A Method for Explaining the Behaviour of Conceptual Models

Antoni Olivé
Maria-Ribera Sancho

Facultat d'Informàtica, Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona - Catalonia
e-mail:{olive|ribera}@lsi.upc.es

Abstract. Traditional information modelling methods have been concerned with the important task of checking whether a model correctly and adequately describes a piece of reality and/or the users' intended requirements, that is, with model validation. In this paper, we present a new method for model validation which can be applied to conceptual models based on the concept of transaction. It provides explanations of the results of model execution. We extend the facilities of methods developed so far in this context by providing answers to questions about the value of derived information, to questions about how an information can be made true or false, and to hypothetical questions.

1 Introduction

This paper describes a new method for explaining the behaviour of conceptual models of information systems. The method aims at improving the validation of conceptual models. By validation we mean the process of checking whether a model correctly and adequately describes a piece of reality and/or the users' intended requirements [Gul93]. It is widely recognised that validating a conceptual model is an important task in Information Systems Engineering, and a broad variety of techniques and tools have been developed over recent years to support designers in that task. Among the support capabilities that are used (or have been investigated) there are [Bub88]: paraphrasing specifications in natural language [RoP92,Dal92], generation of abstractions and abstracts of specifications [JeC92], animation and symbolic execution [LaL93], explanation generation [Gul93,GuW93], infological simulation and semantic prototyping [LTP91,LiK93].

Our method contributes to model validation by providing explanations of the results of model execution. Specifically, we can explain, in several complementary ways, why some facts hold (or do not hold) in the information base (IB), why some facts have been inserted to (or deleted from) the IB, how some intended effect on the IB can be achieved, and what would have happened if some other input had been given (hypothetical explanation).

This paper extends our previous work on model execution explanations, which focused on deductive conceptual models. In [CoO92] we presented a method, based on plan generation techniques, which explains how some intended effect on the information base can be achieved. In [San93,San94] we described a method for explaining the temporal behaviour, through execution, of a model. Both methods were implemented.

We now present the result of extending our previous work to usual conceptual models of information systems, based on the concept of transaction rather than on deductive rules. The new method is limited by the fact that we do not consider the internal structure of transactions, while in our previous methods we know the full details of the effect of each external event (roughly, transactions) on the information base. Even so, we believe that the method provides helpful explanations in a rather simple and precise way.

For the sake of presentation, we describe the system in terms of four levels of explanations, from the simplest to the most complex level. For each level, we describe the explanations that can be obtained, the requirements for an explanation system able to give them, and the techniques and procedures that can be used. In Section 2 we deal with explaining the current contents of the information base. Section 3 deals with explaining the reasons for changes in a single transition of the information base. We will see that we need to record the transactions that are executed, and their effect. Section 4 moves a step forward and considers explanations taking into account the full history of changes. Finally, in Section 5 we describe the hypothetical explanations we can provide. The paper ends with the conclusions.

2 Explaining the Contents of the Information Base

The most elementary level of explanation in our method is that of explaining the current contents of the IB. At this level, we can only provide limited answers to the questions about why a fact is true and why a fact is false in the current IB state. In this section, we describe the requirements for a system to provide such explanations, and the procedures that may be used.

2.1 Requirements

Answering the above questions requires knowing the structure of the IB. This knowledge is, of course, available in all conceptual models. In general, we may assume that the IB consists of two parts: Base and Derived. The *Base* part includes all facts that are inserted, modified and deleted directly by the transactions, while the *Derived* part includes all facts that are derived from base and/or derived facts, by means of deduction rules.

Each conceptual modelling language provides a set of concepts and syntactic features to define the structure of the IB. Our method can be adapted to most languages. We will assume the IB contains facts of a given set of *fact types*. Each fact type consists of a name and a set of arguments. Base facts are updated by transactions, while derived facts are defined by *deduction rules* and, thus, their extension is defined declaratively. We will use the clausal form of logic, augmented with negation, to define deduction rules.

```
companies(company, name)
engaged_in(company, project) derived
   engaged_in(C,P) ← consortium(P,C), belongs_to(E,C), works_on(E,P)
employees(employee, name)
belongs_to(employee, company)
works_on(employee, project)
projects(project, name)
consortium(project, company)
active(project) derived
   active(P) ← projects(P,N), engaged_in(C,P)
inactive(project) derived
   inactive(P) ← projects(P,N), not active(P)
integrity constraints
ic1 ← works_on(E,P), not projects(P,N)
ic2 ← companies(C,N), not engaged_in(C,P)
```

Figure 1. Example of IB structure

We will also take into account, on the next levels of explanation, the integrity constraints on the IB. For the sake of uniformity, we define the constraints in denial form by means of integrity rules, which have the same form as the deduction rules.

Figure 1 is an example of the structure of an IB that we will use throughout this paper. The example has been adapted from [JMS92]. Note that we include two integrity rules. The first states that employees can only work in projects, while the second states that all companies must be engaged in some project.

It can be seen that there is a straightforward correspondence between our IB structure and that of ER [Che76], SDM [HaM81], NIAM [NH89] and many others. Note that not all of them include a derived part in the IB. Our method is even adaptable to languages based on the relational data model. In such case, views are derived fact types and their definition is a deduction rule.

