Information Disclosure in Evolving Information Systems: Taking a Shot at a Moving Target

Version of June 23, 2004 at 10:30

H.A. Proper[1] and Th.P. van der Weide

Department of Information Systems, University of Nijmegen
Toernooiveld, NL-6525 ED Nijmegen, The Netherlands
E.Proper@acm.org

PUBLISHED AS:

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.

Abstract

In this paper, we introduce a query language for evolving information systems. Evolving information systems go beyond the capacity of conventional database systems, not only as they incorporate a time dimension, but also since they allow all aspects of the system to evolve.

The introduced language is related to the philosophy underlying NIAM (Natural language Information Analysis Method). This method investigates the grammar of the communication in the Universe of Discourse. Usually this grammar is depicted as an information structure diagram (NIAM or ER schema).

This paper describes the language Elisa-D, which is based on this grammar. As a result, expressions in this language have a direct meaning in the universe of discourse, while natural language expressions are easily formalised in this language.

keywords: Evolving Information System, Conceptual Query Language EVORM ER, PSM, Elisa-D

1 Introduction

Flexible behaviour of an organisation may entail a rapidly changing information need, and therefore calls for more flexible information systems. This has led to the development of evolving information systems (see for instance: [MS90], [Rod91], [Ari91], [FOP92a], [JMSV92], [FOP92b], [PW93]). In an evolving information system not only the population can evolve, but structural aspects (see figure 1), such as the underlying information structure, constraints and action model, are allowed to evolve as well. Furthermore, the history of such evolutions should be maintained.

One of the most important aspects of information systems is their ability to support information disclosure, for example via a query language. For any kind of information system, the following requirement may therefore be postulated:

> *The system provides an adequate disclosure mechanism for the retrieval of* **all** *stored information.*

This mechanism may be based on a query language, but could also be based on a hypertext-like approach (see for example [BPW93] and [Pro94]). Stored information (see figure 1) refers to the application model as a whole. For evolving information systems, the history of all stored information (the application model) is also subject of discourse. The latter poses extra requirements on the disclosure mechanism. As the

[1] Currently at: Department of Computer Science, University of Queensland, Queensland 4072, Australia

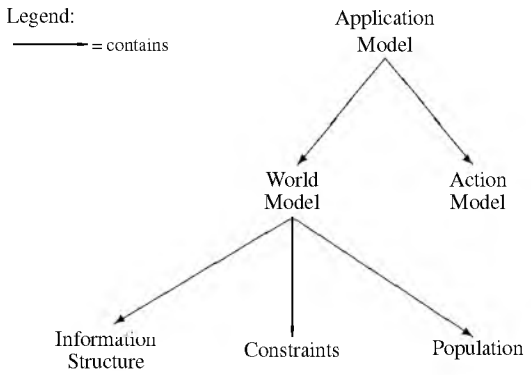


Figure 1: A hierarchy of models

underlying information structure of the stored information changes in the course of time (moving target), traditional query languages such as SQL are inadequate.

In some approaches to evolving information systems, a manipulation language for relational models is extended with historical operations, both on population and schema level (note that a schema is a description of an information structure) An example of this approach can be found in [MS90], in which an algebra is presented allowing relational tables to evolve by changing their arity. This direction is similar to the ORION project ([BKKK87], [KBC+89]), in that a manipulation language is extended with operations supporting schema evolution. However, it is not clear that these languages provide an adequate disclosure mechanism for the information stored in all versions of the underlying information structure, together with the populations conforming to these information structures.

Maintaining the history of an information system does make high demands, both on storage capacity and processing speed. With the increase of these both aspects, the feasability of evolving information systems will at most be a matter of time.

The following criteria are relevant for disclosure (query) languages:

L1 Expressiveness.
The expressiveness of a language provides a classification for the semantical richness of that language.

L2 Suitability.
Suitability addresses the match between the disclosure language and the concepts used in the universe of discourse.

L3 Elegance.
The elegance of a language is concerned with the ease with which searchers can formulate their information need.

L4 Supportiveness.
What (automated) support can be provided to the searcher in formulating an information need.

In this paper, we mainly address the suitability and elegance criteria. We discuss the Elisa-D manipulation language for Evolving Information Systems. In this paper we restrict ourselves to the query abilities of Elisa-D. The update aspects of the Elisa-D language are discussed in full detail in ([Pro94]). Expressiveness will only be addressed briefly. The supportiveness issue has been studied in [BPW93], where a two-level hypermedia approach, an interactive process of query formulation, called *query by navigation*, has been described.

Object modelling techniques can be introduced as modelling techniques that model a universe of discourse via objects and the roles that they play. Some examples are: ER ([Che76]), EER ([HE92], [EGH+92]), NIAM ([NH89], [Win90]), and FORM ([Hal89], [Hal92], [HO92]). The data modelling technique EVORM (EVolving Object Role Model, see [PW94]) has been introduced as a variant of object role

modelling in general, which supports evolution. This modelling technique is an extension of the object role modelling technique PSM ([HW93], [HPW92]).

The Elisa-D language is defined in terms of the EVORM modelling technique for evolving application domains. Due to the generality of EVORM, the language construction which is proposed in the next sections can be used for a wide range of modelling techniques. Since the focus of this article is not the introduction of the EVORM modelling technique, but rather the introduction of Elisa-D, only a brief discussion of EVORM is required to make the paper self-contained. Elisa-D is based on the language LISA-D ([HPW93]). The language allows for the formulation of queries (and constraints), in the form of sentences, which have a direct meaning in the universe of discourse. Other database languages applying the NIAM style of verbalisation of concepts, in order to obtain fluent sentences are: FORML ([HH93]) and RIDL ([Mee82]).

The structure of the paper is as follows. First, in section 2, we shortly summarize the concepts of the data modelling technique EVORM. Section 3 addresses the main properties of EVORM, which are relevant in the context of evolving information systems. In section 4, we continue with the introduction of the notion of *disclosure schema*, which serves as a key concept in a good disclosure of the information stored in an evolving information system. In section 5, we introduce the language Elisa-D. In section 6 we give a number of examples. Finally, section 7 contains a number of conclusions.

2 The Modelling Concepts

In this section we summarize the concepts of EVORM (object types and predicators), and describe how the set $\mathcal{O}$ of object types is constructed from basic data types. Appendix 7 contains an overview of the graphical conventions.

The atomic data types of EVORM are label types and entity types. Label types correspond to concrete objects, and are used to concretize (aspects of) abstract object types, such as entity types. Let $\mathcal{L}$ be the set of label types. As label types are concrete, they are assumed to be directly denotable. The denotation depends on the actual assignment of concrete domains $\mathcal{D}$ (such as integer, real and string) to label types: $\mathsf{Dom} : \mathcal{L} \rightarrow \mathcal{D}$. Entity types ($\mathcal{E}$) are basic abstract object types. The following type constructors can be used to form composed abstract object types:

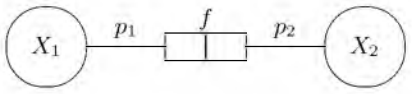


Figure 2: A sample fact type

1. *Factification*
   Fact types describe relations between object types as mappings from an underlying domain of predicators (roles) into these object types. Each fact type has its unique predicators. We identify a fact type by its underlying set (domain) of predicators. For example, in figure 2, we see fact type $f = \{p_1, p_2\}$.

   Let $\mathcal{P}$ be the set of all predicators. The function $\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$ assigns to each predicator its associated object type. Furthermore, the function $\mathsf{Fact} : \mathcal{P} \rightarrow \mathcal{F}$ provides the fact type, associated with each predicator. In figure 2, we have $\mathsf{Base}(p_1) = X_1$ and $\mathsf{Fact}(p_1) = f$

2. *Power Typing*
   Power typing is a type constructor which leads to an object type that is instantiated with subsets over an underlying object type. Let $\mathcal{G}$ be the set of all power types. The function $\mathsf{Elt} : \mathcal{G} \rightarrow \mathcal{O}$ provides the underlying element type of power types. In figure 6, Airforce is an example of a power type, over element type Squadron.

3. *Sequence Typing*
   Sequences of an underlying object type are formed by sequence typing. Let $\mathcal{S}$ be the set of all sequence types. The function Elt : $\mathcal{S} \rightarrow \mathcal{O}$ provides the underlying element type of sequence types.

4. *Schema Typing*
   Schema typing provides the opportunity to describe an information structure in a top-down fashion. As a result, schema typing provides a mechanism for schema decomposition (for example schema type Squadron in figure 6). On the other hand, in some cases, schema types sometimes force themselves, as in the data model of figure 3. Schema types are instantiated by populations of the underlying schema. Let $\mathcal{C}$ be the set of all schema types, then the relation $\prec \subseteq \mathcal{C} \times \mathcal{O}$ describes the decomposition relation.

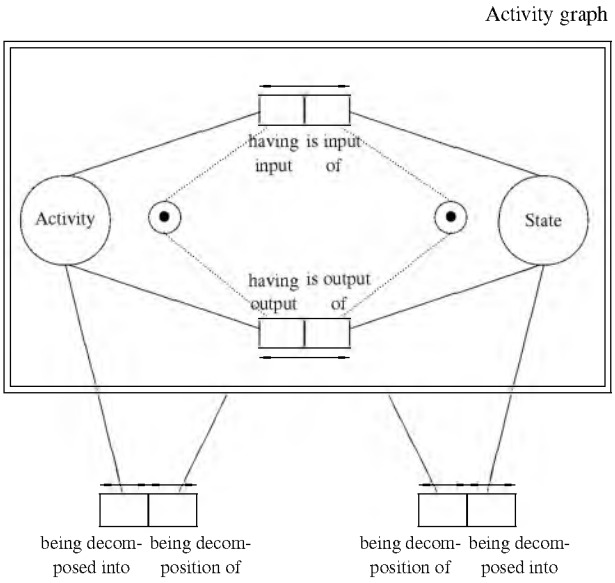


Figure 3: An information structure diagram for Activity Graphs

Besides these type constructors, new object types can be introduced by specialisation and generalisation of other object types. The partial order Spec $\subseteq \mathcal{O} \times \mathcal{O}$ contains the specialization hierarchy, generalization is captured by the relation Gen $\subseteq \mathcal{O} \times \mathcal{O}$. Both relations can be seen as type construction by abstraction. In the case of a generalisation the generalised object type abstracts from the properties specific to the specifiers of the generalisation, whereas in the case of specialisation the supertype allows for a similar abstraction. It should be noted, however, that generalisation and specialisation are distinct concepts which result from differing axioms in set theory ([HW93]) The resulting object types inherit their identification from the abstracted object types.

