On the design and implementation of information systems from deductive conceptual models

Antoni Olivé
Facultat d'Informàtica. Universitat Politècnica de Catalunya
Pau Gargallo, 5. 08028 Barcelona. Catalonia

ABSTRACT

Deductive conceptual models (DCMs) aim at providing a complete specification of information systems, expressing only its logic component. It has been shown that DCMs have some advantages with respect to traditional, operational conceptual models, but they are more difficult to implement. We present a new approach to the design and implementation from a DCM. It consists in deriving from the DCM a new model, which we call the internal events model. This model describes the actions the information system must perform in terms of the inputs. The use of this model for data base and transactions design is discussed.

1. INTRODUCTION

During the last years we have seen the emergence and rapid growth of the logic programming field. This is due to a number of reasons, perhaps one of the most important being the clear distinction found in logic programming between the logic and control components of a program. As Kowalski says, "Logic programs express only the logic component of a program. The control component is exercised by the program executor, either following its own autonomously determined control decisions or else following control instructions provided by the programmer" [7].

There is a parallel trend at the information systems level in the conceptual modelling field [1]. Conceptual models of information systems express more and more the logic component of the system, but unfortunately they still somehow define (part of) the control component. The field is still dominated, both in theory and practice, by the "operational" approach to conceptual modelling. By this we mean models in which:

a) Changes in the Information Base (IB), corresponding to changes in the Universe of Discourse (UoD), are defined by means of **operations**. Occurrence of a real-world, external event triggers the execution of an operation (transaction), which reflects the effects of the event on the IB. These effects are usually expressed by some kind of insertions, updates or deletions to the IB.

b) Operations, as well as queries and integrity constraints, can only access to the current state of the IB.

Deductive Conceptual Models (DCMs) aim at providing a complete specification of an information system expressing only its logic component. The specification of the control component is entirely left to the subsequent phases of information systems development. A DCM relates the contents of the IB to the external events in terms of **deduction rules**. On the other hand, they allow to define integrity constraints and queries as if a complete history of the IB were available.

A detailed comparison of the operational and deductive approaches can be found in [3,13]. The main conclusions are that DCMs provide more local definitions, favour the understanding of informations, are easier to change and to accomodate new requirements, and provide more design freedom. However, DCMs are much more difficult to implement than operational models. The reason is that an operational model already embeddes some architectural design decisions, which are not made in DCMs.

It is clear that progress in deductive conceptual modelling is hampered by the lack of efficient implementation methods. We can expect that progress in the fields of knowledge bases and deductive data bases will make implementation of DCMs easier. However, an ideal solution would be to have methods for the design and implementation of DCMs in conventional hardware/software environments.

This paper is a step in that direction. We present a formal method to derive from a DCM a new model, called the internal events model, which is much easier to implement. The paper is structured as follows. In Section 2 we present the DCMs, including an example used throughout the paper. In Section 3 we review some approaches to design from DCMs. Section 4 presents a method for the derivation of the internal events model, which is the main part of the paper. The use of this model for the design and implementation is briefly discussed in Section 5. Finally, Section 6 gives the conclusions and points out future work.

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

2. DEDUCTIVE CONCEPTUAL MODELS

There are several languages for deductive conceptual modelling of information systems. Among them are CIAM [5], DADES [12] and IPL [16]. These languages differ in many aspects, but all share a common approach. Incidentally, this approach is very similar to the one taken by the recent Event Calculus [8] for representing and reasoning about time and events within a logic programming framework. In this paper, we will abstract from the details of specific languages and try to characterize the deductive conceptual modelling approach in a first order logic framework.

Time plays a major role in this approach. Every possible information i about the UoD is associated with a time point T(i), which states when the information holds in the UoD. It can be called the occurrence or observation time. This time point is a component of the information proper, so informations are self-describing in this respect. We will assume that occurrence times are always expressed in a unique time unit (such as second, day, etc.) small enough to avoid ambiguities.

By life span T of an information system we mean the time interval in which the system operates. It is defined as an ordered set of consecutive time points $T = \{t_0,...,t_f\}$, where t_0 and t_f are the initial and final times, respectively, and where each $t \in T$ is expressed in the given time unit. We can then say that, for any information i, $T(i) \in T$.

A DCM consists of a set B of base predicates, a set D of derived predicates, a set DR of deduction rules, a set IC of integrity constraints, and a set OR of output requirements. Each of them is described in the following, and illustrated with references to the DCM example of figure 1.

2.1 Base predicates

Base predicates model the external event types of the UoD. They are the inputs to the information system. Each fact of a base predicate, called base fact, corresponds to an occurrence of an external event. We assume, by convention, that the last term of a base fact gives the time when the external event occurred. If $p(a_1,...,a_n,t_i)$ is a fact we say that $p(a_1,...,a_n)$ is true or holds at t_i.

In the example of figure 1 we have three base predicates: start, end and assign. A base fact start(p1,e1,d1,t1) means that project p1 has started at time t1, is planned for termination at time e1, and corresponds to department d1. A base fact end(p1,t1) reports that project p1 effectively ends at time t1. A base fact assign(p1,pg1,t1) means that at time t1, programmer pg1 is assigned to project p1.

