

# TOWARDS A GENERAL THEORY FOR THE EVOLUTION OF APPLICATION DOMAINS

H.A. PROPER

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. Towards a General Theory for the Evolution of Application Models. In M.E. Orłowska and M. Papazoglou, editors, *Proceedings of the Fourth Australian Database Conference*, Advances in Database Research, pages 346–362. World Scientific, Brisbane, Australia, February 1993.

## ABSTRACT

In this article we focus on evolving information systems. First a delimitation of the concept of evolution is discussed. The main result is a first attempt to a general theory for such evolution. In this theory, the underlying data model is a parameter, making the theory applicable for a wide range of modelling techniques.

## 1. Introduction

As has been argued in [18] and [7], there is a growing demand for information systems, not only allowing for changes of their information base, but also for modifications in their underlying structure (conceptual schema and specification of dynamic aspects). In case of snapshot databases structure modifications lead to costly data conversions and reprogramming.

The intention of an evolving information system ([6]) is to be able to handle updates of all components of the so-called *application model*, containing the information structure, the constraints on this structure, the population conforming to this structure and the possible operations. The theory of such systems should, however, be independent of whatever modelling technique is used to describe the application model. In this paper, we discuss a general theory for the evolution of application models. The central part of this theory will make weak assumptions on the underlying modelling technique, making it therefore applicable for data modelling techniques such as ER ([3]), NIAM ([16]) and PSM ([10]),

and action modelling techniques such as Task Structures ([21]), DFD ([2]) and ExSpect ([9]).

Version modelling in engineering databases can be seen as a restricted form of evolving information systems ([14], [13]). An important requirement for evolving information systems, not covered by version modelling systems, is that changes to the structure can be made on-line. In version modelling, a structural change requires the replacement of the old system by a new system. Other research regarding evolving information systems can be found in [15], in which an algebra is presented that allows relational tables to evolve by changing their arity.

The structure of the paper is as follows. In section 2 we describe the approach that has been taken to the concept of evolution. We will not focus on a particular modelling technique. In section 3 we describe what the minimal requirements of a modelling technique. By considering all elements that may be subject to evolution, we then introduce in section 4 the application model universe, and describe what constitutes a application model version. Finally, in section 5 the evolution of an application model is formally introduced, and some properties of wellformedness are presented.

## 2. An Approach to Evolving Information Systems

In this section we discuss our approach to evolving information systems. We start with the identification of that part of an information system that may be subject to evolution. From this identification, the difference between a traditional information system, and its evolving counterpart, will become clear. This is followed by a discussion on how the evolution of an information system is modelled.

### 2.1. A hierarchy of models

According to [8], a conceptual (i.e. complete and minimal) specification of a universe of discourse consists of the following components:

1. an *information structure*, a set of *constraints* and a *population* conforming to these requirements.
2. a set of *action specifications* describing the transitions that can be performed by the system.

The set of action specifications is referred to as the *action model*. The *world model* encompasses the combination of information structure, constraints and the population. A conceptual specification of a universe of discourse, containing both the action and world model, is called an *application model* ([6], [7]). The resulting hierarchy of models is depicted in figure 1.



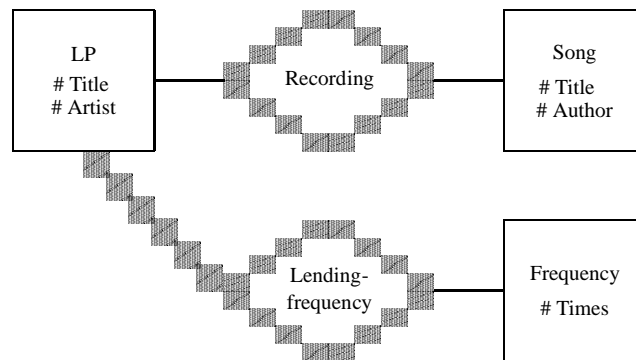


Figure 2: The Data Model of an LP rental store

nects object types to relation types, and the keyword  $\hat{A}$  just the other way around.

After the introduction of the compact disc, and its conquest of a sizeable piece of the market, the rental store has transformed into an  $\hat{A}$ LP and CD rental store $\hat{A}$ . This leads to the introduction of object type Medium as a generic term for LP $\hat{A}$  and CD $\hat{A}$ . The relation type Medium-type effectuates the subtyping of Medium into LP and CD. In the new situation, the registration of songs on LP $\hat{A}$  is extended to cover CD $\hat{A}$  as well. The frequency of lending, however, is not kept for CD $\hat{A}$ , as CD $\hat{A}$  are hardly subject to any wear and tear. As a consequence, the application model has evolved to ægure 3.