In the example of figure 4, the object type Real estate is a specialization of object type Product. Object type Product is a generalization of object types Boat and House. As a result, Real estate inherits its identity from Product, which on its turn derives its identity from its specifiers Boat and House. The partial order IdfBy $\subseteq \mathcal{O} \times \mathcal{O}$ on object types, captures the inheritance hierarchy, and results from specialisation and generalisation. For example figure 4 this leads to:

- Real-estate IdfBy Product
- Product IdfBy Boat
- Product IdfBy House.

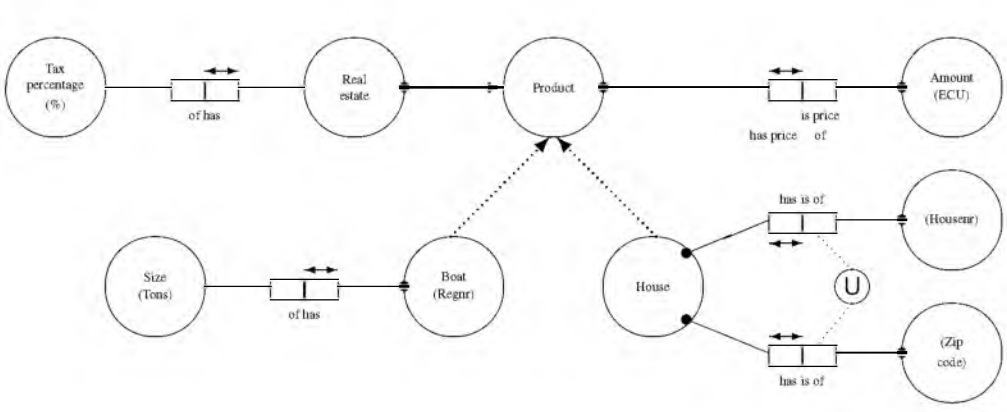


Figure 4: A data model with generalisation and specialisation

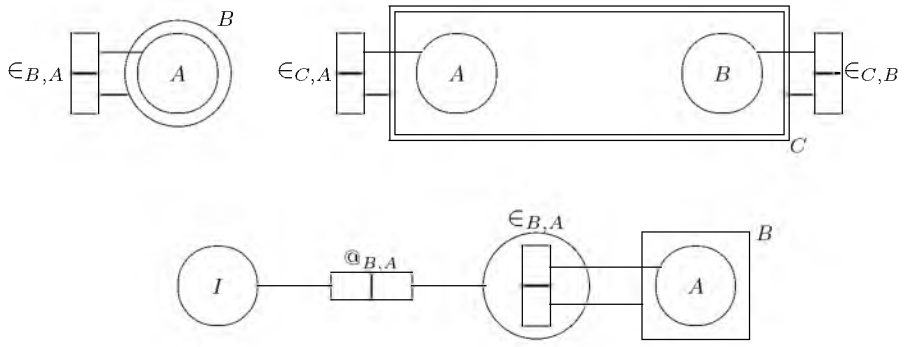


Figure 5: Default names for anonymous concepts

The elementary operators for accessing composed abstract object types are introduced as special fact types, and summarised in figure 5. Usually, these special fact types are omitted from the diagram. Henceforth, we will refer to them as *implicit*. The relation between a power type B and its element type $A = \mathsf{Elt}(B)$ is recorded in the fact type $\in_{B,A}$. As a consequence, power typing corresponds to a polymorphic type constructor, and the fact type $\in_{B,A}$ to an associated polymorphic access operator. The relation between a sequence type B and its element type $A = \mathsf{Elt}(B)$ is recorded by the implicit fact type $\in_{B,A}$. Contrary to power types, this relation $\in_{B,A}$ is augmented with the position of the element in the sequence, via $@_{B,A}$. The object type I is the domain for indexes in sequence types. Usually the natural numbers are used for this purpose. With each schema type C and each object type A in its decomposition, the implicit fact type $\in_{B,A}$ is associated, facilitating the transition from an object to an object from its decomposition.

As an example of an EVORM schema, we consider air-forces and their relation to political entities. This schema depicts a snapshot of a possibly evolving domain. An air-force consists of a set of squadrons, and is assigned to a political entity. For instance the RAF is the air-force 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 1st ATAF (First Allied Tactical Air-force) is an air-force 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. The resulting schema is displayed in figure 6. Besides, this figure depicts the look and feel of the Elisa-`Doolittle` tool (see also [HPW94]). For an explanation of the graphical symbols used, see the appendix. The button $\bigtriangledown$ in this figure signifies object type Squadron as a schema type. Clicking this button will zoom in on schema type Squadron (see figure 7).

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-nr and a C-nr 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

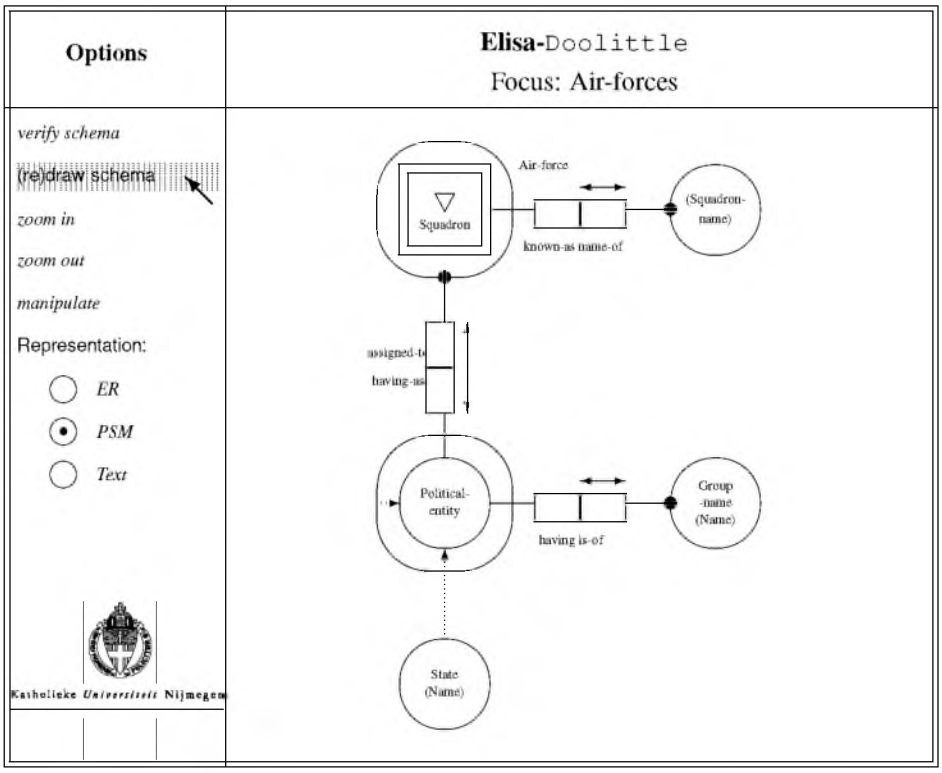


Figure 6: Air Force squadrons

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 7

3 EVORM: Summary of Formalisation

In this section, we provide a short overview of the formal background of EVORM. A more elaborate formal treatment can be found in [PW94]. The complete formalisation consists of several classes of axioms (see figure 8). The typing mechanism is captured by the set of rules (ISU), and forms the basis for version management, leading to the set of rules (AMV) describing wellformedness of versions. Version management, on its turn, serves together with a time axis $\mathcal{T}_s$ as the base for rules (EW) describing what constitutes a wellformed evolution of an information system.

In [PW94] a general theory for evolving application domains has been applied to the data modelling technique PSM, resulting in EVORM. In this paper, we summarise the relevant EU and TR axioms.

3.1 General properties

Label types, entity types, fact types, sequence types and schema types are all interpreted differently:

[EU1] $\mathcal{L}, \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{S}, \mathcal{C}$ form a partition of $\mathcal{O}$.

Bridge types establish the connection between abstract and concrete object types. The term $\mathsf{Bridge}(f)$ qualifies fact type f as a bridge type, and is an abbreviation for the expression

$$\exists_{p,q}\left[f = \{p, q\} \wedge \mathsf{Base}(p) \in \mathcal{L} \wedge \mathsf{Base}(q) \notin \mathcal{L}\right]$$

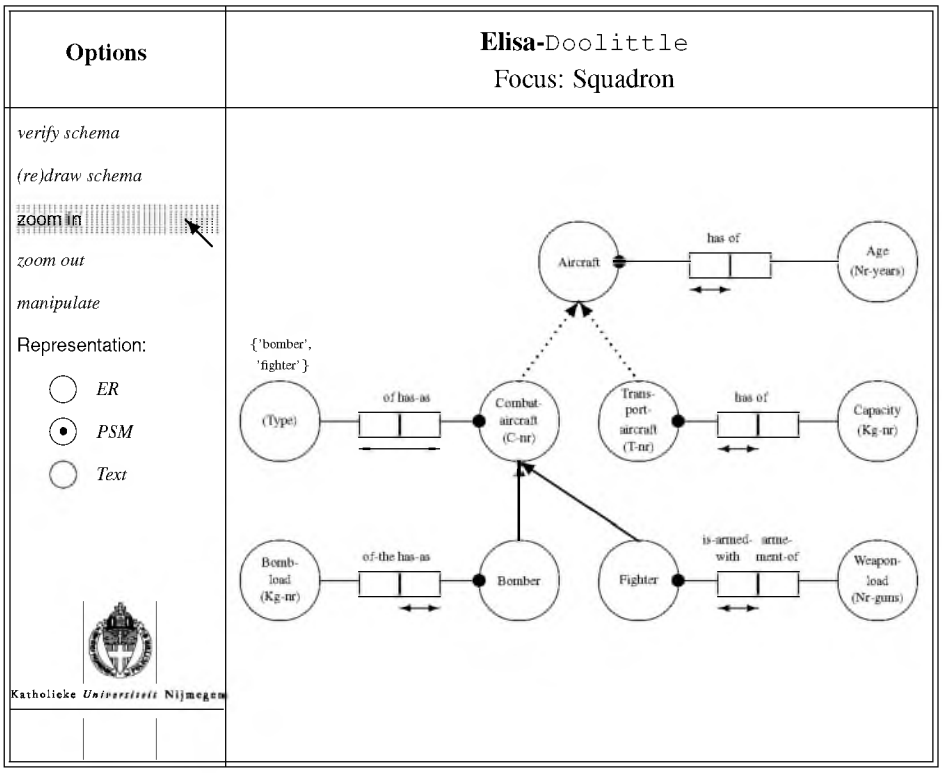


Figure 7: Zoomed in on Squadron

EVORM makes a strict distinction between concrete values, and objects that are percieved as corresponding to an object (without an inherent denotation) in the universe of discourse. The rationale of this distinction is that it may help the system analyst during modelling. The strict separation between the concrete and abstract level is expressed by the following rules. Firstly, label types may only participate in bridge types:

[EU2] $\mathsf{Base}(p) \in \mathcal{L} \Rightarrow \mathsf{Bridge}(\mathsf{Fact}(p))$

Secondly, label types are not allowed to occur as element type of either a power or sequence type:

[EU3] $\mathsf{Elt}(x) \notin \mathcal{L}$

3.2 Identification hierarchy

Object types, that result from construction by abstraction, inherit the structure of their parents in the identification hierarchy. As a result, they do not provide a structure of their own; they have to be basic types. Consequently, such object types are entity types. Furthermore, the strict separation between concrete and abstract object types is extended to the identification hierarchy.

