A Meta Model for Update in Evolving Information Systems*

J.L.H. Oei, H.A. Proper, E.D. Falkenberg
E.Proper@acm.org

Department of Information Systems,
University of Nijmegen,
Toernooiveld, NL-6525 ED Nijmegen,
The Netherlands, +31-80-652645

Keywords: Evolution of Information Systems, Schema Evolution,
Temporal Information Systems, Historical Information Systems.
PUBLISHED AS:

E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. A Metamodel for Update in Information Systems. Technical Report 92-05, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.

Abstract

An evolving information system supports the information needs of an evolving organisation. These systems are able to adapt themselves instantaneously to the changes of the supported organisation, such that there is no need to interrupt the activities of the organisation. Furthermore, evolving information systems support changes of all time- and application-dependent aspects, such as the database and the schema of the application.

The main focus of this paper is on update in evolving information systems. A framework for the processing of updates in evolving information systems is presented. In this framework, update is regarded as recording, correction or forgetting, and state transitions are considered on three different levels of abstraction, viz. the event level, the recording level and the correction level. A formal specification of this framework is provided as well. Furthermore, the notion of evolution dependency is discussed, taking the dependencies of changes within the universe of discourse into consideration.

1 Introduction

As has been argued in [FOP92a], [FOP92b] and [Rod91], there is a substantial demand for information systems which are able to evolve to the same extent and at the same pace at which the supported organisation system evolves due to changes in the *universe of discourse*. A universe of discourse ([ISO87]) is that part of the organisation which is reflected in the information system. Whenever the term information system is used in this paper, we refer to information systems in a narrower sense, i.e. a computerised information system ([Ver89], [FOP92b]).

Most traditional information systems hardly support any aspect of evolution. First of all, most traditional information systems only allow for update of the information base, i.e. the set of facts which obeys a fixed (conceptual) schema with a fixed set of constraints. In other words, update of the conceptual schema constraints, and specifications of dynamic aspects - the activity and behaviour specifications - have not yet been supported by these traditional information systems. The evolving information systems at which we aim, however, do support update of all these specifications. Therefore, a very important requirement

*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).

for these evolving information systems is that no update of any sort should result in the interruption of activities of the evolving organisation ([FOP92a], [FOP92b]), i.e. changes should be performed on line.

Directly related research regarding evolving information systems can be found in [MS90] and [Ari91]. Indirectly related research can be found in the area of version modelling in engineering databases: [BCG$^+$87], [Kat90], [JMSV92]. In [MS90] a relational algebra is presented in which relational tables are allowed to evolve, e.g. a change of their arity. In this paper, we take a more conceptual approach to evolution of information systems. Furthermore, we do not restrict evolution to the data model (and its population). We consider the evolution of the application model (formal model of the universe of discourse) as a whole.

Version modelling in engineering databases offers a fast body of knowledge concerning evolution of several types of engineering applications. The requirements for the evolving information systems regarded in this article (see also [FOP92b]) are related to the requirements for version modelling as presented in [Kat90]. As stated before, an important requirement for evolving information systems, as considered in this paper, is that changes to the structure can be made on-line. In traditional approaches to the evolution of information systems, and software evolution in general, a structural change still requires the replacement of the old system by a new system. In these systems, the support of evolution is restricted to version managament. This latter notion of evolution is the approach to evolving information as taken in e.g. [JMSV92].

The main difference between a traditional information system and an evolving information system, is best explained in more detail by means of the general architecture for an evolving information system as depicted in figure 1 (for a more elaborate discussion on the architecture of an evolving information system see [FOP92b]). The *information processor* (the computer) performs the *information processing* activities. This processing involves the acceptance of input messages (*requests* from the users) which, among other things, may reflect changes of states in the *universe of discourse*. This latter class of messages results in appropriate changes of the *application model*, and may be the cause of other information processing activities being performed by the information processor. The *application model* is a description of the state of the universe of discourse as it is known to the information system. As a result of the input messages, the information processor may generate output messages (*responses*) which are received by the universe of discourse.

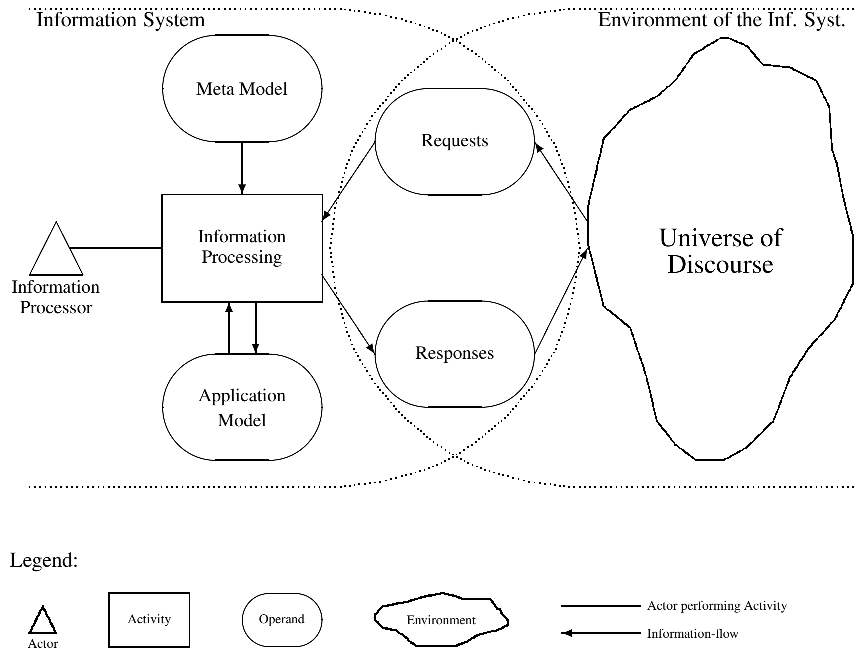

Figure 1: An Evolving Information System and its Environment

During the information processing activities, the information processor uses a *meta model* to maintain the consistency of the application model. The *meta model* is time- and application-independent, it contains (and is restricted to) all rules about the languages used to model the application model and to formulate user *requests*. The *application model*, on the other hand, contains the application or time-dependent elements, i.e. the description of the universe of discourse.

Traditionally, two constituent parts can be distinguished in an application model. The one part is the *conceptual schema*, determining the data base structure. Such a *conceptual schema* is a schema conform a modelling technique like NIAM ([NH89]) or ER ([Che76]), or for more complex applications a schema conform IFO ([AH87]), or PSM ([HW93], [HPW92]). The other part is the *information base*, or population, conform the conceptual schema.

Besides the above two traditional parts, an application model may also contain descriptions of activities and behaviour (see for instance [HSV89], [HV91], [WHO92]). The intention of an evolving information system is, to be able to change all parts of the application model, and not just the information base as is the case in most traditional information systems. In [OHFB92] the evolvability of the composing parts of an information system is taken even further. There the evolution of *meta models* is considered.

