Object-Oriented Modelling based on Logbooks

P. VAN BOMMEL[1], P. J. M. FREDERIKS[2] AND TH. P. VAN DER WEIDE[1]

[1]*Department of Information Systems, University of Nijmegen, Toernooiveld 1, NL-6525 ED Nijmegen, The Netherlands*

[2]*Edmond Research & Development B.V., Van Trieststraat 1a, 6512 CW Nijmegen, The Netherlands*

Email: tvdw@cs.kun.nl

In this paper the notion of *logbook* is introduced as a common basis for various models to be produced during system analysis. A logbook has a unifying format which contains a complete description of the history of some Universe of Discourse. It is intended as a structuring mechanism for initial specifications based on natural language. A typing mechanism is provided as an abstraction mechanism for logbook instances, leading to object-oriented analysis models.

Received October 28, 1996; revised February 17, 1997

1. INTRODUCTION

Many conceptual modelling methods have as point of departure an initial specification of the world to be modelled, also referred to as the *Universe of Discourse* (UoD) [1]. This initial specification is usually expressed in natural language. The goal of the modelling process is to obtain a complete and consistent formal specification from the initial specification. Only a few conceptual modelling methods provide the system analyst with clues and rules which can be applied to the initial specification such that this specification can actually be used in the modelling process. Examples of these modelling methods are NIAM [2], [3] for conceptual data modelling and KISS [4] for object-oriented modelling.

In order to structure this way of working, the (sentences in the) initial specification should be in accordance with some unifying format. In NIAM these sentences are called *elementary sentences*, while KISS uses the term *structure sentences*. Elementary and structure sentences provide a simple and effective handle for obtaining the underlying conceptual model of so-called *Snapshot Information Systems* [5], i.e. information systems where only the current state of the UoD is relevant.

However, even though these initial specifications are an important aid in modelling information systems, they still are too poorly structured. This lack of structure results in several problems during the modelling process. As an example, a system analyst has to come up with a lot of properties of the UoD, which do not follow directly from the initial specification, but have to be specified in order to satisfy the 100% principle for the formal specification. One of these properties is the order and history of events. More concretely, since the mutual order of the sentences in an initial specification is lost, the analyst has to reconstruct this order. Other UoD properties are for instance related to the instances involved in events and the role in which they are involved. This combination, together with properties about the specific involvement, is referred to as an associate of the event. Usually, associates are not clear from the informal specification.

In this paper the notion of *logbook* is introduced as a common basis for various models to be produced during system analysis. A logbook has a unifying format which contains a complete description of the history of some UoD. Generally a logbook will report on changes at the level of *instances* within the UoD, as well as on the *structural* level (information structure) of the UoD. A logbook may thus be seen as the sequence of all events which describe the evolution of the UoD. From a logbook each past state of the UoD can be derived, including the current state. As a consequence, a UoD model basically corresponds to the set of all acceptable logbooks. The goal of the modelling process then is to obtain a formal description of this set of acceptable logbooks.

The logbook is intended as a structuring mechanism for initial specifications in the context of Snapshot Information Systems as well as *Temporal Information Systems* [6]. Furthermore, it supports the development of *Evolving Information Systems* (e.g. [7], [8], [9]). Note that Evolving Information Systems can be used to handle *versioning* problems.

The main reason for introducing the concept of logbook is that it can be used effectively to bridge the gap between a UoD and its specification. The history of a UoD may be described by an application model history, such as that introduced in [7]. In this view, the evolution of a UoD is seen as the evolution of its elements. An element evolution describes the state of the element at each point of time. This dual vision on evolving information systems will not be elaborated further in this paper.

The organization of the paper is as follows. In Section 2 logbooks are formally introduced. We present a meta model

in terms of PSM [10], followed by a further elaboration of logbook functions and properties. These properties include the so-called creation (birth) and destruction (death) of objects. Section 3 describes the modelling process, i.e. the process of obtaining a conceptual model from a logbook. This is centered on the definition of a typing mechanism. Our approach is further illustrated by relating different views on a logbook to concrete models which constitute the overall conceptual model. We consider the object action involvement model, the object life model and the object property model. Finally, conclusions and directions for further research are presented in Section 4.