[EU4] (*strictness*) $\mathsf{IdfBy} \subseteq \mathcal{E} \times (\mathcal{O} \setminus \mathcal{L}) \cup \mathcal{L} \times \mathcal{L}$

Note that subtyping of objectified fact types is allowed in this setup, but that these subtypes are regarded as entity types. In order to avoid identification conflicts, the identification hierarchy must form a partial order:

[EU5] (*asymmetry*) $x\,\mathsf{IdfBy}\,y \Rightarrow \neg y\,\mathsf{IdfBy}\,x$

[EU6] (*transitivity*) $x\,\mathsf{IdfBy}\,y\,\mathsf{IdfBy}\,z \Rightarrow x\,\mathsf{IdfBy}\,z$

We define IdfBy_1 as the one step version of IdfBy:

$$x\,\mathsf{IdfBy}_1\,y \triangleq x\,\mathsf{IdfBy}\,y \wedge \neg\exists_z\,[x\,\mathsf{IdfBy}\,z\,\mathsf{IdfBy}\,y]$$

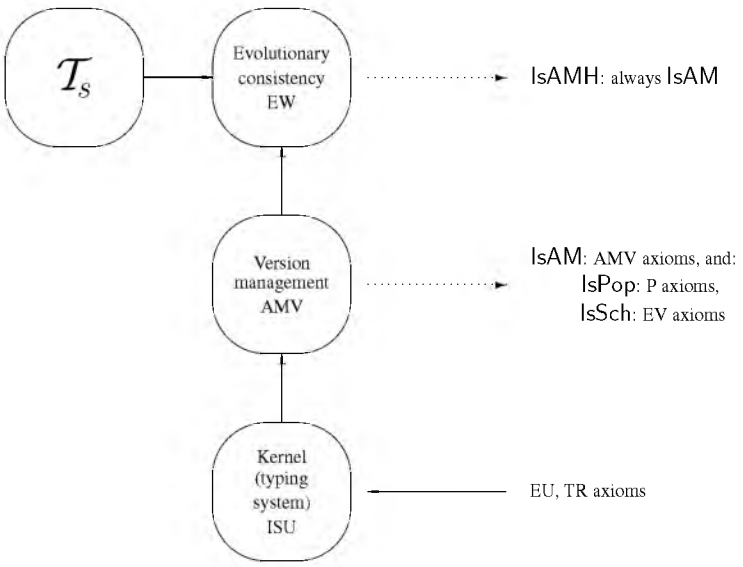


Figure 8: Axiomatic framework

In EVORM, all object types in the identification hierarchy have direct ancestors:

[EU7] (*direct ancestors*)

$$x \,\mathsf{IdfBy}\, y \Rightarrow x \,\mathsf{IdfBy}_1\, y \vee \exists_p \,[x \,\mathsf{IdfBy}_1\, p \,\mathsf{IdfBy}\, y]$$

The finite depth of the identification hierarchy in EVORM is expressed by the following schema of induction:

[EU8] (*identification induction*)

If F is a property for object types, such that for any x

$$\forall_{y:x \,\mathsf{IdfBy}_1\, y} \,[F(y)] \Rightarrow F(x)$$

then $\forall_{x \in \mathcal{O}} \,[F(x)]$

The identification hierarchy is a result of specialisation and generalisation:

[EU9] (*complete span*)

$$x \,\mathsf{IdfBy}\, y \iff x \,\mathsf{Gen}\, y \vee x \,\mathsf{Spec}\, y$$

First we focus in some more detail on specialisation. Specialisation itself should also form a partial order on object types. This is enforced by the following rule:

[EU10] (*transitivity completeness*)

If $x \,\mathsf{IdfBy}\, y \,\mathsf{IdfBy}\, z$ then:

$$x \,\mathsf{Spec}\, y \,\mathsf{Spec}\, z \iff x \,\mathsf{Spec}\, z$$

Note that the asymmetry of Spec follows from the asymmetry of IdfBy. For specialisation hierarchies, we define the *pater familias* relation $\sqcap(x, y)$, characterizing y as pater familias of x. This relation represents the root relation for inheritance when restricted to specialisation. The pater familias relation is identified by:

$$\sqcap(x, y) \triangleq (x \,\mathsf{Spec}\, y \vee x = y) \wedge \neg \,\mathsf{spec}(y)$$

where $\mathsf{spec}(x)$ is a shorthand for $\exists_y \,[x \,\mathsf{Spec}\, y]$. In a specialisation hierarchy, contrary to generalisation hierarchies, there is always one unique top element. This is stipulated by the following axiom:

[EU11] (*unique pater familias*)

$$\sqcap(x, y) \wedge \sqcap(x, z) \Rightarrow y = z$$

As a result of this, we can write $\sqcap(x)$ to denote the pater familias of object type x.

Next we focus in some more detail on generalisation. Generalisation should also form a partial order on objects:

[EU12] (*transitivity completeness*)
If x IdfBy y IdfBy z then:

$$x \,\mathsf{Gen}\, y \,\mathsf{Gen}\, z \iff x \,\mathsf{Gen}\, z$$

In the sequel $\mathsf{gen}(x)$ will be used as an abbreviation for $\exists_y [x \,\mathsf{Gen}\, y]$. Generalisation and specialisation can be conflicting due to their inheritance structure. To avoid such conflicts, generalised object types are required to be pater familias:

[EU13] $\mathsf{gen}(x) \Rightarrow \neg\, \mathsf{spec}(x)$

3.3 Type relatedness

Intuitively, object types can, for several reasons, have values in common in some instantiation of the information structure. For example, each value of object type x will, in any instantiation, also be a value of his pater familias $\sqcap(x)$. As another example, suppose $x \,\mathsf{Gen}\, y$, then each value of y will, in any population, also be a value of x. A third possibility of sharing values arises for power types, when their underlying element types may share values.

Formally, type relatedness is captured by a binary relation $\sim$ on $\mathcal{O}$. Two object types are type related if and only if this can be proven from the following derivation rules:

[TR1] $x \in \mathcal{O} \vdash x \sim x$

[TR2] $x \sim y \vdash y \sim x$

[TR3] $y \,\mathsf{IdfBy}\, x \wedge x \sim z \vdash y \sim z$

[TR4] $x, y \in \mathcal{G} \wedge \mathsf{Elt}(x) \sim \mathsf{Elt}(y) \vdash x \sim y$

[TR5] $x, y \in \mathcal{S} \wedge \mathsf{Elt}(x) \sim \mathsf{Elt}(y) \vdash x \sim y$

[TR6] $x, y \in \mathcal{C} \wedge \mathcal{I}_x = \mathcal{I}_y \vdash x \sim y$

where $\mathcal{I}_x$ denotes the information structure into which schema type x is decomposed.

3.4 Evolution of EVORM schemata

The evolution of an application domain is modelled in EVORM as a set of functions $H \subseteq \mathcal{T} \rightarrowtail \mathcal{AME}$, where $\mathcal{T}$ is a time axis, and $\mathcal{AME}$ the set of all application model elements (see figure 11 for a meta schema). The set $\mathcal{AMH}$ in this schema is the domain for all such evolutions. Examples of application model elements are the object types $\mathcal{O}$ and their instantiations $\wp(\mathcal{O}) \times \Omega$. An instantiation of an object type assigns a population to object types. The underlying domain for populating object types is the set Ω. For a more elaborate definition of Ω, please refer to [PW93], [PW95] or [PW94]. Such a set of functions H is referred to as an application model history. In H, each function $h \in H$ exclusively describes the evolution of some application domain concept, and is referred to as an (application model) element evolution. The state of affairs of history H at some point of time t (the version at time t) is derived as follows. The object types which are available in the version at time t are found by

$$\mathcal{O}_t = \{ h(t) \mid h \in H \wedge h(t) \in \mathcal{O} \}$$

The populations of these object types are derived from instance evolutions. Instance evolutions describe the relation between values and object types in the course of time. The population is then derived from the instance evolutions as follows:

$$\mathsf{Pop}_t(x) = \{v \mid \exists_{h \in H}\, [h(t) \in \wp(\mathcal{O}) \times \Omega \wedge h(t) = \langle X, v\rangle \wedge x \in X]\}$$

In the remainder of this subsection, we focus on well-formedness of schema versions defined by $\mathcal{O}_t$. In the next subsection, focus is on versions of populations.

Let $\mathcal{O}_t$ be a set of object types spanning an information structure version. From this version, we derive the EVORM information structure:

$$\mathcal{I}_t = \langle \mathcal{F}_t, \mathcal{G}_t, \mathcal{S}_t, \mathcal{C}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{P}_t \rangle$$

The set of fact types in the EVORM information structure version is defined as $\mathcal{F}_t = \mathcal{F} \cap \mathcal{O}_t$. The other components are derived analogously. The set of predicators, on the other hand, is defined as: $\mathcal{P}_t = \bigcup \mathcal{F}_t$. On the thus obtained information structure versions, and their populations, some well-formedness rule should hold. Some of these are technique independent, and some of them are EVORM dependent. An elaborate discussion on these well-formedness axioms for information structure versions and their populations can be found in [PW94].