Another limitation of many traditional systems is that they are *snapshot* systems, i.e. they reflect information valid at a certain point of time, but lack the ability to preserve the history of information. An evolving information system, on the contrary, must be *conservative* or *temporal* ([SA85], [SA86], [RBL87], [FOP92a], [FOP92b]) in the sense that it should not forget anything ever fed to the system, unless explicitly asked for, thus allowing for the formulation of queries about the history of the information base (populations). In such systems, the distinction between *event time* - the time at which an event occurs in the universe of discourse - and *recording time* - the time at which the event is recorded in the information systemand recording time - is of great importance ([Bub80], [SA85], [SA86], [Sno90]). Traditional systems can thus be regarded as 'degenerations' of these evolving information systems. For a more elaborate discussion on the difference between traditional and evolving information systems, see [FOP92a] and [FOP92b].

An important concept in evolving information systems are updates. In traditional information systems, updates are a non-trivial aspect. In the field of evolving information systems, this is even more the case. The main focus of this paper will be on updates in evolving information systems. For this purpose, the kinds of updates which can be identified in an evolving information system, and why they are needed, are discussed and formalised first in section 2. In section 3, this is followed by a discussion on, and a formalisation of, how these kinds of updates are treated on different levels of abstraction. Finally, in section 4 we take a broader look at the effects of updates of the application model. We will distinguish some evolution dependencies, capturing the observation that if one part of an application model changes, other parts of the application model may have to be adopted accordingly.

2 Update in Evolving Information Systems

Information systems are meant to fulfill the information needs of organisations. As both the information needs and the information itself change in time, information systems have to be updated from time to time.

In this paper, a framework for update in evolving information systems is introduced, which is based on the possible causes for update requests. As argued in [FOP92b], update in traditional systems can be regarded as a degeneration of update in evolving systems. This is due to the fact that the requirements of traditional information systems, with respect to update, are less demanding than those of evolving information systems.

In this section, three kinds of user requests are identified: recording, correction and forgetting. The definitions of these kinds of user requests are given formally. Currently, a first prototype for the management of updates in evolving information systems is being implemented. This implementation is based on the formalisation as presented in this article, and will be part of an evolving information system shell to be developed.

2.1 A Taxonomy of Update

Updates in (evolving) information systems, result in a change of state of the application model. In traditional information systems an elementary update is usually considered to be an addition, deletion or modification of (pieces of) information contained in the application model. Furthermore, traditional information systems do not support any notion of time. As a consequence, if there is a change of state in the information system due to an update (addition, deletion or modification), the former state cannot be ‘remembered’ by the information system. Such systems are called snapshot systems ([SA86]), due to the fact that these information systems can only model and remember (single) ‘snapshots’ of an organisation’s evolution.

For evolving information systems, as well as for temporal or historical information systems ([SA86]), the traditional notion of update has to be revised. The framework for the processing of updates in evolving information systems is based on the possible causes for update requests. On this basis, three kinds of update are distinguished: recording, correction and forgetting ([FOP92a], [FOP92b]).

Since the application model should be a reflection of the universe of discourse, every *event*, being a change of state at a point of time, in the universe of discourse, (for instance the hiring/firing of personell) should be reported to the information system by means of a *recording* of this event.

In an ideal (evolving) information system, the application model reflects the state of the modelled universe of discourse in a (empirically) correct way at any point of time. It can, indeed, be checked whether the application model is consistent with the meta model, but there is no automatable way to guarantee that the contents of the application model reflect the real state of the universe of discourse. The latter case has to be checked empirically. This implies that in case of a found flaw in the application model, an update should be performed which *corrects* this flaw. This kind of update is called a *correction*, and corrects recordings which have already been performed, or which have not been performed at all.

A third kind of update can be distinguished if information systems are required to be able to remove information if requested. For instance, when a law exists stating that information about former personnel working for a company, may be stored for at most two years. This kind of update is called *forgetting*. The definition of the *forget* operator is highly dependent on the chosen modelling techniques for the application model. In particular on the one chosen for the modelling of the information structure and the information base. How can one, for instance guarantee that deleted information can not be derived from other, still remaining information. Therefore, we consider the forget operator to be beyond the scope of this article.

2.2 Recording of Events

The state of the universe of discourse, at a point of time, is reflected by a set of modelling constructs (e.g. entities, relationships, data base schemata, instances, rules, etc.) in the application model. These modelling constructs are referred to as *application model elements* ([FOP92a]). A recording of an event can thus be regarded as the modification of the set of application model elements constituting the state of the application model. A to be recorded event is specified as a sentence conform a particular language, which is based on the chosen modelling techniques for the application model. In order to be independent of the chosen modelling techniques, we consider the set of event denotations to be a set of abstract objects.

The modification of an application model state is performed by means of a set of elementary transitions. These transitions reflect the events, which occur in the universe of discourse, in the information system. Three kinds of elementary transitions are determined: a birth-transition creating an application model element, a death-transition terminating an element, and a change-transition replacing one element with another ([FOP92b]). The necessity of birth- and death-transitions should be clear. Change-transitions are needed in order to maintain the evolution of an application model element. Such an *application model element evolution* consists of a sequence of consecutive change-transitions, and can alternatively be seen as a partial function assigning application model elements to points of time (see also [PW93]).

As has been argued in e.g. [FOP92b] and [SA85], the time of recording of the events is different from its occurence time. The time at which an event occurs in a universe of discourse is called the event time, and the time at which the event is recorded is referred to as the recording time. This, and the above discussion leads to the following definitions:

Definition 2.1

1. *$\mathcal{ED}$ is the set of event denotations.*
2. *$\mathcal{T}_r$ is the time axis for event recordings, with a total order $<$.*
3. *$\mathcal{T}_e$ is time axis for event occurences, also with a total order $<$.*

4. *$\mathcal{EO} = \mathcal{ED} \times \mathcal{T}_e$ is the set of all event occurences. An event occurence is identified by an event denotation and the point of time at which the event occurs.*

5. *$\mathcal{ER} = \mathcal{EO} \times \mathcal{T}_r$ is the set of all event recordings. A recording of an event occurence is identified by the recorded event occurence, and the point of time at which the recording takes place*

On these concepts, the following functions (access routines) are defined:

Definition 2.2

1. *The point of time at which a recording has taken place:*
   $\mathsf{RcAt} : \mathcal{ER} \to \mathcal{T}_r$
   $\mathsf{RcAt}(x, t) = t$

2. *The event occurence which is recorded by a recording:*
   $\mathsf{RcOf} : \mathcal{ER} \to \mathcal{EO}$
   $\mathsf{RcOf}(x, t) = x$

3. *The point of time at which an event occurence, or a recorded event occurence, has taken place:*
   $\mathsf{OccAt} : \mathcal{ER} \cup \mathcal{EO} \to \mathcal{T}_e$
   $\mathsf{OccAt}(x, t) =$ **if** $x \in \mathcal{EO}$ **then** $\mathsf{OccAt}(x)$ **else** t **fi**

4. *The event denotation of an occured, or recorded, event occurence:*
   $\mathsf{OccOf} : \mathcal{ER} \cup \mathcal{EO} \to \mathcal{ED}$
   $\mathsf{OccOf}(x, t) =$ **if** $x \in \mathcal{EO}$ **then** $\mathsf{OccOf}(x)$ **else** x **fi**