2.2 Procedures

We now describe how answers to the questions given above can be obtained. Our approach to the problem is based on the solutions explaining the success of queries in the field of deductive databases [Llo87]. In fact, we can view the IB modelled by a conceptual model as a deductive database D. At any given state, the extensional part of D consists of all base facts that are true at this state, while the intensional part of D will be defined by the deduction rules of the conceptual model. Now, the problem of explaining the value of a derived fact is equivalent to the problem of explaining the success of a query in a deductive database, as we explain below.

why *f*? Assuming that fact *f* holds in the IB, the explanation depends on whether its fact type is base or derived. If it is base, we cannot provide any explanation at this level. If it is derived, we can give an explanation based on its deduction rules.

Intuitively, an explanation of why a derived fact *f* is true in the current state of the IB is supposed to detail the reasoning involved in proving that *f* is true. We will adopt here the most common approach for explaining the success of queries in the field of deductive databases [Llo87]. This approach considers that exhibiting an interpretation (or a trace) of the SLDNF proof tree is adequate for that kind of reasoning.

We will show this approach using the specifications of Figure 1. Assume that in the current state the following base facts are true:

companies(comp,	name)	**projects**(proj,	name)	**consortium**(proj,	comp)
upc	un_polit_cat	p1	odissea	p1	upc

employees(emp,	name)	**belongs_to**(emp,	comp)	**works_on**(emp,	proj)
toni	toni mayol	toni	upc	toni	p1
joan	joan sistac	joan	upc	joan	p1

At this state, if the user queries the system about the value of derived fact *active(p1)* the answer will be "active(p1) is true". Now, the user can ask *why* this derived fact is true. The explanation given by the system consists of the deduction rule used to prove the desired fact (CM rule) and the corresponding set of instantiated literals. Then, the user can require more explanations for those literals representing derived information, as can be seen in the following.

```
why(active(p1))?
  active(p1) because:
  projects(p1,odissea) and engaged_in(upc,p1).
  CM rule: active(P) ← projects(P,N), engaged_in(C,P)
  engaged_in(upc,p1) can be further explained
```

```
why(engaged_in(upc,p1))?
  engaged_in(upc,p1) because:
  consortium(p1,upc) and belongs_to(joan,upc) and works_on(joan,p1).
  CM rule: engaged_in(C,P) ← consortium(P,C), belongs_to(E,C), works_on(E,P)
```

Sometimes, the truth value of a derived fact has several explanations, each one corresponding to a successful branch of the proof tree. In our the example case engaged_in(upc,p1) has the following alternative explanation:

```
alternative_explanation(engaged_in(upc,p1))?
  engaged_in(upc,p1) because:
  consortium(p1,upc) and belongs_to(toni,upc) and works_on(toni,p1).
  CM rule: engaged_in(C,P) ← consortium(P,C), belongs_to(E,C), works_on(E,P)
```

why_not *f*? A second capability consists in explaining why an information is false at the current state. See [OlS95] for the details and examples of how we answer this kind of question. Our approach is based on the work described in [DeT89,Dec91].

3 Explaining the Changes to the Information Base

The second level of explanation in our method is that of explaining the changes (transitions), induced by a transaction, from the previous to the current state of the IB. At this level, we improve the reasoning capabilities described in the previous level by giving explanations about why a fact has been inserted or deleted in the last transition, and providing the set of possible updates to make a given fact true or false at the next state.

3.1 Requirements

Answering the above kind of question requires knowing the transactions that have been executed, and their effect on the IB. Our method does not require knowing the internal details of the transactions. We will see that many helpful explanations can be given using only the knowledge of which updates have been performed by the transaction.

For each execution of a transaction, we record a fact of type *trans_log(name,parameters,time)*, with the name of the transaction, the list of its parameters (which may be empty) and the execution time. Without loss of generality, we assume that only one transaction is executed at a given time. We will use, in the next section, the execution time of transactions as identifiers for the states of the IB.

A transaction performs, among other things, several updates to the IB. An update may be an insertion or a deletion of a base fact. For each insertion of a fact of type p(x), where x is a set of arguments, we need to record a fact of type ιp(x,time), where time is the transaction time. We assume that inserted facts do not hold at the previous state.

Similarly, for each deletion of a fact of type p(x) we need to record a fact of type δp(x,time). We assume that deleted facts hold at the previous state.

For example, suppose that transaction *change_assignment* removes an employee from a given project and assigns him/her to another. Assume that, at time 10, the transaction is executed, changing employee *maria* from project *odissea* to project *folre*. We would record the following facts:

trans_log (change_assignment,[maria,odissea,folre],10),
δworks_on (maria,odissea,10), and ιworks_on (maria,folre,10)

Note that these facts may be obtained easily in most specification execution environments, and, in fact, some of them already capture traces of the transaction execution [LiK93].

3.2 The Internal Events Model

A transaction performs updates to only base facts of the information base. If we want to be able to give explanations of derived facts, we need to know how updates to base facts propagate to derived facts. This knowledge is given by the *Internal Events Model* (IEM), which is a model that can be obtained automatically from deduction rules. The model has been described in [Oli91,Urp93] and it is briefly reviewed below.