4 Deriving the Disclosure Schema

The main purpose of an information system is supporting information retrieval. Retrieving information from an evolving information system is inherently richer than retrieving information from a traditional information system, due to the availability of a time axis both for the population and the schema. Therefore, a language supporting evolving information systems, should provide the opportunity to query:

- the current state of the system (both population and structure)
- previous states of the system (both population and structure), and their relations

We provide a synergy of the several aspects at three levels of abstraction, leading to the *disclosure schema*. The disclosure schema is the point of departure for the introduction of the way of communicating for an evolving information system.

4.1 The extra-temporal schema

In this section we start by introducing the extra-temporal schema of an application model. The extra-temporal information structure is defined as the union of all versions described by application model history H: $\mathcal{O}_\infty = \bigcup \mathcal{O}_t$.

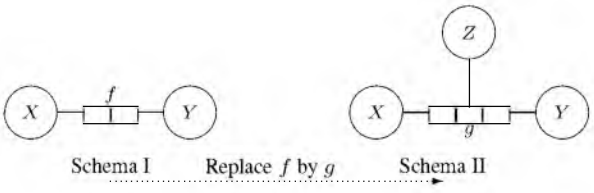


Figure 9: Evolution of schemas

As an illustration of an extra-temporal information structure, consider the two schema versions in figure 9. In this figure, the evolution from a binary fact type (f) to a ternary fact type (g) is modelled. The associated extra-temporal information structure is depicted in figure 10.

Constraints associated to versions of the application model, will not necessarily hold for the extra-temporal population. For evolutionary information systems, constraints must be enforced in their proper temporal context. Because of this, the extra-temporal schema is the same as the extra-temporal information structure.

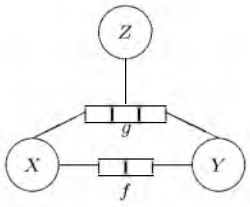


Figure 10: A simple extra-temporal schema

4.2 The disclosure schema

In an evolving information system, the user may want to query past and present of the application model as a whole. Besides, querying is not limited to the population, but may involve such things as the information structure and activity specifications as well. We approach this problem by introducing a disclosure schema, comprising the extra-temporal schema, and the meta schema of the used modelling techniques. The disclosure schema will be defined gradually via a number of subschemas (stepwise refinement). In figure 11, the overall meta schema of the evolution theory from [PW93], [PW95] is provided. Note that this schema also contains constraints ($\mathcal{R}$) and methods ($\mathcal{M}$), which are not discussed in this text.

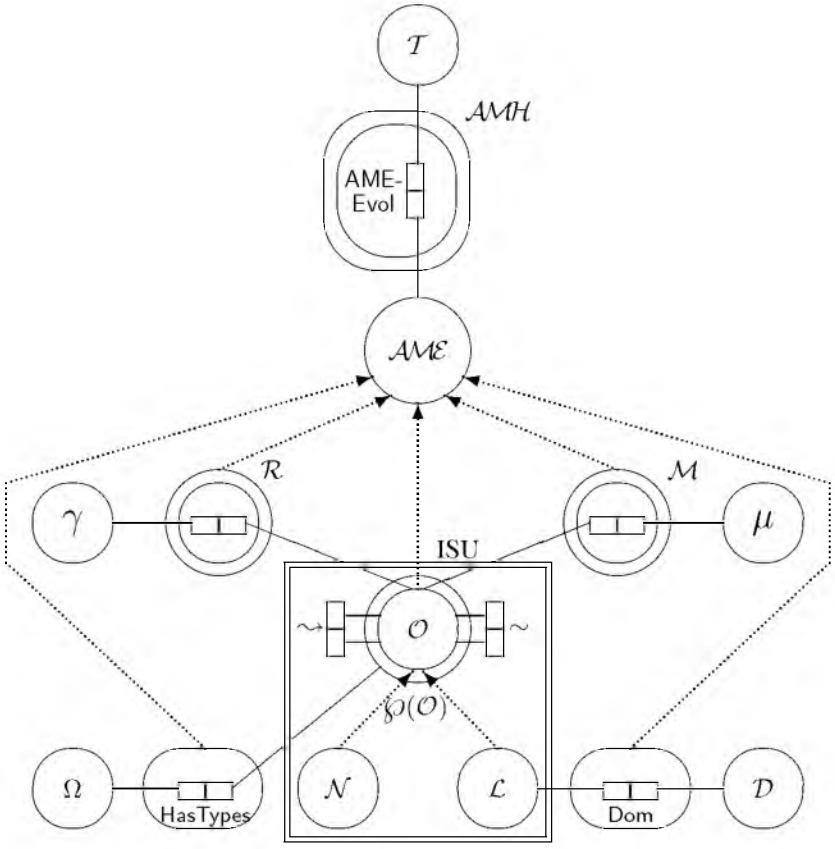


Figure 11: The meta schema for the general evolution theory

As a first refinement, this schema is augmented with a set of (derivable) relations, enabling a convenient formulation of queries involving time (see figure 12 (page 12)). For example, the fact type Change keeps track of all changes undergone by histories $h \in H$ as triples $\langle h(t), h(\triangleright t), t\rangle$. Note that $\triangleright t$ denotes the point of time t increased with the minimal time unit (sometimes referred to as a chronon). A change occurs at

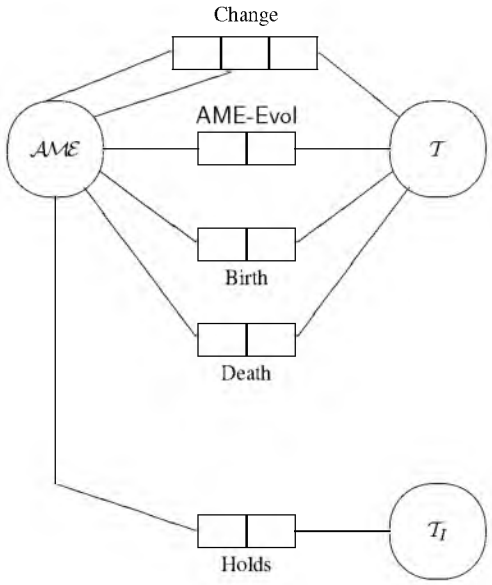


Figure 12: Extension with temporal relations

time t in history h, if h is alive at points of time t and $\triangleright t$, but in different shapes:

$$\mathsf{Pop}_t(\mathsf{Change}) \quad = \quad \left\{ \langle h(t), h(\triangleright t), t \rangle \;\middle|\; h \in H \;\wedge\; h{\downarrow}t, \triangleright t \;\wedge\; h(t) \neq h(\triangleright t) \right\}$$

where the notation $h{\downarrow}t$ is used to express that function h is defined in point t. The population of the fact types Birth and Death can be derived analogously. Fact type Holds relates application model elements to time intervals as follows. Each shape $h(t)$ of application model evolution h is related to the (maximal) time interval T during which this shape holds. Generally, we call a time interval T maximal with respect to predicate P, denoted as $\mathsf{MaxDur}(T, P)$ if:

1. P holds during T, or: $\forall_{t \in T}\,[P(t)]$.
2. no larger interval T' has this property:

$$T \subseteq T' \wedge \forall_{t \in T'}\,[P(t)] \Rightarrow T = T'$$

The population of Holds at time t then is defined as:

$$\mathsf{Pop}_t(\mathsf{Holds}) \quad = \quad \left\{ \langle h(t), T \rangle \;\middle|\; h \in H \wedge T \in \mathcal{T}_I \wedge h{\downarrow}t \wedge \mathsf{MaxDur}(T, P_t) \right\}$$

where P_t is defined as: $P_t(s) = \exists_{g \in H}\,[h(t) = g(s)]$, and $h{\downarrow}t$ signifies that function h is defined for t. Note that T depends on t via MaxDur and the definition of P_t.

On $\mathcal{T}$ and $\mathcal{T}_I$ more relations can be defined. Three classes of operations can be identified ([RP92]). Relations over $\mathcal{T}$, such as BEFORE, and AFTER, capture comparison relations for points of time, and form the first class. Two comparison relations for points of time, which are based on $\triangleright$, are defined as:

$$t_1 \text{ HAS INCREMENT } t_2 \iff \triangleright\, t_1 = t_2$$

$$t_1 \text{ IS INCREMENT OF } t_2 \iff t_1 = \triangleright t_2$$

The second class of comparison relations deals with time intervals ($\mathcal{T}_I$), and includes: BEFORE, and OVERLAPPED-BY. Comparing points of time with time intervals is the objective of the third class. For example, interval I CONTAINS point of time t. The semantics of these relations, depends on the chosen time axis $\mathcal{T}$, and will therefore not be elaborated upon.

Next, the meta schema is refined for the EVORM data modelling technique in figure 13, by being more specific about the $\mathcal{O}, \mathcal{N}, \mathcal{L}$ triplet and filtering the $\rightsquigarrow$-relation. Note that in the general evolution theory as discussed in [PW94], $\rightsquigarrow$ is the general theoretic counterpart of IdfBy which is defined as: $x \rightsquigarrow y \iff y$ IdfBy x. In figure figure 13, a set of EVORM specific (meta) object types are associated to the $\mathcal{O}$ (meta) object type, providing a technique dependent extension of the general meta model. When a concrete modelling technique has been chosen for the modelling of constraints and methods, the meta model of the general theory can be augmented with more details on constraints and activities as well.

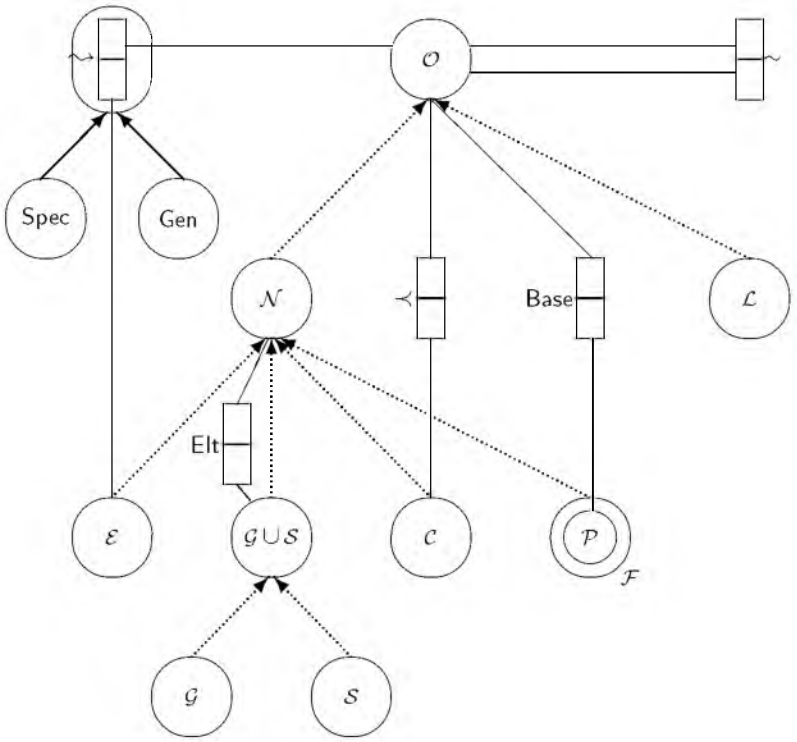


Figure 13: The EVORM meta model

The extra-temporal schema of an evolution forms a population of the $\mathcal{O}, \mathcal{N}, \mathcal{L}$ (meta) triplet. The disclosure schema, however, contains the extra-temporal schema also as a sub schema. In figure 14, the connection between the extra-temporal schema of figure 10, and the meta schema is provided as a sub schema. In general, the Gen relation of the disclosure schema is extended with x Gen Ω for all object types $x \in \mathcal{O}_\infty$.