The order on points of time, from both time axes, can be generalised to an order on recordings and event occurences as follows:

Definition 2.3
Let $x, y \in \mathcal{ER}$ then:

$$\begin{aligned} x <_{\mathcal{T}_r} y &\equiv \mathsf{RcAt}(x) < \mathsf{RcAt}(y) \\ x =_{\mathcal{T}_r} y &\equiv \mathsf{RcAt}(x) = \mathsf{RcAt}(y) \\ x \leq_{\mathcal{T}_r} y &\equiv x <_{\mathcal{T}_r} y \lor x =_{\mathcal{T}_r} y \end{aligned}$$

Let $x, y \in \mathcal{ER} \cup \mathcal{EO}$ then:

$$\begin{aligned} x <_{\mathcal{T}_e} y &\equiv \mathsf{OccAt}(x) < \mathsf{OccAt}(y) \\ x =_{\mathcal{T}_e} y &\equiv \mathsf{OccAt}(x) = \mathsf{OccAt}(y) \\ x \leq_{\mathcal{T}_e} y &\equiv x <_{\mathcal{T}_e} y \lor x =_{\mathcal{T}_e} y \end{aligned}$$

In a traditional snapshot information system, a birth transition of an application model element can be realised by means of an addition, and a death-transition by means of a deletion. In both cases, the 'old' application model states are lost. In an evolving information system, on the other hand, one of the major requirements is that no information may be lost, implying that no application model element may be deleted. (An exception to this rule is formed by the forget operation.) The history of application model elements, in an evolving information system, is kept by storing the birth/death/change transitions of application model elements together with the points of time at which these transitions take place.

2.3 Correction of Recordings

The actual state of an information system depends completely on the processing of update requests formulated by users of the system. These update requests should result in an information system which reflects the modelled universe of discourse in a correct way.

An information system reflects an organisation correctly if and only if there exists a isomorphism between the states and transitions of the application model and the states and transitions in the universe of discourse ([HW93]). From this requirement follows that the order in which the events occured in the universe of discourse should be the same as the order of the recorded events. This is needed because recordings of several events may interfere with each other, such that a different order may result in a different state of the information system. As an illustration of this phenomenon, consider the following two events for a universe of discourse concerned with personell working for departments at a university:

e_1: ADD Person: 'Erik' works-for Department: 'Information Systems'

e_2: DELETE Person works-for Department: 'Information Systems'

The event specifications in the above example are denoted in the semi-natural language LISA-D as defined in [HPW93]. The first event denotation (e_1) represents the adding of person 'Erik' as a coworker of the department of 'Information Systems', whereas the second event denotation (e_2) represents the deletion of all personell working for the department of 'Information Systems'. When recording e_1, e_2 in the obvious order, a state of the system will result in which the department of 'Information Systems' has no coworker. When e_2 is recorded before e_1, i.e. in the wrong order, the result will be that 'Erik' is the only person working for the department.

From the above example can be concluded that events should be recorded correctly, in order of their occurence. In practice, however, recordings may be performed too late, implying a violation of the proper order in which the events occured. Or, alternatively, a recording of an event which actually did not happen at all, or occured differently (i.e. the event denotation is wrong), may have been performed. This means that three kinds of corrections can be identified: the insertion of a (late) recording of an event in the sequence of already performed recordings, a removal of an already performed recording, or a replacement of a recording by a new recording of another event.

To accomplish these kinds of corrections, it must be possible to 'travel' backwards in the sequence of recordings which is ordered on time of recording. The operation which accomplishes this task is called a roll-back, and is the most primitive form of correction.

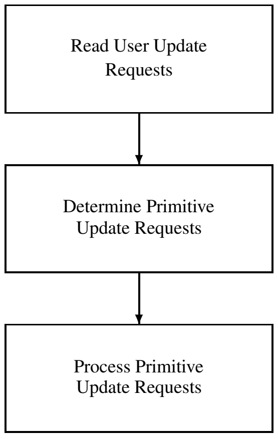
Read User Update Requests
Determine Primitive Update Requests
Process Primitive Update Requests

Figure 2: Processing of User Update Requests

In [OPF92] a formal definition is given of how the above discussed three kinds of corrections can be mapped onto roll-backs and (re)-recordings. It is therefore sufficient to consider roll-backs as the only corrections in the scope of this article. This has been illustrated in figure 2. The user update requests are mapped onto primitive update requests, consisting of recordings and roll-backs. This paper covers the procesing of the primitive update requests. The introduction of the above discussed correction mechanism leads to the following definitions:

Definition 2.4

1. $\mathcal{RB} = \mathcal{T}_r \times \mathcal{T}_r$ *the set of all roll-backs. Every roll-back rolls back to a recording which is identified by a recording time. Therefore, a roll-back is identified by the recording time of the recording to which the roll-back takes place, and the recording time at which the roll-back takes place.*

2. *For reasons of convenience the set of all possible update requests (as taken into account in this paper) is defined as well:* $\mathcal{UR} = \mathcal{ER} \cup \mathcal{RB}$.

Now that roll-backs have been defined formally, it is interesting to note that a recording can be regarded as an operation on an event occurence, whereas a rollback can be regarded as an operation on a recording. On the above defined concepts, the following operations exist:

Definition 2.5

1. *The* RcAt *operation on recordings of events is generalised to all updates. The point of time at which an update took place is given by:*

   $\mathsf{RcAt} : \mathcal{UR} \to \mathcal{T}_r$

   $\mathsf{RcAt}(x, t) = t$

2. *The point of time (of the recording) to which (and including) the given roll-back takes place is determined by:*

   $\mathsf{RbTo} : \mathcal{RB} \to \mathcal{T}_r$

   $\mathsf{RbTo}(t, x) = t$

Finally, the order on points of time on recordings ($<_{\mathcal{T}_r}$) can, based on the generalised RcAt function, be generalised to an order on updates ($\mathcal{UR}$) as:

Definition 2.6
Let $x, y \in \mathcal{UR}$ *then:*

$$\begin{aligned} x <_{\mathcal{T}_r} y &\equiv \mathsf{RcAt}(x) < \mathsf{RcAt}(y) \\ x =_{\mathcal{T}_r} y &\equiv \mathsf{RcAt}(x) = \mathsf{RcAt}(y) \\ x \leq_{\mathcal{T}_r} y &\equiv x <_{\mathcal{T}_r} y \vee x =_{\mathcal{T}_r} y \end{aligned}$$

2.4 User Update Requests

By means of the concepts defined in the previous subsections, the user update requests can be defined formally. As said before, the primitive user's update requests for an evolving information system are assumed to consist of two kinds of update requests: recordings, and roll-backs. This leads to the following definition:

Definition 2.7 *The set of updates entered by the user is modelled as:* $\mathcal{UI} = \langle \mathcal{UI}_{rc}, \mathcal{UI}_{rb} \rangle$ *such that* $\mathcal{UI} \subseteq \wp(\mathcal{ER}) \times \wp(\mathcal{RB})$ *where* $\mathcal{UI}_{rc}$ *(recordings),* $\mathcal{UI}_{rb}$ *(roll-backs) are presumed to be disjoint. As a shorthand, the set of updates* $\mathcal{UI}_{ud} = \mathcal{UI}_{rc} \cup \mathcal{UI}_{rb}$ *is also defined.*

