Expressiveness in Conceptual Data Modelling

A.H.M. ter Hofstede *
Th.P. van der Weide †

Published as: 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.

Abstract

Conceptual data modelling techniques aim at the representation of data at a high level of abstraction. The Conceptualisation Principle states that only those aspects are to be represented that deal with the meaning of the Universe of Discourse. Conventional conceptual data modelling techniques, as e.g. ER or NIAM, have to violate the Conceptualisation Principle when dealing with objects with a complex structure. In order to represent these objects conceptually irrelevant choices have to made. It is even worse: sometimes the Universe of Discourse has to be adapted to suit the modelling technique. These objects typically occur in domains as meta-modelling, hypermedia and CAD/CAM. In this paper extensions to an existing data modelling technique (NIAM) will be discussed and formally defined, that make it possible to naturally represent objects with complex structures without having to violate the Conceptualisation Principle. These extensions will be motivated from a practical point of view by examples and from a theoretical point of view by a comparison with the expressive power of formal set theory and grammar theory.
Keywords. Conceptual data modelling, hypertext, object-role models, ER, NIAM, identification.

1 Introduction

In the past years, many conceptual data modelling techniques have been introduced. Well-known examples are the Relational model ([6]) the ER approach ([5]) and NIAM ([38],[25]). The *Conceptualisation Principle* ([11]) requires that conceptual schemas should deal only and exclusively with aspects of the Universe of Discourse (UoD). Any aspects irrelevant to that meaning, e.g. machine efficiency, should be avoided. In complex application domains contemporary data modelling techniques are not always capable of adhering to the Conceptualisation Principle. Choices that are not relevant with respect to the UoD have to be made (leading to *overspecification*) or worse even, the UoD has to be adapted e.g. extra objects have to be introduced. These problems are caused by the lack of sufficiently powerful construction mechanisms.

Various application domains contain objects with complex structures. Documents are an example in the field of office automation. Such objects are constructed according to specific rules, e.g. a book consists of a *sequence* of chapters, while a chapter consists of a sequence of sections. These rules might have a *recursive* nature. Recursive structures and sequences can not be adequately modelled in conventional modelling techniques such as ER or NIAM.

*This work has been partially supported by SERC project SOCRATES. Software Engineering Research Centre, P.O. Box 424, 3500 AK Utrecht, The Netherlands, E-mail: hofstede@serc.nl

†Dept. of Information Systems, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, E-mail: tvdw@cs.kun.nl

Another domain in which complex objects are important is the field of method engineering or meta-modelling ([34]). In this field, meta-models are constructed for particular modelling techniques. Meta-models capture the (syntactical) structure of such techniques. Meta-models often contain complex object types, for example when a modelling technique allows for decomposition. It is not natural to represent these object types as flat structures (see e.g. [37]).

Finally, Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) also are areas in which complex structures frequently occur. Typical examples are the part-of hierarchies, which have a recursive nature since components are part of other components.

In this paper a general conceptual data modelling technique is introduced, the Predicator Set Model (PSM), which is capable of representing complex structures in a natural way. The Predicator Set Model is an extension of the Predicator Model (PM) ([3]) which on its turn is a formalisation of NIAM. 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 and the NIAM philosophy are not lost in PSM, they only need to be extended to support also the additional constructs. Motivation for the new constructs is, for a practical point of view, provided by a number of examples. From a theoretical point of view the new constructs are motivated by a comparison with the axioms of formal set theory and with grammar theory. This latter comparison is particularly relevant for hypermedia, since document structures are usually described by means of context free grammars (in the style of SGML [28],[18] or ODA [19],[17]). This implies that if context free grammars can be modelled in PSM then also document structures can.

The organisation of the paper is as follows. In section 2, PSM will be introduced informally as well as formally. The main extensions to NIAM are power types and generalisation. The concept of power type can be compared to the concept of powerset in formal set theory. It makes it possible to treat a set of objects as an object itself. A second extension is the concept of generalisation, which enables the union of different types of objects. The concepts in PSM may be combined to define recursive object types. A typical example is the construction of formulas. The semantics of PSM are given in terms of populations (instantiations). Section 2 concludes with some typical (graphical) constraints in relation to power types.

As its name suggests, PSM has been inspired by formal set theory. In section 3 the axioms of set theory will be related to the modelling constructs in PSM, thus providing an indication of its expressive power.

Next the identification of information structures is considered with respect to object instances (*weak identification*) and with respect to object types (*structural identification*). Structural identification acts as a well-formedness rule for complex object types. Schemas which are structurally identifiable can be populated.

In section 5 two notational shorthands will be introduced that further facilitate the representation of complex object types. The first notational shorthand, *schema objectification*, makes it possible to treat a schema as an object type itself. This is particularly useful in the domain of method engineering or meta-modelling. The second shorthand, *sequencing*, is especially important in relation to context-free grammars, which is the subject of the last section. Section 5 concludes with a PSM schema that is a meta-model of PSM. It describes the structure of its formalisation.

In the last section a translation of context-free grammars to PSM schemas will be given. This demonstrates that the expressive power of context-free grammars can be embedded in PSM. An application of the translation is given in which a hypertext information structure, described by means of a context-free grammar, is represented elegantly as a PSM schema.

2 The Predicator Set Model

This section starts with an informal introduction of the basic components and type construction mechanisms in PSM followed by a formalisation of PSM. This formalisation consists of two parts.

In the first part, information structures will be defined. An information structure is a PSM schema without constraints. The second part deals with instantiations of information structures, so-called populations. This gives us a platform for the introduction of constraints, as an exclusion mechanism for populations (see figure 17). Some typical constraints in relation to power types are considered.

2.1 Introduction

One of the key concepts in data modelling is the concept of relation type. In ER ([5]) and NIAM ([25]) a relation type is considered to be an association between object types. In figure 1 the graphical representation of a binary relation R between object types X_1 and X_2 in the NIAM style is shown, while in figure 2 the corresponding ER diagram is depicted. A relation type consists of a number of roles, which denote particular functions that are played by object types in that relation type.

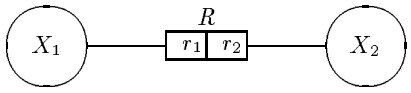


Figure 1: A NIAM relation type

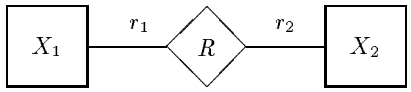


Figure 2: The corresponding ER diagram

In figure 1 we see that the roles r_1 and r_2 are ordered. This corresponds to the *tuple oriented approach* where a relation is defined as a subset of a Cartesian product. A disadvantage of this approach is that algebraic operators lack useful properties as commutativity and associativity. The modern approach is to use the mapping mechanism to describe relations (the *mapping oriented approach*), see also [22]. In the example of figure 1, this approach corresponds to disconnecting the relation R, yielding the situation of figure 3.

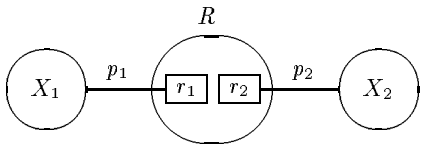


Figure 3: Disconnecting the roles

Note that R is represented in this figure according to the same format as X_1 and X_2. Relation types, also called fact types, can therefore be considered object types, which will be referred to as *fact objectification*. Fact objectification can be compared to the notion of aggregation in semantic data modelling ([16]) as it is a mechanism that allows for the creation of types whose instances are tuples.

In figure 3, the basic building element is the connection between an object type and a role, the so-called *predicator* (this term was first introduced in [9]). In this figure, p_1 is the predicator connecting X_1 to r_1, and p_2 the predicator that connects X_2 to r_2. Note that the concept of predicator appears itself in a NIAM schema (as an unnamed drawing object), while in an ER schema it cannot be visualised. In the sequel, we will therefore prefer the NIAM drawing style.

In PSM, as in PM, a relation type is considered to be a set of predicators. A relation type is therefore considered as an association between predicators, rather than between objects types. The full consequence of this approach would be to draw the relation type of figure 1, as shown in figure 4.

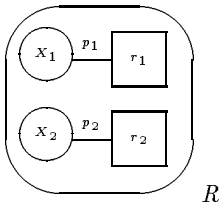


Figure 4: A relation type as a set of predicators

We will prefer, however, the style of figure 1 or figure 3 rather than the style of figure 4, due to its notational simplicity.

2.1.1 Power types

In PSM the notion of *power type* exists. Power types can be compared to power sets in conventional set theory. An instance of a power type is a set of instances of its *element type*. An instance of the power type will be identified by these instances of the element type, just as a set is identified by its elements in formal set theory (axiom of extensionality), see also [16]. A simple example is shown in figure 5.

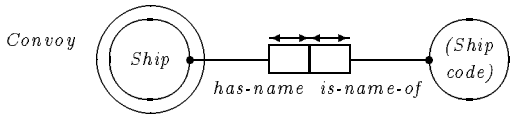


Figure 5: A simple example of a power type

In this example (known as the *Convoy Problem*) the object type *Convoy* is a power type of *Ship*. As a result, each instance of object type *Convoy* is a set of instances of *Ship*. Convoys are identified by their constituent ships, whereas ships are identified by a *Ship code*. In the schema of figure 5 some graphical constraints occur. The black dot on the object type *Ship* is an example of a total role constraint and expresses that each instance of *Ship* has to play the role *has-name*. The double arrow above this role is an example of a uniqueness constraint and expresses that instances of *Ship* play this role at most once. The formal semantics of these graphical constraints can be found in [3],[36].

The Convoy Problem is *not* expressible in terms of a NIAM (or ER) schema. Consider for example the schema from figure 6. This schema only implicitly states that a convoy consists of a number of ships: by means of role name *contains*. Contrary to figure 5, a convoy cannot be identified by its ships in this schema, and needs another form of identification, for example by the artificial introduction of a *Convoy code*. This clearly is a violation of the Conceptualisation Principle.

The notion of power typing is the same as the notion of grouping as introduced in the IFO data model ([1]). This notion provides for a considerable extension of expressiveness. As an illustration of the expressive power of power types the chemical reactions example from [8] is discussed. The considered Universe of Discourse deals with simple chemical reactions. A chemical reaction consumes a set of input substances with their associated quantities, and produces a set of output substances in corresponding quantities.

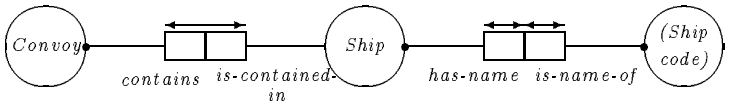
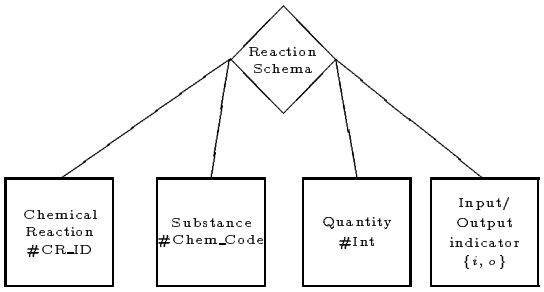


Figure 6: NIAM schema belonging to figure 5



R1	H_2	2	i
R1	O_2	1	i
R1	H_2O	2	o
R2	$CH_3 - CH_2OH$	2	i
R2	O_2	1	i
R2	$CH_3 - CHO$	2	o
R2	H_2O	2	o

Figure 7: Chemical Reactions in ER

This Universe of Discourse could be modelled in an ER schema in terms of a quartenary relationship, as shown in figure 7. In this relation, the attribute *CR_ID* is used to identify chemical reactions. The entity type *Substance* describes which substances are subject to the chemical reaction, and the entity type *Quantity* describes in what quantity. The *Input/Output indicator* makes the distinction between input and output substances to the chemical reaction. A first problem with this solution is the superfluous identification of a chemical reaction. Only some chemical reactions are sufficiently important, to have a name of their own. The others are just identified by their description in terms of what goes in and what comes out. The second problem is that this solution does not allow for the addition of a chemical reaction by one elementary update. This is caused by the fact that in the model of figure 7 several object instances are needed to denote one reaction.