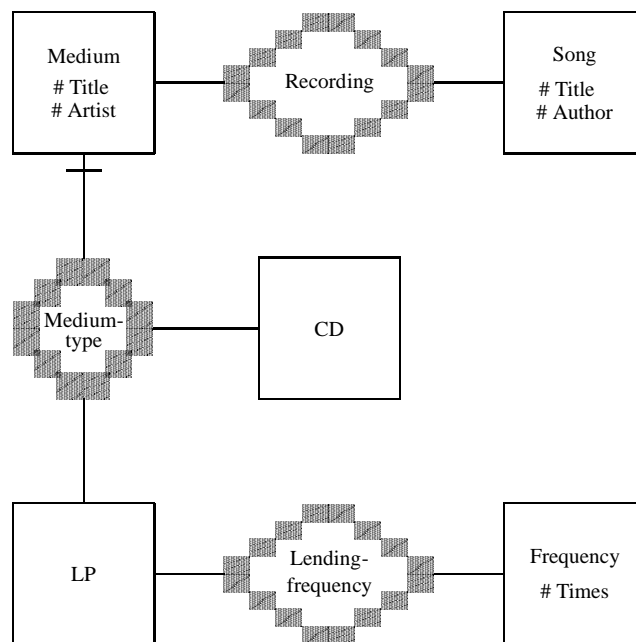
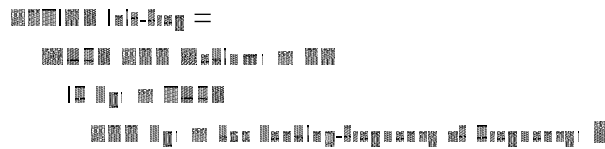


Figure 3: The Data Model of a LP and CD rental store

The action specification  $\hat{A}$  evolves accordingly, now stating that whenever a medium is added to the assortment of the rental store, it $\hat{A}$  lending frequency is set to  $\hat{A}$

provided the medium is an LP:



### 2.3. The approach

The two ER schemata, and the two action specifications, as discussed above, correspond to two distinct snapshots of an evolving universe of discourse. Several approaches can be taken to the modelling of this evolution. A first approach is to model the history of application model elements by adding birth-death relations to all object types in the information structure ([19]). This approach, however, is very limited, as it only enables the modelling of evolution of the population of an information system. For example, the evolution of the Recording relation type can not be modelled in this approach. Evolution of other application model elements than population, can then be described by a meta model approach.

This paper takes another approach, and treats evolution of an application model as a separate concept. There still are two alternatives to deal with the history of application models. The first one is to maintain a version history of application models in their entirety, leading to a sequence of snapshots of application models. The second one is to keep a version history per element, so keeping track of the evolution of object types, instances, methods, etc.

An advantage of this second alternative is that it enables one to state rules about, and query the evolution of distinct objects. This alternative also allows for the formulation of rules concerning the well formedness of the evolution of application model elements ([17]). The first alternative clearly does not offer this opportunity, as it does not provide relations between successive versions of the application model.

Furthermore, the snapshot view from the first alternative can be derived by constituting the application model version of any point of time from the current versions of its components. In the theory of evolving application models we will therefore adapt the second alternative.

An *application model element history* describes the history of an application model element, and is seen as a partial function assigning to points of time the actual occurrence (version) of the element. For example, when CDs are added to the assortment of the rental store, the version of the application model element *Recording* changes from a relation type that registrates songs on LPs to a relation type that registrates songs on Media. An *application model history* is a set of application model element histories. The current version of an application model then is constituted by the current versions of all application model elements.

### 3. A World Model for Application Models

In this section we take a closer look at application models, and define what constitutes an application model version. An application model version is formulated conforming to some modelling technique. The only assumption on this modelling technique is, that it delimits an *application model universe*, i.e., the space for the evolution of application models.

#### 3.1. The Information Structure Universe

The kernel of the application model universe is formed by the *information structure universe*  $\mathcal{ISU} = \langle \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$ , fixing the evolution space for information structures. Further refinements of the information structure universe depend on the chosen data modelling technique (such as NIAM, ER, PSM and Object Oriented data models), and are beyond the scope of the theory in this paper. For our purposes, an information structure universe is assumed to provide (at least) the above components, which are available in all conventional high level data modelling techniques.