As a simplifying assumption it is presumed that only one update occurs at one point of time. Note that this does not mean that an evolving information system has to be a single user system, it is simply demanded that there exists a global order on the user's updates. The assumption is formulated as:

Axiom 2.1 *Different updates are not recorded at the same point of time:*

$$u_1, u_2 \in \mathcal{UI}_{ud} \wedge u_1 =_{\mathcal{T}_r} u_2 \Rightarrow u_1 = u_2$$

The set of event occurences recorded by the recordings in $\mathcal{UI}_{rc}$ is denoted as $\mathcal{UI}_{occ}$, and the set of event denotations associated with the recordings in $\mathcal{UI}_{rc}$ as $\mathcal{UI}_{den}$. They are identified by:

Definition 2.8

$$\begin{aligned}\mathcal{UI}_{occ} &= \{\mathsf{RcOf}(r)\,|r \in \mathcal{UI}_{rc}\}\\ \mathcal{UI}_{den} &= \{\mathsf{OccOf}(r)\,|r \in \mathcal{UI}_{rc}\}\end{aligned}$$

As an overview of the hitherto defined concepts, a (meta) conceptual schema of the update requests, relating all concepts used for user update requests, is given in figure 3. The model depicted there is in the style of the NIAM modelling technique ([NH89], [Win90]).

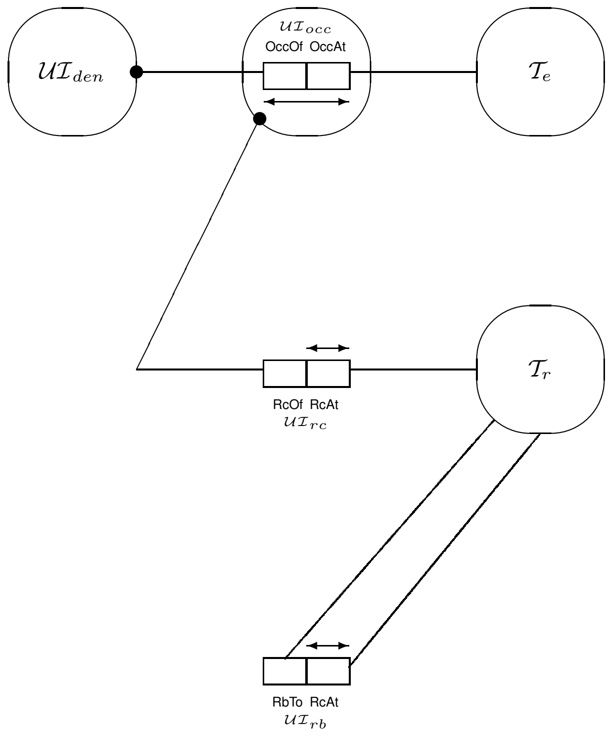

Figure 3: A Meta Model for Update Requests

3 A Conceptual Framework for Update Processing

Based on the notion of update as discussed in the previous section, a conceptual framework for update is presented ([FOP92b]). This framework distinguishes and relates different types of state transitions. Each type of state transition corresponds to a different level of abstraction in the context of update in evolving information systems. These levels are the event level, recording level, and correction level. State transitions on the event level take place due to events occuring in the organisation. The state transitions on the recording level, on the other hand, are caused by recordings of these events, whereas corrections of previous recordings cause state transitions on the correction level.

3.1 The Event Level

It is generally assumed that in the universe of discourse, described in the information system, a set of stable states can be recognised, and that there are a number of actions that result in event occurences (a change of state occuring at a point of time), see e.g. [FOP92a], [HW93].

As stated before, the elements of the application model reflect the application-dependent or time-variant elements in the universe of discourse, implying that the state of a universe of discourse at a particular point of time can be modelled by means of a set of application model elements. This set of application model elements is called the application model state.

An event, and the underlying state transition in the universe of discourse, is communicated to the system by means of an event denotation (together with the occurence time of the events). This event occurence implies a state transition of the application model state. In the previous section it was discussed that there exist three kinds of elementary transitions: birth transitions, death transitions and change transitions. These concepts are captured formally by means of the following definitions:

Definition 3.1

$$\begin{aligned}
\mathcal{AME} &= \textit{“Set of Application Model elements”}\\
\mathcal{B} &= \textit{“Set of Birth transitions”}\\
\mathcal{D} &= \textit{“Set of Death transitions”}\\
\mathcal{C} &= \textit{“Set of Change transitions”}\\
\mathcal{TR} &= \mathcal{B} \cup \mathcal{D} \cup \mathcal{C}
\end{aligned}$$

The transitions operate on application model elements. Which elements they operate upon is presumed to be determined by the following functions:

Definition 3.2

$$\begin{aligned}
\mathsf{Brth} : &\ \mathcal{B} \to \mathcal{AME} && \textit{The element which is born}\\
\mathsf{Dth} : &\ \mathcal{D} \to \mathcal{AME} && \textit{The element which dies}\\
\mathsf{ChTo} : &\ \mathcal{C} \to \mathcal{AME} && \textit{The element which is changed}\\
\mathsf{ChOf} : &\ \mathcal{C} \to \mathcal{AME} && \textit{The element resulting from the change}
\end{aligned}$$

The actual definitions of these functions depend on the chosen meta model, in particular on the semantics definition of the event denotations. These event denotations, as they were introduced in the previous section, are application model elements themselves. This is expressed as:

Axiom 3.1 $\mathcal{ED} \subseteq \mathcal{AME}$

The set of elementary transitions, which are implied by an event occurence can be determined (can be given a Concr*ete* value) by means of the Concr function. The actual set of elementary transitions as implied by a given event occurence, depends on the current state and past states of the application model. These states can be derived from the sequence of sets of already performed elementary transitions. The actual definition of the Concr function will not be given here since its definition depends on the modelling techniques and languages for the specification of the application model. For instance, in the example discussed in the previous section, concerning personnel working for departments at a university, the set of elementary transitions implied by the statement DELETE Person works-for Department: 'Information Systems' depends on the semantics of LISA-D ([HPW93]). The signature of the Concr function, nonetheless, can indeed be given:

Definition 3.3 $\mathsf{Concr} : \mathcal{EO} \times (\mathbb{N} \to \wp(\mathcal{TR})) \to \wp(\mathcal{TR})$

$\mathsf{Concr}(e, T)$ *must be interpreted as the set of elementary transitions needed to perform the event* e*, if the transitions in* T *have already occured, where* $T : \mathbb{N} \to \wp(\mathcal{TR})$.

The actual definition of the Concr is beyond the scope of this paper.

An event taking place in the universe of discourse is considered to occur on the organisational level ([FOP92a]). The corresponding events in the information system, implying transitions on the application model states, are considered to occur on the so called event level. A sequence of such application

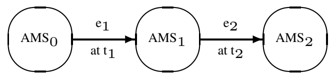

Figure 4: Application Model State (AMS) transitions on the event level

model state transitions is called an application model history. Such an application model history models a sequence of events occurring in the underlying universe of discourse of the information system.