4.3 Discussing the three levels

In the above discussion leading up to the disclosure schema, three levels can be distinghuished (see figure 15). The *instance level* is concerned with the extra-temporal population. The extra-temporal schema forms the second level, the so-called *schema level*. The top level, the *meta schema level*, completes the disclosure schema, and adds the possibility to look beyond temporal boundaries, by taking the temporal relation into account.

The instance level is related to the schema level through the HasTypes relation, while the relation between the schema level and the meta schema level is formed by the generalisation relation.

The three distincted abstraction levels, may be regarded as three layers in a multi-layered hypermedia system ([BW92]). This observation enables the disclosure of the information contained in an evolving information system by means of a hypertext approach. This approach has been studied before in [BPW93].

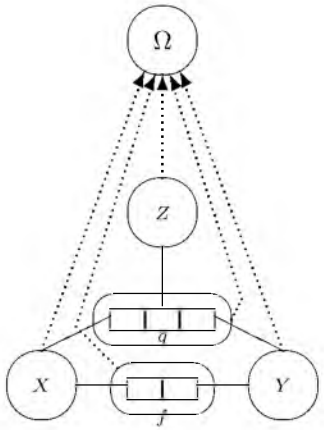


Figure 14: Extra-temporal schema linked to meta schema

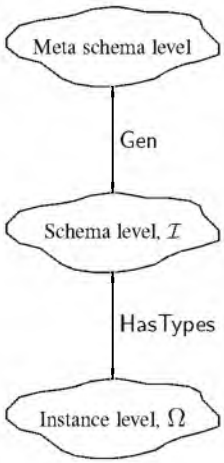


Figure 15: Three levels of abstraction

Such an approach seems to be essential for evolving information system, as the underlying information structure (second level) changes regularly. We motivate this briefly.

For users of a traditional information system, it is already difficult to maintain an overview of the information structure of the stored information. This has led to the idea of building a hypertext browser for such traditional information systems, enabling an improved information disclosure. This browser allows for *query by navigation*, i.e. building a query whilst navigating through the information structure. Similar browsers have also proved their usefulness in CASE-Tools ([Big88], [GS90], [Hag92]). When an evolving information system changes rapidly, it is hardly possible for most users (and information analysts) to keep track of the kind (structure) of the stored information of the past and present.

5 Elisa-D

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. For more details, see [NH89], [CW93].

It is only natural that the language for manipulating and querying also has the format of a semi natural language, and is designed to approximate the expert language as close as possible (*suitability*, L2, see page

2). The rationale behind this has also been addressed in [HH93]. As a result, sentences in this (artificial) language are meaningful expressions within the context of the U∘D, understandable and expressible by domain experts (contributing to *elegance*, L3). The sentences, verbalising the original examples, form extensional specifications, while queries (in general) correspond to intentional specifications.

In this section, an introduction to such a language, Elisa-D, based on the data modelling technique EVORM, is presented. As EVORM generalises Object-Role Modelling techniques such as FORM, NIAM and PSM, as well as ER based modelling techniques, Elisa-D is generally applicable."

5.1 The structure of Elisa-D

In the previous sections, the elements constituting an application model were introduced as abstract concepts (the way of modelling). The intention of this section is to describe a language (way of communicating) by which the U∘D's walk of life can be communicated (by human beings) to the information system. This language is set up as a (semi) natural language, resembling the expert language. Typical for a natural language is the richness to form sentences, even sentences that have no intuitive meaning, or sentences that are ambiguous.

The language should be such that it allows for an elegant description of a user's information need (L3). This does not imply the exclusion of inelegant descriptions independently of subjective ideas of elegance!

Elisa-D provides a set of grammar rules. Complemented with a concrete lexicon, as obtained from a particular application model, a concrete information retrieval and update language is obtained. 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. Besides, the concrete language can be further tuned to the application domain by extending the grammar with new rules. (Note that conventional programming languages use the procedure mechanism for this purpose.)

As a result, Elisa-D defines a class of languages, where each language is based on a particular underlying lexicon, and a set of grammar rules. In the remainder of this subsection, we describe the naming conventions used, and provide the meaning (semantics) of these names.

The main concept in Elisa-D is *information descriptor*, by which information needs can be formulated. An information descriptor is interpreted as a special history, describing the evolution of the result of this descriptor in the course of time. At each moment, this result comprises a binary relation in the mathematical sense (for a motivation for the choice of a binary relation, see [HPW93]). The semantics of information descriptor I are described by the function

$$\rho : \text{Information descriptor} \rightarrow (\mathcal{T} \rightarrowtail \wp(\Omega^2))$$

The semantics of I at point of time t are denoted as $\rho_t(I)$. Thus, $\rho(I) = \lambda t.\rho_t(I)$. The use of the single underlying domain $\mathcal{T} \rightarrowtail \wp(\Omega^2)$ facilitates the concatenation of arbitrary information descriptors. This will result in a liberal, orthogonal syntax for Elisa-D.

Note that LISA-D's semantics has been expressed in terms of binary multisets (bags), whereas Elisa-D's semantics is expressed in terms of binary sets. The motivation for this deviation lies in the easier presentation of the ideas behind Elisa-D. A concrete implementation will indeed be based on multisets. Furthermore, in [Pro94] the full definition of Elisa-D is provided.

5.2 Atomic information descriptors

The basis for Elisa-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 partial naming function

$$\mathsf{Lex} : (\mathcal{T} \rightarrowtail \mathcal{O}) \times (\mathcal{T} \rightarrowtail \mathcal{O}) \times \mathsf{Names} \rightarrowtail (\mathcal{T} \rightarrowtail \wp(\Omega^2))$$

where Names is a set of names. The notation $\mathsf{Lex}(g, h, n)\downarrow$ is used to indicate that $\mathsf{Lex}(g, h, n)$ is defined for object type evolutions g, h and name n. The name n then can be used as a denotation for an instance evolution, which connects, at each point t of time, instances of $g(t)$ to instances of $h(t)$. In this case, name n is qualified as a *defined name*.

Abstract name	*Concrete name*
$\mathcal{T}$	Time
$\mathcal{AMH}$	Amh
AME-Evol	Ame evolution
$\mathcal{AME}$	Ame
$\mathcal{O}$	Object type
$\wp(\mathcal{O})$	Types
$\mathcal{N}$	Non label type
$\mathcal{L}$	Label type
$\rightsquigarrow$	Parent
$\sim$	Type related
Ω	Value
HasTypes	Instance typing
$\mathcal{D}$	Domain
Dom	Domain assignment
$\mathcal{T}_I$	Time interval
Holds	Holds
Change	Change
...	...
$\mathcal{E}$	Entity type
$\mathcal{F}$	Fact type
$\mathcal{P}$	Predicator
...	...

Table 1: Names for meta object types

If a name n is overloaded (multiply defined by the Lex-function), then the meaning of the name is the sum of all possible interpretations:

$$\rho(n) = \bigcup_{\mathsf{Lex}(g,\,h,\,n)\downarrow} \mathsf{Lex}(g, h, n)$$

where the union operator on sets is extended in the obvious way: $p \cup q = \lambda t.p(t) \cup q(t)$.

The function Lex is filled with a number of predefined names. A first class of names verbalises the highest level of abstraction (see figure 15), and provides names for the meta concepts at this level. These names are provided by the function ONm : $\mathcal{O}^M \rightarrowtail$ Names, where $\mathcal{O}^M$ is the set of all object types in the meta schema. This function is summarised in table 1. The meta level consists of:

1. the meta schema for the general evolution theory, see figure 11,
2. the additional temporal aspects as depicted in figure 12,
3. the EVORM-specific details, as contained in figure 13.

The second class of names covers the extra temporal schema level, and provides names for application specific concepts. This level consists of:

1. Names for element evolutions are recorded in the function HNm : $\mathcal{AMH} \rightarrowtail$ Names.
2. Object types ($\mathcal{O}$) are referenced by a unique name, provided by the function ONm. (Note that this function is also used for the naming of meta object types).
3. Names of domains are found by means of the function: DNm : $\mathcal{D} \rightarrowtail$ Names.

Note that γ and μ are left out of consideration in this paper. The naming of instances (Ω) consists of a denotation mechanism for label values (constants), and a naming mechanism for abstract values. This

latter mechanism is modelling technique dependent. For EVORM, the identification mechanism, leading to standard names, is used for this purpose.

Predicators may have assigned a *predicator name* via the (partial) function: $\mathsf{PNm} : \mathcal{P} \rightarrowtail \mathsf{Names}$, and a *predicator reverse name*, via $\mathsf{RNm} : \mathcal{P} \rightarrowtail \mathsf{Names}$. We are now in a position to fill the lexicon with the predefined names and their meaning.

5.2.1 Named concepts

The name function Lex contains a verbalisation of the following concepts:

1. Names of object type histories are defined names.
If h is an object type history with name $\mathsf{HNm}(h)$, then, at any moment t of its existence, this name stands for the population of its version $h(t)$, put in the proper format:

$$\mathsf{Lex}(h, h, \mathsf{HNm}(h)) = \lambda t.\{\langle v, v\rangle \mid v \in \mathsf{Pop}_t \cdot h(t)\}$$

where $\cdot$ denotes functional composition.