The use of a power type offers a much better opportunity to model this Universe of Discourse (see figure 8). In this model, a chemical reaction is modelled as a relationship between a set of input reagents, and a set of output reagents. This schema is better understood by studying a sample population (see figure 8). This sample population is in the style of nested relations as, encountered in the NF^2 datamodel [26]. The main difference is that NF^2 uses a nested table heading (and thus nested tuples).

The solution of figure 8 also solves the update problem which was mentioned before. In this model a chemical reaction is denoted as a single object instance. Therefore, the above mentioned elementary update problem is solved. The consequence is that an update operation of a chemical reaction can be considered as a single operation in the PSM. Furthermore in the PSM schema neither a separate identification for chemical reactions is needed nor an Input/Output indicator.

2.1.2 Specialisation

Specialisation, referred to as subtyping in NIAM, is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. Specialisation is to be applied when only for

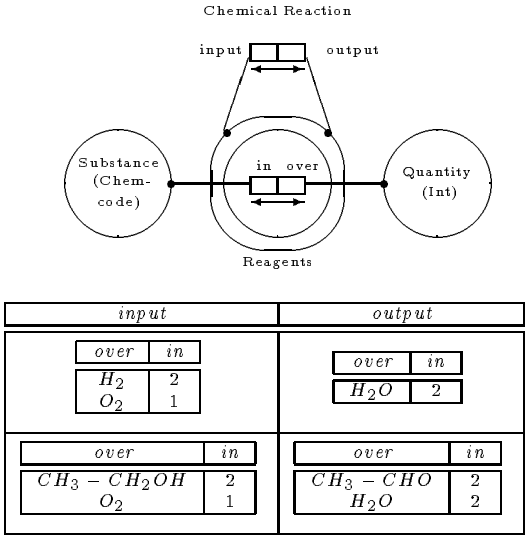


Figure 8: Chemical Reactions

specific instances of an object type certain facts are to be recorded. Suppose for example that only for *Adults*, *Persons* with an *Age* greater or equal than 18, one is interested in the *Cars* they own. This situation is captured by the PSM schema in figure 9.

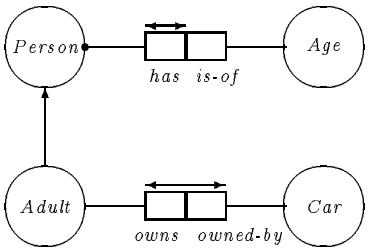


Figure 9: Example of specialisation

A specialisation relation between a subtype and a supertype implies that the instances of the subtype are also instances of the supertype (each *Adult* is also a *Person*). For proper specialisation, it is required that subtypes be defined in terms of one or more of their supertypes. Such a decision criterion is referred to as the *Subtype Defining Rule* (see e.g. [3]). In figure 9 the subtype defining rule for *Adult* would be:

Adult = Person has Age $\geq$ 18

Identification of subtypes is derived from their supertypes. Therefore if, in the ongoing example, *Persons* would be identified by a name, then *Adults* are also identified by the same name.

Specialisation relations are organised in so-called specialisation "hierarchies". A specialisation hierarchy is in fact not a hierarchy in the strict sense, but an acyclic directed graph with a unique top. This top is referred to as the *pater familias* (see [33]).

Objects inherit all properties from their ancestors in the specialisation hierarchy. This characteristic of specialisation excludes fact types and power types occurring as subtypes. Consider for

example the case that a ternary fact type is a subtype of a binary fact type. Clearly this leads to a contradiction. No problems occur when fact types and power types themselves are specialised (see for example figure 15). A fact type *works for* could have a subtype *works temporarily for*. There is no need to explicitly represent this subtype as a fact type since this can be derived from the supertype. Consequently, fact types and power types always act as pater familias. Our approach is in correspondence with the requirements as stated in [1].

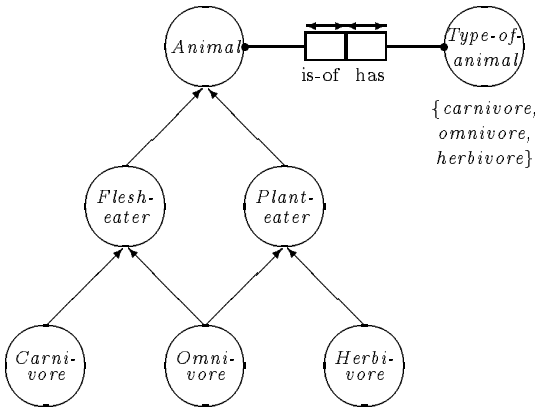


Figure 10: Example of a specialisation hierarchy

Example 2.1 *In figure 10 we have the following specialisation hierarchy:*

Flesh-eater Spec *Animal*
Plant-eater Spec *Animal*
Carnivore Spec *Flesh-eater*
Omnivore Spec *Flesh-eater*
Omnivore Spec *Plant-eater*
Herbivore Spec *Plant-eater*

Each specialisation relation is represented as an arrow in figure 10. As a consequence, the pater familias of object type carnivore *is* animal. *The subtype defining rules are:*

$Flesh\text{-}eater = Animal\ is\text{-}of\ Type\text{-}of\text{-}animal\ \{carnivore, omnivore\}$
$Plant\text{-}eater = Animal\ is\text{-}of\ Type\text{-}of\text{-}animal\ \{herbivore, omnivore\}$
$Carnivore = Animal\ is\text{-}of\ Type\text{-}of\text{-}animal\ \{carnivore\}$
$Omnivore = Animal\ is\text{-}of\ Type\text{-}of\text{-}animal\ \{omnivore\}$
$Herbivore = Animal\ is\text{-}of\ Type\text{-}of\text{-}animal\ \{herbivore\}$

2.1.3 Generalisation

Generalisation is a mechanism that allows for the creation of new object types by uniting existing object types. Contrary to what its name suggests, generalisation is *not* the inverse of specialisation. Specialisation and generalisation originate from different axioms in set theory (see section 3) and therefore have a different expressive power.

For generalisation it typically is required that the generalised object type is covered by its constituent object types (or *specifiers*). Therefore, a decision criterion as in the case of specialisation (the subtype defining rule) is not necessary. Furthermore, properties are inherited "upward" in a generalisation hierarchy instead of "downward", which is the case for specialisation (see also [1]).

This also implies that the identification of a generalised object type depends on the identification of its specifiers. From the nature of generalisation, it is apparent that a fact type or power type cannot be a generalised object type.

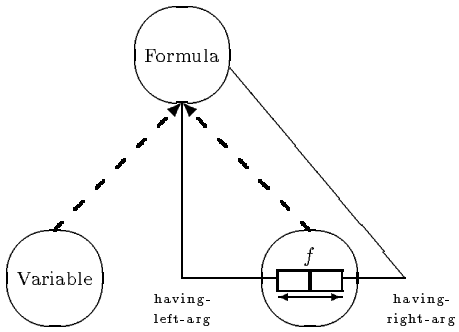


Figure 11: Example of generalisation

Example 2.2 *In figure 11 an example of generalisation is shown. A formula may be either a single variable, or constructed by some function (say f) from simpler formulas. Note that the structure of instances of formula is determined by the specifier (variable or f) from which they come.*

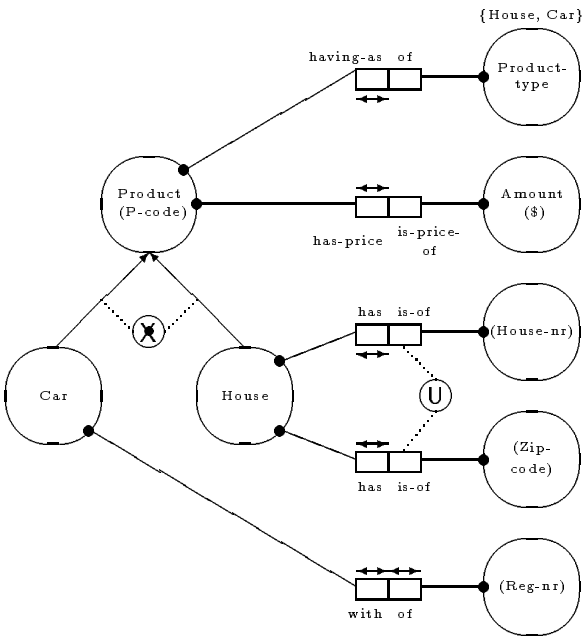
The above example shows that generalisation can be used to define recursive object types. This is not possible in the IFO data model, where object types are hierarchical structures. In the Logical Data Model (see [21]) however, object types are directed graphs, which may contain cycles.

As stated before, generalisation and specialisation are entirely different notions. Some situations can only be solved using specialisation (e.g. when complex subtype defining rules are involved) and some situations can only be solved using generalisation (e.g. when recursive structures are involved as in the formula example). In NIAM generalisation is usually modelled as specialisation. This leads to a violation of the Conceptualisation Principle as will be discussed in the following example.

Consider a pricelist for individually priced *Product*s. A *Product* is either a *Car*, or a *House*. A *Car* is identified by a registration number, while a *House* is identified by the combination of its zip-code and house number. Note that the uniqueness of the combination between a zip code and a house number is modelled by means of an encircled U, which is the graphical notation of a uniqueness constraint over several fact types. *Product* is considered to be a generic term for *House* and *Car*. *Products* have a price associated to them.

In NIAM this Universe of Discourse would be modelled as the schema in figure 12. This schema suffers from overspecification. Firstly, a special label type (*P-code*) has to be introduced in order to identify *Products*. Secondly a constraint between the subtypes *Car* and *House* is needed to express that these subtypes are total and disjunct. Thirdly, a special fact type and a special object type (*Product Type*) are required to determine the type of *Product*. This determination forms the *Subtype Defining Rule* for *Products* (see figure 12). However, these extra object types are not conceptually relevant. Their introduction should therefore be considered as a violation of the Conceptualisation Principle.

Using the concept of generalisation, these overspecifications are avoided. In figure 12 a more appropriate schema for this Universe of Discourse is depicted. In this schema, the label type *P_code* is no longer needed, since *Products* inherit their identification from *Cars* and *Houses*. Furthermore the constraint expressing that *Car* and *House* are disjunct and total with respect to *Product* follows directly from the *Strong Typing Rule* and the *Generalisation Rule* (see section 2.3).