In figure 4 a graphical representation of a sample application model history is given. The circles represent the application model states, whereas transitions between these application model states are represented by arrows. Furthermore, the arrows are labeled with the denotation of the event causing the transition, and the event time of that event. The example illustrates that to every sequence of event occurences an application model history can be associated. This means that a function from a sequence of event occurences to an application model history AMH can be defined. In the definition of AMH, we make use of a slicing operation on lists. The slicing of a list is defined as:

Definition 3.4 *If E is a list, then E^n is the list containing the first n elements of E, identified by:*

$$E^n = \{ \ \langle i, v \rangle \ \mid \langle i, v \rangle \in E \wedge i \leq n \ \}$$

Note that E is a function, and can thus be treated as a binary relation.

For a given sequence of event recordings, the associated application model history is identified by:

Definition 3.5 *Let E be a list of event occurences. Then $\mathsf{AMH}(E) = \langle \sigma^{ams}, H^{ams}, \beta, \delta, \gamma \rangle$, where T is a list of sets of transitions such that $T(i) = \mathsf{Concr}(E(i), T^{i-1})$ and $dom(E) = dom(T)$, and furthermore:*

$$\begin{array}{rcl}
\sigma^{ams} & = & \mathsf{AMS}(T, 0) \\
H^{ams} & = & \{ \ \langle \mathsf{AMS}(T, i), \mathsf{AMS}(T, i+1), E(i+1) \rangle \ \mid 0 \leq i < |E| \ \} \\
\beta & = & \{ \ \langle E(i), \mathsf{Brth}(T(i)) \rangle \ \mid 1 \leq i \leq |E| \ \} \\
\delta & = & \{ \ \langle E(i), \mathsf{Dth}(T(i)) \rangle \ \mid 1 \leq i \leq |E| \ \} \\
\gamma & = & \{ \ \langle E(i), \mathsf{ChOf}(T(i)), \mathsf{ChTo}(T(i)) \rangle \ \mid 1 \leq i \leq |E| \ \}
\end{array}$$

Note that by $dom(E)$ and $dom(T)$ the domains of the sequences is meant, i.e. all the indices such that the sequence has a value at that position.

The Brth, Dth, ChOf *and* ChTo *functions are presumed to be generalised to sets of transitions as:*

$$\begin{array}{rcl}
\mathsf{Brth}(T) & = & \{\mathsf{Brth}(t) \, | t \in T \cap \mathcal{B}\} \\
\mathsf{Dth}(T) & = & \{\mathsf{Dth}(t) \, | t \in T \cap \mathcal{D}\} \\
\mathsf{ChTo}(T) & = & \{\mathsf{ChTo}(t) \, | t \in T \cap \mathcal{C}\} \\
\mathsf{ChOf}(T) & = & \{\mathsf{ChOf}(t) \, | t \in T \cap \mathcal{C}\}
\end{array}$$

In the above definition, there is one 'lose end' in the form of the $\mathsf{AMS}(T, i)$ function. This, recursive, function returns the i-th state of the application model based on a list of sets of elementary transitions. This function is defined as:

Definition 3.6 *Let T be a list of sets of transitions. Then $\mathsf{AMS}(T, i)$ is defined as:*

```
$\mathsf{AMS}(T, i) =$ if $i = 0$ then {initial state of the information system}
                   else $\mathsf{AMS}(T, i-1) - \mathsf{Dth}(T(i)) - \mathsf{ChOf}(T(i)) \cup \mathsf{Brth}(T(i)) \cup \mathsf{ChTo}(T(i))$
                   fi
```

where, again, the Brth, Dth, ChOf *and* ChTo *functions are presumed to be generalised to sets of transitions.*

In general, the initial state of the information system ($\mathsf{AMS}(T, 0)$) will be chosen empty, being a strict pragmatically choice.

3.2 The Recording Level

In this section, a second level is introduced on which state transitions takes place: the recording level. Whenever an event occurs in the organisation, it should be communicated to the information system by means of an update request. The processing of this update request, which is called the recording of an event, should result in an appropriate state transition in the information system. The point of time at which the recording of an event takes place in the information system, is called the recording time of that event.

The resulting state transition is more than a single transition of an application model state, it can be seen as a transition of the complete application model history which modelled the history of the organisation up to the occurence of the newly recorded event. So this transition corresponds to a transition of a complete state transition graph. A sequence of these application model history transitions due to successive recordings is called an application model recording history. Such an application model recording history reflects both the events occurring in the organisation, and the recordings of these events in the information system.

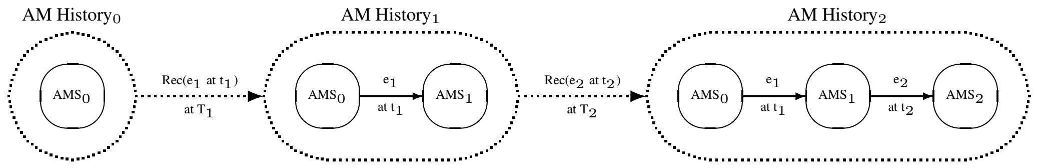

Figure 5: Application Model History (AMH) transitions on the recording level

In figure 5, the graphical representation of application model state transitions (due to events in the organisation) on the event level is extended with the above discussed second level, the recording level, on which the application model history transitions (due to recording of these events) take place. The arrows representing transitions between application model histories are labeled with the denotation of the recording of the event causing that transition, and the recording time of the event in question.

As can be seen in the example, an application model recording history is determined by a set of recordings. In general, the application model recording history, for a given sequence of recordings, is identified by:

Definition 3.7 *Let R be a list of recordings, then* $\mathsf{AMRH}(R) = \langle \sigma^{amh}, H^{amh} \rangle$*, where E is a list of event occurences such that $E(i) = \mathsf{RcOf}(R(i))$ and $dom(R) = dom(E)$, and furthermore that:*

$$\begin{array}{rcl} \sigma^{amh} & = & \mathsf{AMH}(E^0) \\ H^{amh} & = & \{ \ \langle \mathsf{AMH}(E^i), \mathsf{AMH}(E^{i+1}), R(i+1) \rangle \ \mid 0 \leq i < |E| \ \} \end{array}$$

3.3 The Correction Level

As a user may make mistakes when entering recordings of event occurences into the information system, three kinds of correction have been identified in the previous section. A recorded event may have to be replaced by another one, a recording may have to be inserted in the sequence of already recorded events, and a recording may have to be removed. As shown in [OPF92], these three kinds of recordings can be implemented by means of a roll-back and a series of re-recordings of already recorded (correct) events. In all cases which need a correction, a roll-back should take place to the latest application model history which is correct.

In order to process the user's input requests, the roll-backs have to be ordered in time. By means of this order, the recordings can be split up in a list of sets of recordings, in which each of the sets implies an application model recording history. The roll-backs in the user input ($\mathcal{UI}_{rb}$), can be ordered according to the following definition:

Definition 3.8 *Due to axiom 2.1, there exists a strict total order on the elements of $\mathcal{UI}_{rb}$. Let $B_{\mathcal{UI}}(i)$ denote the i-th rollback in $\mathcal{UI}_{rb}$ based on this order. So $dom(B_{\mathcal{UI}}) = \{1, \ldots, |\mathcal{UI}_{rb}|\}$*

In figure 6, the performance of a correction by means of a roll-back is graphically represented in the case of a *replacement* of the recording of an event e_1 having event time t_1 by a recording of event e_1' with the same event time. The replacement is presumed to take place after the recording of event e_2.