2. LOGBOOKS

The most simple description of events occurring in a UoD consists of the following components for each event: (1) when does it occur, (2) what is happening, (3) who are associated, (4) what is the role of each associate and (5) which properties of associates are relevant. A logbook is a set of such event descriptions, ordered by their time-stamps in some unifying format (seconds, hours, days, etc.).

2.1. Informal specifications

The starting point for constructing a logbook is an informal specification consisting of natural language sentences describing time-stamped events. Obtaining an informal specification is usually an incremental and iterative process [11], [12] which requires skills for both the domain expert and system analyst [13]. As an example of an informal specification consider Table 1, where the unifying time format is chosen to be on a daily basis.

2.2. Meta-model of a logbook

Formally, the format of a logbook is defined by the meta model of Figure 1. This meta model is presented as a PSM schema which is a formalized and extended version of the object role modelling technique NIAM. For more literature about the background and formalization of PSM, the reader is referred to [10] or [14].

A *PSM schema* is a structure consisting of an *information structure* and a set of *constraints*. The semantics of a PSM schema is the set of its possible *populations*, which is restricted by both the information structure and the constraints.

The information structure consists of the following basic components:

1. A set of *predicators*. A predicator is intended to specify the role played by an object type in a fact type (see below).
2. A set of *object types* which can be classified as follows:
   (a) *Entity types* and *label types*. The difference is that labels can, in contrast to entities, be represented (reproduced) on a communication medium.
   (b) *Fact types*. The set of fact types is a partition of the set of predicators.

