceptual schema should deal only, and exclusively, with aspects of the underlying Universe of Discourse (UoD). Any aspect irrelevant to that meaning, e.g. machine efficiency, should be avoided. Contemporary data modelling techniques are not capable of adhering to the Conceptualisation Principle for each UoD. Firstly, choices that are not relevant with respect to the UoD may have to be made, leading to *overspecification*. Secondly, even worse, the UoD may have to be adapted to meet the requirements of the modelling technique, e.g. by the introduction of extra object types (see [HPW92]). These problems are caused by the lack of sufficiently powerful construction mechanisms.

A second important principle of conceptual data modelling is the *100% Principle* ([ISO87]), stating that a conceptual schema completely prescribes all the permitted states and transitions of the conceptual data base. As a result, a conceptual data modelling technique should not only be capable of representing complex structures. It should also be able to represent rules (constraints) that must hold for these structures. In most modelling techniques this is not possible, and one must resort to the use of natural language to specify these constraints.

Besides constraints, it would also be convenient to be able to express queries and updates on a conceptual level. Many query and manipulation languages (e.g. SQL) require a fairly high level of training, or are based on a rather primitive data modelling technique (e.g. ER).

In [HW93], the conceptual data modelling technique PSM (Predicator Set Model) has been defined, which is capable of representing complex object structures without violating the Conceptualisation Principle. PSM is an extension of PM (Predicator Model [BHW91]) which on its turn is a formalisation of NIAM ([NH89], [Win90], [HO92]). This means that all NIAM schemas can be seen as PSM schemas. It also means that the design procedure supporting the construction of NIAM schemas (the *way of working* [Wij91]) and the NIAM philosophy are not lost, they only need to be extended to support the additional constructs as well. An alternative formalization of NIAM is FORM ([HO92], [Hal89]).

The NIAM analysis method is based on an analysis method for natural language. The method starts from verbalisations of examples, which form a (partial) description of the underlying domain, and are provided by domain experts. We refer to the language (idiom), in which the examples are verbalised, as the *expert* *language*. The verbalisation leads in a straightforward way to an information structure.

It is only natural that the language for manipulating and querying also has the format of a semi natural language, and is designated to approximate the expert language as close as possible. The rationale behind this has also been addressed in [HH93]. As a result, sentences in this language are meaningful expressions within the context of the U∘D, understandable and expressible by domain experts. The sentences, verbalising the original examples, form extensional specifications, while queries (in general) correspond to intentional specifications.

The language RIDL (Reference and IDea Language [DMP84], [Mee82]) was developed for this purpose. However, due to its informal definition, no rigid base for both its syntax and semantics was provided, the language never got much acceptance. Furthermore, RIDL was based on the restricted binary version of NIAM ([VB82]).

In [HPW93] and [Hof93] the language LISA-D (Language for Information Structure and Access Descriptions) has been formally introduced, covering both its syntax and semantics. This language is based on PSM. Its functionality far exceeds the intended functionality of RIDL. As PSM has been designed as a general object-role modelling technique, LISA-D is also applicable to well-known representatives of the object-role modelling paradigm such as FORM, ER ([Che76]), FDM ([Shi81]) or IFO ([AH87]).

In this paper we provide an informal introduction to LISA-D. In section 3 we propose a CASE-Tool, facilitating a convenient *way of support* for LISA-D. How intentional specifications (queries) can be formulated is discussed in section 4 and section 5. Finally, in section 6 we focus on the part of LISA-D which is concerned with updating of populations.

### The running example

In this paper, we will relate most of the examples to the following case. In the example, we consider airforces and their relation to political entities. An airforce consists of a set of squadrons, and is assigned to a political entity. For instance the RAF is the airforce of the United Kingdom, and the RCAF is the Canadian air-force. Air-forces can, however, be assigned to other political entities than states. The 1-th ATAF (First Allied Tactical Air-force) is an airforce consisting of several squadrons from air-forces of the Northern European states of NATO. As a result, one squadron may be assigned to more than one air-force. Every squadron may be referred to by a squadron name (a code), and political entities may have a name as well. Note that the identification of both a squadron and a political entity will be provided later on. The resulting schema is displayed in figure 5.

A squadron consists of aircrafts, each of which is either a transport aircraft or a combat aircraft. Both classes of aircrafts have their own identification, a T-code and a C-code respectively. For a transport aircraft, its capacity is stored in the database. We distinguish between two classes of combat aircrafts, a bomber and a fighter. A combat aircraft may simultaneously be a bomber and a fighter, for instance the Tornado fighter/bomber as used by some of the European air forces. As a result we have introduced two subtypes for combat aircrafts. For a bomber, its maximum bomb load is stored, whereas for a fighter its number of guns is considered to be relevant. The resulting subschema is presented in figure 8

## 2 Describing an Information Model

In this section a short (informal) introduction to the concepts of PSM is given. In addition to that, we show how a PSM schema is described in the language LISA-D. A schema description in LISA-D not only captures the relevant object *types* and their interrelationships, but also captures the denotation of *instances* of object types.

In the next section we will describe the user interface of a provisional CASE Tool supporting the specification of Lisa-D schemata, and the maintenance of their populations.

### 2.1 Label types

In many conceptual data modelling techniques, a distinction exists between objects that can be represented directly and objects that can not be represented directly. In ER, this distinction is reflected by the difference between entities and attributes, while in NIAM and PSM this distinction corresponds with the difference between entities and labels. Labels (or attributes) are elements from a concrete domain (e.g. strings). As a result, label types have an associated concrete domain. In LISA-D, label types are related to this associated domain in the following way:

LABEL TYPE $L$ HAS DOMAIN $D$;

For the air forces schema we have:

```
LABEL TYPE   T-code   HAS DOMAIN String;
            Nr-guns   HAS DOMAIN Natno;
             C-code   HAS DOMAIN String;
```

Values of label type $L$ can be denoted in LISA-D by the construct $L$ : $d$ where $d$ is a denotation to be interpreted as a value of type $L$. For instance, Nr-guns: 10 denotes a value of type Nr-guns with denotation 10.

## 2.2 Fact types

One of the key concepts in data modelling is the concept of relation type, in NIAM and PSM referred to as *fact type*. Generally, a relation type is considered to be an association between object types. A role denotes the way an object type participates in this association. The participation itself is called a *predicator*. As a result, a fact type can be seen as a set of predicators.