Subtype defining rules:

Car = Product having-as Product-type 'Car'
House = Product having-as Product-type 'House'

Figure 12: Specialisation instead of Generalisation

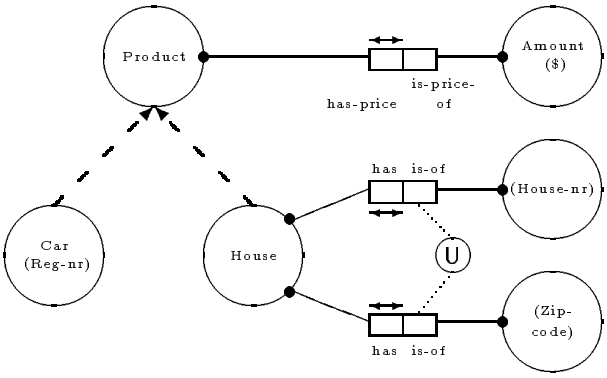


Figure 13: Example of Generalisation

2.2 The Information Structure

The discussion in the previous section leads to the following definition: an *information structure* over a so-called label type set $\mathcal{L}$, is a structure consisting of the following basic components:

1. A finite set $\mathcal{P}$ of *predicators*.
2. A set $\mathcal{O}$ of *object types*, $\mathcal{L} \subseteq \mathcal{O}$.
3. A partition $\mathcal{F}$ of the set $\mathcal{P}$. The elements of $\mathcal{F}$ are called *fact types*. Fact types are also object types, therefore $\mathcal{F} \subseteq \mathcal{O}$.
4. A set $\mathcal{G}$ of *power types*. Power types are also object types, therefore $\mathcal{G} \subseteq \mathcal{O}$.
5. A function $\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$. The base of a predicator is the object part of that predicator.
6. A function $\mathsf{Elt} : \mathcal{G} \rightarrow \mathcal{O}$. This function yields the element type of a power type.
7. A partial order $\mathsf{Spec} \subseteq \mathcal{A} \times \mathcal{O}$ on object types, capturing specialisation. $\mathcal{A}$ is the set of *atomic object types* and is defined as $\mathcal{O} \setminus (\mathcal{F} \cup \mathcal{G})$.
8. A partial order $\mathsf{Gen} \subseteq \mathcal{A} \times \mathcal{O}$ on object types, expressing generalisation.

The auxiliary function $\mathsf{Fact} : \mathcal{P} \rightarrow \mathcal{F}$ is defined by: $\mathsf{Fact}(p) = f \Leftrightarrow p \in f$.

At this point cyclic composition of power types will not be excluded, for example power types that have themselves as element type (formally: those $g \in \mathcal{G}$ for which $\mathsf{Elt}(g) = g$). In a later section, we will however see, that such cyclic structures are excluded if one requires structural identification. Structural identification is then considered a rule for well formedness of object types (see [14]).

Fact types and power types are considered to be different concepts:

(PSM1) $\mathcal{F} \cap \mathcal{G} = \emptyset$

due to the different interpretation that will be given to these concepts.

There are two different sorts of atomic object types: entity types ($\mathcal{E} = \mathcal{A} \setminus \mathcal{L}$) and label types ($\mathcal{L}$). The difference is that labels can, in contrast with entities, be represented (reproduced) on a communication medium. Depending on the medium, we distinguish text, graphics, sound and video. The term multimedia is used as a collective noun. Typical examples of label types are *Name*, *Number* and *Code*. A typical example of an entity type would be *Person*.

It is important to note that instances (occurrences) of object types are *not* part of the information structure. Instantiations (populations) will be introduced in section 2.3.

Example 2.3
In figure 14 an example of an information structure is shown where:

$$\begin{aligned}
\mathcal{P} &= \{p, q, r, s, t, u, v\} \\
\mathcal{O} &= \{A, B, C, D, E, f, g, h\} \\
\mathcal{F} &= \{f, g, h\} \\
\mathcal{G} &= \{E\} \\
\mathcal{A} &= \{A, B, C, D\} \\
\mathcal{E} &= \{A, B, C, D\} \\
\mathcal{L} &= \emptyset
\end{aligned}$$

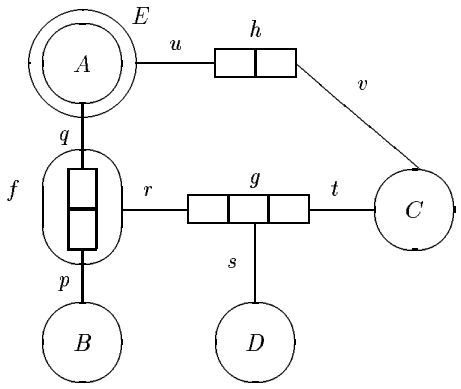


Figure 14: Example information structure

where $f = \{p, q\}$, $g = \{r, s, t\}$ *and* $h = \{u, v\}$. *With respect to the predicators:*

$$\begin{array}{lcllcl} \mathsf{Base}(p) & = & B & \mathsf{Fact}(p) & = & f \\ \mathsf{Base}(q) & = & A & \mathsf{Fact}(q) & = & f \\ \mathsf{Base}(r) & = & f & \mathsf{Fact}(r) & = & g \\ \multicolumn{6}{l}{\quad etc.} \end{array}$$

Finally, $\mathsf{Elt}(E) = A$.

2.2.1 Specialisation

The concept of specialisation is defined as a partial order (asymmetric and transitive) Spec on object types ($\mathsf{Spec} \subseteq \mathcal{A} \times \mathcal{O}$), with the convention that $a\,\mathsf{Spec}\,b$ is interpreted as: a is a specialisation of b, or a is a subtype of b. Note that the infix style of notation is preferred, therefore $a\,\mathsf{Spec}\,b$ will be written instead of $(a, b) \in \mathsf{Spec}$. The signature of Spec implies that if $a \notin \mathcal{A}$, then $\neg a\,\mathsf{Spec}\,b$ for any b. The definition of Spec therefore implies that only atomic object types can occur as subtypes. Specialisation as presented here is an extension of the concept of subtyping as presented in [3], where only atomic object types can be specialised. The following axioms prevent cycles in specialisation structures and require them to be transitively closed.

(PSM2) (*asymmetry*) $a\,\mathsf{Spec}\,b \Rightarrow \neg b\,\mathsf{Spec}\,a$

(PSM3) (*transitivity*) $a\,\mathsf{Spec}\,b \wedge b\,\mathsf{Spec}\,c \Rightarrow a\,\mathsf{Spec}\,c$

The pater familias of specialisation networks is found by the function $\sqcap : \mathcal{O} \rightarrow \mathcal{O}$ (which is similar to the top operator from lattice theory). The fact that $\sqcap$ is a function reflects the requirement stated in section 2.1.2, that each specialisation network should have a *unique* top. This requirement is a direct consequence of the fact that subtypes inherit their identification from their pater familias. $\sqcap$ has the following properties:

(PSM4) (*cohesion*) $a\,\mathsf{Spec}\,b \Rightarrow \sqcap(a) = \sqcap(b)$

(PSM5) (*strictness*) $a \neq \sqcap(a) \Rightarrow a\,\mathsf{Spec}\,\sqcap(a)$

Furthermore it is required that subtype hierarchies for label types and entity types do not interfere:

(PSM6) (*separation*) $a \in \mathcal{L} \Leftrightarrow \sqcap(a) \in \mathcal{L}$

From these axioms some powerful properties can be derived.

Lemma 2.1 $\neg \sqcap (a)\,\mathsf{Spec}\,b$

Proof: Suppose $\sqcap(a)\,\mathsf{Spec}\,b$. We distinguish the following cases:

1. $a \neq \sqcap(a)$
   $\stackrel{(1)}{\Longrightarrow}$ {strictness} $a\,\mathsf{Spec}\,\sqcap(a)$
   $\stackrel{(2)}{\Longrightarrow}$ {transitivity of Spec} $a\,\mathsf{Spec}\,b$
   $\stackrel{(3)}{\Longrightarrow}$ {cohesion} $\sqcap(a) = \sqcap(b)$
   $\stackrel{(4)}{\Longrightarrow}$ {assumption} $\sqcap(b)\,\mathsf{Spec}\,b$
   $\stackrel{(5)}{\Longrightarrow}$ {irreflexivity of Spec} $b \neq \sqcap(b)$
   $\stackrel{(6)}{\Longrightarrow}$ {strictness} $b\,\mathsf{Spec}\,\sqcap(b)$
   (4) and (6) contradict the asymmetry of Spec.
2. $a = \sqcap(a)$
   $\stackrel{(1)}{\Longrightarrow}$ {assumption} $a\,\mathsf{Spec}\,b$
   $\stackrel{(2)}{\Longrightarrow}$ {cohesion} $\sqcap(a) = \sqcap(b)$
   $\stackrel{(3)}{\Longrightarrow}$ {assumption} $\sqcap(b)\,\mathsf{Spec}\,b$
   $\stackrel{(4)}{\Longrightarrow}$ {irreflexivity of Spec} $b \neq \sqcap(b)$
   $\stackrel{(5)}{\Longrightarrow}$ {strictness} $b\,\mathsf{Spec}\,\sqcap(b)$
   (3) and (5) contradict the asymmetry of Spec.

□

Corollary 2.1 *Idempotency of* $\sqcap$

$$\sqcap(\sqcap(a)) = \sqcap(a)$$

Proof: Suppose $\sqcap(\sqcap(a)) \neq \sqcap(a)$. Applying the *strictness* rule yields $\sqcap(a)\,\mathsf{Spec}\,\sqcap(\sqcap(a))$. This however contradicts the previous lemma.

□

Lemma 2.2 *Fact types and power types are always pater familias*

$$x \notin \mathcal{A} \Rightarrow \sqcap(x) = x$$

Proof: Suppose $\sqcap(x) \neq x$, then from the *cohesion* rule it can be derived that $x\,\mathsf{Spec}\,\sqcap(x)$. Using the definition of Spec this implies that $x \in \mathcal{A}$.

□

For the moment we will not require that a power type should not have its element type as subtype (formally: $\mathsf{Elt}(g)\,\mathsf{Spec}\,g$). This will be excluded by the notion of structural identification (see section 4).

Example 2.4 *Figure 15 contains an abstract example of a specialisation hierarchy, where:*

$C\ \mathsf{Spec}\ f$
$D\ \mathsf{Spec}\ f$

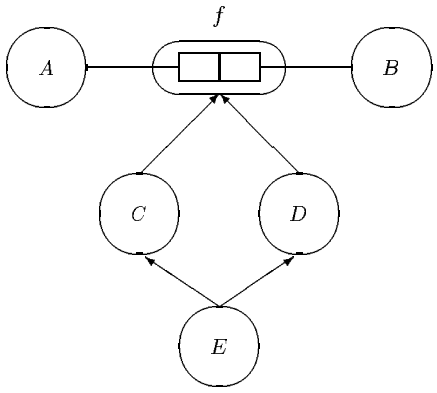


Figure 15: Example of a specialisation hierarchy

E Spec f
E Spec C
E Spec D

Each specialisation relation, except E Spec f *which can be derived by transitivity, is represented as an arrow in figure 15. As a consequence, the pater familias of the object types* f, C, D *and* E *is the composed object type* f.

2.2.2 Generalisation

The concept of generalisation is formally introduced as a partial order (asymmetric and transitive) $\mathsf{Gen} \subseteq \mathcal{A} \times \mathcal{O}$, with the convention that $a \,\mathsf{Gen}\, b$ is interpreted as: a is a generalisation of b, or b is a specifier of a.