##### 3.1.1. Object Types

The central part of an information structure is formed by its object types (referred to as object classes in object oriented approaches). Two major classes of object types are distinguished. Object types whose instances can be represented directly (denoted) on a medium (strings, natural numbers, etc) form the class of label types  $\mathcal{L}$ . The other object types, for instance entity types or fact (relation) types, form the class  $\mathcal{O}$ . The example of figure 2 contains nine object types: three entity types  $\text{Person}$ ,  $\text{Company}$  and  $\text{Product}$ , two relation types  $\text{worksFor}$  and  $\text{owns}$ , and four label types  $\text{Male}$ ,  $\text{Female}$ ,  $\text{Manager}$  and  $\text{Employee}$ .

##### 3.1.2. Type Relatedness

For object types, the (reflexive and symmetrical) relation  $\sim$  expresses *type relatedness* between object types (see [12]). Object types  $\alpha$  and  $\beta$  are termed type related ( $\alpha \sim \beta$ ) iff populations of object types  $\alpha$  and  $\beta$  may have values in common. Type relatedness corresponds to mode equivalence in programming languages ([22]). The relation of type relatedness can be recognised in conventional modelling techniques like ER, NIAM, or PSM, as well as object oriented data modelling techniques. Typically, subtyping and generalisation lead to type related object types. For the data model depicted in figure 2, the type relatedness relation is the identity relation:  $\alpha \sim \alpha$  for all object types  $\alpha$ .

##### 3.1.3. The Identification Hierarchy

In data modelling, a crucial role is played by the notion of object identification: each object type of an information structure should be identifiable. In a subtype hierarchy however, a subtype inherits its identification from its super type, whereas in a generalisation hierarchy

the identification of a generalised object type is inherited from its specialisers. For the data model depicted in figure 3 this means that instances of  $U_1$  and  $U_2$  are identified in the same way as instances of  $U_1 \sqcup U_2$ . An object type from which the identification is inherited, is termed a *parent* of that object type. The inheritance hierarchy (identification hierarchy) is provided by the relation  $\text{parent}$ , meaning that  $U_1$  is the parent of  $U_2$ . For figure 3 this leads to:  $U_1 \sqcup U_2$   $\text{parent}$   $U_1$  and  $U_1 \sqcup U_2$   $\text{parent}$   $U_2$ .

Object types in an information structure that have no parent are called *roots* as they form the roots of the inheritance hierarchy (a directed acyclic graph). The roots of an object type  $U$  are found by:  $\text{roots}(U) = \{U' \mid U' \text{ parent } U \wedge \neg \exists U'' (U' \text{ parent } U'')\}$ .

For every data model from conventional data modelling techniques, a parent ( $\text{parent}$ ) and root ( $\text{roots}$ ) relation can be derived. If no specialisations or generalisations are present in that data model, the parent relation will be empty. As a result, the root relation will be the identity relation. For instance the root relation for figure 2 is:  $\text{roots}(U) = U$  for every object type  $U$ .

### 3.2. Properties of Information Structure Universes

Two interesting properties on the root and parent relation are:

**Lemma 3.1** (*common roots*)

$$\text{roots}(U) \cap \text{roots}(V) = \text{roots}(U \sqcup V)$$

The intuition behind this property is that type relatedness must be based on the type relatedness of some root. Furthermore, type relatedness of roots implies type relatedness of object types:

**Theorem 3.1** (*type relatedness propagation*)

$$U \text{ related } V \wedge U' \in \text{roots}(U) \wedge V' \in \text{roots}(V) \implies U' \text{ related } V'$$

These properties have been proven in [17].

## 4. Secondary elements of Application Models

The hierarchy of models (see figure 1) describes how an application model is constructed from other (sub)models. However, this hierarchy disregards relations that must hold between these submodels, for example, the relation between a population and the information structure. These relations are crucial elements of an application model, as they form the fabric of the application model.

### 4.1. Application Model Universe

An application model version is bound to the *application model universe*  $\mathcal{U}_M$ , which is captured in the tuple:  $(\text{models}, \text{populations}, \text{information structures}, \text{relations})$ . The information structure universe  $\mathcal{U}_I$  was introduced in the previous section.

#### 4.1.1. Constraints

Most data modelling techniques have a language in which constraints can be expressed, resulting in the set of all possible constraint definitions. Constraints are treated as application model elements, that assign to (some) object types a particular constraint definition. A constraint is said to be *owned* by an object type, if the object type has assigned a constraint definition by the constraint. Constraints are inherited via the identification hierarchy. However, as in object oriented data modelling techniques, overriding of constraint definition in identification hierarchies is possible (see for instance [5]).