In LISA-D, a fact type (relation type) is specified by its name, followed by the description of its constituent predicators. A predicator is specified by its name and the name of the corresponding object type, its so-called *base*. The declaration has the following format:

FACT TYPE $F$ ($p_1$:$N_1$, ..., $p_k$:$N_k$)

For example, the fact type Assignment from the Air-Force database may be declared as follows:

```
FACT TYPE Assignment (assigned-to: Air-force,
                      having-as: Political-entity);
```

A fact type instance is denoted as an enumeration of the (denotations of the) instances involved: $F$ : $d_1, \ldots, d_k$, where $d_i$ is a constant denotation for an instance of the object type named $N_i$, being the base of predicator $p_i$.

Bridge types are a special kind of binary fact types. They cross the gap between the concrete and the abstract world, by connecting abstract object types with concrete object types (label types). Usually the bridge type and its predicators remain anonymous. This explains the following format:

BRIDGE TYPE FROM $N$ TO $L$

where $N$ is the name of an entity type, and $L$ the name of a label type. Some examples are:

```
BRIDGE TYPE FROM Combat-aircraft TO C-code,
            FROM Weaponload      TO Nr-guns
```

An alternative formulation, conforming to ER conventions, is the following:

ENTITY TYPE $N$ HAS ATTRIBUTE $L$

This format is discussed in the next section.

## 2.3 Entity types

Entities correspond to relevant objects in the UoD. As a result, they are abstract objects, which have to be identified by label values. Entity types are entered into a schema in two steps. In the first step, the name of the entity type is introduced (and optionally its attributes):

ENTITY TYPE $E$ HAS ATTRIBUTES $L_1, \ldots, L_k$

Other variants of the entity type declaration will follow. Some examples are:

```
ENTITY TYPE
    Combat-aircraft HAS ATTRIBUTE C-code;
    Weaponload HAS ATTRIBUTE Nr-guns;
```

In the second step, the identification of the entity type is specified, in the form of an identification descriptor:

IDENTIFICATION $E : x_1 \ldots x_n =$
$E$ (information descriptor)

Information descriptors are introduced in a next section. This declaration can be regarded as a macro declaration with parameters $x_1, \ldots, x_n$. The expression that results after substitution of the parameters by constants, should be a proper identifier for instantiations of object type $E$. This means that this information descriptor may retrieve at most one object instance (in *any* population). Testing whether this property is universally valid, is part of the schema verification process.

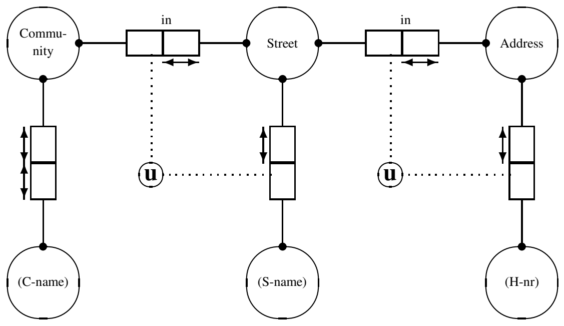


Figure 1: An example of a complex identification

As an example, consider the information structure from figure 1. This schema (omitting the constraints) is described as follows:

```
LABEL TYPE
    H-nr HAS DOMAIN Int;
    S-name HAS DOMAIN String[30];
    C-name HAS DOMAIN String[30];
```

```
ENTITY TYPE
    Address HAS ATTRIBUTE H-nr;
    Street HAS ATTRIBUTE S-name;
    Community HAS ATTRIBUTE C-name;

FACT TYPE
    In-street (in: Address, from: Street);
    In-community (in: Street, from: Community);
IDENTIFICATION
   Address: x y = Address(in Street: x
                           AND-ALSO
                           WITH H-nr: y);
   Street: x y = Street(in Community: x
                         AND-ALSO
                         WITH S-name: y);
   Community: x = Community WITH C-name: x;
```

In this example, an address now can be denoted as: Address: ('New York' 'Fifth Avenue') 17. In this expression, the parentheses may be omitted.

## 2.4 Specialisation

Specialisation is a mechanism for the introduction of special subclasses of objects, which all satisfy some property, referred to as the *subtype defining rule*. These subclasses inherit all properties of their supertypes, but may have other properties as well. The specialisation hierarchy is described by specifying for each subtype $S$ its direct supertypes $(X_1, \ldots, X_k)$, together with the associated subtype defining rule $P$, where $P$ is an information descriptor. Information descriptors are discussed in a subsequent section.

ENTITY TYPE $S$ SUBTYPE OF $X_1, \ldots, X_k$
ACCORDING TO $P$

An example of such a declaration is:

```
ENTITY TYPE Bomber SUBTYPE OF Combat-aircraft
  ACCORDING TO Combat-aircraft having Type 'bomber'
```

## 2.5 Generalisation

Generalisation provides the opportunity to group different kind of object types (specifiers) and to assign new properties to this group. Generalised objects inherit their properties from their specifiers. The generalisation hierarchy is described by specifying for each generalised entity type its specifiers:

ENTITY TYPE $G$ GENERALISATION OF $X_1, \ldots, X_k$

The object type Aircraft is used as a generic name for both Combat-aircraft and Transport-aircraft.

```
ENTITY TYPE Aircraft
  GENERALISATION OF Combat-aircraft, Transport-aircraft
```

## 2.6 Power types

Power types are the data modelling pendant of powersets from conventional set theory. An instance of a power type is a set of instances of its *element type*. Power types are introduced by a statement of the following format:

POWER TYPE $P$ OF $X$

This statement introduces a power type with name $P$, having as element type the object type named $X$. The denotation of an instance of a power type consists of an enumeration of denotations of its elements. For example, an instance of Convoy in figure 2 is denoted as follows: Convoy: $\{\text{Ship}_1, \ldots, \text{Ship}_n\}$ where each $\text{Ship}_i$ is a denotation for a ship.

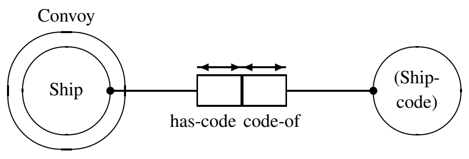


Figure 2: Convoys of ships

## 2.7 Sequence types

Sequence typing forms the data modelling counterpart of the mathematical notion of (homogeneous) tuple. The following statement introduces a sequence type with name $S$ and element type named $X$.

SEQUENCE TYPE $S$ OF $X$

The denotation of instances of a sequence type consists, similarly to power types, of an enumeration of denotations of its elements. For sequence types, however, the order of the elements is of importance. For example, the format Freight-car-sequence: $\langle \text{Freight-car}_1, \ldots, \text{Freight-car}_n \rangle$ is the way to denotate an instance of the sequence type Freight-car-sequence from figure 3, provided each $\text{Freight-car}_i$ is a denotation of a freight car.