The key concept of an IEM is that of *internal event*. Let IB be an information base, U an update and IB' the updated information base. We say that U induces a transition from IB (the previous state) to IB' (the new state). We assume, for the moment, that U consists of an unspecified set of base facts to be inserted and/or deleted. Due to the deduction rules, U may induce other updates on some derived facts. Let p be one of such fact types, and let p' denote the same fact type evaluated in IB'. We associate with p an *insertion event predicate* ιp, and a *deletion event predicate* δp, defined as:

(1) $\forall X \ (\iota p(X) \leftrightarrow p'(X) \wedge \neg p(X))$

(2) $\forall X \ (\delta p(X) \leftrightarrow p(X) \wedge \neg p'(X))$

where X is a vector of variables. From the above we have the equivalences [Urp93]:

(3) $\forall X \ (p(X) \leftrightarrow (p(X) \wedge \neg\, \delta p(X)) \vee \iota p(X))$

(4) $\forall X \ (\neg p(X) \leftrightarrow (\neg p(X) \wedge \neg\, \iota p(X)) \vee \delta p(X))$

Let us consider a derived predicate p. Assume that the definition of p consists of m rules, m≥1. For our purposes, we rename predicate symbols at the head of the rules as $p_1,...,p_n$ and we add the set of clauses:

(5) $p(X) \leftarrow p_i(X) \qquad i = 1..m$

Consider now one of the rules $p_i(X) \leftarrow L_{i,1} \wedge ... \wedge L_{i,n}$. When this rule is to be evaluated in the new state, its form is $p'_i(X) \leftarrow L'_{i,1} \wedge ... \wedge L'_{i,n}$, where $L'_{i,r}$ (r = 1..n) is obtained by replacing the predicate q of $L_{i,r}$ by q'. Then, if we replace each literal in the body by its equivalent expression given in (3) or (4) we get a new rule, called *transition rule*, which defines the new state predicate p'_i in terms of old state predicates and events.

For example, the transition rule corresponding to *inactive'*$_1$ in Figure 1 is given by:

$inactive'_1(P) \leftarrow ((projects(P,N) \wedge \neg\, \delta projects(P,N)) \vee \iota projects(P,N)) \wedge$
$\qquad ((\neg active(P) \wedge \neg\, \iota active(P)) \vee \delta active(P))$

which, after distributing ∧ over ∨ , is equivalent to the four transition rules:

$inactive'_{1,1}(P) \leftarrow projects(P,N) \wedge \neg\, \delta projects(P,N) \wedge \neg active(P) \wedge \neg\, \iota active(P)$

$inactive'_{1,2}(P) \leftarrow projects(P,N) \wedge \neg\, \delta projects(P,N) \wedge \delta active(P)$

$inactive'_{1,3}(P) \leftarrow \iota projects(P,N) \wedge \neg active(P) \wedge \neg\, \iota active(P)$

$inactive'_{1,4}(P) \leftarrow \iota projects(P,N) \wedge \delta active(P)$

with:

$inactive'_1(P) \leftarrow inactive'_{1,j}(P) \qquad j = 1..4$

The transition rules for predicates *engaged_in* , *active* , *ic1* and *ic2* would be obtained similarly.

Insertion predicates ιp were defined in (1) as: $\forall X \ (\iota p(X) \leftrightarrow p'(X) \wedge \neg p(X))$

If there are *m* rules for predicate p, then $p'(X) \leftrightarrow p'_1(X) \vee ... \vee p'_m(X)$. Replacing p'(X) in (1) we obtain:

$\iota p(X) \leftarrow p'_i(X) \wedge \neg p(X) \qquad i = 1..m$

which are called the *insertion internal events rules* of predicate p. In the example above, there would be only one rule (since m = 1):

$\iota inactive(P) \leftarrow inactive'_1(P) \land \neg inactive(P)$

Similarly, deletion predicates δp were defined in (2): $\forall X \ (\delta p(X) \leftrightarrow p(X) \land \neg p'(X))$

If there are *m* rules for predicate p, we then have:

$\delta p(X) \leftarrow p_i(X) \land \neg p'(X) \qquad i = 1..m$

and replacing p'(X) by its equivalent definition $p'(X) \leftrightarrow p'_1(X) \lor ... \lor p'_m(X)$ we obtain:

$\delta p(X) \leftarrow p_i(X) \land \neg p'_1(X) \land ... \land \neg p'_m(X) \qquad i = 1..m$

This set of rules is called the *deletion internal events rules* for predicate p. In our example, there would be only one rule (since m = 1):

$\delta inactive(P) \leftarrow inactive(P) \land \neg inactive'_1(P)$

The set of transition, insertion internal event and deletion internal event rules is called the *Internal Events Model* (IEM). These rules allow us to deduce which induced insertions and deletions happen in a transition, in terms of old state facts and events. In most cases, these rules can be substantially simplified, using the procedure described in [Oli91,Urp93]. Note that the Internal Events Model will be relevant even if our method is used in a language that does not consider derived facts. Recall that integrity constraints can be seen as rules defining when an inconsistency fact (ic1, ic2 in our example) holds. The insertion internal events rules for inconsistency predicates give the conditions upon which consistency is violated.