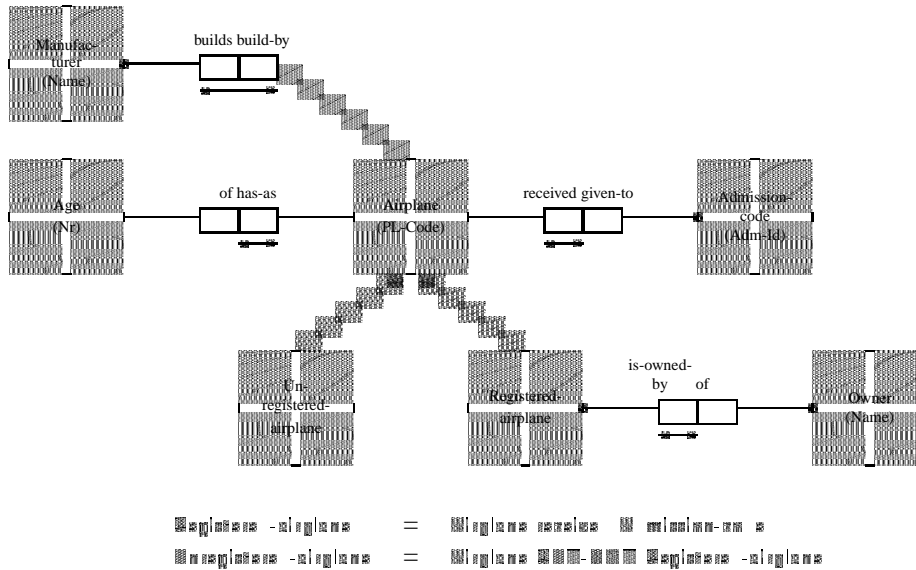
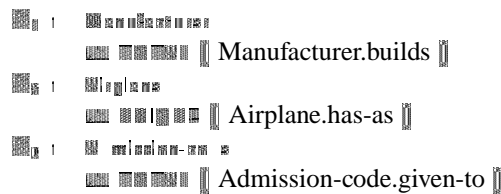


Figure 4: Constraint assignment

As an illustration of the assignment of constraints to object types, consider figure 4. The depicted data model is conforming to NIAM, while the subtype defining rules have been formulated in LISA-D. The modelled universe of discourse is concerned with the administration of airplanes. As airplanes should be replaced in time, the age of an airplane is an important attribute. Furthermore, an airplane may be registered by an aviation association, in which case it has associated an admission code. The owner of registered planes is maintained by the administration.

The graphical constraints contained in this data model, are assigned to object types in the following way:





All these constraints are owned by a single object type. A more interesting case with respect to inheritance results by adding the following constraint:

The object type assignment for this constraint is:

The constraint  $\text{C}_{\text{OCC}}^{\text{S}}$  is considered to be owned by object types  $\text{O}_{\text{P}}^{\text{S}}$ ,  $\text{O}_{\text{R}}^{\text{S}}$ ,  $\text{O}_{\text{D}}^{\text{S}}$ ,  $\text{O}_{\text{A}}^{\text{S}}$ ,  $\text{O}_{\text{M}}^{\text{S}}$  and  $\text{O}_{\text{U}}^{\text{S}}$ .

#### 4.1.2. Methods

### 4.1.3. Domains

which can evolve by itself. Therefore, (non empty) sets of object types are associated to instances, specifying the object types having the instance as an instantiation. This association is contained in the relation  $\text{inst\_obj\_types} : \text{Inst} \rightarrow \mathcal{P}(\text{ObjType})$ , where  $\text{Inst}$  is the set of all possible instantiations of object types. An example of such an association is:  $\{i_1, i_2, i_3\} \rightarrow \{o_1, o_2\}$ , meaning  $i_1$  is an (abstract) instance of entity types  $o_1$  and  $o_2$ . The population of an object type, traditionally provided as a function  $\text{pop} : \text{ObjType} \rightarrow \text{Inst}$ , can be derived from the association between instances and object types:  $\text{pop}(o) = \{i \in \text{Inst} \mid i \rightarrow o\}$ .

#### 4.1.5. Evolution dependency

Every method and constraint specification refers to (uses) a set of object types and denotable instances. This relation is provided in the application model universe by means of the dependency relation ( $\text{depends}$ ). The relation  $\text{depends}$  is modelling technique dependent, but is not subject to evolution.