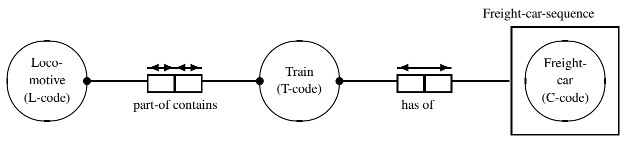


Figure 3: Freight trains

## 2.8 Schema types

The PSM modelling technique offers the possibility of schema (de)composition. A schema type is a composition of some underlying (sub)schema, called its decomposition. An instance of a schema type is a population of its decomposition. For example, figure 4 contains a meta model of activity graphs (see [Sch84]), using schema decomposition. Activity graphs are used for modellling processes and information passing between processes. They are bipartite directed graphs, consisting of *Activities* (processes) and *States*. States can be input for, or output of activities. In an activity graph, both activities and states may be subject to refinement. This is modelled in figure 4, where Activity-graph is a schema type, corresponding to the notion of activity graph.

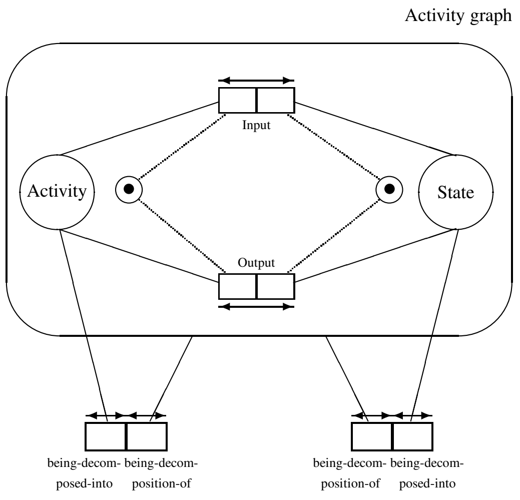


Figure 4: A meta model for activity graphs

A new schema type named $G$, being the composition of the object types named $X_1, \ldots, X_k$, is introduced as follows:

SCHEMA TYPE $G$ OF $X_1$, ..., $X_k$

For instance, the Activity-graph schema type is declared by:

SCHEMA TYPE Activity-graph OF Activity, State, Input, Output

Usually a schema type is described in a separate PSM schema, and is imported in the current schema by:

SCHEMA TYPE $G$

## 3 Doolittle: A Provisional LISA-D Tool

In this section we discuss a way of support for the LISA-D language, in the form of the provisional CASE-Tool Doolittle. The behaviour of this CASE-Tool is described in terms of an interpreting automaton (see [Win90]). Furthermore, we introduce a user interface for this CASE-Tool supporting the special constructs of LISA-D. Several prototype implementations of Doolittle have been set up. We mention [PEPW93], an implementation in the functional language Clean ([BELP87]), and [Hub93], which is based on the language Prolog.

The conceptual schema for the Universe of Discourse for the air-forces running example is depicted by the LISA-D Tool, conforming to the drawing style of NIAM, in figure 5.

### 3.1 Decomposition

In figure 5, the decompose button $\bigtriangledown$ qualifies Political-entity as an object type with an underlying decomposition. In this case, the decomposition is a refinement of Political-entity, resulting in a more detailed schema. This refinement deals with states (such as Canada or Norway), and the grouping of states (or political entities) into political entities (for instance the NATO or the Western European Union).

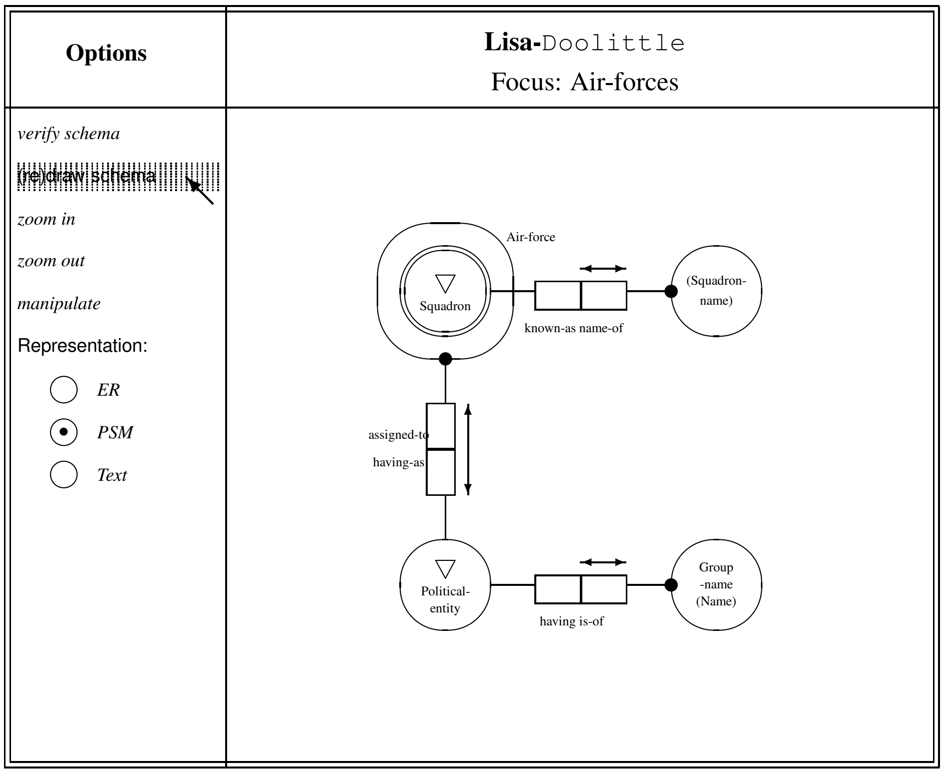
The decomposition of Political-entity may be included in the current diagram by clicking its decompose button. This will lead to the screen of figure 7, in which Political-entity has associated a compose button $\bigtriangleup$. Subsequently selecting this compose button removes the expansion of Political-entity from the screen, resulting in the original screen from figure 5.

The decompose button of Squadron indicates an underlying decomposition for this object type. However, the object type Squadron has the status of a schema type. In PSM, a schema type has an associated (sub)schema. The population of a schema type thus consists of a set of populations of this underlying subschema.

Besides the (de)compose mechanism described above, LISA-D offers navigation as another mechanism for hiding details, in particular for handling of object types with an underlying decomposition. The zoom in item from the Options menu results in a transfer to the (sub)schema associated with the selected object type. For instance, zooming in on object type Squadron in figure 5 results in the screen from figure 8. The zoom out option performs the inverse navigation. Note that zooming in on the object type Political-entity leads to a screen, which displays the construction of political entities from states and groups of political entities.