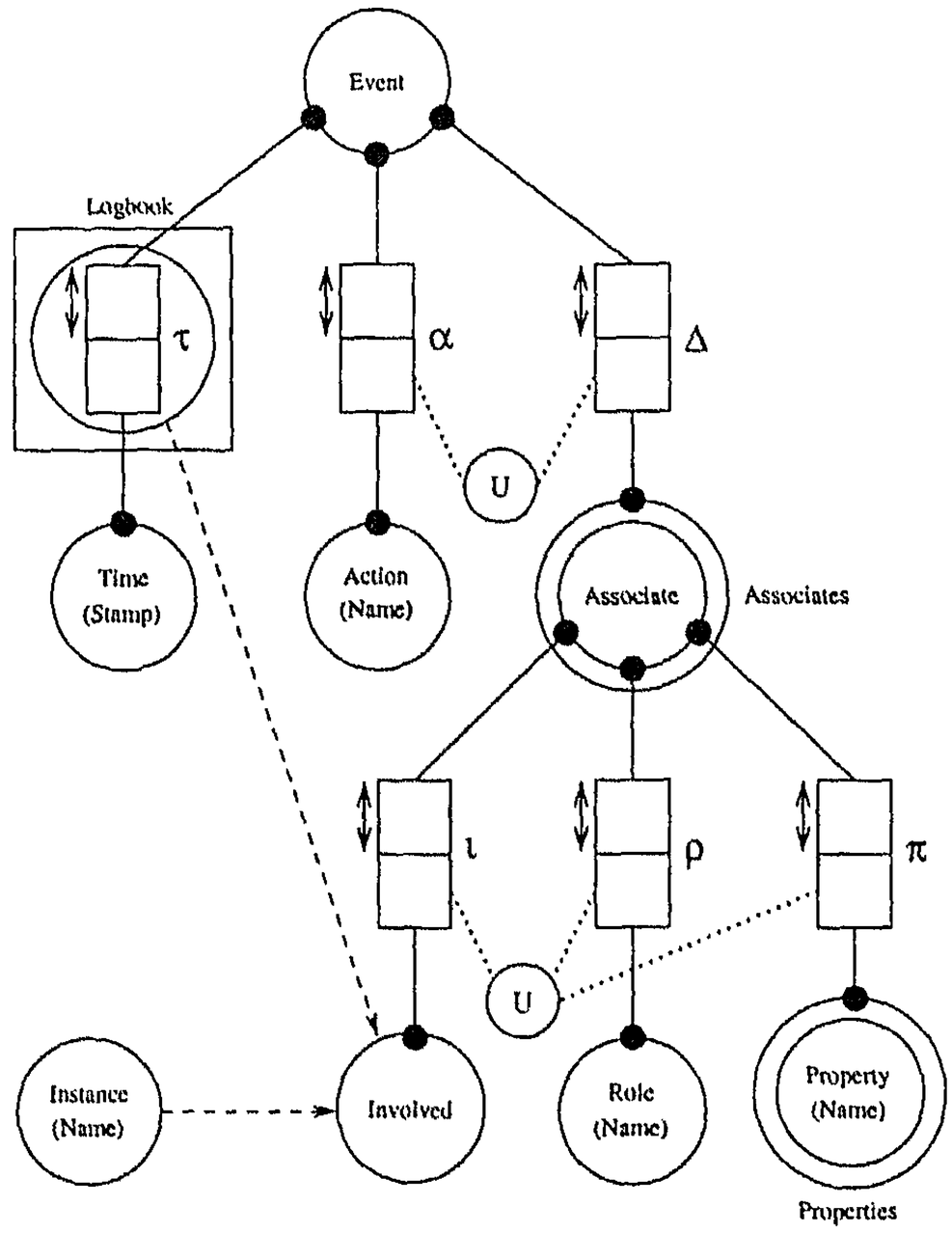

FIGURE 1. Meta model for the format of a logbook.

   (c) Complex object types such as *power types* (also called set types), *sequence types* (for modelling list structures) and *schema types* (which can be used as an abstraction mechanism for information structures).
3. Relations for specifying (a) *specialization* and (b) *generalization*.
4. Functions for specifying (a) the object type associated to a predicator, (b) the fact type of an associated predicator and (c) the underlying element type (or object types) for power types and sequence types (and schema types).

In Figure 1 we have a number of entity types, e.g. *Event*, represented by circles. Label types, also represented by circles, appear between parentheses. Examples of label types are *Name* and *Stamp*. The convention is used that if a label type is an identifier for an entity type, the label type is represented within the same circle. For example, in the logbook meta-model, an *Action* is identified by its *Name*. Fact types consist of predicators represented by boxes connected to circles for their associated object types. For example fact type τ is a binary fact type consisting of two predicators, with associated object types *Event* and *Time*. Object type *Properties* is modelled as a group type with underlying element type *Property*. A *Logbook* is seen as a sequence type of tuples consisting of an *Event* and *Time* part. Finally, the information structure of a logbook contains a generalization hierarchy. Object type *Involved* is either an *Instance* or an *Event/Time* tuple.

TABLE 1. Example informal specification

01-05-1963:	Mick Jagger and Keith Richards set up band 'The Rolling Stones'.
03-05-1967:	Mick Jagger and Keith Richards write Paint It Black.
21-06-1967:	Paint It Black is recorded, on tape nr 666 in studio nr 11, by The Rolling Stones.
23-06-1967:	The recording of Paint It Black, on tape nr 666 in studio nr 11, is produced by Mick Jagger and Keith Richards.
12-12-1988:	The song 'I Want You' is written by A. Knijff.
10-02-1989:	P. Frederiks and A. Knijff set up The Playful Plebs.
29-04-1991:	The band 'The Playful Plebs' record, in studio nr 2 on tape nr 3, the song 'I Want You'.
05-05-1991:	The Playful Plebs produce the recording of the song 'I Want You', in studio nr 2 on tape nr 3, which is recorded by this band.

The PSM schema of Figure 1 also contains a number of constraints. A *Total Role* constraint, denoted by a black dot, expresses that each instance of the corresponding object type must be involved in the corresponding fact type. For example each *Event* must have a *Time* component. The arrows and 'circles with an *U*' represent so-called *Uniqueness* constraints over a predicator and set of predicators, respectively. The uniqueness constraint is used to specify that each instance of the corresponding object type is involved only once in the corresponding fact type if it plays that role. Note that both a total role constraint and a uniqueness constraint over a predicator state that an instance of the corresponding object type is involved exactly once in the corresponding fact type. Thus such a predicator is practically a function. In the remainder of this paper we will use this observation and name this function by the name of its corresponding fact type.