The intuition behind this relation is as follows:  $m \text{ depends } o$  means that if  $o$  is not present in an application model version, then  $m$  can not be used in that version. A consequence is, that in case of evolution of application models, when  $o$  evolves to  $o'$ , then  $m$  must be adapted appropriately.

As an example, consider the second action specification from the rental store example. This action specification depends on object types  $o_1$ ,  $o_2$  and  $o_3$ . It, furthermore, depends on the domain assignment:  $o_1 \rightarrow o_2$ . If one of the object types, or the domain assignment, is terminated or changed, the action specification has to be terminated or changed accordingly.

## 4.2. Application Model Versions

An application model version ( $\text{AMV}$ ) over information structure universe  $\text{ISU}$  is determined by the following components:  $\text{ObjType}$ ,  $\text{Inst}$ ,  $\text{Method}$ ,  $\text{Constraint}$ ,  $\text{Domain}$ . With a version of the application model, we can associate the following version of the information structure:  $\text{ISU} \rightarrow \text{AMV}$  where:

$$\begin{aligned} \hat{\text{AMV}} &= \text{ObjType} \cup \text{Inst} \cup \text{Method} \cup \text{Constraint} \cup \text{Domain}, \\ \hat{\text{AMV}} &= \text{ObjType} \cup \text{Inst} \cup \text{Method} \cup \text{Constraint} \cup \text{Domain}, \\ \hat{\text{AMV}} &= \text{ObjType} \cup \text{Inst} \cup \text{Method} \cup \text{Constraint} \cup \text{Domain}, \text{ and} \\ \hat{\text{AMV}} &= \text{ObjType} \cup \text{Inst} \cup \text{Method} \cup \text{Constraint} \cup \text{Domain}. \end{aligned}$$

As an overview of the components of an application model version, a meta model is provided in figure 5. This (meta) data model is conforming to the PSM data modelling technique, an extension of the NIAM modelling technique. The object types  $o_1$  and  $o_2$  in figure 5 are power types, the data modelling pendant of power sets in set theory.

Every application model version must adhere to some rules of wellformedness. Some of these rules are modelling technique dependent. Nonetheless, some general rules about application model versions can be stated. In [17] a formalisation of these rules is provided.

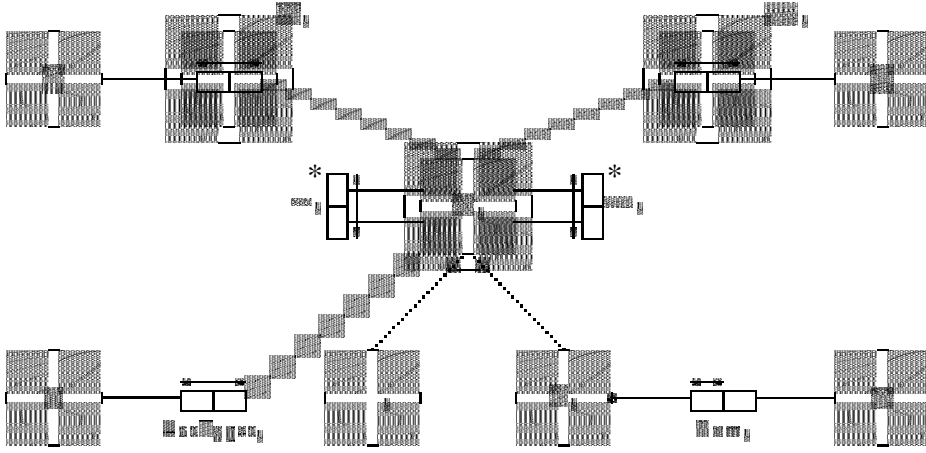


Figure 5: A meta model for information structures

An object type  $\mathbb{O}$  is called *alive* at a certain point of time  $\mathbb{T}$ , if it is part of the application model version at that point of time ( $\mathbb{O} \in \mathbb{M}_{\mathbb{T}}$ ). Furthermore, an object type  $\mathbb{O}$  is termed *active* at a certain point of time, if there is an instance with  $\mathbb{O}$  as one of its associated types ( $\exists \mathbb{I} \in \mathbb{I}_{\mathbb{T}} : \mathbb{O} \in \mathbb{A}(\mathbb{I})$ ).