From the IEM, we compute the induced updates on derived facts, and store the result in a way similar to the base updates. For example, assume that, at the current state, the information base contains the following base facts:

companies(comp,	name)
upc	un_polit_cat

projects(proj,	name)
p1	odissea
p2	folre
p3	dream

consortium(proj,	comp)
p1	upc
p2	upc
p3	upc

employees(emp,	name)
toni	toni mayol
joan	joan sistac
maria	maria costal

belongs_to(emp,	comp)
toni	upc
joan	upc
maria	upc

works_on(emp,	proj)
toni	p3
joan	p3
maria	p1

which, implicitly, induce the derived facts:

engaged_in(company,	project)
upc	p1
upc	p3

active(project)
p1
p3

inactive(project)
p2

Assume now that transaction:

trans_log (change_assignment,[maria,odissea,folre],10),
δworks_on (maria,p1,10), ιworks_on (maria,p2,10)

is executed at time 10. The computed induced updates would be recorded as:

δengaged_in (upc,p1,10), ιengaged_in (upc,p2,10), δactive (p1,10), ιactive (p2,10),
δinactive (p2,10), ιinactive (p1,10)

3.3 Procedures

Figure 2 depicts the architecture of our explanation system. There must be some kind of tracing system that traces the transactions executed (and adds a trans-log fact to 'Updates'), captures the insertions to (and deletions from) the Information Base (and adds an ιp or δp fact to 'Updates') and computes the induced insertions and deletions (adding also the corresponding ιp or δp facts to 'Updates'). Such a tracing system may be implemented in several ways, depending on the execution environment. The figure also shows all the sources used in giving explanations.

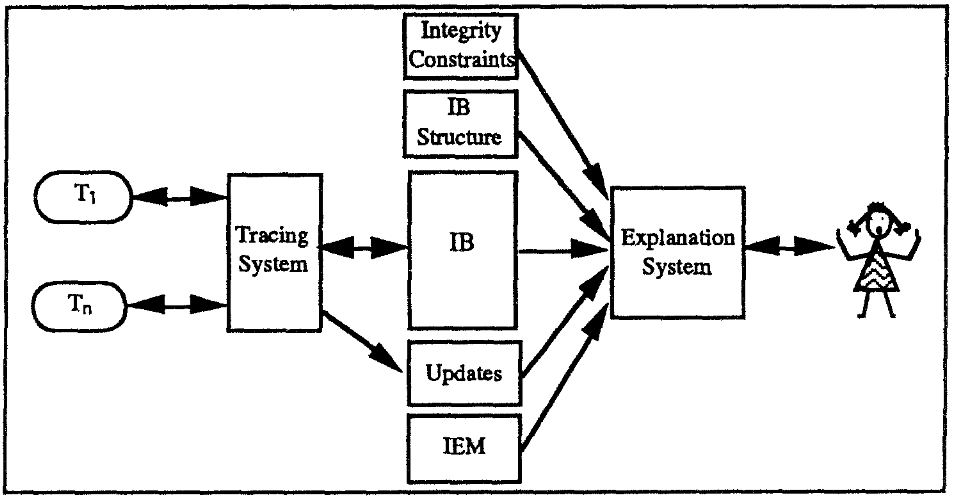

Figure 2. System architecture

We will now describe how answers to the questions described above can be obtained.

why *f*? Assuming that fact *f* holds in the IB, the explanation depends on whether its fact type is base or derived. If it is base, there are two possible explanations for fact *f* being true:

- *f* has been inserted by last transaction, or
- *f* was already true at previous state (and it has not been deleted).

Let us resume the execution of our example case at the point it was left in the previous subsection. Now, assume that at time 13, a transaction *change_project*(dream,odissea) moves all employees working on project dream to project odissea. The resulting set of base facts will be:

companies(comp,	name)
upc	un_polit_cat

projects(proj,	name)
p1	odissea
p2	folre
p3	dream

consortium(proj,	comp)
p1	upc
p2	upc
p3	upc

employees(emp,	name)
toni	toni mayol
joan	joan sistac
maria	maria costal

belongs_to(emp,	comp)
toni	upc
joan	upc
maria	upc

works_on(emp,	proj)
toni	p1
joan	p1
maria	p2

and the set of derived facts:

engaged_in(company,	project)
upc	p1
upc	p2

active(project)
p1
p2

inactive(project)
p3

At this time, if the user queries the system about the value of base fact works_on(toni,p1) the answer will be "works_on(toni,p1) is true", and the explanation will be as follows:

```
why(works_on(toni,p1))?
  works_on(toni,p1) has been inserted by last transaction change_project(dream,odissea)
```

If fact *f* is derived, the explanation of *why* this fact *f* is true given on the first level can be substantially improved using the IEM. In particular, the use of the IEM allows us to reason in terms of the change induced in the transition from previous to current state. Reasoning in this way, there are two possible explanations for fact *f* being true:

- An insertion of *f* has been induced, or
- *f* was already true at previous state (and its deletion has not been induced).