In order to be able to represent logbooks in a way such as that presented in Table 1, some additional (non-graphical) constraints are required. The PSM modelling technique is equipped with the query language Lisa-D to express such constraints. Lisa-D as such falls outside the scope of this paper, for more information see [15]. The additional constraints are informally expressed as:

1. the order of the tuples of a logbook is according to the order of its time stamps and
2. if two events occur at the same time, these events cannot be the same.

Table 2 represents a population of the meta model of the logbook in Figure 1 and thus is a unifying format for the sample logbook of Table 1. Note that the action *to write* may have multiple agents. However, each of these events is elementary and thus cannot be split into smaller events. As a consequence, the modelling technique must be able to handle the case of multiple agents on a conceptual level.

A population of the logbook meta-model can be obtained by grammatical analysis of the sentences of the informal specification. Methods such as KISS and SACIS (as described in [16]) provide the analyst with clues for grammatical analysis of informal specifications. Grammatical analysis as such falls outside the scope of this paper.

2.3. Logbook functions

In this section a number of functions are defined in order to elaborate the notion of a logbook as defined by its meta model. The function $\tau(e)$ is such a function which produces the point of time of an event e. The function $\alpha(e)$ results in the performed action and $\Delta(e)$ is the mapping which assigns the set of associates occuring in event e.

Let a be an associate of event e, i.e. $a \in \Delta(e)$. The function $\iota(a)$ yields the involved object, $\pi(a)$ assigns the (possibly empty) set of properties of this object, whereas $\rho(a)$ provides its role in this event. Note that a role may be played more than once in an event.

An event e is thus identified by the action $\alpha(e)$ performed, and the set $\Delta(e)$ of associates. An associate a is identified by its involved object $\iota(a)$, its corresponding set of properties $\pi(a)$ and its role $\rho(a)$.

We introduce for given functions $f : B \rightarrow C$ and $g : A \rightarrow P(B)$ the composition operator $*$ as:

$$(f * g)(a) = \{f(b) \mid b \in g(a)\}$$

EXAMPLE 1. Consider the event e_1 with time stamp (*23-06-1967*) from Table 2:

$$\left\langle \text{is produced}, \left\{ \begin{array}{l} (e_2, \text{object}, \{\text{tape nr 666}, \text{studio nr 11}\}), \\ (\text{Mick Jagger}, \text{agent}, \emptyset), \\ (\text{Keith Richards}, \text{agent}, \emptyset) \end{array} \right\} \right\rangle$$

and the event e_2 with time stamp (*21-06-1967*):

$$\left\langle \text{is recorded}, \left\{ \begin{array}{l} (\text{Paint It Black}, \text{object}, \emptyset), \\ (\text{The Rolling Stones}, \text{agent}, \emptyset) \end{array} \right\} \right\rangle$$

which correspond with the following sentences of the informal specification:

The recording of Paint It Black, on tape nr 666 in studio nr 11, is produced by Mick Jagger and Keith Richards.

Paint It Black is recorded, on tape nr 666 in studio nr 11, by The Rolling Stones.

Applying the functions τ, α, Δ, ι, ρ and π to events e_1 and e_2 results in:

$$\begin{aligned} \tau(e_1) &= \text{23-06-1967} \\ \alpha(e_1) &= \text{is produced} \\ \Delta(e_1) &= \{a_1, a_2, a_3\} \end{aligned}$$

TABLE 2. A sample population of Figure 1

Time	Event			
	Action	Associate		
		Involved	Role	Properties
01-05-1963	set up	Mick Jagger	agent	Ø
		Keith Richards	agent	Ø
		The Rolling Stones	object	Ø
03-05-1967	write	Mick Jagger	agent	Ø
		Keith Richards	agent	Ø
		Paint It Black	object	Ø
21-06-1967	record	Paint It Black	object	Ø
		The Rolling Stones	agent	Ø
23-06-1967	produce	$\langle$21-06-1967, $\langle$record, $\left\{ \begin{array}{l} \langle \text{Paint It Black, object, } \emptyset\rangle \text{ ,} \\ \langle \text{The Rolling Stones, agent, } \emptyset\rangle \end{array} \right\}\rangle\rangle$	object	{tape nr 666, studio nr 11}
		Mick Jagger	agent	Ø
		Keith Richards	agent	Ø
12-12-1989	write	I Want You	object	Ø
		A. Knijff	agent	Ø
10-02-1989	set up	P. Frederiks	agent	Ø
		A. Knijff	agent	Ø
		The Playful Plebs	object	Ø
29-04-1991	record	The Playful Plebs	agent	Ø
		I Want You	object	Ø
05-05-1991	produce	H. Honer	agent	Ø
		$\langle$29-04-1991, $\langle$record, $\left\{ \begin{array}{l} \langle \text{I Want You, object, } \emptyset\rangle \text{ ,} \\ \langle \text{The Playful Plebs, agent, } \emptyset\rangle \end{array} \right\}\rangle\rangle$	object	{tape nr 3, studio nr 2}