A first rule of wellformedness states that every active object type must be alive as well. This rule can be popularised as:  $\hat{\mathbb{A}}$  am active, therefore I am alive $\hat{\mathbb{A}}$ . Furthermore, the set of object types associated to an instance must be mutually type related, as their populations share at least this value.

From the very nature of the root relation it immediately follows that instances are included upwards, towards the roots. As a result, every instance of an object type is also an instance of its parents (if any). This rule has a structural pendant as well. Every living object type should be accompanied by one of its parents (if any). If all object types in a data model have a non empty instantiation, i.e. every object type is active, the latter (structural) rule immediately follows from the former (population) rule. However, when some non-root (alive) object type is not active, implying that its parent(s) do not have to be active, it does not follow that at least one of its parents must be alive. Therefore, a rule demanding this explicitly is required.

Constraints and methods are defined as mappings from object types to constraint and method definitions respectively. Thus, object types that own a constraint or a method must be alive. Furthermore, object types that own the same constraint or method, have to be type related. Finally, due to inheritance, if a constraint is defined for a parent object type, it must be defined for its children as well.

For populations some interesting properties have been proven [17]. An first example states: every instance of an object type is also an instance of one of its roots.

**Lemma 4.1**  $\forall \mathbb{O} \in \mathbb{O}_{\mathbb{T}} : \exists \mathbb{R} \in \mathbb{R}_{\mathbb{T}} : \mathbb{O} \in \mathbb{R}$

From the nature of type relatedness it follows that the populations of roots which are not type related are mutually exclusive. This rule can be generalised to all object types, leading to:

**Lemma 4.2**  $\forall \alpha, \beta \in \mathcal{O} \quad \alpha \neq \beta \implies \alpha \not\sim \beta$

For the proofs of these properties, refer to [17].

## 5. Evolution of Application Models

The basis of the theory for evolving application models is formed by the concept of *evolution continuum*, capturing both dimensions of evolution of application models, being the universe of application model and time. The evolution continuum serves as a state space for evolutionary navigation:  $\mathcal{EC} = \langle \mathcal{U}, \mathcal{T} \rangle$

Time, essential to evolution, is incorporated into the evolution continuum through the algebraic structure  $\mathcal{EC} = \langle \mathcal{U}, \mathcal{T} \rangle$ , where  $\mathcal{T}$  is a (discrete) time axis, and  $\mathcal{U}$  a set of functions over  $\mathcal{T}$ . In this paper, the time axis is regarded as an abstract data type. Several ways of defining a time axis exist, see e.g. [4], [20] or [1]. The time axis is the axis along which the application model evolves (through the application model universe).

An application model element history is a partial function, that assigns to points of time the actual version of the element, and thus is a partial mapping  $\mathcal{H} \subseteq \mathcal{T} \times \mathcal{U}$ , where  $\mathcal{U}$  is the set of all evolvable elements of an application model:  $\mathcal{U} = \{ \alpha \in \mathcal{O} \mid \alpha \text{ is an object type, constraint, method, or population} \}$ . An application model history over an information structure universe  $\mathcal{U}$  then is defined by:

### Definition 5.1

$\mathcal{H} \subseteq \mathcal{T} \times \mathcal{U} \implies \mathcal{H} \text{ is a partial mapping from } \mathcal{T} \text{ to } \mathcal{U}$

As a first rule of wellformedness, the evolution of application model elements is bound to classes. For example, an object type may not evolve into a method, and a constraint may not evolve into an instance. A consequence of this rule is that an application model history can be partitioned into the history of its object types, its constraints, its methods, its populations, and its concretisations (of label types):

### Definition 5.2

*object type histories:*

$\mathcal{H}_{\alpha} \subseteq \mathcal{H} \implies \mathcal{H}_{\alpha} \text{ is a partial mapping from } \mathcal{T} \text{ to } \mathcal{O}$

*constraint histories:*

$\mathcal{H}_{\beta} \subseteq \mathcal{H} \implies \mathcal{H}_{\beta} \text{ is a partial mapping from } \mathcal{T} \text{ to } \mathcal{C}$

*method histories:*

$\mathcal{H}_{\gamma} \subseteq \mathcal{H} \implies \mathcal{H}_{\gamma} \text{ is a partial mapping from } \mathcal{T} \text{ to } \mathcal{M}$

*population histories:*

$\mathcal{H}_{\delta} \subseteq \mathcal{H} \implies \mathcal{H}_{\delta} \text{ is a partial mapping from } \mathcal{T} \text{ to } \mathcal{P}$

[illegible]

### Definition 5.3

Figure 1 is a flowchart illustrating the experimental design. It begins with a box labeled 'Selection of a sample of 1000 subjects'. This leads to a box labeled 'Random assignment to two groups: Control and Experimental'. From here, the path splits into two: 'Control' and 'Experimental'. Each path leads to a box labeled 'Assessment of Outcome'. Finally, both paths converge into a box labeled 'Comparison of Results', which leads to a box labeled 'Significance'.

