A Procedural Approach to Schema Evolution

Catherine A. Ewald and Maria E. Orlowska

Key Centre for Software Technology
The University of Queensland, Qld 4072, Australia

Abstract. Like data, conceptual schemata change and need to be updated. If not performed with care, updates can cause problems. In this paper, we present a procedure which safely adds a fact type to a conceptual schema. This procedure, as well as being used for simple schema updates, may be applied to design and integration problems. We also present procedures for the safe removal of a fact type, and the safe elimination of redundant information from the conceptual schema.

1 Introduction

Conceptual schemata cannot be regarded as static. A schema should be able to evolve in response to changes in user requirements. Design faults, once discovered, should be easily corrected. Conceptual schema updates include the addition and removal of fact types and constraints, and modifications to information and constraints. Constraints may also be added and removed. The design process may be seen as an evolution process, in which the schema is built up by adding information units or pieces of information one at a time, starting with a single piece of information. We call such elementary, irreduceable pieces of information fact types. Schema integration (and view integration) are also evolutionary processes. Starting from two or more source schemata [1] , an integrated schema is constructed. This is done by choosing one schema as the target schema, and adding the others to it (one piece of information at a time).

Conceptual schema design, and schema integration may be seen as special cases of a more general problem. We call this problem the schema evolution problem. Another case is the ad hoc modification of a schema to meet changing requirements or fix errors. Clearly, any solution to the schema evolution problem may be applied to the classical design and integration problems. The design problem is a special cases, requiring more decisions on the part of the human designer. However, the interactive procedures presented here may be applied to conceptual schema design. In this paper, we propose an interactive, procedural solution, in which some automatic support is provided to the designer during checking. We present a procedure which adds a single elementary unit of information to a conceptual schema. This procedure, as well as being used for simple schema updates, may be applied to design and integration problems. The addition of an information unit is the most important operation of conceptual

[1] These may range from simple, small user views to large "local schemas" which describe the sites of a distributed database.

schema design and integration. Since such an addition may lead to redundancy on the conceptual schema level, other redundant information may need to be removed after the addition. However, more information may have to be added after the redundancy is removed. Obviously, care has to be taken not to introduce a cyclic problem here. For this reason, we introduce a special procedure for the replacement of a fact type which never requires addition of another fact type. This is possible because we assume that the fact type to be replaced is redundant (some other representation of the fact type exists). We also present a procedure for the removal of a fact type, to be used when the designer wishes to delete a non-redundant fact type.

Most previous research in the area of schema integration is based on the relational, and Entity Relationship (ER) models. Constraint integration in a global relational database is discussed by Tanaka and Kambayashi [17]. The UNIBASE system [14] uses the relational model as a common data model to handle different database structures. An augmented relational model using connectors to impose conditions on attributes has been developed by Czejdo,Rusinkiewicz,and Embley [2]. A large number of papers focus on aspects of the integration of ER schemas. Attribute integration, view integration, and a scheme for the comparison of relationships have been examined by Larson, Navathe, and Elmasri and by Navathe et. al. in [6, 8, 9]. A set of incremental, reversible schema operators using a calculus oriented ER schema notation was presented by Markowitz and Makowsky [7]. Other relevant issues include the resolution of semantic conflicts, which is examined in a paper by Siegal and Madnick [16], and a version model designed to handle schema changes, presented by Andany et. al. in [1] . An object- oriented database programming language has been applied to the domain and schema mismatch problems by Kent [5]. The binary relationship and NIAM methodologies both use a fact based approach to database design. In fact, the binary relationship approach is based on an early version of NIAM (Nijssen's Information Analysis Methodology) . One advantage of fact-based design methodologies is that they represent a rich variety of constraints, and therefore are semantically expressive. Fact-based schemata are usually easier to integrate than ER, or relational schemata. This is because relationships are represented by fact types. Therefore the problem of structural (entity/relationship) conflict is greatly reduced in fact-based models. A view integration methodology using the binary relationship data model has been developed by Shoval and Zohn [15]. Other research uses a version of NIAM which is not restricted to binary relationships. Updates to conceptual schemata, in particular the addition and removal of fact types are examined by Orlowska and Ewald in [12, 11].