Base predicates

start(project,end,department,time)
end(project,time)
assign(project,programmer,time)

Derived predicates

DR.1 active(p,t) ← start(p,e,d,t1) ∧ t1 ≤ t ∧ ¬ completed(p,t)
DR.2 p-end(p,e,t) ← start(p,e,d,t1) ∧ t1 ≤ t ∧ active(p,t)
DR.3 dpt(p,d,t) ← start(p,e,d,t1) ∧ t1 ≤ t ∧ active(p,t)
DR.4 completed(p,t) ← end(p,t1) ∧ t1 ≤ t
DR.5 assigned(p,pg,t) ← assign(p,pg,t1) ∧ t1 ≤ t ∧ active(p,t)
DR.6 has-worked(d,pg,t) ← starts-working(d,pg,t1) ∧ t1 ≤ t
DR.7 starts-working(d,pg,t) ← assign(p,pg,t) ∧ dpt(p,d,t)
DR.8 new-in-dept(d,pg,t) ← starts-working(d,pg,t) ∧ ¬has-worked(d,pg,t-1)
DR.9 critical(p,t) ← p-end(p,e,t) ∧ t > e - 8 ∧ t ≤ e

Integrity constraints

IC.1 ← start(p,e,d,t) ∧ e ≤ t
IC.2 ← start(p,e,d,t) ∧ active(p,t1) ∧ t1 < t
IC.3 ← end(p,t) ∧ ¬active(p,t-1)
IC.4 ← assign(p,pg,t) ∧ assigned(p,pg,t-1)
IC.5 ← assign(p,pg,t) ∧ ¬active(p,t-1)

Output requirements

OR.1 O1(t) ← Q1(t)
O1(t) = {<p,e> | critical(p,t) ∧ p-end(p,e,t)}
OR.2 O2(t1) ← Q2(t1,t) ∧ t1 ≤ t
O2(t1) = {<p,d> | dpt(p,d,t1)}
OR.3 O3(pg,t) ← new-in-dept(d,pg,t)
O3(pg,t) = {<pg,p> | assigned(p,pg,t)}

Figure 1. Example of Deductive Conceptual Model

2.2 Derived predicates

Derived predicates model the relevant types of knowledge about the UoD. Each fact of a derived predicate, called derived fact, represents an information about the state of the UoD, at a particular time point. We will also assume that the last term of a derived fact gives the time when the information holds. In the example there are nine derived predicates: active, p-end, dpt, completed, assigned, has-worked, starts-working, new-in-dept and critical. The semantics of a derived predicate is given by its deduction rules.

2.3 Deduction rules

There are one or more deduction rules for each derived predicate. Let p(x1,...,xn,t) be a derived predicate, with n+1 terms. A deduction rule for p has the form $p(x1,...,xn,t) \leftarrow \phi$ where ϕ is a literal or a conjunction of (positive or negative) literals, and all variables are assumed to be universally quantified.

The terms in the rule head must be distinct variables. The terms in ϕ (rule body) must be constants or variables. We assume every rule to be range-restricted, i.e. every variable occurring in the head, or in a negative literal in ϕ, occurs in a positive literal in ϕ as well [4]. We also assume every rule to be time-restricted. This means that, for every base or derived predicate q occurring in the body as a positive literal q(...,t1), the condition $\phi \rightarrow t1 \leq t$ must hold. This condition ensures that p(x1,...,xn,t) is defined in terms of q-facts holding at time t or before.

In the example, the nine deduction rules are (hopefully) self-explanatory. Note the use of variable t-1 as a syntactic convention instead of t' and t' = t - 1. Thus, in DR.8, new-in-dept(d,pg,t) holds if programmer pg starts working for department d at time t, and pg had not worked for d at time t - 1.

2.4 Integrity constraints

Integrity constraints are closed formulae that base and/or derived facts must satisfy to be consistent. An integrity constraint can only be falsified by the presence (or absence) of new base facts, and it is assumed that some mechanism of integrity constraints enforcement will reject (or require) those facts to maintain the informations consistent. We deal here (as in [9]) with constraints that have the form of a denial $\leftarrow L_1 \wedge ... \wedge L_n$ where the L_i are literals, and variables are assumed to be quantified over the whole formula.

In the example, there are five integrity constraints. If a project starts at t, its planned end time must be greater than t (IC.1), and it can not have been active before (IC.2). IC. 3 requires a project to be active at time t-1 if it ends at t. Programmer pg can be assigned to a project p at time t if pg was not assigned to p at t-1 (IC.4) and project p was active at t-1 (IC.5).

2.5 Output requirements

An output requirement defines an output that must be produced by the information system. It has the form:

Output ← Condition
Output = Contents

meaning that whenever the condition is satisfied, the system must produce an output with the defined contents.

In its simplest form, the condition is a query Qi with some parameters. In the example, OR.1 and OR.2 are of this form. The meaning is:
- When query Q1 is issued at time t, the system must provide the set of projects which are critical at t, and their planned end times.
- When query Q2 is issued at time t, with parameter t1, the system must provide the projects that were active at time t1, and their corresponding departments. Note that this query requires the system to know the complete history of predicate dpt.