[illegible][illegible]

Entity type:  $\hat{A}$   
 Fact type:  $\hat{A}$   
 Fact type:  $\hat{A}$   
 Entity type:  $\hat{A}$

[illegible]

where  $\mathbb{I}_1, \dots, \mathbb{I}_n$  are instances such that:

$\mathbb{I}_1$	$\hat{=}$	$\hat{\text{Brothers in Arms}}$
$\mathbb{I}_2$	$\hat{=}$	$\hat{\text{Money for nothing}} \hat{\text{Brothers in Arms}}$
$\mathbb{I}_3$	$\hat{=}$	$\hat{\text{Brothers in Arms}} \hat{\text{Brothers in Arms}}$
$\mathbb{I}_4$	$\hat{=}$	$\hat{\text{Brothers in Arms}} \hat{\text{Brothers in Arms}}$

The interpretation of this table leads to:

$\mathbb{I}_1, \mathbb{I}_2$	$\hat{=}$	$\mathbb{I}_1, \mathbb{I}_2$	means:	$\hat{\text{Brothers in Arms}}$	is both a $\mathbb{I}_1$ and a $\mathbb{I}_2$ at $\mathbb{I}_1$ ,
$\mathbb{I}_2, \mathbb{I}_3$	$\hat{=}$	$\mathbb{I}_2, \mathbb{I}_3$	means:	$\mathbb{I}_2$	$\hat{\text{Brothers in Arms}}$ $\mathbb{I}_3$ $\hat{\text{Brothers in Arms}}$
$\mathbb{I}_3, \mathbb{I}_4$	$\hat{=}$	$\mathbb{I}_3, \mathbb{I}_4$	means:	$\mathbb{I}_3$	$\hat{\text{Brothers in Arms}}$ $\mathbb{I}_4$ $\hat{\text{Brothers in Arms}}$