2. Names of object types are defined names.
As stated in section 2, EVORM distinguishes between implicit and explicit object types. For implicit object types (such as fact type $\in_x$) no names are assumed. Rather, special names will later on be introduced to handle their manipulation. Let x be an explicit object type, then the name $\mathsf{ONm}(x)$ refers, at any moment, to its population at that moment:

$$\mathsf{Lex}(\lambda_x, \lambda_x, \mathsf{ONm}(x)) = \lambda t.\{\langle v, v\rangle \mid v \in \mathsf{Pop}_t(x)\}$$

where λ_x is a shorthand for: $\lambda t.x$.

3. Predicator names are defined names.
If p is a named predicator, then the name $\mathsf{PNm}(p)$ describes a path from the base of p to its corresponding fact type, while $\mathsf{RNm}(p)$ describes the reverse path:

$$\mathsf{Lex}(\lambda_{\mathsf{Base}(p)}, \lambda_{\mathsf{Fact}(p)}, \mathsf{PNm}(p)) = p$$

$$\mathsf{Lex}(\lambda_{\mathsf{Fact}(p)}, \lambda_{\mathsf{Base}(p)}, \mathsf{RNm}(p)) = p^{\leftarrow}$$

where predicator p, in this context, is used as the following history:

$$p = \lambda t.\{\langle v(p), v\rangle \mid v \in \mathsf{Pop}_t \cdot \mathsf{Fact}(p)\}$$

and $p^{\leftarrow}$ denotes relation inversion extended to histories: $p^{\leftarrow} = \lambda t.(p(t)^{\leftarrow})$.

4. Denotations for label values are defined names.
Defined names for label values, make it possible to use such denotations as regular information descriptors. If c is a constant and x an object type such that $c \in \mathsf{Dom}_t(x)$, then the denotation $\mathsf{CNm}(c)$ may be interpreted according to:

$$\mathsf{Lex}(\lambda_x, \lambda_x, \mathsf{CNm}(c)) = \lambda t.\{\langle c, c\rangle\}$$

5.2.2 Anonymous concepts

A schema may contain a number of implicit fact types. Being implicit, such fact types, and their constituent predicators, are not named. However, it may be necessary to address these concepts in verbalisations of information needs. Therefore, a set of default names for these concepts is introduced (see figure 16).

1. Names for bridge types
The names WITH and IS NAME OF can be used for quickly relating object types to label types. The name

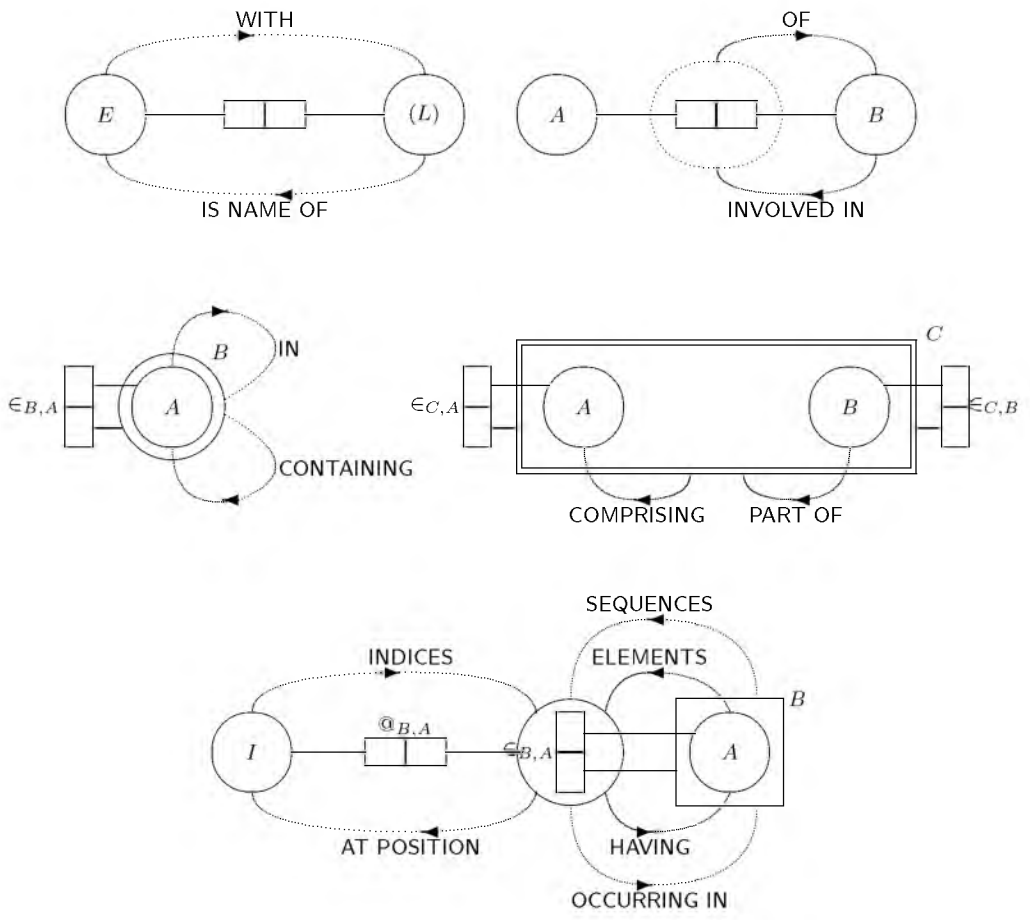


Figure 16: Names of implicit fact types

WITH relates object types via bridge types to label types, the name IS NAME OF being its reverse. Let $b = \{p, q\}$ be a bridge type, relating non-label type $\mathsf{Base}(p)$ to label type $\mathsf{Base}(q)$, then:

$$\mathsf{Lex}(\lambda_{\mathsf{Base}(p)}, \lambda_{\mathsf{Base}(q)}, \mathsf{WITH}) = p \circ q^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_{\mathsf{Base}(q)}, \lambda_{\mathsf{Base}(p)}, \mathsf{IS\ NAME\ OF}) = q \circ p^{\leftarrow}$$

Note that $\circ$ is the temporal extension of the concatenation operator for relations $A \circ B = \lambda t.A(t) \circ B(t)$.

2. Names for power types
The names CONTAINING and IN verbalise the implicit relation between a power type and its underlying element type. CONTAINING relates a power type to its corresponding element type, IN being its inverse. For all $x \in \mathcal{G}$:

$$\mathsf{Lex}(\lambda_x, \lambda_{\mathsf{Elt}(x)}, \mathsf{CONTAINING}) = \in^c_{x,\mathsf{Elt}(x)} \circ \in^e_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_{\mathsf{Elt}(x)}, \lambda_x, \mathsf{IN}) = \in^e_{x,\mathsf{Elt}(x)} \circ \in^c_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

3. Names for sequence types
The implicit fact types for sequence types describe the indexing mechanism. For all $x \in \mathcal{S}$:

$$\mathsf{Lex}(\lambda_x, \lambda_{\in_{x,\mathsf{Elt}(x)}}, \mathsf{SEQUENCES}) = \in^c_{x,\mathsf{Elt}(x)}$$

$$\mathsf{Lex}(\lambda_{\in_{x,\mathsf{Elt}(x)}}, \lambda_x, \mathsf{OCCURRING\ IN}) = \in^c_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_{\mathsf{Elt}(x)}, \lambda_{\in_{x,\mathsf{Elt}(x)}}, \mathsf{ELEMENTS}) = \in^{e}_{x,\mathsf{Elt}(x)}$$

$$\mathsf{Lex}(\lambda_{\in_{x,\mathsf{Elt}(x)}}, \lambda_{\mathsf{Elt}(x)}, \mathsf{HAVING}) = \in^{e}_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_I, \lambda_{\in_{x,\mathsf{Elt}(x)}}, \mathsf{INDICES}) = @^{i}_{x,\mathsf{Elt}(x)} \circ @^{s}_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_{\in_{x,\mathsf{Elt}(x)}}, \lambda_I, \mathsf{AT\ POSITION}) = @^{s}_{x,\mathsf{Elt}(x)} \circ @^{i}_{x,\mathsf{Elt}(x)}{}^{\leftarrow}$$

4. Names for schema types
The relations between a schema type, and the object types from its decomposition are handled by the keywords COMPRISING and PART OF. COMPRISING relates each instance of a schema type to the instances of the object types of its decomposition; PART OF is the reverse of COMPRISING. For all $x \in \mathcal{C}, x \prec y$:

$$\mathsf{Lex}(\lambda_y, \lambda_x, \mathsf{COMPRISING}) = \in^{e}_{x,y} \circ \in^{e}_{x,y}{}^{\leftarrow}$$

$$\mathsf{Lex}(\lambda_x, \lambda_y, \mathsf{PART\ OF}) = \in^{e}_{x,y} \circ \in^{e}_{x,y}{}^{\leftarrow}$$

5.2.3 Generic names

The names OF and INVOLVED IN are intended to facilitate the handling of objectification. They are also useful as shorthands for predicator names. OF represents all relations between fact type instances and their constituent object type instances, INVOLVED IN being its inverse. For all $x \in \mathcal{O}$ and $f \in \mathcal{F}$:

$$\mathsf{Lex}(\lambda_x, \lambda_f, \mathsf{INVOLVED\ IN}) = \bigcup_{q \in f, \mathsf{Base}(q)=x} q$$

$$\mathsf{Lex}(\lambda_f, \lambda_x, \mathsf{OF}) = \bigcup_{q \in f, \mathsf{Base}(q)=x} q^{\leftarrow}$$

The union operator in this definition is required to handle fact types that contain predicators with the same base.

5.3 Information descriptors

A number of operators are available to construct (composed) information descriptors. The most important construction mechanism is juxtaposition. The general rule for concatenating information descriptors P and Q is:

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

where $\circ$ denotes concatenation of binary relations (in the mathematical sense).