$$
\begin{array}{rcl}
\iota(a_1) & = & e_2 \\
\iota(a_2) & = & \text{Mick Jagger} \\
\iota(a_3) & = & \text{Keith Richards} \\
\rho(a_1) & = & \text{object} \\
\rho(a_2) & = & \text{agent} \\
\rho(a_3) & = & \text{agent} \\
\pi(a_1) & = & \{\text{tape nr 666, studio nr 11}\} \\
\pi(a_2) & = & \emptyset \\
\pi(a_3) & = & \emptyset \\
 & & \\
\tau(e_2) & = & \text{21-06-1967} \\
\alpha(e_2) & = & \text{is recorded} \\
\Delta(e_2) & = & \{a_4, a_5\} \\
\iota(a_4) & = & \text{Paint It Black} \\
\iota(a_5) & = & \text{The Rolling Stones} \\
\rho(a_4) & = & \text{object} \\
\rho(a_5) & = & \text{agent} \\
\pi(a_4) & = & \emptyset \\
\pi(a_5) & = & \emptyset
\end{array}
$$

2.4. Properties of a logbook

In this approach, a concrete logbook is an instantiation of the meta model from Figure 1, i.e. a mapping which instantiates each object type. A population thus assigns a set of values to each object type from the meta model. Let Log be such a mapping (population). Applying function application leads simply to some results. For example the set of all involved objects occurring in this logbook can be obtained with Log(Involved). This set corresponds to what is called *the extra temporal population* in [7].

The logbook can be restricted to the history (evolution) Log_x of a single involved object x by removing all events from the logbook in which x is not involved. This population of the meta model is obtained by first determining the relevant events:

$$\mathsf{Log}_x(\tau) = \{\langle t, e\rangle \in \tau \mid x \in (\iota * \Delta)(e)\}$$

where, for convenience, $\langle t, e\rangle \in \tau$ is used as a shorthand for $\langle t, e\rangle \in \mathsf{Log}(\tau)$ (thus overloading the $\in$-operator). The instantiations of the other object types of the meta model are obtained as the minimal subset of their original instantiation required for $\mathsf{Log}_x(\tau)$.

Restricting a logbook Log to a particular point of time (time stamp) t results analogously in a *snapshot* of the logbook:

$$\mathsf{Log}_t(\tau) = \{\langle t, e\rangle \in \tau \mid \tau(e) = t\}$$

The history of a single action a is obtained from:

$$\mathsf{Log}_a(\tau) = \{\langle t, e\rangle \in \tau \mid \alpha(e) = a\}$$

Note that $\mathsf{Log}_a(\mathsf{Involved})$ results in the set of objects involved in action a.

Many object-oriented modelling methods have models describing the life of objects. The creation (birth) of an object has an especially prominent position in these life models. The creation $\mathsf{Init}(x)$ of an object x is defined as the first event in the logbook which mentions this object:

$$\mathsf{Init}(x) \in \mathsf{Log}_x(\mathsf{Event}) \wedge \forall_{e \in \mathsf{Log}_x(\mathsf{Event})} \left[\tau(\mathsf{Init}(x)) \leq \tau(e)\right]$$

As a result $\alpha(\mathsf{Init}(x))$ denotes the action which causes the birth of object x. Note that the death of an object cannot be derived from the logbook since objects are not removed from the logbook. Usually there will be a criterion $\mathsf{Alive}(x, t)$ which decides whether object x is alive at some point of time t. The following property then is obvious:

$$\mathsf{Alive}(x, t) \wedge \forall_{s<t} \left[\neg\mathsf{Alive}(x, s)\right] \Rightarrow \tau(\mathsf{Init}(x)) = t$$

The death of an object is not derivable, however, from the Alive predicate but the death of an object may be concluded. Suppose object x is alive at time t, but not alive at a later point of time s, then x has died somewhere in between. The set Log(Role) of all roles has a central position during the modelling process. In the case of evolving information systems, this set will also be subject to evolution. In this paper, we restrict ourselves to a stable set of roles and thus to the specification of temporal information systems.