### 3.2 Drawing styles

Information structures from PSM can be represented (as complete as possible) according to several drawing styles. For example, figure 6 shows the schema from figure 5 according to ER-conventions ([You89]). The LISA-D Tool offers the user the possibility to choose between ER and PSM (NIAM). However, some extensions to ER are required, for example, ER can



Representation: ( ) ER [✓ PSM] ( ) Text

Figure 5: Air Forces

not handle power types, sequence types, schema types and generalisation. In figure 6 we have chosen to represent the power type Air-force by putting an extra square around the corresponding element type Squadron (the graphical symbol for entity types in ER). Some extended ER versions (e.g. [EWH85]) can indeed handle some of these concepts.

Generally, graphical descriptions tend to be incomplete, as, for example, not all constraints can be represented graphically. A complete description of the underlying information structure can be provided by a textual description (see figure 9 for a textual description of the schema from figure 5). As a result, a drawing style can be seen as a special (graphical) view on the textual description. This (sub)viewing mechanism is intended to offer the possibility to employ (as far as possible) one's favourite design methodology.

## 4 Retrieving Information

LISA-D provides a set of grammar rules which, complemented with a concrete lexicon as obtained from a particular information structure, leads to a concrete information retrieval language tuned for the particular information structure, i.e. closely resembling the expert language. As a result, LISA-D defines a class of languages, where each language is based on a particular underlying lexicon. This is analogous to, say, predicate calculus, in which a set of predicate symbols provides the lexicon, whereas the construction rules for formulas correspond to the grammar. Starting from an example, we will first provide the construction rules of the lexicon, followed by a discussion of the grammar rules.

An information descriptor is interpreted as a rule for the retrieval of information, fulfilling some information need. An example of an information need is:

*All political entities having an air-force,*
*that contains a squadron known under*
*the squadron-name '316'.*

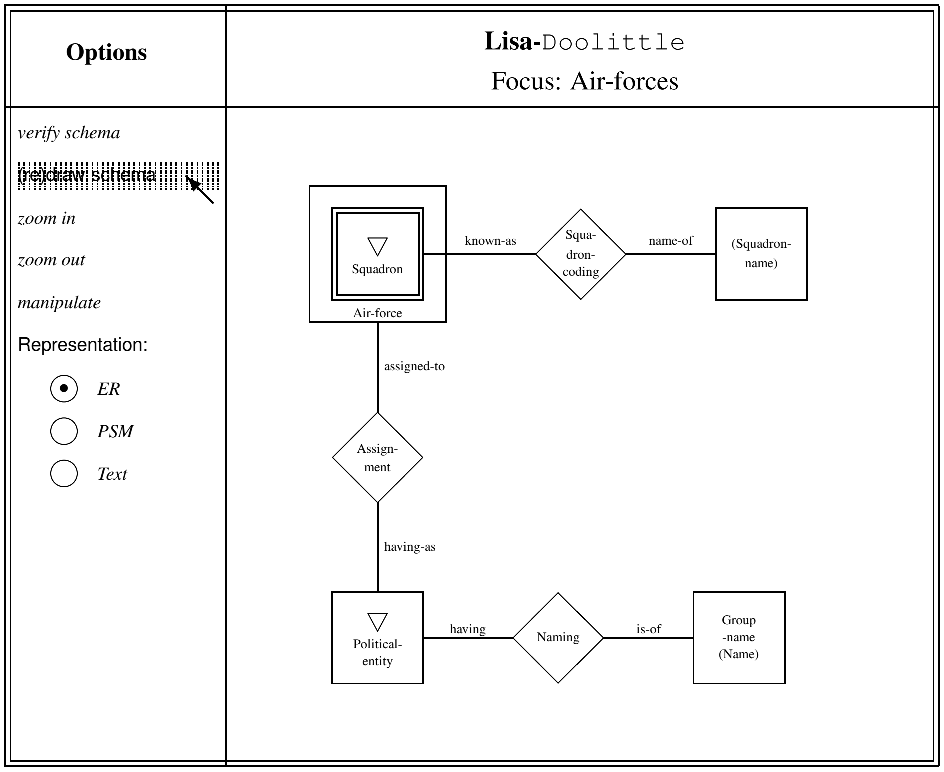
This sentence corresponds to the following LISA-D information descriptor:

```
Political-entity having-as Air-Force CONTAINING
  Squadron known-as Squadron-name '316'
```

This information descriptor is composed of the following simple information descriptors:

```
Political-entity, having-as, Air-force,
CONTAINING, Squadron, known-as,
Squadron-name, '316'
```

The semantics of an information descriptor $I$ is expressed as a binary relation $\rho(I)$. The simplest form of information descriptor is the name of an object



Representation: [✓ ER] PSM Text

Figure 6: Air Forces according to ER

type, such as: Political-entity, Air-force and Squadron. In this case, the associated binary relation consists of all tuples $\langle v, v\rangle$, with $v$ an instance of the object type:

$$\rho(x) = \big\{ \langle v, v\rangle \;\big|\; v \in \mathsf{Pop}(x) \big\}$$

The transition of one object type to another, through a fact type, is called a connector. For instance, having-as is the name of the connector corresponding to the transition from Political-entity to Air-force via fact type Assignment. Connector names are information descriptors as well, having as semantics the binary relation consisting of all tuples $\langle v, w\rangle$ such that instance $v$ is connected to $w$ via an instance of the corresponding fact type.

LISA-D provides a number of generic connector names, which are also information discriptors. The name CONTAINING, is an example of such a generic connector name. It denotes the transition from any power type (such as Air-force) to its associated element type (Squadron). The semantics of a generic connector name is the sum of all associated concrete connectors.