(PSM7) (*asymmetry*) $a \,\mathsf{Gen}\, b \Rightarrow \neg b \,\mathsf{Gen}\, a$

(PSM8) (*transitivity*) $a \,\mathsf{Gen}\, b \wedge b \,\mathsf{Gen}\, c \Rightarrow a \,\mathsf{Gen}\, c$

Note that the definition of Gen implies that generalised object types are atomic. A predicate $gen \subseteq \mathcal{A}$ will be used for generalised object types, defined as follows:

$$gen(a) \Leftrightarrow \exists_{x \in \mathcal{O}} [a \,\mathsf{Gen}\, x]$$

Generalisation and specialisation can be conflicting due to their inheritance structure. Consider for example figure 16. In this figure, a is a specialisation of c and a generalisation of b. The specialisation relation requires the identification of a to depend on the identification of c, while the generalisation relation requires the identification of a to depend on the identification of b. In terms of populations (see also the next section) this contradiction can be formulated in a different way. Specialisation in this case requires that, in every population, the instances of object type a are those instances of object type c that fulfil the subtype defining rule, while generalisation requires the instances of a to be exactly the instances of object type b. To avoid such conflicts, generalised object types are required to be pater familias:

(PSM9) $gen(a) \Rightarrow \sqcap(a) = a$

Generalisation should go beyond the boundaries of a specialisation hierarchy:

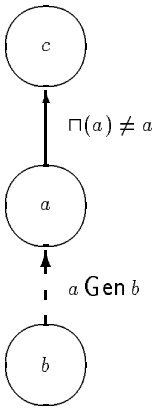


Figure 16: Conflicting generalisation and specialisation

(PSM10) $a\,\mathsf{Gen}\,b \Rightarrow \sqcap(a) \neq \sqcap(b)$

Lemma 2.3

$$a\,\mathsf{Gen}\,b \vee b\,\mathsf{Gen}\,a \Rightarrow \neg a\,\mathsf{Spec}\,b$$

Proof: Suppose $a\,\mathsf{Gen}\,b \vee b\,\mathsf{Gen}\,a$. Application of PSM10 yields $\sqcap(a) \neq \sqcap(b)$. From the *cohesion* rule (PSM4) we conclude $\neg a\,\mathsf{Spec}\,b$.

□

2.3 Populations

An information structure is used as a frame for some part of the (real) world, the so-called Universe of Discourse (U∘D). A *state* of the U∘D then corresponds to a so-called instantiation or population of the information structure, and vice versa. The idea of states was previously mentioned in [10], [35], [31]. Furthermore, a state transition of the U∘D has a corresponding transition on populations of the information structure. This can be formulated as:

> The Universe of Discourse is *isomorphic* with the set of possible populations of the information structure and a transition relation hereupon.

This statement cannot be verified formally however, since the Universe of Discourse is part of the real world.

In our approach, a population Pop of an information structure

$$\mathcal{I} = \langle \mathcal{P}, \mathcal{O}, \mathcal{F}, \mathcal{G}, \mathsf{Gen}, \mathsf{Spec}, \mathsf{Base}, \mathsf{Elt} \rangle$$

is a value assignment to the object types in $\mathcal{O}$

$$\mathsf{Pop} : \mathcal{O} \rightarrow \wp(\Omega)$$

conforming to the structure as prescribed by $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{G}$, and respecting the generalisation and specialisation hierarchies Gen and Spec. This is denoted as $\mathsf{IsPop}(\mathcal{I}, \mathsf{Pop})$. In the sequel of this paper we will omit quantifications over Pop.

A population assigns a set of instances to each object type in $\mathcal{O}$. These instances come from a (pre-existing) set Ω. Instances may have a structure (they may e.g. be a set). This structure

should conform to the structure of the object types to which they are assigned. This is specified by means of a number of axioms stated in the sequel of this section.

Respecting the specialisation hierarchy is reflected by the *Specialisation Rule*:

(P1) $x \,\mathsf{Spec}\, y \Rightarrow \mathsf{Pop}(x) \subseteq \mathsf{Pop}(y)$

Intuitively, object types can, for several reasons, have values in common in some instantiation. For example, each value of object type x will, in any instantiation, also be a value of object type $\sqcap(x)$. As another example, suppose $x \,\mathsf{Gen}\, y$, then any value of y in any population will also be a value of x. A third example, where object types may share values is when two power types have element types that may share values. This is formalised in the concept of *type relatedness*, which 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:

(T1) $\vdash x \sim x$

(T2) $x \sim y \vdash y \sim x$

(T3) $\sqcap(x) = \sqcap(y) \wedge y \sim z \vdash x \sim z$

(T4) $x \,\mathsf{Gen}\, y \wedge y \sim z \vdash x \sim z$

(T5) $x, y \in \mathcal{G} \wedge \mathsf{Elt}(x) \sim \mathsf{Elt}(y) \vdash x \sim y$

The *Strong Typing Rule* expresses that instantiations of object types can *only* have instances in common, if they are type related.

(P2) $\mathsf{Pop}(x) \cap \mathsf{Pop}(y) \neq \emptyset \Rightarrow x \sim y$

Respecting the Generalisation hierarchy is reflected by the *Generalisation Rule*:

(P3) $gen(x) \Rightarrow \mathsf{Pop}(x) = \bigcup \{\, \mathsf{Pop}(y) \mid x \,\mathsf{Gen}\, y \,\}$

The population of an atomic object type is just a set of values. The population of a fact type is a set of tuples, which are mappings from the predicators of the fact type to values of the appropiate type. This is referred to as the *Conformity Rule*.

(P4) $\forall_{f \in \mathcal{F}, p \in f, t \in \mathsf{Pop}(f)} \left[t : f \rightarrow \Omega \wedge t(p) \in \mathsf{Pop}(\mathsf{Base}(p)) \right]$

The population of power types consists of sets of instances in the population of their element types. This is called the *Power Type Rule*:

(P5) $g \in \mathcal{G} \wedge y \in \mathsf{Pop}(g) \Rightarrow y \in \wp(\mathsf{Pop}(\mathsf{Elt}(g))) - \{\emptyset\}$

Example 2.5 *An example of a population of the information structure as presented in figure 14 could be:*

$$
\begin{array}{rcl}
\mathsf{Pop}(A) & = & \{a_1, a_2\} \\
\mathsf{Pop}(B) & = & \{b_1\} \\
\mathsf{Pop}(C) & = & \{c_1\} \\
\mathsf{Pop}(D) & = & \{d_1\} \\
\mathsf{Pop}(E) & = & \{\{a_1\}, \{a_1, a_2\}\} \\
\mathsf{Pop}(f) & = & \{\{q \rightarrow a_1, p \rightarrow b_1\}, \{q \rightarrow a_2, p \rightarrow b_1\}\} \\
\mathsf{Pop}(g) & = & \{r \rightarrow \{q \rightarrow a_1, p \rightarrow b_1\}, s \rightarrow d_1, t \rightarrow c_1\} \\
\mathsf{Pop}(h) & = & \{\, \{u \rightarrow \{a_1\}, v \rightarrow c_1\}, \\
 & & \{u \rightarrow \{a_1, a_2\}, v \rightarrow c_1\}\}
\end{array}
$$

2.4 Constraints

The specification of a Universe of Discourse will contain the following components:

1. a description of the set of states, also called the underlying *information structure*.
2. a description of the set of transitions, usually as a set of actions that bring about those transitions.

In this paper, focus is on the description of the set of states. A well accepted method for describing sets is the use of (formal) grammars. In fact, the information structure is sometimes also denoted as the grammar in terms of which the communication with the information system is formulated (see [38]).

For information systems the description more closely follows the style of *abstract datatypes*. The information structure is described as a set of (elementary) fact types over a set of object types. A state then is a *population* (or instantiation) of these fact types. Usually not all populations correspond to states of the U∘D, and therefore have to be excluded. This is done by imposing constraints on the information structure. Static constraints simply exclude invalid populations. Dynamic constraints forbid certain transitions, and therefore may render states unreachable from the initial state. These unreachable states are also not valid.

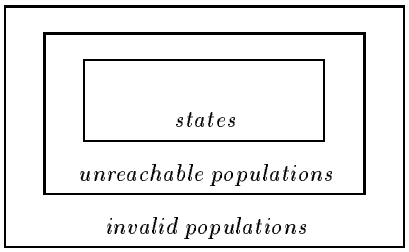


Figure 17: The classification of populations

This style of describing is called the *superset technique*. This technique is used in cases where a precise desciption is cumbersome. One then looks for an easy way to describe a superset, and exclude invalid elements by imposing constraints on the elements of the superset. For more information about formalisation, see [14].

In this section, we will restrict ourselves to some typical static constraints in relation to power typing. It should be noted that the constraints that were introduced for PM, including well-known constraints as e.g. the total role and the uniqueness constraint, are also applicable here. For their definition, which makes use of relational operators, such as projection, selection and unnesting, that are part of a relational algebra that has been introduced for PM, we refer to [3]. Appendix A contains a complete list of all types of constraints and their graphical representations.

2.4.1 Exclusion Constraint

A power type may have the property that in any population all its instances are disjunct. This property can be expressed using the *exclusion constraint*. Graphically, this constraint has the same representation as the exclusion constraint for fact types, it is however connected to the power type involved. In figure 18 an example of the exclusion constraint is shown. In this example, groups of students receive a mark for a course. The exclusion constraint expresses that a student should not be a member of several tutorial groups.

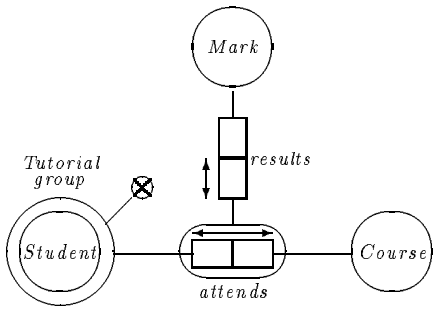


Figure 18: Example of exclusion constraint

Formally, an exclusion constraint τ is a singleton set $\{g\}$ with $\sqcap(g) \in \mathcal{G}$. A population Pop satisfies an exclusion constraint τ, denoted as $\mathsf{Pop} \models \mathsf{exclusion}(\tau)$, if:

$$\forall_{p,q \in \mathsf{Pop}(g)} [p \cap q \neq \emptyset \Rightarrow p = q]$$

2.4.2 Cover Constraint

The requirement that every instance of a certain object type has to occur in an instance of its power type can be expressed by the *cover constraint*. Graphically this constraint is depicted as a total role constraint connected to the power type involved. An example is shown in figure 19: each student should be member of at least one tutorial group.

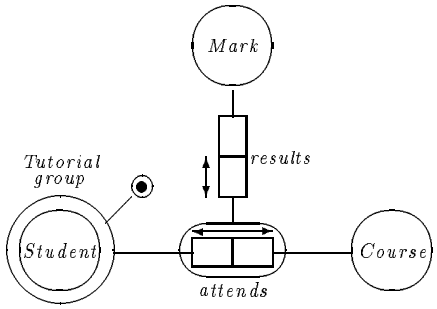


Figure 19: Example of cover constraint

A cover constraint may also be assigned to a subtype of a power type. The population of the subtype should then cover the population of the element type of its pater familias.

Formally, a cover constraint τ is a singleton set $\{g\}$ with $\sqcap(g) \in \mathcal{G}$. A population Pop satisfies a cover constraint τ, $\mathsf{Pop} \models \mathsf{cover}(\tau)$, if:

$$\bigcup \mathsf{Pop}(g) = \mathsf{Pop}(\mathsf{Elt}(\sqcap(g)))$$

Naturally, it is possible to combine the exclusion constraint and the cover constraint. In this way it can be expressed that a power type is a partition. An example is shown in figure 20: each student should be member of precisely one tutorial group.

2.4.3 Cardinality Constraint

It may be the case, that instances of power types have a minimum or maximum number of elements. A *cardinality constraint* enforces that the instances of a power type consist of at least n and at

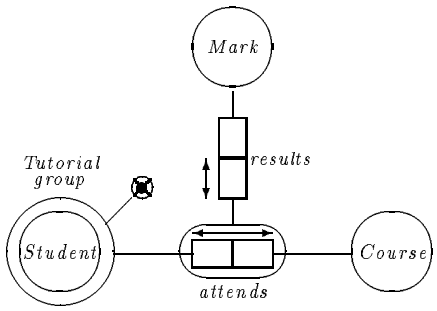


Figure 20: Example of partition constraint

most m elements (n and m are arbitrary natural numbers). Graphically a cardinality constraint is depicted as an occurrence-frequency constraint connected to the power type involved. For an example consider figure 21: tutorial groups should have a reasonable size.