3. MODELLING A LOGBOOK

A number of conceptual data modelling techniques are based on using natural language. The goal of such modelling techniques is to derive the grammar that governs the communication within the UoD. Some examples of such techniques are NIAM and PSM. This grammar, also referred to as *information grammar* (see e.g. [17], [18], [19], [20]), can be depicted as an information structure diagram. Information grammars achieved with these techniques only describe static aspects of the communication in the UoD. In order to find an information grammar which also includes dynamic parts of the UoD, the modelling process may start from a informal specification such as presented in Table 1.

Once a stable sample logbook is obtained from this informal specification the main task of the system analyst is to find abstractions for the concrete instances within the sample logbook, leading to a structural description (conceptual model). The information grammar can also be used to paraphrase the conceptual model, resulting in a textual description of the UoD. A major advantage of this approach is that the domain expert can validate this textual description [21].

In the remainder of this section we will first informally introduce a number of analysis models for object-orientation which compose the conceptual model. Each analysis model describes a specific view on the logbook. For an in-depth treatment of the analysis models see [22]. Then we will introduce a typing mechanism which describes the modelling process, i.e. a way to obtain the analysis models from the sample logbook.

3.1. Views on a logbook

During the several stages of the modelling process the sentences of a logbook are analysed according to three perspectives, leading to a number of abstractions of the logbook, also referred to as *analysis models*. Each abstraction provides a specific view on the nature of the application domain and results in a corresponding model. The models together compose a conceptual model of the UoD. This conceptual model is also referred to as the *information architecture*. The information architecture composes the following analysis models:

- object action involvement model,
- object life model and
- object property model.

The purpose of the *object action involvement model* is to develop a first approximation of the information grammar, covering the main structure of the logbook language. This model provides an abstraction (type) for three columns of the logbook, i.e. *Action*, *Involved* and *Role* (see Table 2). What is special is that different types of events may have the same type of action. Eventually this can lead to the introduction of generalized object types. Subtyping hierarchies as such falls outside the scope of this paper.

The object action involvement model does not restrict the order of the action types. The object life model elaborates on the object action involvement model. It (the object life model) considers the application domain from a historical perspective and describes for each object type the *course of life* of its instances. Each object type starts with its birth action type (creation). From the object action involvement model one can derive in which action types an object type is involved. Using the column *Time* of the logbook, clues for the order of action types can be derived. Note that the object life model describes a possible order in which action types can be performed for each object type. By collecting all life courses the (the restriction on the) dynamics of the UoD can be derived.

The *object property model* deals with such static aspects of the application domain by modelling *properties* of object types. The properties of an object type form a so-called *state record* which reports the current state of an object. Usually the property is set during the birth of the associated object type. Properties are only useful if they can be retrieved via *retrieval* action types and updated by so-called *update* action types. Strictly speaking, properties thus can also be modelled via their initialization action type, retrieval action types and update action types. The object property model is introduced to provide a more simple description mechanism dealing with properties, obtained by consulting the *Property* column of the logbook.

3.2. Typing a logbook

In order to obtain structural information from a logbook, all instances that occur in the sentences (tuples) of the logbook have to be typed according to some *typing mechanism*. This typing mechanism is guided by the three analysis models that constitute the information grammar. The modelling process may be informally described as follows: Find a consistent typing mechanism, i.e. a typing mechanism satisfying:

1. Events with a similar action have (a) similar objects involved (b) in similar roles.
2. Similar objects have a similar course of life.
3. Similar objects have a similar state record.

Two instances of the logbook meta-model are said to be *similar* if they have the same type assigned. Two sets of instances are called similar if each instance in the first set has a similar instance in the second set and vice versa. Recognizing similar lives of objects is hard to formalize. In fact this remains the main task of the system analyst: ordering and comparing the course of life of objects. In any case, two objects are candidates for similar lives if they have a similar birth action.