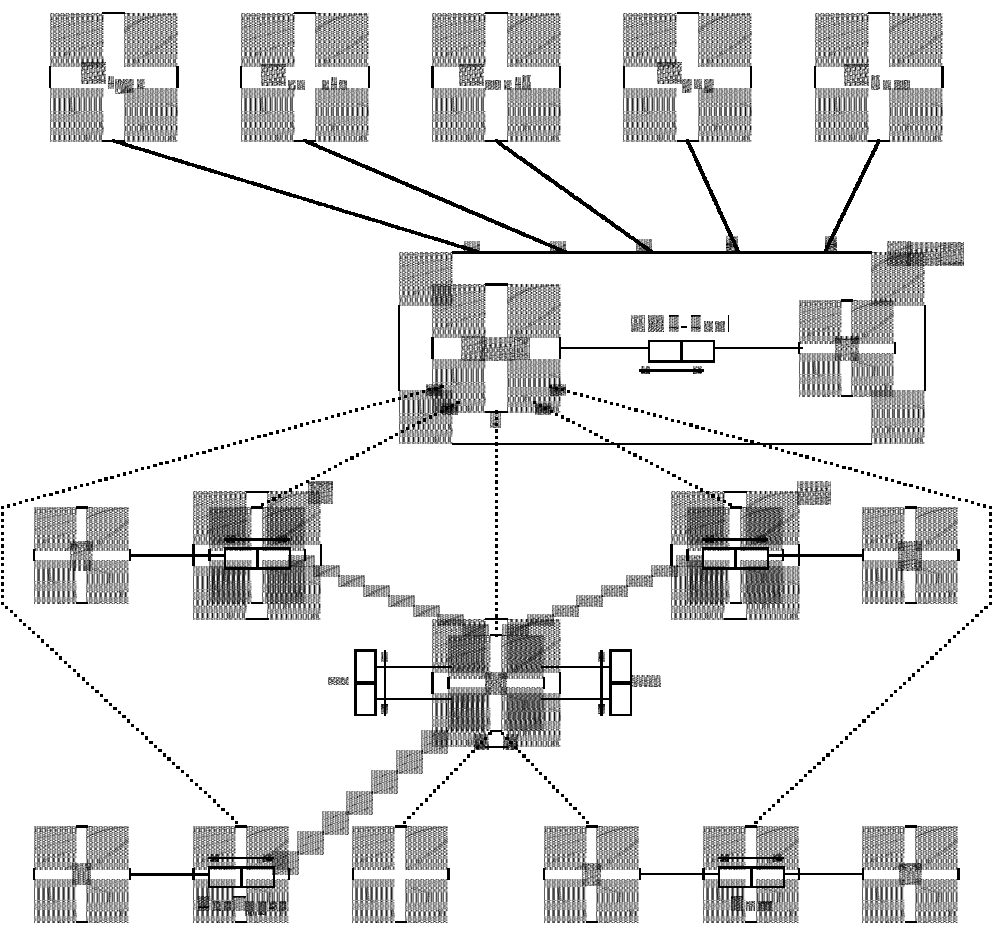


Figure 6: A meta model for the evolution system

As an outline of the hitherto deæned concepts, a (meta) data model, relating all deæned concepts, is provided in ægure 6. The data model depicted there is conform the PSM modelling technique, and uses the notion of schema objectiæcations (object type  $\mathbb{I}_1$ ), and powertyping (object type  $\mathbb{I}_2$  ). The population of an objectiæd schema at hand is

to be looked upon as one single abstract object instance of the object type corresponding to the objectified schema. Powertyping is, as stated before, the data modelling pendant of powersets from set theory.

## 6. Conclusions and Further research

In this paper we presented a first attempt to a general theory for the evolution of application models, supporting evolving information systems. As a next step a set of well formedness rules on evolution steps has to be defined, since not all evolution steps of application models will correspond to sensible/valid evolution steps in an actual universe of discourse.

Furthermore, in order to validate the theory, it must be applied to some modelling techniques. In the near future we will therefore apply this theory to PSM, a data modelling technique serving as a common base for data modelling techniques such as NIAM, ER, and IFO, and to Task Structures, a powerful action modelling technique. A query and manipulation language must be defined supporting the evolution of information systems. Finally, the consequences of evolution for the internal representation of information structures should be studied.

## Acknowledgements

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

## References

1. J.F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 1984(23):123-154, 1984.
2. P.D. Bruza and Th.P. van der Weide. The Semantics of Data Flow Diagrams. In N. Prakash, editor, *Proceedings of the International Conference on Management of Data*, Hyderabad, India, 1989.
3. P.P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9-36, March 1976.
4. J. Clifford and A. Rao. A simple, general structure for Temporal Domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in information Systems*, pages 17-28. North-Holland/IFIP, Amsterdam, The Netherlands, 1987.
5. O.M.F. De Troyer. The OO-Binary Relationship Model: A Truly Object Oriented Conceptual Model. In R. Andersen, J.A. Bubenko, and A. Søberg, editors, *Proceedings of the Third International Conference CAISE'91 on Advanced Information*

- Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 561–578, Trondheim, Norway, May 1991. Springer-Verlag.
6. 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, 1992.
  7. 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, September 1992. Springer-Verlag.
  8. J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036, 1982.
  9. K.M. van Hee, L.J. Somers, and M. Voorhoeve. Executable Specifications for Distributed Information Systems. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*, pages 139–156. North-Holland/IFIP, Amsterdam, The Netherlands, 1989.
  10. 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, May 1992. Springer-Verlag.
  11. 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.
  12. 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.
  13. 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.
  14. R.H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.
  15. E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.
  16. G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989.
  17. 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.
  18. J.F. Roddick. Dynamically changing schemas within database models. *The Australian Computer Journal*, 23(3):105–109, August 1991.
  19. R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of*



*the ACM SIGMOD International Conference on the Management of Data*, pages 236–246, Austin, Texas, 1985.

20. G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with the Granularity of Time in Temporal Databases. In R. Andersen, J.A. Bubenko, and A. Søberg, editors, *Proceedings of the Third International Conference CAISE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 124–140, Trondheim, Norway, May 1991. Springer-Verlag.
21. G.M. Wijers, A.H.M. ter Hofstede, and N.E. van Oosterom. Representation of Information Modelling Knowledge. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 167–223. IOS Press, 1992.
22. A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.T. Meertens, and R.G. Fisker. *Revised Report on the Algorithmic Language ALGOL 68*. Springer-Verlag, Berlin, Germany, 1976.
23. E. Yourdon. *Modern Structured Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.