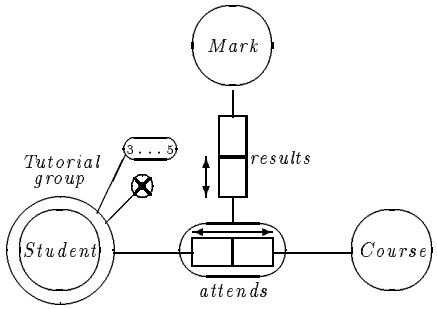


Figure 21: Example of cardinality constraint

Formally, a cardinality constraint τ is a tuple $\langle g, n, m \rangle$ with $\sqcap(g) \in \mathcal{G}$ and $n, m \in \mathcal{N}$. A population Pop satisfies a cardinality constraint τ, $\mathsf{Pop} \models \mathsf{cardinality}(\tau)$, if:

$$\forall_{p \in \mathsf{Pop}(g)} [n \leq |p| \leq m]$$

2.4.4 Membership Constraint

A *membership constraint* can be used to express that instances should be element of other instances. Consider for example figure 22, which is an extension of our convoy example. A convoy can have a *flagship*. The membership constraint expresses that the flagship of a convoy should be part of that convoy.

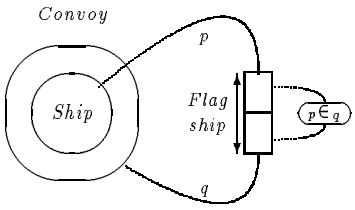


Figure 22: Example of membership constraint

Formally, a membership constraint μ is a tuple $\langle p, q \rangle$ with $p, q \in \mathcal{P}$ such that $\mathsf{Fact}(p) = \mathsf{Fact}(q)$ and $\mathsf{Elt}(\sqcap(\mathsf{Base}(q))) = \sqcap(\mathsf{Base}(p))$. A population satisfies a membership constraint μ, $\mathsf{Pop} \models \mathsf{member}(\mu)$, if:

$$t \in \mathsf{Pop}(\mathsf{Fact}(p)) \Rightarrow t(p) \in t(q)$$

In the sequel we will write $\mathsf{IsPop}(\Sigma, \mathsf{Pop})$ if Pop is a population of the information structure $\mathcal{I}$ of Σ (i.e. $\mathsf{IsPop}(\mathcal{I}, \mathsf{Pop})$) and Pop satisfies all the constraints specified.

3 Motivation from the Axioms of Set Theory

In this section the premises from PSM are motivated, and the universality of this data model argued. The approach is to compare the model with the axioms from formal set theory, the so-called Zermelo-Fraenkel system of axioms (see for example [7]).

3.1 Extensionality

First we consider the Axiom of Extensionality, which states that a set is characterised by its elements:

$$\forall_z \left[z \in x \Leftrightarrow z \in y \right] \Rightarrow x = y$$

The axiom states that two sets, having the same elements, are equal. This axiom is encountered in all set oriented data models, where entity types and fact types are determined by their instances. In PSM, this axiom is also used as a motivation for the concept of power type. Instances of power types are equal if and only if their elements are the same. This results in the observation, already mentioned in section 2.1.1, that power types in PSM can be identified in terms of their element types.

3.2 Specialisation

The Subset Schema (Comprehension Schema) is an axiom schema, that allows for the construction of a new set b by resticting a set a via selection formula ϕ, in the context of sets $x_1, \ldots, x_n$:

$$\forall_y \left[y \in b \Leftrightarrow (y \in a \wedge \phi(x_1, \ldots, x_n, y)) \right]$$

Note that the superset a is essential in this schema: omission would yield the well known Russell paradox, which results by choosing the selection formula $\phi(x) \equiv x \notin x$. The paradox then easily is derived from the expression $b \in b$.

The Comprehension Schema corresponds to the notion of specialisation (also referred to as subtyping) in the usual data modelling techiques. In PSM b would be defined as a subtype of a. Formula ϕ then corresponds to the subtype defining rule.

$$\mathsf{Pop}(b) = \left\{ x \mid x \in \mathsf{Pop}(a) \wedge \phi(x) \right\}$$

3.3 Derivation

The Replacement Schema is a more general schema than the Comprehension Schema. A formula ψ is considered, that has a functional behaviour:

$$\psi(x_1, \ldots, x_n, u, v) \wedge \psi(x_1, \ldots, x_n, u, w) \Rightarrow v = w$$

Then a new set b can be constructed by applying ψ to all elements of a given set a. The elements of b are exactly those y which correspond, under ψ, to some x from a:

$$\forall_y \left[y \in b \Leftrightarrow \exists_x \left[x \in a \wedge \psi(x_1, \ldots, x_n, x, y) \right] \right]$$

In this schema each $x \in a$ is "replaced" by the corresponding y. The Replacement Schema corresponds to the notion of derived object types (derived information).

$$\mathsf{Pop}(b) = \{\ y \mid \exists_x \left[x \in \mathsf{Pop}(a) \wedge \psi(x, y) \right] \}$$

The population of b thus is defined in terms of the population of a by derivation rule ψ. To express complex derivation rules, a constraint language has to be introduced for PSM. The graphical constraints are, in general, not powerful enough for this purpose.

3.4 Power Type

A special notion in formal set theory is the notion of power set. By applying a clever numbering schema for subsets (see [7]) it is easily shown that this concept is not derivable. The powerset is defined as:

$$\wp(x) = \{\ y \mid y \subseteq x \}$$

The Power Set Axiom enforces the existence of this operator. This notion of power set corresponds to the notion of power type in PSM. Instances of power types are sets of instances of their element type, though not necessarily all possible sets of those instances.

3.5 Generalisation

The motivation for the concept of generalisation can be found in the Union Axiom. This axiom states that one can construct a set b from a set a, by uniting all elements of a. The set b is denoted as $\bigcup a$.

$$\bigcup a = \{\ z \mid \exists_y \left[y \in a \wedge z \in y \right] \}$$

In the context of PSM, we call $\bigcup a$ the generalisation of its elements. As a special case, suppose

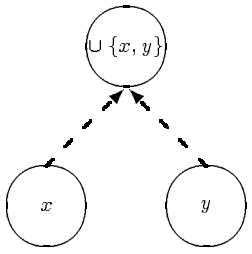


Figure 23: Union of object types

$a = \{x, y\}$, then we have (see figure 23):

1. $\cup \{x, y\}\, \mathsf{Gen}\, x$
2. $\cup \{x, y\}\, \mathsf{Gen}\, y$

Applying the *Generalisation Rule* then yields:

$$\mathsf{Pop}(\cup\{a, b\}) = \mathsf{Pop}(a) \cup \mathsf{Pop}(b)$$

3.6 Enumeration

The Pair Set axiom states that if x and y are sets, then there is another set, denoted as $\{x, y\}$, which has exactly x and y as members. In PSM this can be interpreted using the enumeration constraint ([3]), which is a set of objects associated with a label type, restricting the population of that label type to these objects. In case of a label type L and objects x and y, an enumeration constraint associating $\{x, y\}$ to L enforces:

$$\mathsf{Pop}(L) \subseteq \{x, y\}$$

3.7 Finite hierachy

The Axiom of Foundation states that sets are hierarchically constructed, there are no infinite descending element chains. This corresponds with the notion of structural identification, which will be discussed in the next section. Structural identification excludes the definition of types which are not well-formed. An example would be cyclic power typing. Suppose that we have object types A and B such that $\mathsf{Elt}(A) = B$ and $\mathsf{Elt}(B) = A$. This would correspond to sets A and B such that $A \subseteq \wp(B)$ and $B \subseteq \wp(A)$. In this case A and B are not well-founded. Another example would be a fact type $f = \{p_1, p_2\}$ with $\mathsf{Base}(p_1) = \mathsf{Base}(p_2) = f$. This would correspond to a set f for which $f \subseteq f \times f$, which would also not be well-founded. In general, object types which are not structurally identifiable correspond to sets which are not well-founded. In figure 24 we see an example of a finite hierarchy. In this case, no infinite descending element chains occur. A *Company* is a set of *Platoons* (at least 5 and at most 8), a *Platoon* is a set of *Soldiers* (at least 9 and at most 20) and *Soldiers* are atomic entities.

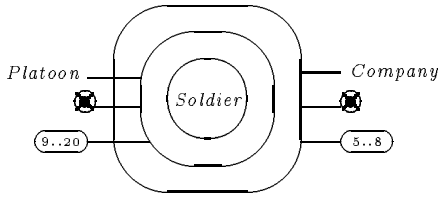


Figure 24: A finite hierarchy

4 Identification

In this section we will discuss identification. A distinction is made between weak identification and structural identification. Informally, a population is weakly identified if objects with the same properties are equal. Structural identification is defined on the schema level and guarantees weak identification for every population from schema properties.

4.1 Weak Identification

A population of a PSM schema maps all object types to sets of instances. In some applications, the difference between instances is not relevant, and need not even be expressible. A typical application is a chicken farm where eggs are important entities. However, the difference between eggs need not be expressible, only their number will be of importance. In cases, where the difference between objects is not expressible within the model, we speak of *implicit identification*.

Usually the difference between objects must be expressible in terms of the model. In such cases we use the term *explicit identification*.

Labels are the elementary data types, and are considered to be representable directly. As a result, each label can be identified by itself. We assume an equality operator for labels. Entities on the other hand can only be represented by their properties. As a consequence, entities with the same properties are not distinguishable, and therefore considered to be the same. The properties of an entity are recorded by the facts in which it plays a role. A population is *weakly identified* if all entity values with the same properties are equal. For a formal definition, we refer to [3].

Composite objects were introduced as tuples, and are identified by their components. This means that instances of fact types are equal if their components are equal. Power type instances are sets, which are identified by their elements. Consequently, in any population Pop we have for all values s and t of any power type g:

$$\forall_e [e \in s \Leftrightarrow e \in t] \quad \Rightarrow \quad s = t$$

Information structures may contain cycles, where a fact type contains itself directly or indirectly

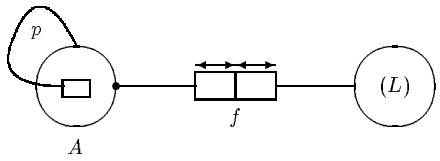


Figure 25: Cyclic object type

as a component. An example is given in figure 25. As a consequence, the population of such a structure may contain an object, that contains itself as a component. For example, an object $a \in \mathsf{Pop}(A)$, such that $a = \langle p : a \rangle$. Such cyclic objects are called *virtual objects*.

4.2 Structural Identification

In this section it is considered how weak identification can be guaranteed from properties of the schema (the constraints in particular).

A PSM schema Σ is *structural identifiable* iff:

1. Σ is closed over labels, i.e., each label type occurs in some total role constraint:

   $$\forall_{x \in \mathcal{L}} \exists_{p \in \mathcal{P}} \exists_{\mathsf{total}(\tau) \in \mathcal{C}} [\mathsf{Base}(p) = x \wedge p \in \tau]$$

   The motivation behind this is to enforce the absence of unused labels.

2. All object types can be identified:

   $$\forall_{x \in \mathcal{O}} [\mathsf{Identifiable}(x)]$$

   The identification of an object can be seen as a fixed set of properties that provide a unique description in terms of labels. This identification then guarantees that any valid population is weakly identified.

The predicate $\mathsf{Identifiable}$ is defined in terms of the structure of objects. The respective object classes are discussed subsequently.