In the first case, the user may be interested in the reasons for the change from the previous to the current state, that is, in explanations of why the insertion of a fact *f* has been induced. This is catered for the *why_inserted* explanations.

why_inserted *f*? The answer gives the reasons why derived fact *f* was inserted in the last transition. The desired explanations can be obtained from the internal events rules. From an analysis of the SLDNF proof tree of the corresponding insertion internal event fact (ιf) we can obtain the set of updates on base facts, performed by the last transaction, that induced the insertion of *f*.

Assume that the current IB state is the result of the execution of transaction *change_project*(dream,odissea) at time 13. At this state, if the designer wants to know why engaged_in (upc,p1) holds, the explanation will be as follows:

why(engaged_in (upc,p1))?
engaged_in (upc,p1) because:
consortium(p1,upc) and belongs_to(toni,upc) and works_on(toni,p1).
CM **rule:** engaged_in(C,P) ← consortium(P,C), belongs_to(E,C), works_on(E,P).
engaged_in(upc,p1) has been induced by last transaction: *change_project*(dream,odissea)

why_inserted(engaged_in(upc,p1))?
inserted(engaged_in(upc,p1)) because:
works_on (toni,p1) has been inserted.

alternative_explanation(inserted(engaged_in(upc,p1)))?
inserted(engaged_in(upc,p1)) because:
works_on (joan,p1) has been inserted.

why_not *f*? Assuming that fact *f* does not hold in the IB, the explanation depends on whether its fact type is base or derived. If it is base, there are two possible explanations for fact *f* being false:

- *f* has been deleted by last transaction, or
- *f* was already false at previous state (and it has not been inserted).

If *f* is derived, we can provide explanations of why it is false at the current state in terms of the change induced in the transition from the previous state. Now, the use of the IEM allows us to give two possible explanations for fact *f* being false:

- A deletion of *f* has been induced), or
- *f* was already false at previous state (and its insertion has not been induced).

As before, the reasons for the induced deletion can be obtained using the *why_deleted* explanations.

why_deleted *f*? The answer gives the reasons why derived fact *f* was deleted in the last transition. As in the case of *why_inserted*, the desired explanations can be obtained from the internal events rules. From an analysis of the SLDNF proof tree of the corresponding deletion internal event fact (δf) we can obtain the set of updates on base facts, performed by the last transaction, that induced the deletion of *f*.

In our example case, if at time 13 the designer wants to know why active(p3) does not hold, the explanation will be as follows:

```
why_not(active(p3))?
 active(p3) is false because:
 [engaged_in(p3,C)] is false
 active(p3) has been implicitly deleted by last transaction: change_project(dream,odissea)
why_deleted(active(p3))?
 deleted(active(p3)) because:
 works_on (toni,p3) has been deleted and works_on (joan,p3) has been deleted.
```

how *f*? Assuming that fact *f* is currently false, the answer to this question consists in a set of base updates to the current state such that *f* will hold in the next state, and the information base will mantain its consistency. Note that we cannot provide here a simple solution for base facts (like "insert *f* into the IB"), because some integrity constraint could be violated.

Providing answers to this question is equivalent to view updating in deductive databases. The information base can be seen as a deductive database, and the request to make a fact *f* true can be seen as an insertion request of *f* in a view updating method.

Several methods for the solution of the view update problem do exist. We use the Events Method presented in [TeO92,Ten92]. The method is based on the Internal Events Model described above.

For example, if we assume that the current IB state is the result of the execution of transaction *change_project*(dream,odissea) at time 13, derived fact engaged_in(upc,p3) is false. If the designer wants to know how to make it true, the system answers in the following way:

```
how(engaged_in(upc,p3))?
   possible translations:
   [insert  works_on(maria,p3)]
   [insert  works_on(toni,p3)]
   [insert  works_on(joan,p3)]
   [insert  belongs_to(E,upc) and insert works_on(E,p3)]
```

In order to apply the desired update, the designer should choose and execute a transaction that modifies the IB in the way shown by one of the proposed translations. Note that the proposed translations maintain the information base consistency. Our method takes into account the integrity constraints, and translations that would leave the information base inconsistent are not generated.

In our example case, the designer could choose a transaction *assign(joan, dream)* to assign employee *joan* to project *p3*. This would imply the engagement of company *upc* to project *p3*.

how_not *f*? Assuming that fact *f* currently holds, the answer to this question consists in a set of base updates to the current state, such that *f* will be false in the next state, and the information base will maintain its consistency. We solve this problem as in the previous question, but now considering a delete request.

4 Explaining the Historical Evolution of the Information Base

The third level of explanation in our method shows the temporal evolution of the IB. At this level we can explain why a fact was true or false at a past state, what was known about a fact type at a given state, the intervals during which a fact has been true, and the set of states when a fact has been updated. In this section, we describe the procedures that may be used to obtain this kind of explanation. The knowledge needed to provide such explanations is exactly the same as on the previous level.

4.1 Procedures

If we want to be able to give explanations about the historical evolution of the IB, we need to know the value of base and derived facts at each state, from the initial to the current one. We will show that this knowledge can be obtained from the requirements established at the previous level.