In its most general form, the condition is any base or derived predicate (in fact, we regard queries as base predicates although, for convenience, we do not require to define them explicitly as such). An example is predicate new-in-dept(d,pg,t) in OR.3. The meaning is that as soon as a new fact of new-in-dept is known, the output must be produced. Thus, every time that a programmer pg is new in a department, the system must give the set of projects to which pg is assigned at t.

2.6 DCM vs. deductive data bases

As can be observed, there is a strong similarity in form between a DCM and a deductive data base. However, there are some fundamental differences between both, and it may be worthwhile to comment them briefly before closing this section.

A deductive data base consists of a set of base facts, a set of deduction rules and a set of integrity constraints [4]. The base facts are explictly stored in the data base, the deduction rules are used to augment the knowledge of the data base, and the integrity constraints are used to control its consistency.

A DCM does not include any fact. It is a model of a future information system, independent of any particular fact. On the other hand, a DCM does not assume any particular data base contents. It is left to the designer to decide which facts will be explictly stored and which ones will be deduced, and when and how will this deduction take place. Furthermore, a DCM includes a list of output requirements that the system must satisfy. This list (completed with quantitative data such as frequence, volume and response times) is the basis from which the designer must develop an efficient implementation. Finally, a DCM adopts a temporal view of informations, instead of the "single state" view of most current deductive data bases.

3. DESIGNING FROM A DCM

In this Section we first present the main decisions a designer must make when designing an information system from a DCM, then review some relevant work in this context, and finally outline our contribution.

3.1 Design decisions

The two main design decisions are: Data base design and Transactions design. In turn, data base design consists of data base contents design and data base schema design. In data base contents design, the designer must decide which predicates of the Information Base will be explicitly stored in the data base and, for each of them, the time interval for which the informations will be stored. There are, of course, many valid alternatives in this decision, and they must be evaluated with respect to the design objectives (cost, time, etc.).

In data base schema design, the designer must decide how to represent the data base contents in the data model used. Several options can also be available. If, for example, we use the relational data model, we can group two or more predicates in a single relation scheme. Full time history of a predicate can be represented by means of "events" data bases [6] or by associating a "time start" and "time end" components to each tuple [10]. When only the current state is stored we can eliminate the time component of the facts, making it implicit.

In transactions design, the designer must decide which transactions will exist and, for each of them, when will be executed, its pre-conditions and the actions to be performed, including data base updates and output production.

3.2 Previous work

The CIAM methodology [5] discusses in detail the design decisions required once the DCM is defined, but it does not provide a formal procedure for making them. CIAM suggests two phases: Conceptual information processing system (CIPS) design and DBMS adaptation. During CIPS design, the designer defines two models: A Conceptual data base model (CDBM), and a Conceptual processing model (CPM). The former describes the information which will be stored in the data base, while the latter decribes the processes for the maintanance of the data described in the CDBM. The CIPS level is independent of the DBMS that will be used. Adaptation to the chosen DBMS is done in the second phase.

The DADES methodology [12] also discusses the design decisions, and emphasizes the verification aspect, but again no precise method is given. It suggests an initial transaction structure, which is then refined to improve efficiency.

The DADES/GP prototype generator system [15] generates prototypes of information systems from DCMs written in the DADES language, with a minimum of designer intervention. DADES/GP stores in the data base the full history of all base and derived predicates. When a new fact is received all derived facts implied by it are generated and stored. Outputs are derived directly from the data base. The prototypes are thus a convenient means for validation purposes, but inefficient for the final system.

The same applies to the implementation discussed by Weigand [17]. He uses DCMs written directly in PROLOG, which are then interpreted. New base facts are added to the facts base. Derived facts are produced only when requested in some output, and never stored. Output response times can thus be unacceptable for a final system.

Some work from the deductive data bases field is also relevant here. Nicolas and Yazdanian [11] discuss the use of deduction rules for the generation of derived facts when a new fact is inserted in the data base. Their strategy is not directly applicable to a DCM, mainly because a "single state" view of the data base is taken. If that strategy were to be used in our example, this would imply that facts of predicates like active and has-worked would be generated for each instant of the life span. They cannot recognize that a project remains active until terminated and, thus, it is not necessary to generate and store the state of a project in all time points. We will see, however, that their strategy can be used once the DCM is transformed.

3.3 Our work

Our work is based on the idea of internal events. We define an internal event as a change in the extension of a predicate, and thus it corresponds to an update of the Information Base. An internal event rule is a rule that defines when an internal event happens. Internal events rules can be derived formally from the DCM. The set of internal events rules corresponding to a DCM is the internal events model. The internal events model allows us to know which internal events are implied by a given occurrence of an external event (base fact). We can then use the model as a basis for data base and transaction design. Both uses are discussed in section 5.