Label Types
If x is a label type, then obviously $\mathsf{Identifiable}(x)$.

Fact Types
If x is a composed object type (or, generally, a set of predicators), then we may conclude $\mathsf{Identifiable}(x)$ if all components of x are identifiable:

$$\forall_{p \in x} \left[\mathsf{Identifiable}(\mathsf{Base}(p))\right]$$

Power Types
If x is a power type then x is identifiable if:

$$\mathsf{Identifiable}(\mathsf{Elt}(x))$$

As stated before, sets are identified by their elements.

Entity Types
If x is an entity type, then we can distinguish the following cases.

If x is *not* pater familias ($\sqcap(x) \neq x$) then x takes (inherits) its identification from its pater familias ($\mathsf{Identifiable}(\mathsf{Pop}(x))$), provided that the subtype membership is decidable from the properties (constraints) of the schema. This latter is expressed in the schema by subtype defining rules (see [3]).

A second case of identification inheritance arises from object generalisation. In this case the object type inherits its identification from some of its specifiers. More precisely, if x is a generalised object type ($gen(x)$), then x is identifiable if and only if:

$$\exists_{y \in \mathcal{O}} \left[x \,\mathsf{Gen}\, y \wedge \mathsf{Identifiable}(y)\right]$$

Note that we do not require for the identification of a generalised object the identification of *all* its specifiers. The reason is that the identification of a specifier may depend on the identification of the generalised object type. For example, the identification of fact type f in figure 11 is dependent on the identification of *formula*. The identification of *formula* depends on its turn on the identification of *variable*.

This leaves us with the identification of object types that are pater familias, but not the generalisation of another object type. In this case we are looking for identification paths (denominations). These denominations form a recipee for uniquely denoting each object of the object type, in any population. The set of possible first names for denominations is defined by:

$$N(x) = \left\{\, p \mid \mathsf{Base}(p) = x \wedge \mathsf{total}(\{p\}) \wedge \mathsf{unique}(\{p\}) \,\right\}$$

The identification of object type x now depends on the existence of a set of middle names (constituting a so-called identifier), i.e., a set τ of predicators such that:

- $\mathsf{unique}(\tau)$
- $\forall_{p \in \tau} \exists_{q \in \mathsf{Fact}_{(p)}} \left[q \in N(x)\right]$
- $\forall_{p \in \tau} \left[\mathsf{Identifiable}(p)\right]$

In this definition, $\mathsf{unique}(\tau)$, expresses that there is a uniqueness constraint on the predicators in τ.

The following theorem demonstrates that we attained our purpose of guaranteeing weak identification from structural identification. The proof is straightforward, and analogous to the proof of the corresponding theorem in [3].

Theorem 4.1

$$\Sigma \textit{ identifiable} \Rightarrow \textit{each population weakly identified}$$

Identifiability for power types or fact types, expresses whether they can be populated with non-virtual objects. An example of a virtual object would be an infinite list. In figure 26 a PSM schema modelling an infinite list is shown.

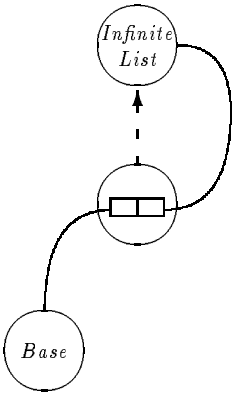


Figure 26: Schema of an infinite list

Virtual objects could be introduced as fruitful elements of computing (see for example [2]). In this paper however we restrict ourselves to non-virtual objects.

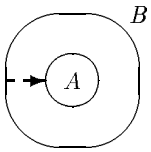


Figure 27: Non-identifiable object type

Example 4.1 Figure 27 shows an example of an object type that is not structurally identifiable. In this figure $A\,\mathsf{Gen}\,B$ and $\mathsf{Elt}(B) = A$. Object type A cannot be populated with non-virtual objects. This situation interpreted in set theory would yield two sets A and B for which $B \subseteq \wp(A)$ and $B = A$. These sets would not be well-founded, see section 3.7. No problems occur however, if A has another specifier C, which is identifiable.

Note that $B\,\mathsf{Gen}\,A$ is not possible, as Gen is a relation over $\mathcal{A} \times \mathcal{O}$, while $B \in \mathcal{G}$.

The problem of structural identification can also be related to predictive type theory (see [13]). Considering figure 27 it is clear that instances from object type B cannot be properly typed. Their types would have to be cyclic structures, while predictive type theory enforces types to be hierarchical structures.

In some cases it may be difficult to establish whether object types fulfil the conditions of the predicate $\mathsf{Identifiable}$. The following lemma, based upon topological sort (see [20]), may then be useful.

Lemma 4.1 (Nesting Criterion) *If and only if a PSM schema is identifiable, there exists a monotone nest counter, i.e. a function* $h : \mathcal{O} \rightarrow \mathbf{N}$*, such that:*

1. $f \in \mathcal{F} \wedge p \in f \Rightarrow h(f) > h(\mathsf{Base}(p))$
2. $g \in \mathcal{G} \Rightarrow h(g) > h(\mathsf{Elt}(g))$
3. $gen(x) \Rightarrow \exists_{y \in \mathcal{O}} [x \ \mathsf{Gen}\ y \wedge h(x) > h(y)]$

Proof: A monotone nest counter is obtained by numbering the atomic (non-generalised) object types in arbitrary order, and then deriving numbers for object types according to the rules of identifiability.

□

Corollary 4.1 *If a PSM schema is identifiable, then the information structure cannot contain cyclic power type structures.*

Example 4.2 Consider again figure 27. Using the Nesting Criterion, identifiability of this schema would imply the existence of a monotone nest counter h that satifies the conflicting requirements $h(A) > h(B)$ and $h(A) < h(B)$.

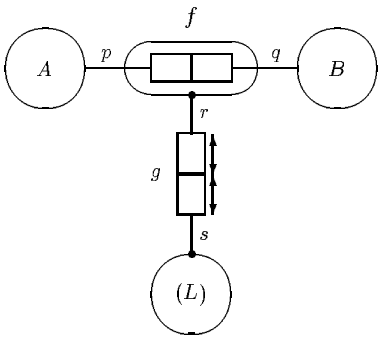


Figure 28: Standard names but not identifiable

In figure 28 an information structure is depicted, where the identification of object type f depends on the identification of both A and B. However, f has also a standard name, that is not dependent on either A or B. Therefore it can be concluded that the presence of standard names is not sufficient to conclude identifiability.

5 Schema Objectification and Sequencing

In this section two notational shorthands are introduced, schema objectification and sequencing, that facilitate the specification of complex object types. Examples will be discussed, which make use of these shorthands in order to demonstrate their elegance in modelling. This section concludes with a meta-model of PSM expressed as a PSM schema. This is an indication of the capability of PSM to model complex structures.

5.1 Schema Objectification

The need for decomposition in large systems, has been generally recognised. A well known example is the decomposition mechanism for Activity Graphs ([27]). In an Activity Graph, both processes and data may be subject to decomposition. However, data modelling techniques usually

do not provide a decomposition mechanism. In PSM, schema objectification is introduced for this purpose.

Schema objectification is a construction mechanism that allows to define part of a schema as an object type. Instances of such object types are then populations of their corresponding schemas. As a result, these objectified schemas have to be valid information structures. Formally, if Σ is a schema which is objectified into object type O, we have the syntactical requirement that Σ should be a valid PSM schema. Furthermore, populations should satisfy the *decomposition rule*, meaning, that instances of O should be valid populations of Σ:

(P6) $p \in \mathsf{Pop}(O) \Rightarrow \mathsf{IsPop}(\Sigma, p)$

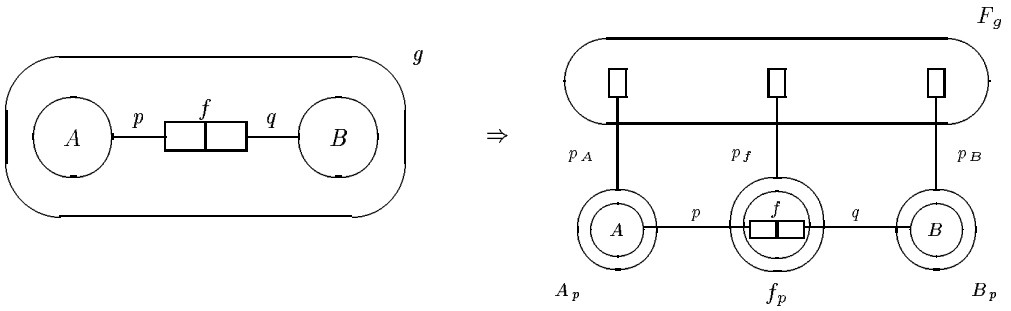


Figure 29: Schema objectification

Schema objectification, however, is not an elementary concept, since it can be defined in terms of the concepts of power type and fact type. The idea is to construct a power type x_p for each object type x from the schema g to be objectified. Each of these power types x_p is the base of a predicator p_x, that is part of a fact type F_g (see figure 29). This fact type is to relate sets of instances of the object types involved in the schema objectification, which are part of the same schema instance.

5.2 Example: Activity Graphs

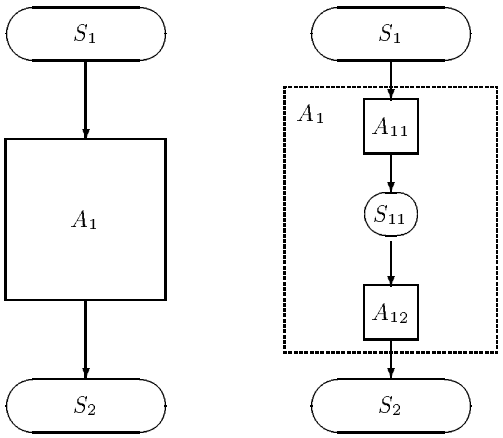


Figure 30: Sample activity graphs

Using schema objectification one can elegantly define the meta-model of Activity Graphs. In figure 30 two sample activity graphs are depicted.

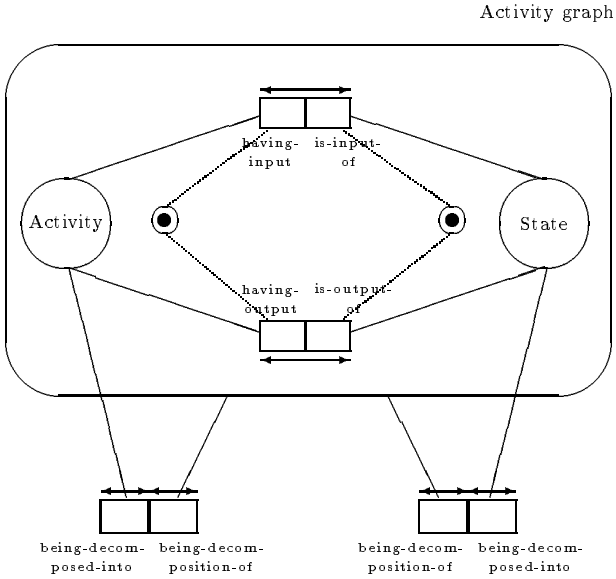


Figure 31: Meta-model of activity graphs

Activity Graphs are bipartite directed graphs consisting of activities and states. The direction of the arrow between an activity and a state indicates whether that state is input or output of that activity. Activities and states can be decomposed in other Activity Graphs. In figure 30 the rightmost Activity Graph shows the decomposition of activity A1.

In figure 31, the meta-model of Activity Graphs is depicted. As can be seen, an Activity Graph is an objectified schema consisting of activities, states and input and output relations. The binary relations between activity and Activity Graph and state and Activity Graph represent the decomposition relation.