The first requirement was to store all transactions executed from the initial state, and the corresponding updates. With this information we can 'reconstruct' the IB contents at any state. With respect to base predicates, it is easy to define the value of a base predicate p(X) at a given state *s* in terms of the insertion and deletion events occurred until *s*, as follows:

$\forall X \ (p(X,S) \leftrightarrow \iota p(X,S1) \land S1 \leq S \land \neg \exists S2 \ (\delta p(X,S2) \land S1 < S2 \leq S)$

meaning that a fact p(x) is true at state *s* if it has been inserted in a state *s1* before *s* and has not been deleted between *s1* and *s*. Recall that we identify states by the execution time of the transaction.

With respect to derived predicates, we only have to apply a minor transformation to deduction rules to take the state into account. For example, the deduction rule for predicate engaged_in would be transformed as:

engaged_in(C,P,S) ← consortium(P,C,S), belongs_to(E,C,S), works_on(E,P,S)

why *f* at *s*? Assuming that fact *f* was true at state *s*, the explanation depends on whether its fact type is base or derived. If it is base, there is only one possible explanation for fact *f* being true at *s*, which consists in identifying the transaction that inserted it. Then, if *f*=p(k) we have to look for a transaction such that:

$\text{trans\_log(name,parameters},S1) \land S1 \leq s \land \iota p(k,S1) \land \neg \exists S2 \ (\delta p(k,S2) \land S1 < S2 \leq s)$

Assume, for example, that the current IB state is the result of the execution of transaction *assign(joan, dream)* at time 16. At this time, if the user queries the system about the value of base fact works_on(maria,p2) at state 13 the answer will be "works_on(maria,p2) was true at state 13", and the explanation is as follows:

why(works_on(maria,p2)) at 13? works_on(maria,p2) was true at 13 because: transaction*change_assignment*(maria,odissea,folre) executed at time 10 has inserted it, and no transaction between 10 and 13 has deleted it.

If fact *f* is derived, the explanation of *why f* was true in a given state *s* is exactly the same as that provided on previous levels for the current state. It can be obtained with the same technique, but taking into account the set of facts that were true at *s*.

why_not *f* at *s*? Assuming that fact *f* was false at state *s*, the explanation depends on whether its fact type is base or derived. If it is base, there are two possible explanations for fact *f* being false at *s*:

- fact *f* was never inserted in the IB before *s*.
- fact *f* was deleted by a transaction before *s*.

If *f* = p(k), we identify the first case when the following condition holds:

$\neg \exists S1 \ (\iota p(k,S1) \land S1 \leq s)$

In the second case, we have to look for a transaction such that:
$\text{trans\_log(name,parameters,S1)} \wedge S1 \le s \wedge \delta p(k,S1) \wedge \neg \exists S2\ (\iota p(k,S2) \wedge S1 < S2 \le s)$

For example, if at state 16 the user queries the system about the value of base fact works_on(toni,p3) at state 13, the answer will be "works_on(toni,p3) was false at state 13", and the explanation is:

why_not(works_on(toni,p3)) **at** 13?
works_on(toni,p3) was false at 13 because:
transaction *change_project*(dream,folre) executed at time 13 has deleted it.

If fact *f* is derived, the explanation of *why f* was false in a given state *s* is exactly the same as that provided on previous levels for the current state. It can be obtained with the same technique, but taking into account the set of facts that were true at *s*.

what_known_about fact type **at** *s*? To obtain a list of all facts of the given type that were true at state *s* we have only to evaluate the extension of the corresponding predicate at this state, as explained at the beginning of this section.

when *f*? The answer to the question about *when* a fact *f* holds in the IB can be obtained from the stored updates. It does not depend on whether the corresponding fact type is base or derived. Basically, we have to look for all the pairs of consecutive insertion-deletion events of the required fact.

There are two possible answers to the question *when f*?:
- fact *f* has never been true.
- the set of intervals during which fact *f* has been true.

If $f = p(k)$, we identify the first case when the condition $\neg \exists S\ (\iota p(k,S))$ holds. If this condition does not hold, we have to look for all the intervals [S1,S2] such that:
$\iota p(k,S1) \wedge \delta p(k,S2) \wedge S2 > S1 \wedge \neg \exists S3\ (\delta p(k,S3) \wedge S1 < S3 \le S2)$ or
$\iota p(k,S1) \wedge \neg \exists S3\ (\delta p(k,S3) \wedge S3 > S1) \wedge S2 = \text{current\_state}$

In our example case, the following explanations could be obtained at state 16.

when(employees(enric, enric pastor))?
employees(enric, enric pastor) has never been true
when(works_on(joan,p3))?
works_on(joan,p3) has been true **from** state 1 **to** state 13 **and at** current state

In the case of derived facts there is an alternative solution to answer this query, which does not require the use of the internal events. It consists in the evaluation of fact *f*, using the deduction rules, for each state from the initial to the current one.

when [ι|δ]*f*? The answer to the question about *when* an update [ι|δ]*f* occurred can be obtained from the stored updates. It does not depend on whether the corresponding fact type of *f* is base or derived. We have to look for all the events updating fact *f* which have occurred from the initial to the current state.