The next category of elementary information descriptors is formed by denotations of constants. They are interpreted in the obvious way:

$$\rho(\text{'316'}) = \big\{ \langle 316, 316\rangle \big\}$$

Next we focus at the grammer rules for forming information descriptors. The main construction mechanism for information descriptors is juxtaposition. The general rule for concatening information descriptors $P$ and $Q$ is:

$$\rho(P\ Q) = \rho(P) \circ \rho(Q)$$

where $\circ$ denotes concatenation of binary relations. The semantics of the example information descriptor can thus be expressed as:

$$\begin{aligned}
&\rho(\text{Political-entity has-as} \ldots \text{ Squadron-name '316'}) \\
&\quad = \rho(\text{Political-entity}) \circ \rho(\text{has-as}) \circ \rho(\text{Air-Force}) \circ \\
&\qquad \rho(\text{CONTAINING}) \circ \rho(\text{Squadron}) \circ \rho(\text{known-as}) \circ \\
&\qquad \rho(\text{Squadron-name}) \circ \rho(\text{'316'})
\end{aligned}$$

Thus far, an information descriptor corresponds to a linear path through the information structure, with a unique begin and end point. These information descriptors are qualified as linear, and have the following property:

> if $P$ is a linear information descriptor from object type $x$ to $y$, then:
>
> $$\rho(P) \subseteq \mathsf{Pop}(x) \times \mathsf{Pop}(y)$$

The Doolittle CASE tool supports the formulation of such information descriptors by succesively clicking (using a mouse) the elements of the path. Next we consider construction rules leading to non-linear information descriptors. A first group contains

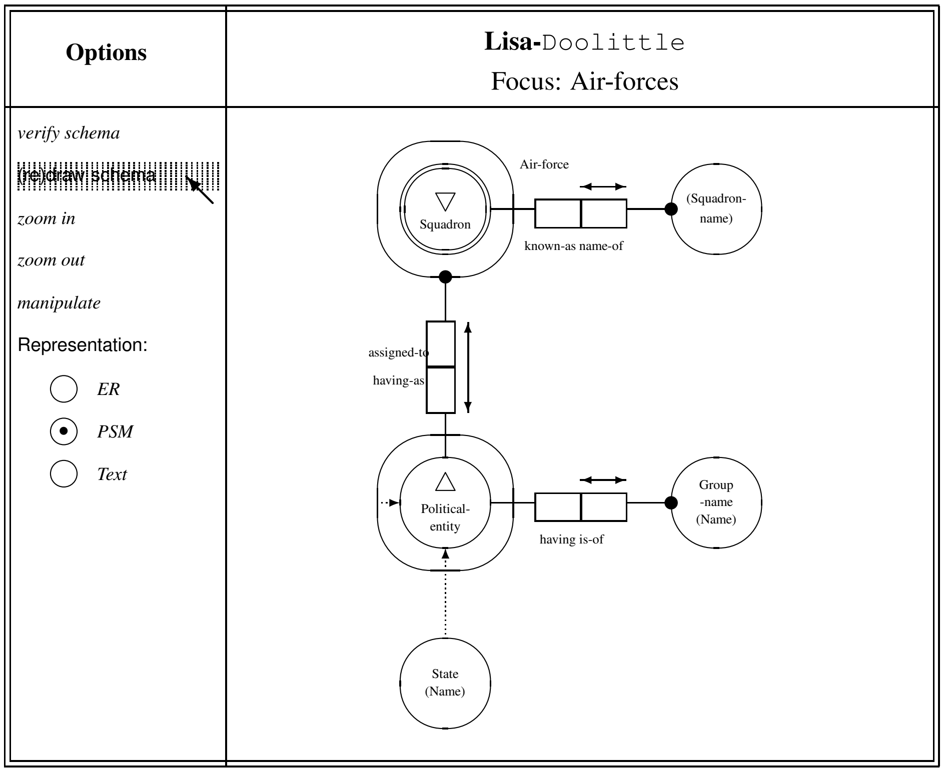


Figure 7: Squadron decomposed

operators such as union, intersection and set difference:

| expression | $\rho$(expression) |
|---|---|
| $P$ INTERSECTION $Q$ | $\rho(P) \cap \rho(Q)$ |
| $P$ UNION $Q$ | $\rho(P) \cup \rho(Q)$ |
| $P$ MINUS $Q$ | $\rho(P) \setminus \rho(Q)$ |

Information descriptors relate beginning and ending points of paths through the information structure. The operator THE makes a restriction to the beginning points of these paths:

| expression | $\rho$(expression) |
|---|---|
| THE $P$ | $\{\langle x, x\rangle \mid \langle x, y\rangle \in \rho(P)\}$ |

The second group of binary operators operates on first elements of (binary) tuples:

| expression | interpretation |
|---|---|
| $P$ AND-ALSO $Q$ | (THE $P$) INTERSECT (THE $Q$) |
| $P$ OR-ELSE $Q$ | (THE $P$) UNION (THE $Q$) |
| $P$ BUT-NOT $Q$ | (THE $P$) MINUS (THE $Q$) |

The third group of binary operators consists of arithmetic operators:

| expr. | $\rho$(expression) |
|---|---|
| $P$ **op** $X$ | $\left\{\langle x \text{ \textbf{op} } y, z\rangle \;\middle|\; \begin{array}{l}\langle x, x\rangle \in \rho(\text{THE}P) \wedge \\ \langle y, z\rangle \in \rho(Q)\end{array}\right\}$ |

where **op** $\in \{+, -, *, /\}$.

## 5 Specifying Constraints

The basis for LISA-D predicates is formed the following existential operator for information descriptors:

$$\text{SOME } P \quad \triangleq \quad \rho(P) \neq \varnothing$$

Instead of SOME $P$ we will also write $P$. From these atomic predicates, new predicates can be formed in the usual fashion, using logical connectives and quantification:

$C_1$ AND $C_2$
$C_1$ OR $C_2$
NO $C$
FOR EACH $x$ IN $P$ HOLDS $C$
FOR SOME $x$ IN $P$ HOLDS $C$

where $C_1, C_2, C$ are predicates, and $P_1, P_2, P$ information descriptors. In the quantification constructs, $x$ is a variable bound to $P$, as follows:

$$\text{FOR EACH } x \text{ IN } P \text{ HOLDS } C \triangleq \forall_{\langle a,b\rangle \in \rho(P)} [C|^x_a]$$
$$\text{FOR SOME } x \text{ IN } P \text{ HOLDS } C \triangleq \exists_{\langle a,b\rangle \in \rho(P)} [C|^x_a]$$

Using the above predicates, the following comparison operators on information descriptors can be formulated.

$$P_1 \text{ INCLUDES } P_2 \;\triangleq\; \text{NONE } (P_1 \text{ MINUS } P_2)$$