5.3 Sequencing

Sequencing is a construction mechanism that allows to define a new object type whose instances are tuples of arbitrary length of another object type. Graphically, the sequence type is depicted as a rectangle around its associated object type.

This construction mechanism however, is also not elementary. In figure 32, the translation of a sequence type to a generalised object type is shown.

5.4 Example: parameter passing mechanism

As an example, consider function overloading in a language like C++ ([30]). In C++ a function may have several definitions, where each definition has different types of parameters. In some languages, the number of parameters is *not* even fixed (e.g. ALGOL68). We assume, that each parameter type sequence for some function may contain at most one output parameter type, which is uniquely determined by the input parameter types and there ordering.

This peculiar Universe of Discourse is described in figure 33 as a PSM schema. In this schema *Parameter Type Sequence* is a sequence type.

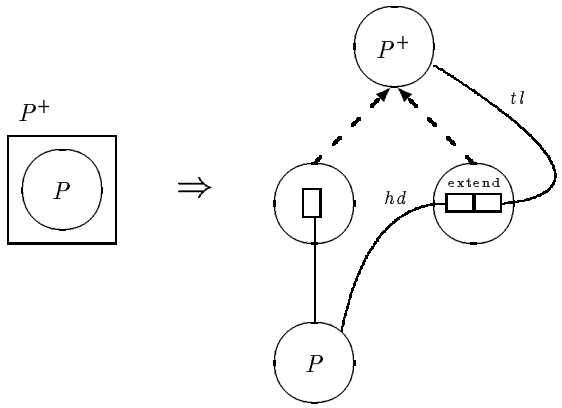


Figure 32: Sequencing

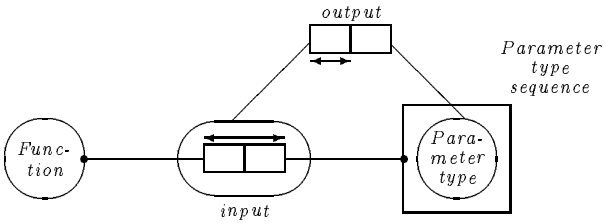


Figure 33: Example of sequencing

5.5 Example: a Meta-model

This section is concluded with a meta-model of PSM (without its constraints as e.g. introduced in section 2.4) expressed as a PSM schema. This meta-model is shown in figure 34.

The meta-model contains the following label types:

$$\mathcal{E},\ \mathcal{L},\ \mathcal{G},\ \mathcal{P}$$

The meta-model has $\mathcal{A}$ and $\mathcal{O}$ as generalised object types, while $\mathcal{F}$ is a power type, with $\mathsf{Elt}(\mathcal{F}) = \mathcal{P}$. The meta-model contains the following fact types and predicators:

$$\begin{aligned} \mathsf{Gen} &= \{g_1, g_2\} \\ \mathsf{Spec} &= \{s_1, s_2\} \\ \sqcap &= \{\sqcap_1, \sqcap_2\} \\ \mathsf{Base} &= \{b_1, b_2\} \\ \mathsf{Elt} &= \{x_1, x_2\} \end{aligned}$$

The bases of the predicators are given informally in figure 34.

The meta-model of figure 34 demonstrates the expressive power of PSM with regard to the modelling of complex structures. It can express its own complex structure.

6 Relation with Context-Free Grammars

In this section it is discussed how the expressive power of context-free grammars can be embedded in PSM. First it is shown how a context-free grammar is translated into a PSM schema. Since

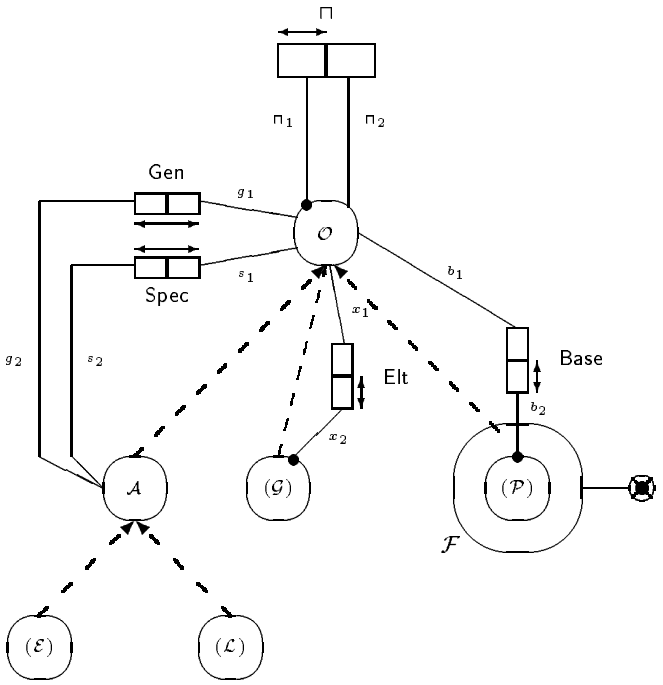


Figure 34: Meta-model of PSM

context-free grammars are often employed for describing hypertext information structures (see for example [4],[29]), the translation also shows the feasibility of PSM for describing hypertext information structures. In the translation, generalised object types will play a crucial role. The reverse translation, describing how a PSM schema can be translated into a context-free grammar, can be found in [15]. We conclude with a larger example in which the syntactical description of path expressions, which form the basis of more advanced languages for data manipulation in the style of NIAM, is given.

6.1 Embedding Context-Free Grammars

We start with the well-known definition of a context-free grammar.

Definition 6.1 *A context-free grammar G is a tuple $\langle N, \Sigma, \Pi, S\rangle$, where N is a finite set of nonterminal symbols, Σ is a finite set of terminal symbols, $S \in N$ is the initial symbol and Π is a set of production rules of the form $A \rightarrow \omega$ where $A \in N$ and $\omega \in (N \cup \Sigma)^*$.*

In the sequel we restrict ourselves to production rules with a non-empty right-hand side. This restriction is not very severe, as only the possibility of generating the empty string is lost (see for example [12]). The empty string would correspond to an information structure without object types.

We describe the translation Δ of a context-free grammar G into a PSM schema $\Delta(G)$. Each symbol of G is interpreted as an atomic object type. Nonterminal symbols are interpreted as entity types, while terminal symbols are interpreted as label types. Let $P \in \Pi$ be a production rule, and let P be of the form $t \rightarrow s_1, \ldots, s_n$. We interpret this rule as a description of a generalisation for object type t. The right-hand side is the specifier of this generalisation, and considered as an objectified fact type P_f. This fact type consists of predicators $P_{s_1}, \ldots, P_{s_n}$, which have the object

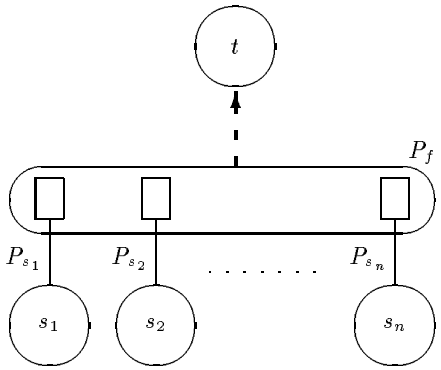


Figure 35: Translation of production rules

types $s_1, \ldots, s_n$ respectively as bases. In figure 35 this translation of production rules is depicted graphically.

This concludes the translation of a context-free grammar into a PSM schema. Note that power types do not result from this translation.

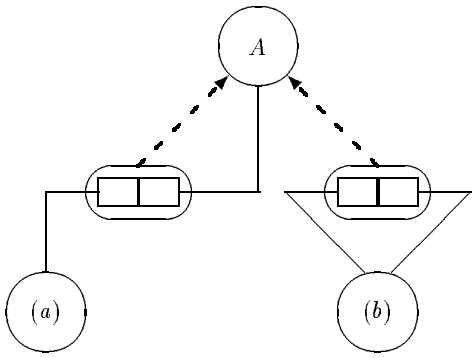


Figure 36: Example of translation of context-free grammar

Example 6.1 *Let G be the context-free grammar with the following production rules:*

$$\begin{aligned} \langle A \rangle &\rightarrow \langle a \rangle \langle A \rangle \\ \langle A \rangle &\rightarrow \langle b \rangle \langle b \rangle \end{aligned}$$

In figure 36 the resulting PSM schema $\Delta(G)$ is shown.

It may be convenient to allow grammar rules of the form $t \rightarrow s^+$, which are shorthand for the grammar rules $t \rightarrow s$ and $t \rightarrow st$. In this case sequence types, introduced in section 5.3, can be used in the grammar translation. In figure 37 a simplified translation of the production rule $t \rightarrow s^+$ is shown, which is possible in the case of non-terminals with a single defining production rule. The following example makes full use of this simplified translation.

Example 6.2 *Consider the following grammar in the style of SGML ([18]) for describing the structure of a book.*

$$\langle book \rangle \rightarrow \langle title \rangle \langle contents \rangle$$

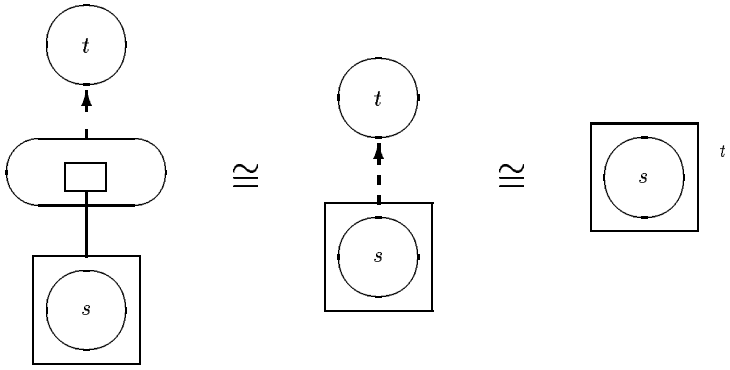


Figure 37: A simplified translation of $t \rightarrow s^+$

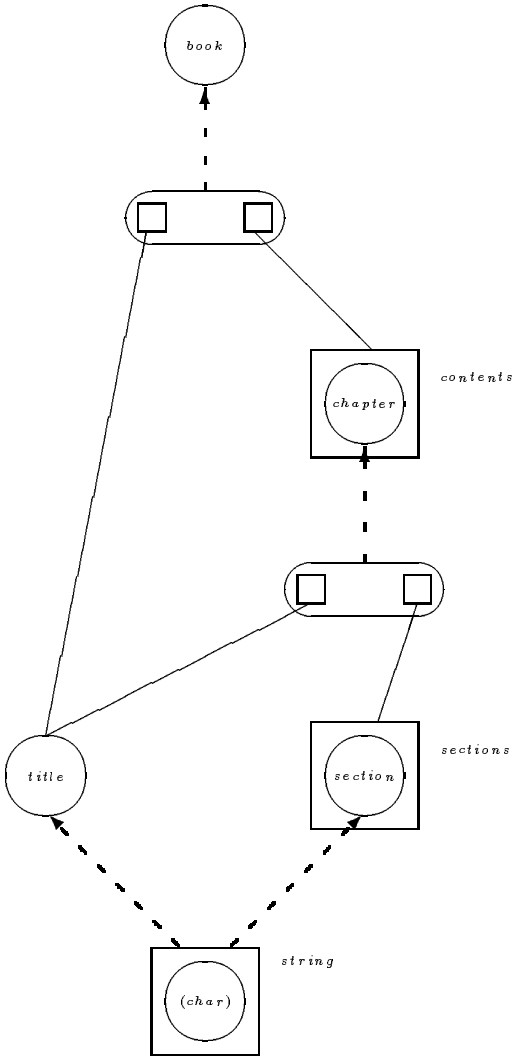


Figure 38: Example of translation of SGML structure

$$
\begin{aligned}
\langle contents \rangle &\rightarrow \langle chapter \rangle^{+} \\
\langle chapter \rangle &\rightarrow \langle title \rangle \, \langle sections \rangle \\
\langle sections \rangle &\rightarrow \langle section \rangle^{+} \\
\langle section \rangle &\rightarrow \langle string \rangle \\
\langle title \rangle &\rightarrow \langle string \rangle \\
\langle string \rangle &\rightarrow \langle char \rangle^{+}
\end{aligned}
$$

This results in the PSM schema of figure 38.