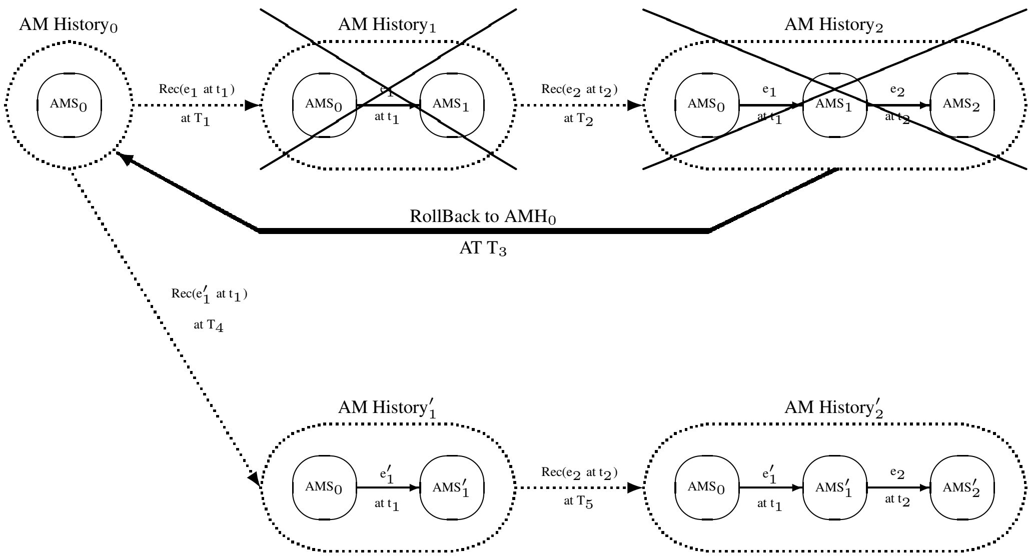

Figure 6: Correction by means of a roll-back

A sequence of successive recordings, an application model recording history, can be seen as the belief of the world (organisation) by the information system. This means that a correction of this belief of the world is performed by means of a roll-back which causes a transition of the current application model recording history in the information system. A sequence of these application model recording history transitions due to roll-backs is called the application model evolution, which is said to take place on the correction level.

In figure 7 the situation of figure 6 is represented in an alternative way identifying three levels of state transitions more clearly. Note that the roll-back performed by the correction is implicitly present in this figure. In the same way corrections requiring the removal or insertion of a recording of an event can be represented. In [FOP92a] more examples are given and elaborated.

As stated before, the recordings of the user's update requests have to be split up, based on the roll-backs, in a list of sets of recordings. Each of these sets will correspond to an application model's recording history. This distribution of recordings is given by:

Definition 3.9 *$U_{\mathcal{UI}}$ is a list of sets of recordings such that $dom(U_{\mathcal{UI}}) = \{0, \ldots, |\mathcal{UI}_{rb}|\}$, and:*

```
$U_{\mathcal{UI}}(i) =$ **if** $i = 0$ **then**
        $\{r \in \mathcal{UI}_{rc} \mid |dom(B_{\mathcal{UI}})| \geq 1 \Rightarrow r <_{\mathcal{T}_r} B_{\mathcal{UI}}(1)\}$
    **else**
        $\{r \in U_{\mathcal{UI}}(i-1) \mid \mathsf{RcAt}(r) < \mathsf{RbTo}(B_{\mathcal{UI}}(i))\} \cup$
        $\{r \in \mathcal{UI}_{rc} \mid |dom(B_{\mathcal{UI}})| \geq i+1 \Rightarrow r <_{\mathcal{T}_r} B_{\mathcal{UI}}(i+1) \wedge B_{\mathcal{UI}}(i) <_{\mathcal{T}_r} r\}$
    **fi**
```

The initial set of recordings $U_{\mathcal{UI}}(0)$ contains all recordings in $\mathcal{UI}_{rc}$ which have not been undone by the first roll-back (if present). The successive sets of recordings contain all recordings in the previous set of recordings $U_{\mathcal{UI}}(i-1)$ which have not been rolled-back by $B_{\mathcal{UI}}(i)$, and the set of new recordings done before the next roll-back at $B_{\mathcal{UI}}(i+1)$ (if present).

The sets of recordings in the above defined list have to be ordered themselves as well. Each of the resulting ordered list of recordings of events implies an application's model recording history. Due to axiom 2.1, and

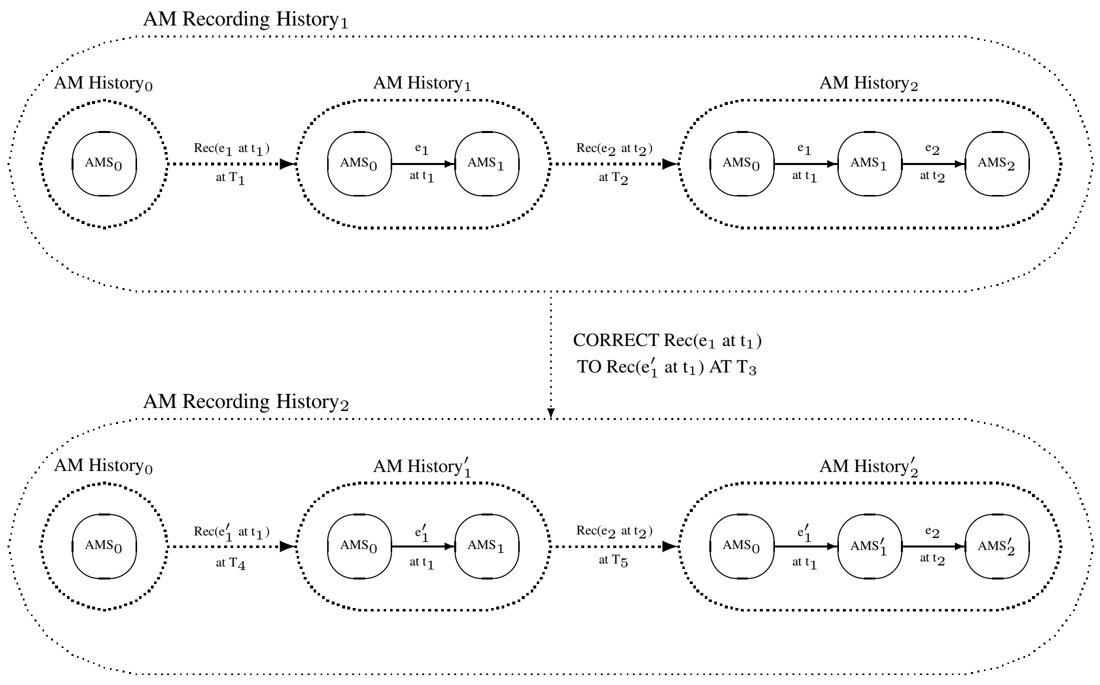

Figure 7: Application Model recording History (AMRH) transition on the correction level

definition 3.9, $<_{\mathcal{T}_r}$ defines a strict total order on the recordings in $U_{\mathcal{UI}}(i)$. This allows for the following order on the sets of recordings in $U_{\mathcal{UI}}$:

Definition 3.10 *$R_{\mathcal{UI}}$ is a list of lists of recordings such that $R_{\mathcal{UI}}(i)(j)$ is the j-th recording based on the order $<_{\mathcal{T}_r}$ in $U_{\mathcal{UI}}(i)$.*
So $dom(R_{\mathcal{UI}}) = dom(U_{\mathcal{UI}})$, and $i \in dom(R_{\mathcal{UI}}) \Rightarrow dom(R_{\mathcal{UI}}(i)) = \{1, \ldots, |R_{\mathcal{UI}}(i)|\}$.