There are two possible answers to the question *when* [ι|δ]*f*:
- the update has never occurred.
- the set of states in which the update has occurred.

If $f = p(k)$, we identify the first case when the condition $\neg \exists S\ ([\iota|\delta]p(k,S))$ holds. If this condition does not hold, we have to look for all the states S such that $[\iota|\delta]p(k,S)$.

We can also consider the case in which *f* is not fully instantiated. That is, f has the form p(k,Y), Y being a set of variables. Then, the system will answer giving the set of tuples (y,s) such that $[\iota|\delta]p(k,y,s)$ holds, if any.

5 Explaining the Effect of Hypothetical Past Updates

The last level of explanation in our method is that of explaining the effect of a potential update in the past. In this section, we describe the procedures that may be used. The knowledge needed to provide such explanations is exactly the same as on the third level.

5.1 Procedures

what_if [ι|δ]p(k) **at** *s* **on** *f*? The system has to evaluate the impact that a potential update [ι|δ]p(k) performed at state *s* would have on fact *f* at the current state. Given that a transaction performs updates on only base facts of the IB, the fact type of p(k) is always base. The fact type of *f* can be base or derived. The answer consists in the difference between the current value of *f* and its hypothetical value if the update had been performed at state *s*.

As already mentioned, our explanation method does not consider the internal structure of transactions. In particular, we ignore the conditions under which each transaction performs updates to base facts. Then, the only way to evaluate the impact of a hypothetical update on a past state is to simulate a new execution of all transactions occurring since that state. This simulation can be done using the information about the transactions and their corresponding updates stored at execution time.

Also, we have to ensure that the IB consistency will be mantained from state *s* to the current state. Our method guarantees this condition by checking that the update does not violate any integrity constraint at the state in which it is performed, and by checking that transactions executed after *s* preserve this consistency. In addition, we may also ensure that, at every state, some implicit assumptions are preserved. Namely, that insertions and deletions are effective (insertions add non-existing information and deletions remove existing information). Such conditions are treated as constraints.

As a consequence, the procedure to obtain explanations regarding what_if [ι|δ]p(k) at *s* on *f* is as follows:

- Make the hypothesis that the update [ι|δ]p(k) is performed at state *s*.
- Ensure that the update does not violate any integrity constraint at *s*, by evaluating the inconsistency rules at this state.
- For each transaction occurring from *s* to the current state, check if the state resulting from the application of its updates is consistent, by evaluating the inconsistency rules. Then, two cases are possible:
  (1) The transaction violates some integrity constraints. In this case, the explanation will be: "This update leads the IB to an inconsistent state when transaction *name(parameters)* is simulated at time *t*."
  (2) No integrity constraint is violated. In this case, the system evaluates fact *f* at the resulting state, and gives the difference with respect to the real value of *f* at the current state.

Note that to evaluate the integrity constraints at each state, our method does not have to rebuild each state from *s* to the current one. We can obtain the value of base and derived facts at any state from the stored events, as explained in Subsection 4.1. Therefore, the inconsistency rules can be evaluated at each state like any deduction rule. In our example case, the integrity constraints would be transformed to take the state into account as:

ic1(S) ← works_on(E,P,S), not projects(P,N,S)
ic2(S) ← companies(C,N,S), not engaged_in(C,P,S)

Assume, for example, that the current IB state is the result of the execution of transaction *new_project(p4,bloom)* at time 20. The following explanations could be obtained at this state.

what_if δworks_on(maria,p2) **at** 13 **on** active(p2)?
active(p2) is currently true, but it would be false if the update was performed.

what_if ιcompanies(uab,univ_autòn_de_barna) **at** 16 **on** engaged_in?
This update would violate integrity constraint ic1 at state 16.

what_if δprojects(p3) **at** 13 **on** active?
This update leads the IB to an inconsistent state when transaction assign(joan, dream) is simulated at time 16. Integrity constraint ic2 would be violated.

what_if_not [ι|δ]p(k) **at** s **on** *f*? Assuming that an update [ι|δ]p(k) was performed by a transaction at state s, the system has to evaluate the impact that the absence of this update would have on fact *f* at the current state. Given that a transaction performs updates on only base facts of the IB, the fact type of p(k) is always base. The fact type of *f* can be base or derived. The answer consists on the difference between the current value of *f* and its hypothetical value if the update was not performed at state *s*. The procedure to obtain explanations about what_if_not [ι|δ]p(k) at *s* on *f* is similar to the previous case. Both kinds of hypothetical explanations can also be combined.

6 Conclusions

We have presented a method for explaining the behaviour of conceptual models of information systems. The method assumes a conceptual model in terms of information base structure (with base and, optionally, derived facts), integrity constraints and transactions. Therefore, the method may be adapted in most current methodologies.

Our method contributes to model validation by providing explanations about the results of model execution. It provides, with a simple architecture, useful answers to questions about why (or why not) a fact holds in the current state, why a fact has been inserted (or deleted) in a transition, how a fact can be made true (or false), why (and when) a fact has been true in the past and what would have happened if past updates had been different.