It is important to note that the PSM schema resulting from the translation of a context-free grammar does not exhibit explicitly the order of the symbols in the right-hand side of production rules. This corresponds to a *mapping oriented* view to the right-hand side of a production rule, rather than the usual tuple oriented view. The resulting PSM schema can be viewed as a representation of the abstract syntax ([24]) corresponding to the grammar at hand.

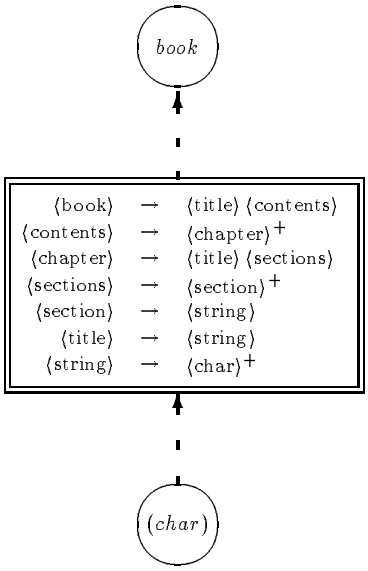


Figure 39: A grammar box

Grammars can be incorporated in a PSM schema via the grammar box. An example of a grammar box is depicted in figure 39. The grammar box takes as inputs the object types that correspond to terminal symbols. The output of the grammar box is the start symbol. With respect to this use of context-free grammars, a bad schema will result if the context-free grammar does not satisfy some aesthetical rules. Firstly, there can be useless symbols, i.e. symbols that do not occur in any derivation from the start symbol. In terms of PSM these symbols correspond to superfluous object types. Secondly, the object types that correspond to the terminal symbols can be identified without making use of the grammar box, since they are interpreted as label types. The identification of the object type, corresponding with the start symbol of the grammar, then depends on the structure of the grammar. The following theorem shows that proper identification precisely corresponds to the productiveness of the nonterminal symbols in the grammar. A nonterminal x that is not productive (i.e. $L(x) = \emptyset$) will not be identifiable. A grammar with only productive nonterminal symbols is called universally productive. The test of this property can be done in linear time ([12]).

Theorem 6.1 *Let $G = \langle N, \Sigma, \Pi, S \rangle$ be a context-free grammar and let $\Delta(G)$ be the corresponding Predicator Set schema, then:*

$$\forall_{T \in N} \exists_{\omega \in \Sigma^*} \left[T \xrightarrow{*} \omega \right] \Leftrightarrow \forall_{x \in N} \left[\mathsf{Identifiable}(x) \right]$$

Proof: Object types that correspond to terminal symbols are structurally identifiable, since they correspond with label types. First consider the translation of production rules into schema fragments (see figure 35). We see that the identification requirement for object types that correspond to a nonterminal symbol, can be rewritten, by applying the identification rule for generalised object types, as:

$$\forall_{x \in N} \exists_{\text{rule } p: x \rightarrow s_1 .. s_n} \forall_{1 \leq i \leq n} \left[\mathsf{Identifiable}(s_i) \right]$$

Furthermore, if x is a productive nonterminal symbol, then there exists a derivation from x resulting in a string that does only contain terminal symbols. This derivation is a frame for the identification of x.

□

6.2 An Extended Example

We conclude with an example that describes a relevant subset of the language RIDL, dealing with path expressions. This language was introduced as an "almost natural language" for the manipulation of NIAM schemata ([32], [23]).

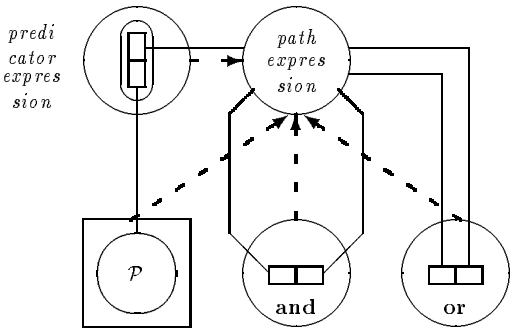


Figure 40: The definition of path expressions

In object role models the concept of path expressions is introduced as a means for conveniently selecting specific sets of information. We consider a very simple subset, that is governed syntactically by the following syntax:

$$\begin{aligned}
\langle \text{pathexpr} \rangle &\rightarrow \langle \text{predicator} \rangle^+ \\
\langle \text{pathexpr} \rangle &\rightarrow \langle \text{pathexpr} \rangle \ \textbf{and} \ \langle \text{pathexpr} \rangle \\
\langle \text{pathexpr} \rangle &\rightarrow \langle \text{pathexpr} \rangle \ \textbf{or} \ \langle \text{pathexpr} \rangle \\
\langle \text{pathexpr} \rangle &\rightarrow \langle \text{predexpr} \rangle^+ \\
\langle \text{predexpr} \rangle &\rightarrow \langle \text{predicator} \rangle \ \langle \text{pathexpr} \rangle
\end{aligned}$$

Consider figure 40, that shows the corresponding PSM schema.

The meaning of this abstract language is as follows. A path expression in its simplest form is a sequence of predicators. In this case, the sequence describes a chain for joining. The logical

connectives are the counterpart for intersection and union. New relations are formed by *predexpr*, by specifying the new predicators, and their associated value. As we use the mapping oriented view for relations, we prefer the power type construct in figure 40 rather than the sequence mechanism for *predexpr*.

Acknowledgement

We would like to thank Peter Bruza for his contribution in the translation of context-free grammars to Predicator Set schemas. Furthermore, we would like to thank Gerard Wijers and Denis Verhoef for their comments on an earlier version of this paper.

Appendix: Legend of graphical symbols

This appendix contains an overview of the symbols used in this paper. In table 1 we see all symbols for object types. Table 2 we see the notational conventions for generalisation and specialisation. Finally, table 3 shows the graphical representations for constraints.

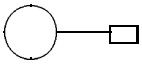
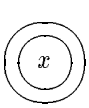
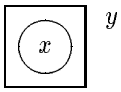
	*object type*
	*label type x*
	*role*
	*predicator*
	*y power type of x*
	*y sequence type of x*
	*grammar box*

Table 1: Legend for object type symbols

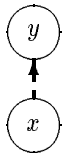
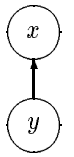
	*y is generalisation of x*
	*y is specialisation of x*

Table 2: Legend for ISA relationships

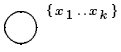
	uniqueness constraint over a single fact type
	*uniqueness constraint over several fact types*
	*total role or cover constraint*
	*occurrence frequency constraint or cardinality constraint*
	*exclusion constraint*
	*membership constraint*
	*subset constraint*
	*equality constraint*
	*enumeration constraint*

Table 3: Legend for constraints

References

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

[2] R. Bird and P. Wadler. *Introduction to functional programming.* Prentice Hall, New York, 1988.

[3] 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.

[4] P.D. Bruza and Th.P. van der Weide. Two Level Hypermedia - An Improved Architecture for Hypertext. In A.M.Tjoa and R.Wagner, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 90)*, pages 76–83. Springer-Verlag, 1990.

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

[6] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications ACM*, 13(6):377 – 387, 1970.

[7] J.N. Crossley, C.J. Ash, C.J. Brickhill, J.C. Stillwell, and N.H. Williams. *What is mathematical logic?* Oxford University Press, 1972.

[8] E.D. Falkenberg. Deterministic Entity-Relationship Modelling. Technical Report 88-13, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, 1988.

[9] E.D. Falkenberg and Th.P. van der Weide. Formal Description of the TOP Model. Technical Report 88-01, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, 1988.

[10] A.L. Furtado and E.J. Neuhold. *Formal Techniques for Data Base Design.* Springer-Verlag, 1985.

[11] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base.* Publ. nr. ISO/TC97/SC5-N695, 1982.

[12] M. Harrison. *Introduction to Formal Language Theory.* Addison Wesley, Reading, MA, 1978.

[13] W.S. Hatcher. *The Logical Foundations of Mathematics.* Pergamon Press, 1982.

[14] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

[15] 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.

[16] R. Hull and R. King. Semantic Database Modelling: Survey, Applications and Research Issues. *Computing Surveys*, 19(3):201–260, September 1987.

[17] R. Hunter, P. Kaijses, and F. Nielsen. ODA: a document architecture for open systems. *Computer Communications*, 12(2):69–79, April 1989.

[18] ISO. *Information Processing - Text and Office Systems - Standard General MarkUp Language (SGML).* ISO8879, 1986.

[19] ISO. *Information Processing - Text and Office Systems - Office Document Architecture (ODA) and interchange format, Part 1,2,4,5,6,7,8.* ISO8613, 1989.

[20] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, 1975.

[21] G.M. Kuper and M. Vardi. On the Expressive Power of the Logical Data Model. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 180–187, Austin, Texas, 1985. ACM.

[22] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1988.

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

[24] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

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

[26] H.J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[27] G. Scheschonk. *Eine auf Petri-Netzen basierende Konstruktions, Analyse und (Teil)Verifikationsmethode zur Modellierungsunterstützung bei der Entwicklung von Informationssystemen*. PhD thesis, Berlin University of Technology, Berlin, Germany, 1984.

[28] J.M. Smith. Standard Generalized Markup Language and related standards. *Computer Communications*, 12(2):80–84, April 1989.

[29] P.L. van der Spiegel, J.T.W. Driessen, P.D. Bruza, and Th.P. van der Weide. A Transaction Model for Hypertext. In D. Karagiannis, editor, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 91)*, pages 281–286. Springer-Verlag, 1991.

[30] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Publishing Company, 1987.

[31] O.M.F. de Troyer. On Rule-Based Generation of Conceptual Database Updates. In *Proceedings of the IFIP TC 2 Working Conference on Knowledge and Data*, 1986.

[32] 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, 1984.

[33] O.M.F. de Troyer, R. Meersman, and P. Verlinden. RIDL* on the CRIS Case: A Workbench for NIAM. In T.W. Olle, A.A. Verrijn-Stuart, and L. Bhabuta, editors, *Computerized Assistance during the Information Systems Life Cycle*, pages 375 – 459, Amsterdam, The Netherlands, 1988. North-Holland/IFIP.

[34] T.F. Verhoef, A.H.M. ter Hofstede, and G.M. Wijers. Structuring modelling knowledge for CASE shells. In R. Andersen, J.A. Bubenko, and A. Sølvberg, editors, *Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 502–524, Trondheim, Norway, May 1991.

[35] Y. Wand and R. Weber. An Ontological Analysis of some fundamental Information Systems Concepts. In *Proceedings of the Ninth International Conference on Information Systems*, pages 213–226, November 1988.

[36] Th.P. van der Weide, A.H.M. ter Hofstede, and P. van Bommel. Uniquest: Determining the Semantics of Complex Uniqueness Constraints. *The Computer Journal*, 35(2):148–156, April 1992.

[37] R.J. Welke. The CASE Repository: More than another database application. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[38] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer Academic Publishers, 1990.