The evolution of the application model (AMEV) for a given user input $\mathcal{UI}$ is now defined as:

Definition 3.11 $\mathsf{AMEV}(\mathcal{UI}) = \langle \sigma^{amrh}, H^{amrh} \rangle$, *where:*

$$
\begin{aligned}
\sigma^{amrh} &= \mathsf{AMRH}(R_{\mathcal{UI}}(0)) \\
H^{amrh} &= \{ \ \langle \mathsf{AMRH}(R_{\mathcal{UI}}(i)), \mathsf{AMRH}(R_{\mathcal{UI}}(i+1)), B_{\mathcal{UI}}(i+1) \rangle \ \mid 0 \leq i < |\mathcal{UI}_{rb}| \ \}
\end{aligned}
$$

Finally, an overview of all the defined concepts is provided in the conceptual schema (meta schema) of the framework of the processing of the user update requests, in figure 8. The schema depicted there is in the style of PSM ([HPW92]) which is an extension of the modelling technique from NIAM ([NH89]). The extension of NIAM used in figure 8 deals with schema objectifications. In a schema objectification, a population of the objectified schema at hand is considered as one single abstract object instance. The encircled schemas in figure 8 labeled with AMRH, AMH and AMS are graphical representations of objectified schemas, each reflecting a level of the discussed three level framework for updates. Note that the three objectified schemata are nested.

4 Evolution Dependencies when Performing Updates of Application Models

In this section we take a broader view on updates, and look at updates of the application model at a broad level. Whenever one part of an application model is updated, other parts may have to be updated

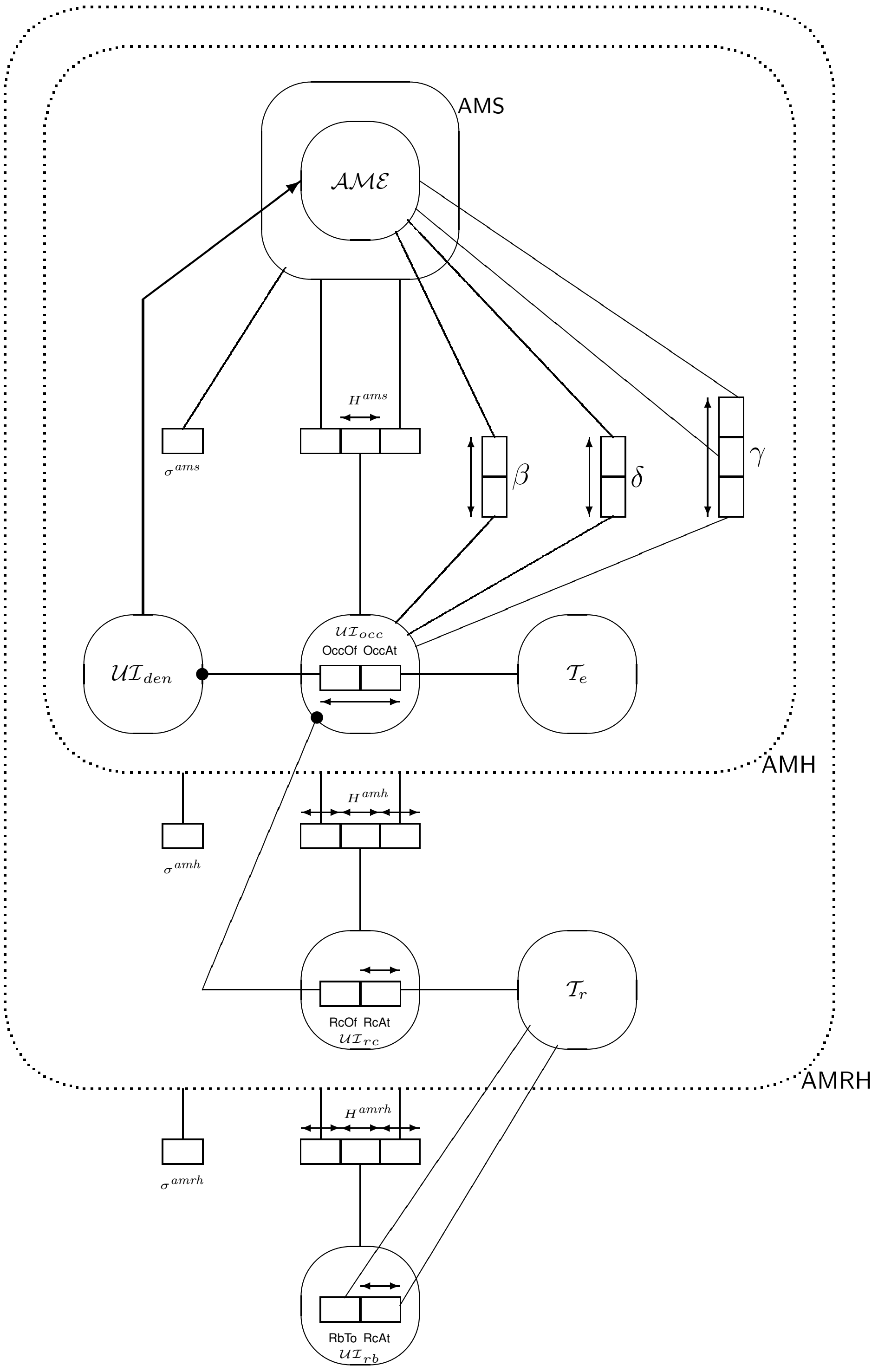

Figure 8: A Meta Model for the Update Request Processing Framework

accordingly. For instance, if an object type is deleted all action specifications in the action model refering to this object type will have to be adopted, or deleted. In this section we provide an overview of these *evolution dependencies*. Before doing this, we define the contents of an application model in more detail. An application model, being a complete specification of a universe of discourse typically ([ISO87], [HW92]) contains the following components:

1. An intentional description of the set of populations of the universe of discourse. This is referred to as the underlying *information structure*.
2. A further refinement of the set of allowed populations by means of *static constraints*.
3. An intentional description of the set of transitions that can be performed automatically (by the system). This description is usually provided as a set of *action specifications* bringing about those transitions as a reaction to other transitions, caused automatically or manually.
4. A refinement of the set of possible transitions (between valid states), both automatically and manually, by means of so called *dynamic constraints*.
5. An extentional specification of the current state of the universe of discourse, i.e. the *population*, or information base, of the underlying *information structure*.