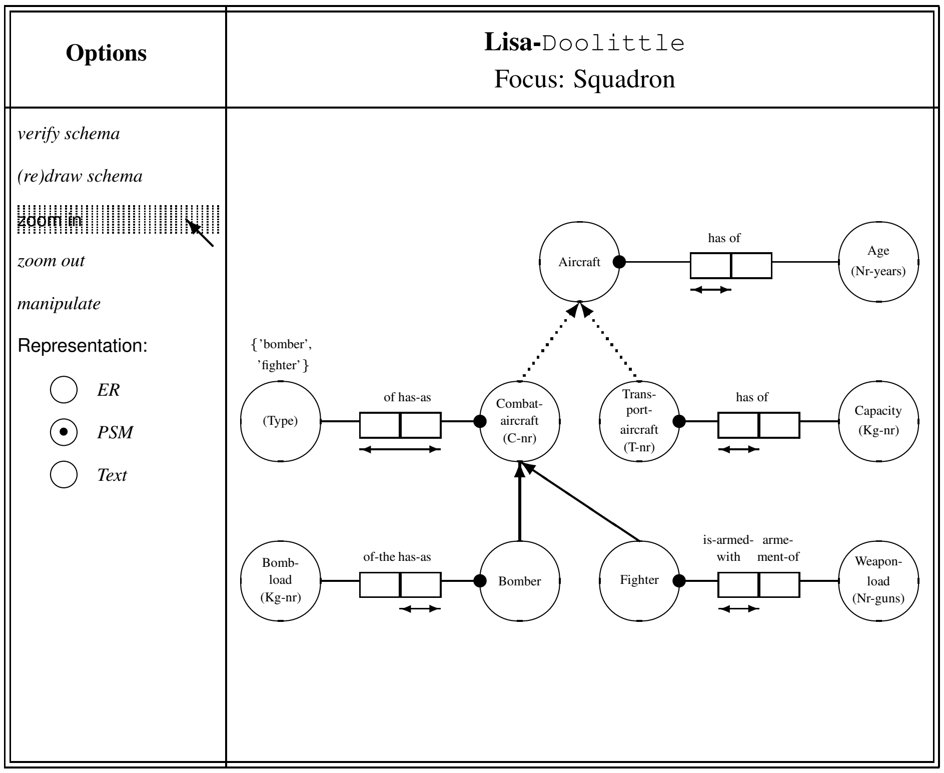


Figure 8: Zooming in on Squadron

$$P_1 \text{ EQUALS } P_2 \triangleq (P_1 \text{ INCLUDES } P_2) \text{ AND } (P_2 \text{ INCLUDES } P_1)$$

$$P_1 \text{ OVERLAPS } P_2 \triangleq \text{SOME}(P_1 \text{ INTERSECTION } P_2)$$

Predicates can be used to resolve yes-no information requests, such as: *is it the case that ...?* Besides, these expressions are also useful for the description of constraints. An example is the following sentence:

> *An aircraft may be in a squadron in an air-force assigned to a political entity at most once.*

This constraint cannot be represented graphically in PSM. The corresponding LISA-D expression would be:

```
CONSTRAINT
  p6: FOR EACH x IN Aircraft:
      NUMBER-OF(Political-entity having-as Air-force CONTAINING
                 Squadron COMPRISING Air-Craft x) ≤ 1
```

In this expression, COMPRISING embodies the transition from a schema type instance to instances from its decomposition, and NUMBER-OF is a function on information descriptors yielding a relation with one tuple $\langle n, n \rangle$ where $n$ is the number of elements of the information descriptor involved.

## 6 Updates

In this section the LISA-D constructs for updating populations are discussed. In LISA-D update statements either add or delete object instances to populations. For a proper introduction, a partial ordering $\sqsubseteq$ on populations of an information structure is useful.

**Definition 6.1**

*Let* Pop *and* Pop′ *be populations of a PSM information structure, then* Pop $\sqsubseteq$ Pop′ *if and only if:*

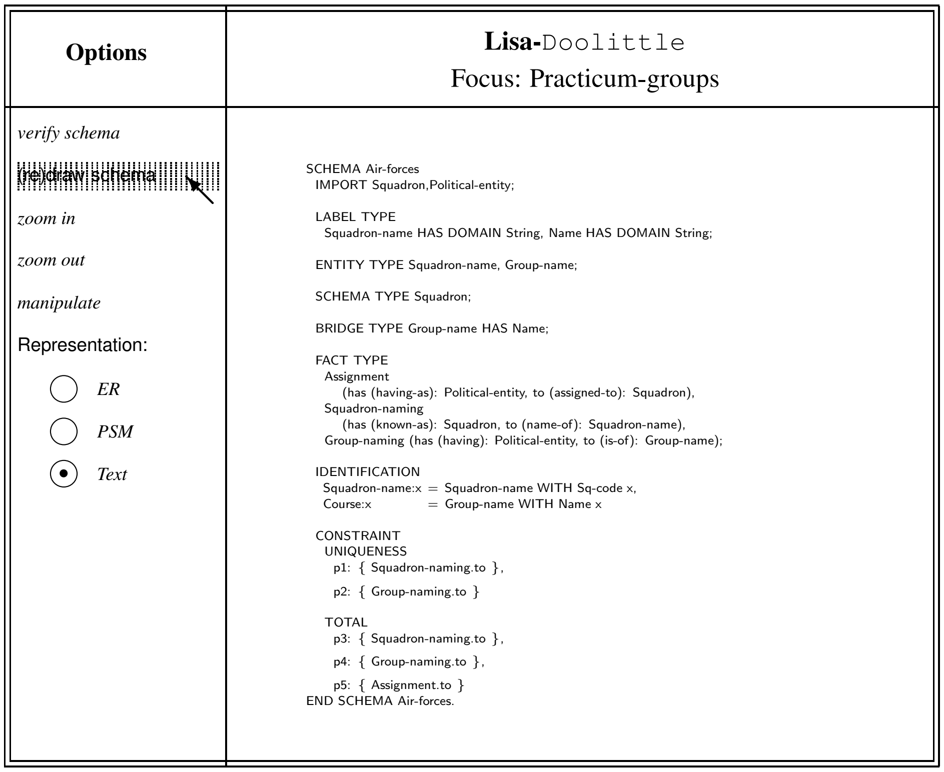
$$\forall_{x \in \mathcal{O}} \left[ \text{Pop}(x) \subseteq \text{Pop}'(x) \right]$$

□

Clearly $\sqsubseteq$ is a reflexive partial ordering.

Adding instances to a population is performed by the add statement, with the format ADD $P$, where $P$ is any information descriptor. The meaning of this statement is to enforce *a* minimal extension of the current population, that populates $P$, i.e. a minimal extension Pop′ of the current population Pop, such that information descriptor $P$ has no empty result (i.e., SOME $P$) in the extended population Pop′.