4. INTERNAL EVENTS MODEL

4.1 Classification of derived predicates

In trying to develop a method for the design and implementation of an information system from a DCM, we have observed that not all derived predicates should be treated in the same way. There are fundamental differences among several kinds of derived predicates, and an effective method should be able to recognize them. For this reason, we have built a classification of derived predicates that captures those differences. It consists of two dimensions, that we call the Positive and Negative dimensions. In the former, we distinguish among five types, and among two types in the latter. Each derived predicate has a type in the Positive dimension (P-type) and a type in the Negative one (N-type).

We may infer from the deduction rules to which P- and N-type a given predicate belongs to. However, we believe that our classification is so natural, and so intrinsically related to the intended meaning of a predicate, that we would be in favour of requiring the designer to explicitly define these types. Once the model is complete, we can then formally deduce them. In case of disagreement, either the deduction rules are incorrect or the perception the designer has of the predicate meaning is wrong.

Positive dimension

Let p be a derived predicate, x a vector of variables, c a vector of constants, and p(c) any fact that is true at time t-1. What can we say about the truth value of p(c) at time t?. Three cases are possible:

a) p(c) will be true at t. We say that p is P-steady.
b) p(c) will be false at t. We say that p is P-momentary.
c) p(c) may be true or false at t.

In this third case, assume now that no external events happen at time t. We find then three subcases:

c1) p(c) will be true at t. We say that p is P-state.
c2) p(c) will be false at t. We say that p is P-transient.
c3) p(c) will be true or false, depending on the truth value of a predicate ϕ(c,t). We say that p is P-spontaneous.

In what follows we define each case formally, and give some examples.

P-steady A predicate p is P-steady iff:
(4.1) $\forall x,t\ [p(x,t\text{-}1) \rightarrow p(x,t)]$
That is, if p(x) is true at time t-1, then it is also true at time t, independently of the external events that have occurred at t, and of any internal condition. In the example of figure 1, predicates completed and has-worked are of this type. If a project is completed at a given time, then it remains completed at any later time.

P-momentary A predicate p is P-momentary iff:
(4.2) $\forall x,t\ [p(x,t\text{-}1) \rightarrow \neg p(x,t)]$
In the example, new-in-dept is of this type. In a programmer is new in a department at time t-1, he cannot be new again in the same department at t.

P-state A predicate p is P-state iff:
(4.3) $\forall x,t\ [p(x,t\text{-}1) \wedge \text{noextev}(t) \rightarrow p(x,t)]$
where noextev(t) means that no external events have occurred at time t. Formally, if $p_1(y_1,t),...,p_n(y_n,t)$ are the base predicates:
$\forall t\ [\text{noextev}(t) \leftrightarrow \neg(\exists y_1 p_1(y_1,t) \vee ..... \vee \exists y_n p_n(y_n,t))]$

In the example, predicates active, p-end, dpt and assigned are of this type.

P-transient A predicate p is P-transient iff:
(4.4) $\forall x,t\ [p(x,t\text{-}1) \wedge \text{noextev}(t) \rightarrow \neg p(x,t)]$
In the example, predicate starts-working is of this type. If a programmer starts working for a department at time t-1, and is not assigned to any project at t, he will not "start working for" any department at t.

P-spontaneous A predicate p is P-spontaneous iff:
(4.5) $\forall x,t\ [p(x,t\text{-}1) \wedge \text{noextev}(t) \wedge \phi(x,t) \rightarrow p(x,t)]$
and $\forall x,t\ [p(x,t\text{-}1) \wedge \text{noextev}(t) \wedge \neg\phi(x,t) \rightarrow \neg p(x,t)]$
where ϕ(x,t) is distinct from the true predicate. We call ϕ(x,t) the persistence condition. The meaning is: If p(x) holds at t-1 and there are not external events at t, and the persistence condition is true, then p(x) holds at t (the fact persists). If the persistence condition is false, then p(x) does not hold at t. In the example, predicate critical is of this type.

Negative dimension

Let p be a derived predicate and let p(c) be any fact that is false at time t-1. Assume that no external events happen at t. What can we say about the truth value of p(c) at t?. Two cases are possible:

a) p(c) will remain false at t. We say that p is N-state.
b) p(c) will be true or false at t, depending on the truth value of a predicate ϕ'(c,t). We say that p is N-spontaneous.

In the Negative dimension we do not consider the types equivalent to P-steady, P-momentary and P-transient, because they do not seem to be of any practical interest. In what follows, we define both cases formally, and give some examples.

N-state A predicate p is N-state iff:
(4.6) $\forall x,t\ [\neg p(x,t\text{-}1) \wedge \text{noextev}(t) \rightarrow \neg p(x,t)]$
In the example, all derived predicates, except critical, are of this type.

N-spontaneous A predicate p is N-spontaneous iff:
(4.7) $\forall x,t\ [\neg p(x,t\text{-}1) \wedge \text{noextev}(t) \wedge \phi'(x,t) \rightarrow p(x,t)]$
and $\forall x,t\ [\neg p(x,t\text{-}1) \wedge \text{noextev}(t) \wedge \neg\phi'(x,t) \rightarrow \neg p(x,t)]$
where ϕ'(x,t) is distinct from true. We call ϕ'(x,t) the creation condition. In the example, predicate critical is of this type.