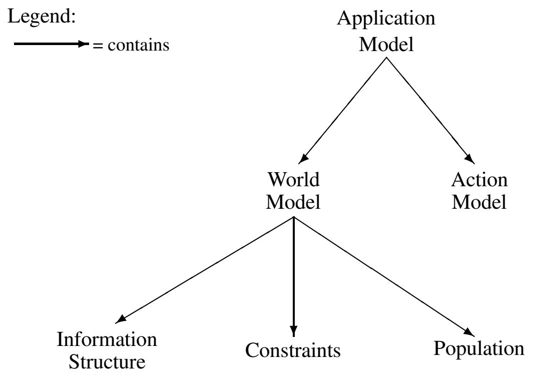

Figure 9: The Structure of the Application Model

An *action model* consists of a set of action specifications, describing the transitions that can be performed by the system. A *world model* encompasses the combination of information structure, both static and dynamic constraints, and a population conforming to these requirements. The complete specification of a universe of discourse containing all these components is referred to as the *application model* ([FOP92a], [FOP92b]). These definitions, result in the hierarchy of models, as denoted in figure 9. There the distinct models, their interrelationships, and their respective components are depicted.

Most updates of the application model will be concerned with the update of the information base. These ‘normal’ updates of instances of object (entity) types in the information base hardly ever lead to updates of other parts of the application model, unless, for example, the violation of a constraint leads to the conclusion that the violated constraint is wrong. However, a change of the information structure, for example the addition/deletion/change of an object type, almost inevitably implies the necessity of updates of other parts of the application model. The following *evolution dependencies* can be identified:

- The constraints in the application model, depend on the information structure.

  If an object type is deleted, all the constraints defined on that object type, or using that object type in their definition, have to be deleted - or changed - too. If an object type is added, one probably has to add some appropriate constraints as well.

- The information base is highly dependent on the information structure, as well as the constraints.

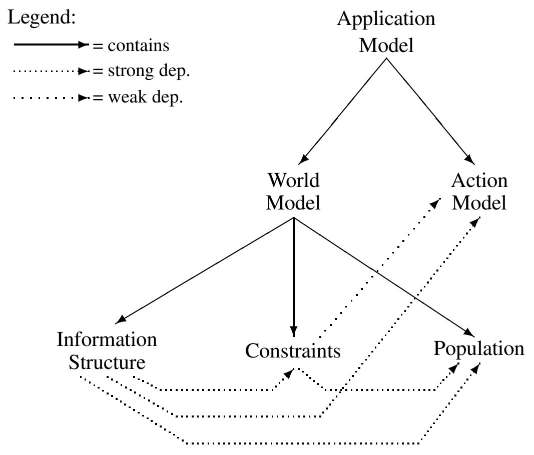

Figure 10: Evolution Dependency

If an object type is deleted or changed, all the instances of this object type have to be deleted or changed as well. Furthermore, whenever a new constraint is added, or a constraint is replaced by a more limitative one, the population may have to be updated accordingly, i.e. instances may have become illegal. So schema evolution inevitably leads to changes of the information base.

- The action model depends on the information structure as well.

  Whenever an object type is deleted, all the activities operating on objects of this type have to be deleted - or changed - too. If an object type is added, one probably has to specify appropriate activities operating on the instances of this new object type as well.

Apart from these 'strong' dependencies, we can also recognize a 'weak' dependency. The 'weak' dependency should be interpreted in this context as a deontic rule, specifying the 'usual' case.

- If one of the constraints in the application model changes, one might have to change activity specifications as well.

  If, for instance, a constraint is changed/added, an activity's operation might lead to a violation of a constraint, although it did not do so before the change.

All these dependencies are illustrated in figure 4. A proper formalisation of the notion of evolution dependency depends on the syntacical and semantical definitions of the modelling techniques used for the application model.

5 Conclusions & Further Research

In order to handle temporal and evolutionary aspects in an evolving information system, the traditional notion of update was revised, resulting in the triple: recording, correction and forgetting. It was stated that update should not forget any aspect ever fed to the system, unless explicitly asked for. The notion of updating an application model was described by introducing a state-transition-oriented model on three levels of abstraction (event, recording and correction level). This model has been formalised.

The conceptual framework for update as proposed in this article positions evolving information systems with respect to traditional systems, in the sense that traditional information systems can be regarded as degenerations of evolving information systems. Currently, the formalised update framework as presented in this article is being implemented, as a first step towards an evolving information system shell.

The identified evolution dependencies serve as the input for a continuing meta-modelling process, leading to the complete meta model for an evolving information system shell. This work involves the formalisation of the meta model, and the design of a language for the manipulation and specification of application models. A complete evolving information system shell will be implemented based on that meta model and language. Furthermore, a design method for the process of building up and maintaining an application model of an evolving information system will be developed.

References

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

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

[BCG$^+$87] J. Banerjee, H.-T. Chou, J.F. Garza, W. Kim, D. Woels, and N. Ballou. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5(1):3–26, 1987.

[Bub80] J.A. Bubenko. Information Modelling in the Context of System Development. In S.H. Lavington, editor, *Information Processing 80*, pages 395–411. North-Holland/IFIP, Amsterdam, The Netherlands, 1980.

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

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

[FOP92b] E.D. Falkenberg, J.L.H. Oei, and H.A. Proper. Evolving Information Systems: Beyond Temporal Information Systems. In A.M. Tjoa and I. Ramos, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA'92)*, pages 282–287, Valencia, Spain, EU, September 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3211824006

[HPW92] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 364–377, Manchester, United Kingdom, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815

[HPW93] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[HSV89] 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.

[HV91] K.M. van Hee and P.A.C. Verkoulen. Integration of a data model and high-level petri nets. In *Proceedings of the Twelfth International Conference on Applications and Theory of Petri Nets*, pages 410–431, Gjern, Denmark, 1991.

[HW92] 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.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

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

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

[Kat90] R.H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.

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

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

[OHFB92] J.L.H. Oei, L.J.G.T. van Hemmen, E.D. Falkenberg, and S. Brinkkemper. The Meta Model Hierarchy: A Framework for Information System Concepts and Techniques. Technical Report 92-17, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, 1992.

[OPF92] J.L.H. Oei, H.A. Proper, and E.D. Falkenberg. Modelling the Evolution of Information Systems. Technical Report 92-36, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.

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

[RBL87] C. Rolland, F. Bodart, and M. Leonard, editors. *Temporal Aspects in Information Systems*. North-Holland/IFIP, Amsterdam, The Netherlands, 1987.

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

[SA85] 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.

[SA86] R. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9):35–42, 1986.

[Sno90] R. Snodgrass. Temporal Databases Status and Research Directions. *SIGMOD Record*, 19(4):83–89, December 1990.

[Ver89] A.A. Verrijn-Stuart. Some Reflections on the Namur Conference on Information Systems Concepts. In E.D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*. North-Holland/IFIP, Amsterdam, The Netherlands, 1989.

[WHO92] 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.

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

Contents