Usually, the extension will involve label type instances as well as abstract objects. The possible extension with abstract instances, shows why it is necessary to speak of *a* minimal population instead of *the* minimal population: any abstract instance may be added when needed, as long as the requirements are fulfilled (especially the minimality of the extention).



```
SCHEMA Air-forces
  IMPORT Squadron,Political-entity;

  LABEL TYPE
    Squadron-name HAS DOMAIN String, Name HAS DOMAIN String;

  ENTITY TYPE Squadron-name, Group-name;

  SCHEMA TYPE Squadron;

  BRIDGE TYPE Group-name HAS Name;

  FACT TYPE
    Assignment
        (has (having-as): Political-entity, to (assigned-to): Squadron),
    Squadron-naming
        (has (known-as): Squadron, to (name-of): Squadron-name),
    Group-naming (has (having): Political-entity, to (is-of): Group-name);

  IDENTIFICATION
    Squadron-name:x = Squadron-name WITH Sq-code x,
    Course:x        = Group-name WITH Name x

  CONSTRAINT
    UNIQUENESS
      p1: { Squadron-naming.to },

      p2: { Group-naming.to }

    TOTAL
      p3: { Squadron-naming.to },

      p4: { Group-naming.to },

      p5: { Assignment.to }
END SCHEMA Air-forces.
```

Figure 9: Textual representation of the Air Forces case

It is a good convention to use object denotations as objective for the add statement. However, the definition of the add statement makes it possible to formulate such things as

ADD Aircraft

This statement adds an arbitrary aircraft if and only if there is no aircraft in the population at hand. An other example is:

ADD State having-as Air-force

This statement assigns an arbitrary air-force to an arbitrary state if and only if such a relation is not available in the current population. Besides, it may lead to the creation of a state, and the creation of an air-force.

Instances can be deleted from a population by the delete statement, with the format DELETE $P$, where $P$ is any information descriptor. The meaning of this statement is to enforce a minimal reduction of the current population, that unpopulates $P$, i.e. a maximal part $\mathsf{Pop}'$ of the current population $\mathsf{Pop}$, such that information descriptor $P$ has an empty result (i.e., NO $P$) in the reduced population $\mathsf{Pop}'$. The reduced population must be a correct population with respect to the PSM modelling technique. However, constraints may be violated, in which case the delete statement is rejected, as we will see in the example below.

As a more elaborate example of updates, let us suppose that the schema of figure 8 has been populated as follows:

| | |
|---|---|
| Aircraft | $c_1, c_2, c_3, t_1, t_2$ |
| Combat-aircraft | $c_1$('F270'), $c_2$('F271'), $c_3$('F401') |
| Type | 'bomber', 'fighter' |
| Bomber | $c_3$ |
| Fighter | $c_1, c_2$ |
| Weaponload | $w_1(4), w_2(6)$ |
| Bombload | $b_1(200)$ |
| Transport-aircraft | $t_1$('TR300'), $t_2$('TR301') |
| Capacity | $tl_1$('2000 kg'), $tl_2$('2500 kg') |
| Combat-aircraft-typing | $\langle c_1, \text{'fighter'}\rangle$, $\langle c_2, \text{'fighter'}\rangle$, $\langle c_3, \text{'bomber'}\rangle$ |
| Fighter-weaponry | $\langle c_1, w_1\rangle, \langle c_2, w_2\rangle$ |
| Bomber-weaponry | $\langle c_3, b_1\rangle$ |
| Transport-capacity | $\langle t_1, tl_1\rangle, \langle t_2, tl_2\rangle$ |

In the above example population, $c_i$, $w_i$, $b_i$, $t_i$ and $tl_i$'s denote instances of abstract object types. The identification (denotation) of these abstract instances is given between parentheses. Suppose the following fact is entered:

*the transport aircraft with code 'TR400'*
*has a capacity of 3000 kg.*

This can be formulated in LISA-D as follows:

ADD Transport-aricraft: 'TR400' has-a Capacity: '3000 kg'

This statement is *accepted* by the LISA-D Tool. This acceptance is communicated to the user by the message:

```
Command completed.
```

and leads to the following population (the new instances are underlined):

| | |
|---|---|
| Aircraft | $c_1, c_2, c_3, t_1, t_2, \underline{t_3}$ |
| Combat-aircraft | $c_1$('F270'), $c_2$('F271'), $c_3$('F401') |
| Type | 'bomber', 'fighter' |
| Bomber | $c_3$ |
| Fighter | $c_1, c_2$ |
| Weaponload | $w_1(4), w_2(6)$ |
| Bombload | $b_1(200)$ |
| Transport-aircraft | $t_1$('TR300'), $t_2$('TR301'), $\underline{t_3\text{('TR400')}}$ |
| Capacity | $tl_1$('2000 kg'), $tl_2$('2500 kg'), $\underline{tl_3\text{('3000 kg')}}$ |
| Combat-aircraft-typing | $\langle c_1, \text{'fighter'}\rangle$ , $\langle c_2, \text{'fighter'}\rangle$ , $\langle c_3, \text{'bomber'}\rangle$ |
| Fighter-weaponary | $\langle c_1, w_1\rangle$ , $\langle c_2, w_2\rangle$ |
| Bomber-weaponary | $\langle c_3, b_1\rangle$ |
| Transport-capacity | $\langle t_1, tl_1\rangle$ , $\langle t_2, tl_2\rangle$ , $\underline{\langle t_3, tl_3\rangle}$ |

Elements may also be deleted from a population. Suppose the capacity 2000 kg is to be deleted. In order to achieve this, the following statement can be used:

```
DELETE Capacity: '2000 kg'
```

This statement is, however, *not accepted* by the system. The removal of the capacity 2000 kg would lead to the deletion of the Transport-capacity instance: $\langle t_1, tl_1\rangle$, and thus leave transport aircraft TR300 without an associated capacity. This is in contradiction with the total role constraint for the capacity of a transport aircraft. The LISA-D Tool reports this looming violation by:

```
Command rejected due to constraint p3
   in schema Squadron.
```

The above discussed updates can be entered in the LISA-D-Tool as illustrated in figure 10 (page ).

As can be seen in the above example, the population resulting from an update statement may not fulfil all constraints. To avoid constraint violations, transactions are introduced. A transaction is a sequence of update statements, enclosed between START-TRANSACTION and END-TRANSACTION. The constraints then serve as invariant relations (i.e., pre- and post-conditions) for these transactions.

## 7 Conclusions and Further Research

In this paper the conceptual language LISA-D based on the data modelling technique PSM, has been introduced informally. In LISA-D constraints, queries and updates can be expressed in a way closely following the naming of roles and object types in the conceptual schema. This makes LISA-D statements (generally) easy to read and interpret intuitively.