Based on this intuition the modelling process can be formally stated as follows: find a set $\mathcal{T}$ of types and a typing function

$$\mathsf{Type} : \left\{ \begin{array}{c} \mathsf{Log(Involved)} \cup \\ \mathsf{Log(Action)} \cup \\ \mathsf{Log(Role)} \cup \\ \mathsf{Log(Property)} \end{array} \right\} \rightarrow \mathcal{T}$$

such that:

(1) Events e_1 and e_2 with a similar action have (a) similar objects involved (b) in similar roles:

$$\mathsf{IsSim}(\alpha(e_1), \alpha(e_2)) \Rightarrow \left\{ \begin{array}{c} \mathsf{IsSim}(\iota * \Delta(e_1), \iota * \Delta(e_2)) \\ \wedge \\ \mathsf{IsSim}(\rho * \Delta(e_1), \rho * \Delta(e_2)) \end{array} \right.$$

(2) Similar objects x and y have a similar course of life:

$$\mathsf{IsSim}(x, y) \Rightarrow \mathsf{SimLife}(x, y)$$

(3) Similar objects x and y have a similar state record:

$$\mathsf{IsSim}(x, y) \Rightarrow \mathsf{IsSim}(\mathsf{Log}_x(\mathsf{Properties}), \mathsf{Log}_y(\mathsf{Properties}))$$

The similarity predicate is defined on instances x and y as:

$$\mathsf{IsSim}(x, y) \Leftrightarrow \mathsf{Type}(x) = \mathsf{Type}(y)$$

whereas on sets X and Y of instances:

$$\mathsf{IsSim}(X, Y) \Leftrightarrow \forall_{x \in X} \exists_{y \in Y} \left[\mathsf{IsSim}(x, y)\right] \wedge \forall_{y \in Y} \exists_{x \in X} \left[\mathsf{IsSim}(y, x)\right]$$

Note that this definition does not require similar sets of instances to have equal cardinality. Similarity of life of objects is defined as:

$$\mathsf{SimLife}(x, y) \Leftrightarrow \mathsf{IsSim}(\alpha(\mathsf{Init}(x)), \alpha(\mathsf{Init}(y)))$$

The similarity predicate forms an equivalence relation over the instances of the logbook.

This typing mechanism is very strict. The typing mechanism can be enhanced, using the *type relatedness* predicate (see e.g. [22]), which extends the typing mechanism with a subtyping mechanism (specialization and generalization). For example, in some UoD regarding music companies, different object types, such as *Person* and *Musician*, can share common instances. A *Musician* might be modelled in this UoD as a specialization of a *Person*. In this UoD *Person* and *Musician* are type-related but not of similar type. Introduction of a subtyping mechanism requires a refinement of the typing mechanism. For example, similar objects within the same subtype hierarchy need not have a similar state record, e.g. object type *Musician* can have properties which are not relevant for object type *Person*.

EXAMPLE 2. Suppose the typing mechanism contains the following:

$$\begin{aligned} \mathsf{Type}(\text{Mick Jagger}) &= \text{Person} \\ \mathsf{Type}(\text{The Rolling Stones}) &= \text{Person} \end{aligned}$$

Then we have no consistent typing, as Mick Jagger and The Rolling Stones do not have a similar life.

EXAMPLE 3. Suppose the typing mechanism contains the following:

$$\begin{aligned} \mathsf{Type}(\text{Mick Jagger}) &= \text{Person 1} \\ \mathsf{Type}(\text{Keith Richards}) &= \text{Person 2} \end{aligned}$$

This typing is not plausible, as Mick Jagger and Keith Richards occur in events with similar actions in similar roles.

4. CONCLUSIONS

In this paper the notion of logbook was introduced as a common basis for various information system models. A logbook is intended as a structuring mechanism for initial specifications, which is obtained via a unifying format containing a complete description of the history of some UoD. A typing mechanism is provided as an abstraction mechanism for logbook instances, leading to object-oriented analysis models.

Further research may address the completeness of this approach (for more information, see [23]). For example, does the logbook contain all information relevant for the modelling process? Another direction could be to make a prototype to support the approach of this paper.