4.2 Internal events

Internal events play a critical role in our approach. As we will see in Section 5, we use internal events as triggers for data base updates, output production and process execution. There are two classes of internal events: insertion and deletion. To each base or derived predicate p corresponds an insertion internal event predicate ιp, with the same number of terms, and to each P-state or P-spontaneous predicate q corresponds a deletion internal event δq, with the same number of terms.

Insertion events are defined as follows. If p is a P-steady, P-state or P-spontaneous predicate then:

(4.8) $\forall x,t\ [\iota p(x,t) \leftrightarrow p(x,t) \wedge \neg p(x,t-1)]$

That is, event $\iota p(c)$ occurs at t if $p(c)$ is true at t and it was false at time t-1. If p is a P-momentary, P-transient or base predicate:

(4.9) $\forall x,t\ [\iota p(x,t) \leftrightarrow p(x,t)]$

That is, event $\iota p(c)$ occurs at t if $p(c)$ is true at t. Note that we also associate an internal event to each base fact (external event).

Similarly, deletion events are defined as follows. If p is a P-state or P-spontaneous predicate:

(4.10) $\forall x,t\ [\delta p(x,t) \leftrightarrow p(x,t-1) \wedge \neg p(x,t)]$

Note that we do not define deletion events for P-steady, P-momentary, P-transient and base predicates. Deletion events for P-steady predicates would never happen, because if $p(c)$ holds at t-1 it also holds at any $t' > t$. We are not interested in deletion events of P-momentary, P-transient and base predicates, because we require an insertion event for every fact.

Rules 4.8, 4.9 and 4.10 above are called internal events deduction rules. They allow us to deduce at any time which internal events occur at that time. Thus, for example, applying rule 4.8 to predicate active we have:

(4.11) $\forall p,t\ [\iota active(p,t) \leftrightarrow active(p,t) \wedge \neg active(p,t-1)]$

Evaluating this rule at time t (and using DR.1) we get the internal events $\iota active(c)$ that occur at t. However, it is obvious that this evaluation would be greatly inefficient, and that there are alternative rules much more efficient. In this case, for example, the rule:

(4.12) $\forall p,t\ [\iota active(p,t) \leftrightarrow \iota start(p,e,d,t)]$

is equivalent to 4.11 and quite easy to evaluate. Rule 4.12 says that an occurrence of event $\iota start$ at time t produces an occurrence of $\iota active$ at t. The rule can be obtained from 4.11 and DR.1 by taking into account that active is P-state and N-state, and the deduction rules and integrity constraints of the DCM. This is explained in the next sections.

4.3 Transformation of deduction rules

We first transform the deduction rules of the DCM into a set of equivalent ones. Let $p(x,t) \leftarrow B(x,y,t)$ be a deduction rule, where $B(x,y,t)$ is a conjunction of one or more literals. The idea is to replace each literal in B, corresponding to a base or derived predicate, and whose time variable may range in the rule over a set including t, by an equivalent expression containing internal event predicates, whose time variables range only over {t}, and base or derived predicates, whose time variables range over a set not including t.

Let L1,...,Ln be the literals in B corresponding to base or derived predicates and whose time variable may range in the rule over a set including t. The transformation we apply to each Li (i = 1,...,n) is:

If $Li = q(z,t)$ or $Li = \neg q(z,t)$

If q is P-momentary, P-transient or base predicate

$q(z,t) \equiv \iota q(z,t)$

$\neg q(z,t) \equiv \neg \iota q(z,t)$

If q is P-steady

$q(z,t) \equiv q(z,t-1) \vee \iota q(z,t)$

$\neg q(z,t) \equiv \neg q(z,t-1) \wedge \neg \iota q(z,t)$

If q is P-state or P-spontaneous

$q(z,t) \equiv (q(z,t-1) \wedge \neg \delta q(z,t)) \vee \iota q(z,t)$

$\neg q(z,t) \equiv (\neg q(z,t-1) \wedge \neg \iota q(z,t)) \vee \delta q(z,t))$

If $Li = q(z,t1)$ or $Li = \neg q(z,t1)$

with t1 distinct from t and ranging over a set including t:

If q is P-momentary, P-transient or base predicate

$q(z,t1) \equiv [t1 < t \wedge q(z,t1)] \vee [t1 = t \wedge \iota q(z,t)]$

$\neg q(z,t1) \equiv [t1 < t \wedge \neg q(z,t1)] \vee [t1 = t \wedge \neg \iota q(z,t)]$

If q is P-steady

$q(z,t1) \equiv [t1 < t \wedge q(z,t1)] \vee [t1 = t \wedge \iota q(z,t)]$

$\neg q(z,t1) \equiv [t1 < t \wedge \neg q(z,t1)] \vee$
$[t1 = t \wedge \neg q(z,t-1) \wedge \neg \iota q(z,t)]$

If q is P-state or P-spontaneous

$q(z,t1) \equiv [t1 < t \wedge q(z,t1)] \vee$
$[t1 = t \wedge q(z,t-1) \wedge \neg \delta q(z,t)] \vee$
$[t1 = t \wedge \iota q(z,t)]$