Further research is necessary to establish the expressive power of LISA-D in relation to other query languages such as SQL or DataLog ([Ull89]), and to provide the language with a more powerful typing mechanism to support static semantic checks. Furthermore, research is being performed in the development of Elisa-D, a version of LISA-D supporting the (on line) evolution of information systems ([FOP92], [PW93], [PW94], [PW95], [Pro94]). Currently prototype implementations of LISA-D are being further developed, and used in an educational environment.

In order to improve the support of the query formulation process the suitability of a hypertext approach is being studied ([BPW93], [BW92], [Pro94]). When using such an approach, the user is able to navigate through the information structure (index), meanwhile formulating the information need (*Query By Navigation*).

## Acknowledgements

This work has been partially supported by:

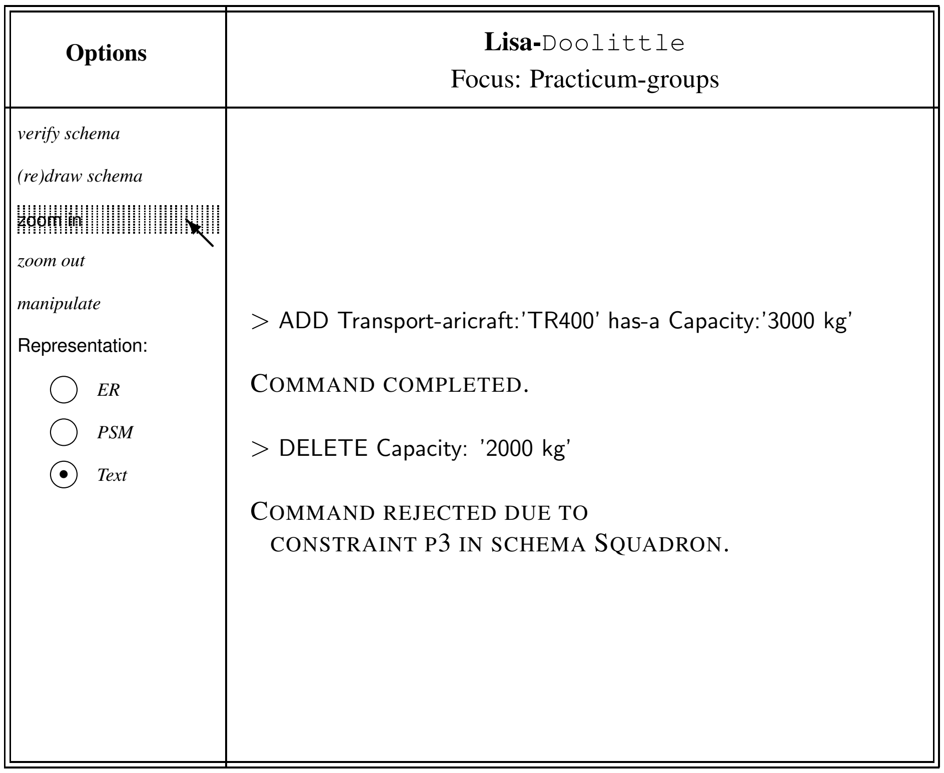
The Foundation for Computer Science in the Netherlands (SION) with financial support from the Netherlands Organization for Scientific Research (NWO), The Netherlands.

## References

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

[BELP87] T. Brus, M.C.J.D. van Eekelen, M. van Leer, and M.J. Plasmeijer. Clean - A Language for Functional Graph Rewriting. In *Proceedings of the Third International Conference for Functional Programming Languages and Computer Architectures (FPCA '87)*, volume 274 of *Lecture Notes in Computer Science*, pages 364–384, Portland, Oregon, 1987. Springer-Verlag.

[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.



ER PSM [✓ Text]

```
> ADD Transport-aricraft:'TR400' has-a Capacity:'3000 kg'

COMMAND COMPLETED.

> DELETE Capacity: '2000 kg'

COMMAND REJECTED DUE TO
  CONSTRAINT P3 IN SCHEMA SQUADRON.
```

Figure 10: Manipulating schemas

[BPW93] C.A.J. Burgers, H.A. Proper, and Th.P. van der Weide. Organising an Information System as Stratified Hypermedia. In H.A. Wijshoff, editor, *Proceedings of the Computing Science in the Netherlands Conference*, pages 109–120, Utrecht, The Netherlands, EU, November 1993.

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

[Che76] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[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.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1:75–116, 1985.

[FOP92] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. Evolving Information Systems: Beyond Temporal Information Systems. In A.M. Tjoa and I. Ramos, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA'92)*, pages 282–287, Valencia, Spain, EU, September 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3211824006

[Hal89] T.A. Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.

[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.

[HO92] T.A. Halpin and M.E. Orlowska. Fact-oriented modelling for data analysis. *Journal of Information Systems*, 2(2):97–119, April 1992.

[Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[HPW92] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. 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, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815

[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.

[Hub93] J.W.G.M. Hubbers. Automated Support for Verification & Validation of Graphical Constraints in PSM. Technical Report 93/01, Software Engineering Research Centre (SERC), Utrecht, The Netherlands, 1993.

[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.

[ISO87] *Information processing systems – Concepts and Terminology for the Conceptual Schema and the Information Base*, 1987. ISO/TR 9007:1987. `http://www.iso.org`

[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

[PEPW93] A.Th. Peters, M.C.J.D. van Eekelen, M.J. Plasmeijer, and Th.P. van der Weide. A case-tool for the development of information systems implemented in a pure functional language. Technical report, Computing Science Institute, Nijmegen, The Netherlands, EU, 1993.

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[PW93] H.A. Proper and Th.P. van der Weide. Towards a General Theory for the Evolution of Application Models. In M.E. Orlowska and M.P. Papazoglou, editors, *Proceedings of the Fourth Australian Database Conference*, Advances in Database Research, pages 346–362, Brisbane, Australia, February 1993. World Scientific, Singapore. ISBN 981021331X

[PW94] H.A. Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

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

[Sch84] G. Scheschonk. *Eine auf Petri-Netzen basier-en-de Konstruk-tion-s, Ana-ly-se und (Teil)Veri-fica-tion-s-me-tho-de zur Modellierungsunterstützung bei der Entwicklung von Informationssystemen*. PhD thesis, Berlin University of Technology, Berlin, Germany, 1984. (In German).

[Shi81] D.W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[Ull89] J.D. Ullman. *Principles of Database and Knowledgebase Systems*, volume I, chapter 3. Computer Science Press, Rockville, Maryland, 1989.

[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.

[Wij91] G.M. Wijers. *Modelling Support in Information Systems Development*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 1991. ISBN 9051701101

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

[You89] E. Yourdon. *Modern Structured Analysis*. Printice-Hall, Englewood Cliffs, New Jersey, 1989.