Information descriptors can be combined into new information descriptors by a number of operators. The first group contains operators such as union, intersection and set difference:

expression	ρ_t(expression)
P INTERSECTION Q	$\rho_t(P) \cap \rho_t(Q)$
P UNION Q	$\rho_t(P) \cup \rho_t(Q)$
P MINUS Q	$\rho_t(P) \setminus \rho_t(Q)$

Note that as Elisa-D is designed as an *open* language (see the language enrichments in subsection 5.5), it is quite possible to define – as an alternative for MINUS"

Information descriptors relate beginning and ending points of paths through the information structure. The operator THE restricts an information descriptor to its beginning point:

expression	ρ_t(expression)
THE P	$\{\langle x, x\rangle \mid \langle x, y\rangle \in \rho_t(P)\}$

The second group of binary operators consists of arithmetic operators:

expression	ρ_t(expression)
P **op** X	$\left\{ \langle x \textbf{ op } y, z\rangle \;\middle|\; \begin{array}{l} \exists_v \left[\langle x, v\rangle \in \rho_t(P)\right] \wedge \\ \langle y, z\rangle \in \rho_t(Q) \end{array} \right\}$

where **op** $\in \{+, -, *, /\}$. The third group of binary operators is derived from comparison relations on the underlying domains ($\mathcal{D}$), such as $<$, $\geq$, BEFORE, OVERLAPPED BY. If R is such a relation, then:

$$\rho_t(P \text{ R } Q) = \rho_t(P) \circ \text{R} \circ \rho_t(Q)$$

Finally, we introduce *set comprehension*. This operator restricts the result of a information descriptor P to values adhering to a condition C:

expression	ρ_t(expression)
$\{v \text{ IN } P \mid C\}$	$\{\langle x, x\rangle \in \rho_t(\text{THE } P) \mid \rho_t(C\|_x^v) \neq \varnothing\}$

The expression $C|_x^v$ represents the Elisa-D information descriptor C, in which all free occurrences of name v are interpreted as x, regardless of any previous meaning assignment (from the lexicon, or another variable binding).

5.4 Predicates

In Elisa-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	ρ_t(expression)
TRUE	$\{\langle v, v\rangle \mid v \in \Omega\}$
FALSE	$\varnothing$

The conditional clause construction is introduced as follows:

expression	ρ_t(expression)
IF C THEN P ELSE Q	$\begin{cases} \rho_t(P) & \text{if } \rho_t(C) \neq \varnothing \\ \rho_t(Q) & \text{otherwise} \end{cases}$

The test whether an information descriptor has an empty result provides an illustration of the usage of the conditional clause:

$$\text{SOME } P \triangleq \text{IF } P \text{ THEN TRUE ELSE FALSE}$$

For the moment, $\triangleq$ is employed as an abbreviation mechanism. In the next section we will 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:

C_1 AND C_2 $\triangleq$
(SOME C_1) INTERSECTION (SOME C_2)

C_1 OR C_2 $\triangleq$
(SOME C_1) UNION (SOME C_2)

NOT C $\triangleq$
TRUE MINUS SOME C

FOR SOME x IN P HOLDS C $\triangleq$
SOME $\{x \text{ IN } P \mid C\}$

FOR EACH x IN P HOLDS C $\triangleq$
NOT FOR SOME x IN P HOLDS NOT C

where C_1, C_2, C and P are information descriptors.

5.5 Language enrichment

In the previous subsection, we saw some examples of the introduction of macros. In this subsection, this mechanism is introduced formally. The purpose of macros is the enrichment of the retrieval language by assigning meaning to more constructs. Each macro can be seen as a new grammar rule, extending the retrieval language.

The general format of a macro definition is as follows:

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

where $X_1, \ldots, X_n$ are names of variables ($\notin$ Names), and $\omega_0, \ldots, \omega_n$ form the name of the macro. Due to this definition, the expression $\omega_0\ P_1\ \omega_1 \ldots P_n\ \omega_n$ is meaningful. It is evaluated according to the actual grammar, at each point of time, with the following meaning:

$$\rho_t(\omega_0\ P_1\ \omega_1 \ldots P_n\ \omega_n) \triangleq \rho_t(E|^{X_1,\ldots,X_n}_{P_1,\ldots,P_n})$$

This approach is in line with the style of evaluation in functional programming languages ([EP93]).

Adding new grammar rules may easily render a grammar ambiguous. A possible strategy is to accept ambiguity on the basis of the observation that natural language is ambiguous as well. Note that overloading of names in the lexicon already introduced ambiguity to some extent. Another approach is the introduction of a validity check upon entering new grammar rules. A possible means to overcome ambiguity is the introduction of priorities for macros.

Note that the macro mechanism offers the possibility to introduce recursion. As an example, consider the following macro definition:

$$\begin{array}{l}\underline{\mathsf{LET}}\ \mathsf{Fac}\ n\ \underline{\mathsf{BE}}\\ \quad \mathsf{IF}\ n = 0\ \mathsf{THEN}\ 1\ \mathsf{ELSE}\ n\ \mathsf{*}\ \mathsf{Fac}\ n - 1\end{array}$$

where the equality operator is defined by the macro definition $\underline{\mathsf{LET}}\ =\ \underline{\mathsf{BE}}$. The working of this mechanism is illustrated by the following proof of SOME (Fac 2 = 2):

$$\begin{array}{ll}\mathsf{Fac}\ 2 & \equiv \{\mathsf{NOT\ SOME}\ 2 = 0\}\\ 2\ \mathsf{*}\ \mathsf{Fac}\ 1 & \equiv \{\mathsf{NOT\ SOME}\ 1 = 0\}\\ 2\ \mathsf{*}\ 1\ \mathsf{*}\ \mathsf{Fac}\ 0 & \equiv \{\mathsf{SOME}\ 0 = 0\}\\ 2\ \mathsf{*}\ 1\ \mathsf{*}\ 1 & \equiv \{\textit{elaborate}\}\\ 2 & \end{array}$$

5.6 Special evolution related names

The language introduced so far, bascially does not exceed the functionality of LISA-D. In order to, indeed handle queries concerned with evolution, only a limited number of extensions is sufficient. These extensions range from extensions to the lexicon up to macro definitions.

5.6.1 Now

The name NOW is introduced to get grip on the point of time at which the query is evaluated:

$$\mathsf{Lex}(\lambda\mathcal{T}, \lambda\mathcal{T}, \mathsf{NOW}) = \lambda t.\ \langle t, t\rangle$$

5.6.2 Validity

The time intervals, during which an expression exists (i.e., has a non-empty result), can be obtained by the following operator:

expression	ρ_t(expression)
VALIDITY OF C	$\{\langle T, T\rangle \mid \mathsf{MaxDur}(T, \rho_t(C) \neq \varnothing)\}$

5.6.3 Gathering over time

The following operator unites the result of an information descriptor over its existence.

expression	ρ_t(expression)
ALL P EVER	$\bigcup_{s \in \mathcal{T}} \rho_s(P)$

Another verbalisation of this operator is introduced by: LET WHICHEVER P BE ALL P EVER.

5.6.4 Relating to points of time

The name AT is introduced in the lexicon of Elisa-D to relate, in a generic sense, histories to points of time. When using the AT keyword, we do not want to distinguish between elements from $\mathcal{AME}$ and Ω. Therefore, we first define Element as a generic term for application model elements, and their (possible) relation with values (from Ω):

```
LET Element BE
  Ame UNION Value INVOLVED IN Instance typing
```

Intuitively, AT should have the following semantics:

1. It relates values from Ω to points of time from $\mathcal{T}$, via fact type HasTypes and object type $\mathcal{AME}$ (see figure 11 (page 11)). This connection allows for the handling of time stamping.
2. It relates application model elements to $\mathcal{T}$. This offers the possibility to query about different versions of the application model.

This semantics is summarised in the following macro definition:

```
LET AT BE
    Element INVOLVED IN Ame evolution OF Time
```

The special name WHEN is the reverse of AT:

```
LET WHEN BE
    Time INVOLVED IN Ame evolution OF Element
```

Some examples of the usage of these constructions are:

1. THE Time WHEN X
   This yields all points of time at which information descriptor X has a non-empty result.
2. THE Squadron AT Now
   This results in all squadrons existing at the point of time at which the information descriptor is evaluated.

The names AT and WHEN relate points of time to the generalised notion Element. The fact type Holds relates time intervals and application model elements. In order to obtain better readable sentences, we also enrich Element also with this relation:

```
LET HAVING LIFESPAN BE
  Element INVOLVED IN Holds OF Time interval

LET LIFESPAN OF BE
  Time interval INVOLVED IN Holds OF Element
```

The operations on $\mathcal{T}$ and $\mathcal{T}_I$, as introduced in section 4, are made available for Elements, by defining for instance:

```
LET DURING BE
    HAVING LIFESPAN DURING

LET BEFORE BE
    (HAVING LIFESPAN  UNION  AT) BEFORE

LET AFTER BE
    (HAVING LIFESPAN  UNION  AT) AFTER

LET CONTAINING BE
    (HAVING LIFESPAN  UNION  AT) CONTAINS

LET INCREMENT OF BE
    AT IS INCREMENT OF
```

Note that BEFORE, AFTER and CONTAINS refer to the relations from section 4.

5.6.5 Time modalities

A special class of time related operations is formed by the so called time modalities. We will demonstrate the expressiveness of Elisa-D by showing how such operators are introduced via the macro mechanism.

The first operator corresponds to the next operator ($\Box$) in temporal logic. This operator is (verbosely) introduced in Elisa-D by:

```
LET AT THE NEXT POINT OF TIME C WILL HOLD BE
  SOME
    VALIDITY OF C
    CONTAINING
    INCREMENT OF NOW
```

Statements about system behaviour in the future can be formulated via the following constructs, from which the first corresponds to the ever ($\Diamond$) operator from temporal logic:

```
LET AT SOME TIME C WILL HOLD BE
  SOME
    VALIDITY OF C
    AFTER NOW

LET NEVER C WILL HOLD BE
  NOT   AT SOME TIME C WILL HOLD

LET FROM NOW ON C BE
  NOT SOMETIME
    NOW BEFORE
    VALIDITY OF NOT C
```

Properties from the past can be established by:

```
LET AT SOME TIME C HAS HELD BE
  SOME
    VALIDITY OF C
    BEFORE NOW

LET NEVER C HAS HELD BE
  NOT AT SOME TIME C HAS HELD
```

Properties over the entire history, including past, present and future, of the system can be addressed by:

LET ALWAYS C BE
NOT SOME VALIDITY OF NOT C

6 An example of evolution

As an illustration of an evolving universe of discourse, we consider an insurance company for cars. For each policy sold, the insured car and client are recorded. Every insured car has associated its registration number and type (Opel Corsa 1.2S, Ford Sierra 1.8, etc). A client is identified by name and address. The information structure of this universe of discourse is modelled in figure 17. As an illustration of the Elisa-D's ability to be employed for the disclosure of ER schemas as well, we use the ER modelling technique for this example. Note that we have prefixed the attributes of an entity type with a $\#$, and have associated predicator and predicator reverse names to predicators, in the format p/r. Note furthermore, that although EVORM supports the definition of primary identifiers, they are not shown in the ER examples as the used graphical language does not feature the symbols apt for this purpose.