$\neg q(z,t1) \equiv [t1 < t \wedge \neg q(z,t1)] \vee$
$[t1 = t \wedge \neg q(z,t-1) \wedge \neg \iota q(z,t)] \vee$
$[t1 = t \wedge \delta q(z,t)]$

As an example, consider the application of the above rules to DR.1 (where the time variable of start ranges over the set $\{t_0,...,t\}$ and the time variable of completed ranges over {t}):

$active(p,t) \leftarrow ([start(p,e,d,t1) \wedge t1 < t] \vee$
$[\iota start(p,e,d,t) \wedge t1 = t]) \wedge t1 \leq t$
$\wedge \neg completed(p,t-1) \wedge \neg \iota completed(p,t)$

which is equivalent to:

$active(p,t) \leftarrow start(p,e,d,t1) \wedge t1 < t \wedge \neg completed(p,t-1) \wedge$
$\neg \iota completed(p,t)$

$active(p,t) \leftarrow \iota start(p,e,d,t) \wedge$
$\neg completed(p,t-1) \wedge \neg \iota completed(p,t)$

Similarly, the application to DR.4 gives:

$completed(p,t) \leftarrow end(p,t1) \wedge t1 < t$

$completed(p,t) \leftarrow \iota end(p,t)$

4.4 Normalized internal events deduction rules

Once we have transformed the deduction rules, we easily derive the normalized internal events deduction rules, which form the internal events model. These rules, we believe, are a useful basis from which the designer can develop an efficient implementation. The normalized internal events deduction rules are the same as rules 4.8, 4.9 and 4.10 (in Section 4.2), but replacing literal p(x,t) by the transformed body of the corresponding deduction rule. Thus, if for a predicate p we have n transformed deduction rules of the form:

$p(x,t) \leftrightarrow \phi_i(x,y_i,t) \quad i = 1,...,n$

and we use rule 4.8, replacing p(x,t) yields:

$\iota p(x,t) \leftrightarrow [\phi_1(x,y_1,t) \vee ... \vee \phi_n(x,y_n,t)] \wedge \neg p(x,t\text{-}1)$

which is equivalent to the rules:

$\iota p(x,t) \leftarrow \phi_i(x,y_i,t) \wedge \neg p(x,t\text{-}1) \quad i = 1,...,n$

As an example, using rules 4.8 and 4.10 for predicate active, we have:

(4.13) $\iota active(p,t) \leftarrow start(p,e,d,t1) \wedge t1 < t \wedge \neg completed(p,t\text{-}1) \wedge \neg \iota completed(p,t) \wedge \neg active(p,t\text{-}1)$

(4.14) $\iota active(p,t) \leftarrow \iota start(p,e,d,t) \wedge \neg completed(p,t\text{-}1) \wedge \neg \iota completed(p,t) \wedge \neg active(p,t\text{-}1)$

(4.15) $\delta active(p,t) \leftarrow active(p,t\text{-}1) \wedge \neg[start(p,e,d,t1) \wedge t1 < t \wedge \neg completed(p,t\text{-}1) \wedge \neg \iota completed(p,t)] \wedge \neg[\iota start(p,e,d,t) \wedge \neg completed(p,t\text{-}1) \wedge \neg \iota completed(p,t)]$

Similarly, using rule 4.8 for predicate completed, we get:

(4.16) $\iota completed(p,t) \leftarrow end(p,t1) \wedge t1 < t \wedge \neg completed(p,t\text{-}1)$

(4.17) $\iota completed(p,t) \leftarrow \iota end(p,t) \wedge \neg completed(p,t\text{-}1)$

4.5 Simplification of rules

These normalized rules can now be subjected to a set of simplifications, by combining them with the properties of the predicate types, (4.1) to (4.7), the deduction rules and the integrity constraints of the DCM.

We can only show here the potential of these simplifications, by means of an example. Let us take rules 4.16 and 4.17. Being completed a N-state predicate, we have, by 4.6:

$noextev(t) \rightarrow \neg \iota completed(p,t)$

and, if ιend is false no $\iota completed$ events can happen, which means that no p, t1, t can satisfy the body of rule 4.16. Therefore, we eliminate rule 4.16. On the other hand, we have by IC.3:

(4.18) $\iota end(p,t) \rightarrow active(p,t\text{-}1)$

and by DR.1: $active(p,t\text{-}1) \rightarrow \neg completed(p,t\text{-}1)$

and, thus, we can remove literal $\neg completed(p,t\text{-}1)$ from rule 4.17. So, rules 4.16 and 4.17 are equivalent to:

(4.17') $\iota completed(p,t) \leftarrow \iota end(p,t)$

We give in figure 2 the resulting set of rules after simplification.