Answers to some of the above questions are given by some existing explanation systems. We extend them by providing answers to questions about derived facts, to questions about how a fact can be made true or false, and to hypothetical questions.

Our method is based mainly on results obtained in the field of deductive databases. We have seen how the procedures developed in that field for explaining the results of queries, or their failure, and for updating consistent knowledge bases may be useful for behaviour explanation of conceptual models. In this sense, our method links these two fields.

Acknowledgements

The authors wish to thank the ODISSEA group for their comments and suggestions. This work has been supported by the CICYT PRONTIC program, project TIC 680.

References

[Bub88] Bubenko,J.A. "Selecting a Strategy For Computer-aided Software Engineering (CASE)", SYSLAB Rep. 59, University of Stockholm, 1988.

[Che76] Chen.P.P. "The Entity-Relational model. Towards a unified view of data". ACM Trans. on Database Systems, vol. 1, no. 1, March 1976, pp. 9-36.

[CoO92] Costal,D.; Olivé,A. "A method for reasoning about deductive conceptual models", Proc. of CAiSE 92, Manchester, May 1992, pp. 612-631.

[Dal92] Dalianis,H. "A method for validating a conceptual model by natural language discourse generation", Proc. of CAiSE 92, Manchester, 1992, pp. 425-444.

[Dec91] Decker,H. "On Explanations in Deductive Databases", Proc. Third Workshop on Foundations of Models and Languages for Data and Objects, Aigen, September 1991, pp. 173-186.

[DeT89] Decker,H.; Tomasic,A. "Towards a foundation of explanations in deductive databases", Internal report ECRC, Munich 1989.

[Gul93] Gulla,J.A. "Explanation generation in information systems engineering", PhD. Thesis. Norwegian Institute of Technology, Trondheim, 1993.

[GuW93] Gulla,J.A.;Willumsen,G. "Using Explanations to Improve the Validation of Executable Models", Proc. CAiSE 93, Paris, June 1993, pp. 118-142.

[HaM81] Hammer,M.;McLeod,D. "Database description with SDM: A semantic database model". ACM TODS, vol. 6, no. 3, 1981, pp. 351-386.

[JeC92] Jesus,L.;Carapuça,R. "Automatic Generation of Documentation for Information Systems",Proc. CAiSE 92, Manchester, May 1992, pp. 48-64.

[JMS92] Jarke,M.; Mylopoulos,J.W.;Schmidt,J.W.;Vassiliou,Y. "DAIDA: An Environment for Evolving Information Systems". ACM Trans. on Information Systems, vol. 10, no. 1, January 1992, pp. 1-50.

[LaL93] Lalioti,V.; Loucopoulos,P. "Visualisation for Validation", Proc. CAiSE 93, Paris, June 1993, pp. 143-164.

[LiK93] Lindland,O.I.;Krogstie,J."Validating Conceptual Models by Transformational Prototyping", Proc. CAiSE 93, Paris, June 1993, pp. 165-183.

[Llo87] Lloyd, J.W. "Foundations of logic programming". Springer-Verlag, 1987.

[LTP91] Loucopoulos,P.;Theodoulidis,B.;Pantazis,D. "Business Rules Modelling: Conceptual Modelling and Object-Oriented Specifications". In Van Assche, F., Moulin,B.; Rolland,C. (eds.) "Object Oriented Approach in Information Systems", North-Holland, 1991, pp. 322-342.

[NH89] Nijssen, G.M.; Halpin, T.A. "Conceptual Schema and Relational Database Design. A fact oriented approach". Prentice Hall, 1989.

[Oli91] Olivé, A. "Integrity constraints checking in deductive databases", Proc. of the 17th. VLDB, Barcelona, 1991, pp. 513-523.

[OlS95] Olivé, A.; Sancho,M.R. "A Method for Explaining the Behaviour of Conceptual Models - Extended version" Tech. Report LSI/95-R

[RoP92] Rolland,C.;Proix,C. "A Natural Language Approach for Requirements Engineering", Proc. CAiSE 92, Manchester, May 1992, pp. 257-277.

[Ten92] Teniente, E. "El mètode dels esdeveniments interns per actualització de vistes en bases de dades deductives" (in catalan), PhD. Thesis, Universitat Politècnica de Catalunya, Barcelona, 1992.

[TeO92] Teniente, E; Olivé, A. "The Events Method for View Updating in Deductive Databases", Proc. EDBT'92, Vienna, 1992, pp. 245-260.

[San93] Sancho,M.R. "Explaining the behaviour of a deductive conceptual model", Proc. Fourth Intl. DAISD Workshop, Tech. Report LSI/93-25-R, Universitat Politècnica de Catalunya, 1993, pp. 27-50.

[San94] Sancho,M.R. "Disseny de transaccions a partir de models conceptuals deductius" (in catalan). PhD. Thesis, Universitat Politècnica de Catalunya, Barcelona, 1994.

[Urp93] Urpi, A. "El mètode dels esdeveniments interns per al càlcul de canvis en bases de dades deductives" (in catalan). PhD. Thesis, Universitat Politècnica de Catalunya, Barcelona, 1993.