Figure 17: The information structure of a car insurance company

After some time, the insurance company noticed a substantial difference between damage claims made for private cars, and for company cars. Rather than raising overall policy prices, a price differentiation was effectuated. For company owned cars, prices for new policies were increased by some percentage. Prices for new policies for private cars, however, were made dependent on the car usage, measured in kilometers per year.

As these changes in price only involve new policies, the current population of the schema did not have to be altered. The evolved information structure is depicted in figure 18. The differentiation between private and company cars, has led to a subtyping of cars, and the dependency of the policy price on the amount of driven kilometers has led to the introduction of an extra entity type (Kilometrage) and relation type (Usage).

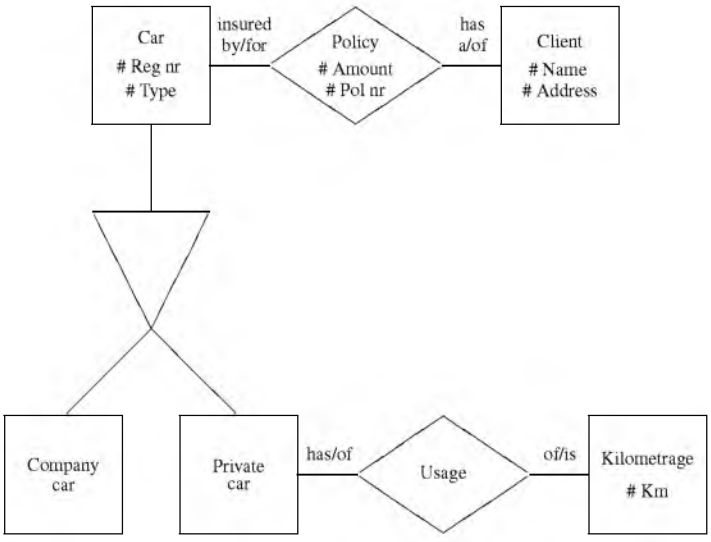


Figure 18: Car insurance with differentiated pricing

A large number of small companies, not intensively using their cars, started to protest against new policy pricing, threatening to accommodate their policies elsewhere. Thereupon, the insurance company decided to differentiate pricing for business cars as well. As a result, subtyping cars into business cars and private cars was abolished. A further means to be more competitive, was found in the introduction of a reduction for clients not claiming much damage. This reduction depends on the number of damage free years.

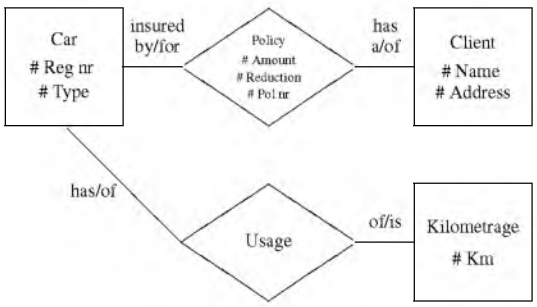


Figure 19: The final information structure

For the information structure, this leads to the introduction of the attribute # Reduction for relationship Policy. This results in figure 19. The introduction of the reduction, also requires a change in the current population of the information structure, as an initial reduction must be issued. Some illustrative examples of queries on instances, and the schema level, in the context of this example are:

1. Provide a list of all policies ever sold

   Policy

2. Provide a list of all policies during the year 1992

   Policy DURING Interval: 1992-01-01 1992-12-31

3. Provide all fact types ever known about cars

   Fact type IN Types INVOLVED IN
     Instance typing OF Car

4. Provide all fact instances known about the cars registered during the 22th of may in 1967

   Value OF Car DURING 1967-05-22

5. When was the concept of company car introduced

   BEGIN OF VALIDITY OF Object-type 'Company-cars'

   Where BEGIN OF $\subseteq \mathcal{T} \times \mathcal{T}_I$ is the relation providing the starting point of a time interval.

6. Which customers, having left the company after the differentiation in between private and company car (which happened at t_1), have returned after the newest change (at t_2).

   Client
     WHICH EVER has-a Policy AT t
   AND-ALSO
     WHICH EVER NOT has-a Policy HAVING
       LIFESPAN CONTAINING Interval: t_1 t_2
   AND-ALSO
     AT NOW

7 Conclusions

In this paper we introduced a language for the handling of evolving information systems, with focus on queries. If also constructs are added to evolve all application model elements, then Elisa-D seems to meet the requiremenets of what may be called a 6-th generation language. Currently, prototype implementations are being prepared, based on prototypes of LISA-D([Hub93]).

Further research may involve an improved disclosure mechanism, and extensions with uncertainty and relevance feedback, bridging the gap with expert systems and information retrieval systems (see also [Pro94]).

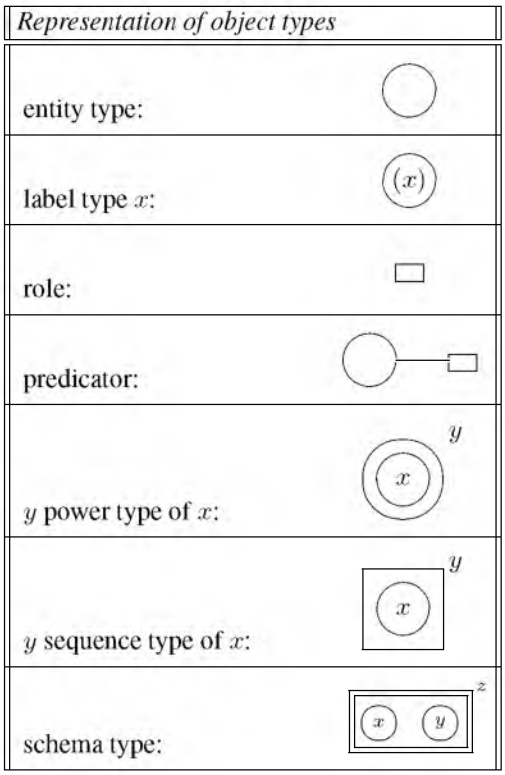
Acknowledgements

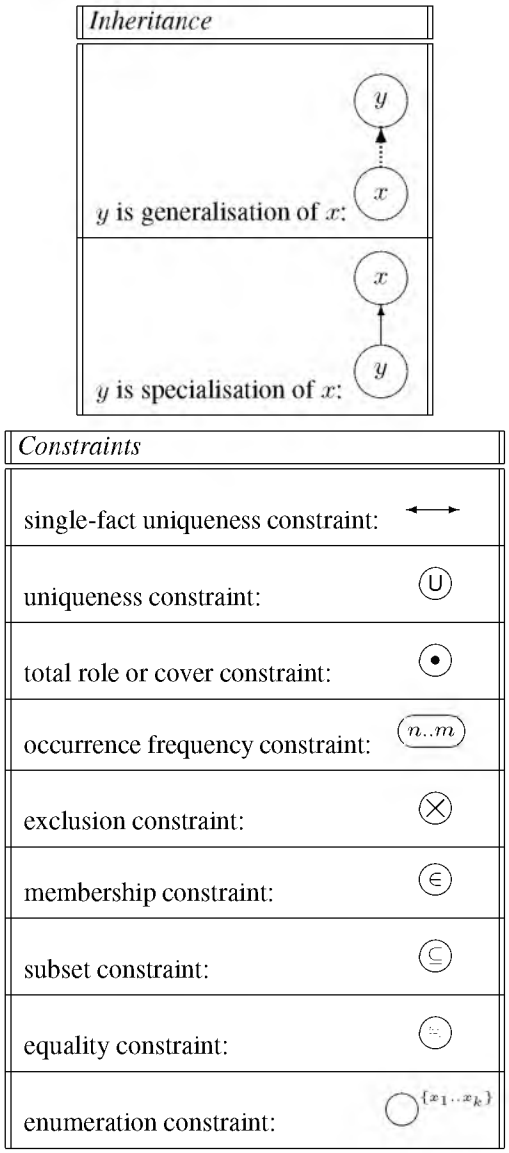
The investigations were partly supported by the Foundation for Computer Science in the Netherlands (SION) with financial support from the Dutch Organization for Scientific Research (NWO).

We would like to thank the anonymous referees for their many valuable comments on earlier versions of this paper.

Appendix: EVORM Graphical Conventions

This appendix contains an overview of the EVORM symbols for object types, generalisations and specialisations, and graphical constraints used in this article.





References

[Ari91] G. Ariav. Temporally oriented data definitions: Managing schema evolution in temporally oriented databases. *Data & Knowledge Engineering*, 6(6):451–467, 1991.

[Big88] J. Bigelow. Hypertext and CASE. *IEEE Software*, 5(2):23–27, 1988.

[BKKK87] J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record*, 16(3):311–322, December 1987.

[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.

[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.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.

[EP93] M.C.J.D. van Eekelen and M.J. Plasmeijer. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, Reading, Massachusetts, 1993.

[FOP92a] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. A Conceptual Framework for Evolving Information Systems. In H.G. Sol and R.L. Crosslin, editors, *Dynamic Modelling of Information Systems II*, pages 353–375. North-Holland, Amsterdam, The Netherlands, EU, 1992. ISBN 0444894055

[FOP92b] 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

[GS90] P.K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software*, 7(3):90–98, 1990.

[Hag92] T.M. Hagensen. Hyperstructure CASE Tools. In B. Theodoulidis and A. Sutcliffe, editors, *Proceedings of the Third Workshop on the Next Generation of CASE Tools*, pages 291–297, Manchester, United Kingdom, May 1992.

[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.

[Hal92] T.A. Halpin. WISE: a Workbench for Information System Engineering. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 38–49. IOS Press, 1992.

[HE92] U. Hohenstein and G. Engels. SQL/EER-syntax and semantics of an entity-relationship-based query Language. *Information Systems*, 17(3):209–242, 1992.

[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.

[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.

[HPW94] 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

[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.

[JMSV92] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. DAIDA: An Environment for Evolving Information Systems. *ACM Transactions on Information Systems*, 20(1):1–50, January 1992.

[KBC+89] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza, and D. Woelk. Features of the ORION Object-Oriented Database. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Frontier Series, pages 251–282. Addison-Wesley, Reading, Massachusetts, 1989.

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

[MS90] E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[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

[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. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[Rod91] J.F. Roddick. Dynamically changing schemas within database models. *The Australian Computer Journal*, 23(3):105–109, August 1991.

[RP92] J.F. Roddick and J.D. Patrick. Temporal semantics in information systems - A survey. *Information Systems*, 17(3):249–267, 1992.

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