IDR.1 $\iota active(p,t) \leftarrow \iota start(p,e,d,t)$
IDR.2 $\delta active(p,t) \leftarrow \iota completed(p,t)$
IDR.3 $\iota p\text{-}end(p,e,t) \leftarrow \iota start(p,e,d,t)$
IDR.4 $\delta p\text{-}end(p,e,t) \leftarrow p\text{-}end(p,e,t\text{-}1) \wedge \delta active(p,t)$
IDR.5 $\iota dpt(p,d,t) \leftarrow \iota start(p,e,d,t)$
IDR.6 $\delta dpt(p,d,t) \leftarrow dpt(p,d,t\text{-}1) \wedge \delta active(p,t)$
IDR.7 $\iota completed(p,t) \leftarrow \iota end(p,t)$
IDR.8 $\iota assigned(p,pg,t) \leftarrow \iota assign(p,pg,t) \wedge \neg \delta active(p,t)$
IDR.9 $\delta assigned(p,pg,t) \leftarrow assigned(p,pg,t\text{-}1) \wedge \delta active(p,t)$
IDR.10 $\iota has\text{-}worked(d,pg,t) \leftarrow \iota starts\text{-}working(d,pg,t) \wedge \neg has\text{-}worked(d,pg,t\text{-}1)$
IDR.11 $\iota starts\text{-}working(d,pg,t) \leftarrow \iota assign(p,pg,t) \wedge dpt(p,d,t\text{-}1) \wedge \neg \delta dpt(p,d,t)$
IDR.12 $\iota new\text{-}in\text{-}dept(d,pg,t) \leftarrow \iota starts\text{-}working(d,pg,t) \wedge \neg has\text{-}worked(d,pg,t\text{-}1)$
IDR.13 $\iota critical(p,t) \leftarrow p\text{-}end(p,e,t\text{-}1) \wedge \neg \delta p\text{-}end(p,e,t) \wedge t\text{-}1 = e\text{-}8$
IDR.14 $\iota critical(p,t) \leftarrow \iota p\text{-}end(p,e,t) \wedge t > e\text{-}8$
IDR.15 $\delta critical(p,t) \leftarrow critical(p,t\text{-}1) \wedge \delta p\text{-}end(p,e,t)$
IDR.16 $\delta critical(p,t) \leftarrow critical(p,t\text{-}1) \wedge p\text{-}end(p,e,t\text{-}1) \wedge t = e+1$

Figure 2. Internal events model example

5. USE OF THE INTERNAL EVENTS MODEL

In this Section we outline a possible use of the internal events model for the design and implementation of information systems, and show its application to the example.

5.1 Data base design

An alternative of data base contents DB(t) is characterized by:

- A set of base or derived predicates of the DCM to be explicitly stored in the data base, and
- For each of them, a time interval for which its facts will be stored.

We will consider here only two extreme cases of time intervals: either current state or full history. If p(x,t) is a stored predicate, in the first case we store in the data base the set $\{<x,t> \mid p(x,t)\}$, and in the second case the set $\{<x,t1> \mid p(x,t1) \wedge t1 \leq t\}$. We

usually do not store current states of P-momentary, P-transient and base predicates, since this would imply to store their facts when produced (at time t), and to delete them immediately (also at time t).

A valid alternative of data base contents DB(t) is one that satisfies [12]:

1) Internal events that happen at t can be derived from DB(t-1) and base facts received at t. DB(t_0-1) is assumed to be empty.
2) Contents of the outputs to be produced at t can also be derived from DB(t-1) and base facts received at t.
Note that the alternative of storing the full history of only base predicates is always a valid one.

In our example, base facts are ιstart, ιend and ιassign. A valid alternative would then be:

DB(t) = {<p,d,t1> | dpt(p,d,t1) ∧ t1 ≤ t},
{<p,e,t> | p-end(p,e,t)},
{<p,pg,t> | assigned(p,pg,t)},
{<d,pg,t> | has-worked(d,pg,t)}

The database schema corresponding to a given DB(t) can also be designed systematically, although we do not have developed the method yet. If we decide to represent each stored predicate as an independent relation then the following rules could be used:

1) For a current state predicate p(x,t) define a relation scheme p(x).

2) For a full history predicate p(x,t):

If p is P-momentary, P-transient or base, define a relation scheme p(x,t).

If p is a P-steady predicate, define a relation scheme p(x,t_s), where attribute t_s will represent the first time p(x) is true.

If p is a P-state or P-spontaneous predicate, define a relation scheme p(x,t_s,t_E), where attributes t_s , t_E represent an interval in which p(x) holds.

In our example, a possible data base schema could be:

DEPARTMENT(PROJECT,DPT,T-START,T-END)
P-END(PROJECT,END)
ASSIGNED(PROJECT,PROGRAMMER)
HAS-WORKED(DEPT,PROGRAMMER)

5.2 Transaction design

Once the data base has been designed we can proceed to transaction design. Again, many alternatives exist, and several structuring mechanisms could be used. We only want to outline here a simple approach, that can work well in some applications, as an illustration of the use of the internal events model.

We first define **relevant internal events** as follows:

1) Insertion and deletion events corresponding to stored predicates are relevant internal events.
2) Insertion events corresponding to output requirements conditions are relevant internal events.
In the example, relevant internal events are: ιdpt, δdpt, ιp-end, δp-end, ιassigned, δassigned, ιhas-worked and ιnew-in-dept.