ACKNOWLEDGEMENTS

We wish to thank Caspar Derksen and Kees Koster for their interest and advice. We also wish to thank the anonymous referees for their valuable comments which led to an improvement of the paper.

REFERENCES

[1] van Griethuysen, J. J. (ed.) (1982) *Concepts and Terminology for the Conceptual Schema and the Information Base.* Publ. nr. ISO/TC97/SC5-N695, Addison-Wesley, Reading, MA.

[2] Halpin, T. A. (1995) *Conceptual Schema and Relational Database Design*, 2nd edn. Prentice-Hall, Sydney, Australia.

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

[4] Kristen, G. · (1994) *Object Orientation, the KISS Method: From Information Architecture to Information System.* Addison-Wesley, Reading, MA.

[5] Date, C. J. (1991) *An Introduction to Data Base Systems*, Vol 1, 5th edn. Addison-Wesley, Reading, MA.

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

[7] Proper, H. A. and van der Weide, Th. P. (1995) A General Theory for the Evolution of Application Models. *IEEE Trans. Knowl. Data Engng*, **7**, 984–996.

[8] Tresch, M. T. (1991) A Framework for Schema Evolution by Meta Object Manipulation. In *Proceedings of the 3d International Workshop on Foundations of Models and Languages for Data and Objects*, Aigen, Austria, September. Institut für Informatik, TU Clausthal.

[9] Nguyen, G. T. and Rieu, D. (1989) Schema evolution in object-oriented database systems. *Data Knowl. Engng*, **4**, 43–67.

[10] ter Hofstede, A. H. M. and van der Weide, Th. P. (1993) Expressiveness in conceptual data modelling. *Data Knowl. Engng*, **10**, 65–100.

[11] Redmond-Pyle, D. (1996) Software development methods and tools: some trends and issues. *Software Engng J.*, **11**, 99–103.

[12] Frederiks, P. J. M., Koster, C. H. A. and van der Weide, Th. P. (1995) *Object-Oriented Analysis using Informal Language.* Technical Report CSI-R9516, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands.

[13] Frederiks, P. J. M. and van der Weide, Th. P. (1996) *Cognitive Requirements for Natural Language Based Conceptual Modelling.* Technical Report CSI-R9610, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands.

[14] ter Hofstede, A. H. M. (1993) *Information Modelling in Data Intensive Domains.* Ph.D. Thesis, University of Nijmegen, Nijmegen, The Netherlands.

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

[16] Graham, I. (1994) *Object-Oriented Methods.* Addison-Wesley, Reading, MA.

[17] Weber, R. and Zhang, Y. (1996) An analytical evaluation of NIAM's grammar for conceptual schema diagrams. *Info. Syst. J.*, **6**, 147–170.

[18] ter Hofstede, A. H. M., Proper, H. A. and van der Weide, Th. P. (1994) *Grammar Based Information Modelling.* Technical Report CSI-R9414, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands.

[19] van der Lek, H. (1993) On the Structure of an Information Grammar. In Nijssen, G. M. (ed.), *Proceedings of NIAM-ISDM.* NIAM-GUIDE, September.

[20] Frederiks, P. J. M. and van der Weide, Th. P. (1996) *From a File-Oriented View to an Object-Oriented View.* Technical Report CSI-R9601, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands.

[21] Derksen, C. F., Frederiks, P. J. M. and van der Weide, Th. P. (1996) Paraphrasing as a Technique to Support Object-Oriented Analysis. In van de Riet, R. P., Burg, J. F. M. and van der Vos, A. J. (eds), *Proceedings of the Second Workshop on Applications of Natural Language to Databases (NLDB'96)*, pp. 28–39, Amsterdam, The Netherlands, June.

[22] Frederiks, P. J. M. and van der Weide, Th. P. (1996) *Formalization, Integration and Validation of Object-Oriented Analysis Models leading to an Information Grammar.* Technical Report CSI-R9625, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands.

[23] Frederiks, P. J. M. (1997) *Object-Oriented Modelling based on Information Grammars.* Ph.D. Thesis, University of Nijmegen, Nijmegen, The Netherlands.