We then build a transaction type for each base fact type. The purpose of each transaction will be:

1) Read the corresponding base fact.
2) Verify the applicable integrity constraints.
3) Derive the relevant internal events produced by the base fact. This can be easily determined bottom-up from the internal events model. The generation strategy described in [11] can be used for this purpose.
4) For each internal event thus produced, if it corresponds to a stored predicate perform the appropiate data base operation; and if it corresponds to an output condition produce the defined output contents.

If some relevant internal event corresponds to a P- or N-spontaneous predicate, then we would also need a special transaction, to be executed at each time point of the life span.

Finally, we also build a transaction type for each query type. The purpose of these transactions will be:

1) Read the query parameters, if any, and
2) Produce the defined output contents.

As an example, consider transaction end(p,t). The actions to be performed are:

1) Read project identifier p (time t is implicit).
2) Verify integrity constraints (not considered here).
3) Delete tuple with PROJECT = p from P-END relation. (This is deduced from IDR.7, 2 and 4).
4) Read DEPARTMENT with PROJECT = p from the data base, and mark it as inactive: T-END = t (This is deduced from IDR.6).
5) Remove all tuples with PROJECT = p from ASSIGNED relation. (This is deduced from IDR.9).

6. CONCLUSIONS

We have first characterized deductive conceptual models (DCMs) in a first order logic framework. Then we have discussed the main design decisions involved in the design and implementation of an information system from a DCM. A new approach has been presented, which consists in deriving an internal events model from a DCM. This model is a useful basis from which several design alternatives can be systematically developed and evaluated. We have outlined a possible use of the internal events model in data base and transaction design, but other uses can be imagined as well. We also expect that the internal events model, and the classification of predicates we have proposed, can be useful in the field of deductive data bases, although this has not been elaborated in the paper.

Several research directions can be explored from the work reported here. First, the problem of integrity constraints enforcement should be taken into account. This has not been considered in the paper, but it is felt that work from deductive data bases can be easily integrated here [2]. Second, specific procedures for the simplification of normalized internal events deduction rules should be developed, perhaps integrated in an interactive man-machine system. Finally, one or more approaches to design and implementation from internal events models should be completely developed, and integrated in existing or new methodologies.

ACKNOWLEDGMENTS

I would like to thank D. Costal, J.A. Pastor, M.R. Sancho and J. Sistac for many useful comments on earlier drafts of this paper.

7. REFERENCES

[1] Brodie,M.;Mylopoulos,J.;Schmidt,J. "On conceptual modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages", Springer-Verlag, 1983.

[2] Bry,F.; Decker,H.; Manthey,R. "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases". Proc. 1st. Int. Conf. Extending Data Base Technology (EDBT), March 1988.

[3] Bubenko,J.; Olivé,A. "Dynamic or temporal modelling?. An illustrative comparison". SYSLAB Working Paper No. 117, University of Stockholm, 1986.

[4] Gallaire,H.; Minker,J.; Nicolas,J-M. "Logic and databases: A deductive approach". Computing Surveys, Vol. 16, No. 2, June 1984, pp. 153-185.

[5] Gustafsson,M.; Karlsson,T.; Bubenko,J. "A declarative approach to conceptual information modelling", In [14], pp. 93-142.

[6] Kobayashi,I. "Temporal aspects of databases: Interaction between state and event relations". IFIP WG 2.6 Working Conference DS-1, Belgium, Jan. 1985.

[7] Kowalski,R. A. "Logic for problem solving", North-Holland, 1979.

[8] Kowalski,R.A.; Sergot,M.J. "A logic based Calculus of Events", New Generation Computing, 4, No. 1, 1986, pp. 67-95.

[9] Kowalski,R.;Sadri,F.;Soper,P. "Integrity checking in deductive databases". Proc. 13th. VLDB, Sept. 1987, pp. 61-69.

[10] Navathe,S.B.; Ahmed,R. "TSQL- A language interface for history databases", in Proc. WG 8.1 Conf. on Temporal aspects in information systems, Sophia-Antipolis, France, May 1987, pp. 113-128.

[11] Nicolas, J-M.; Yazdanian, K. "An outline of BDGEN: A deductive DBMS", IFIP 1983, North-Holland, pp. 711-717.

[12] Olivé,A. "DADES: A methodology for specification and design of information systems", In [14], pp. 285-334.

[13] Olivé,A. "A comparison of the operational and deductive approaches to conceptual information systems modelling", Proc. IFIP-86, Dublin, pp. 91-96.

[14] Olle,T.W.; Sol,H.G.; Verrijn-Stuart,A.A. (Eds.) "Information systems design methodologies: A comparative review", North-Holland, 1982.

[15] Sistac,J.; Olivé,A. "DADES/GP: A prototype generator from deductive conceptual models of data base systems", Convention Informatique 1986, Paris.

[16] van Griethuysen, J.J. (Ed.)."Concepts and terminology for the conceptual schema and the information base", ISO/TC97/SC5/WG3, March 1982.

[17] Weigand,H. "Conceptual models in PROLOG", IFIP WG 2.6 Working Conference DS-1, Belgium, Jan. 1985.