NIAM was originally developed as an information analysis methodology to assist humans in designing a conceptual schema for the universe of discourse which they wish to model. As such, its focus is on representation of application semantics, rather than formal methods for design and analysis. In fact, it is difficult to formalize some concepts and techniques used in fact-based modelling since they have been designed specifically for ease of use by humans. The conceptual schema design procedure [10] relies to a large extent on the expertise of

the human designer. Development of formal algorithms for such a data model is difficult and, even if successful, would place many restrictions on the human desinger. For this reason, we present interactive procedures designed to exploit the benefits of fact-based modelling,rather than formal algorithms.

This paper is organized as follows. Section 2 gives an overview of the NIAM methodology, Section 3 deals with the addition of a fact type , and Section 4 with the removal of a fact type. Both addition and removal of a fact type are operations which take a well formed conceptual schema and correct fact type as input and produce a well formed conceptual schema. Interactive checking ensures that problems with constraints are not introduced, and that (apart from the change desired by the human designer) the semantics of the old schema are preserved. Conclusions and directions for further research will be discussed in Section 5.

2 The NIAM methodology

A conceptual schema consists of elementary fact types, and constraints on the facts. Entities are graphically represented by circles or ellipses. A graphical representation of all constraints exits. Relationships are represented by rectangles, divided into "role boxes". The familiar concepts of generalization and aggregation are represented by subtypes. Nesting allows a role (relationship) to be treated as an entity.

Figures 1-3 show the most common constraints expressed on conceptual schema diagrams. For simplicity, only binary fact types are shown although fact types of higher arity are allowed. Only the relevant constraints are shown on each diagram. In particular, most uniqueness constraints are omitted. For this reason, the diagrams of Figures 1-3 should be regarded as constraint patterns, rather than complete schema diagrams. Although our main focus in this paper will be on interfact constraints, we also include intrafact constraints for completeness.

Using the conceptual schema design procedure (CSDP) [10, 4] the designer produces a schema diagram which represents the universe of discourse(UoD). The procedure, if followed correctly, ensures that the schema is correct. This means that the ONF algorithm may be applied to produce a normalized relational schema.

The ONF algorithm [10, 4] transforms the conceptual schema into a set of relational tables. For this reason, there is a very close relationship between the relational schema and the conceptual schema. In fact, a conceptual schema could be produced from a relational schema by means of a reverse engineering process, which will not be discussed in this paper. This process cannot be carried out entirely automatically, due to semantic problems. Interaction with the human designer is required. Assuming that the conceptual schema exists and is well formed, most of the automatic checking required during integration can be performed on the relational schema, rather than the conceptual schema. One or more populated databases may be involved in the integration process. Evolution of populated databases will be considered in depth as part of our con-

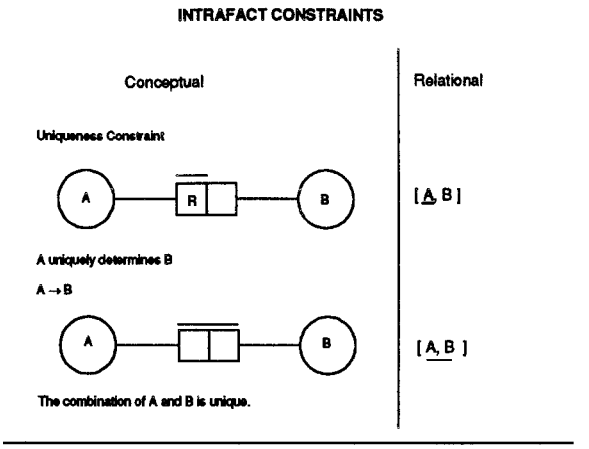

Fig. 1. Intrafact uniqueness constraints

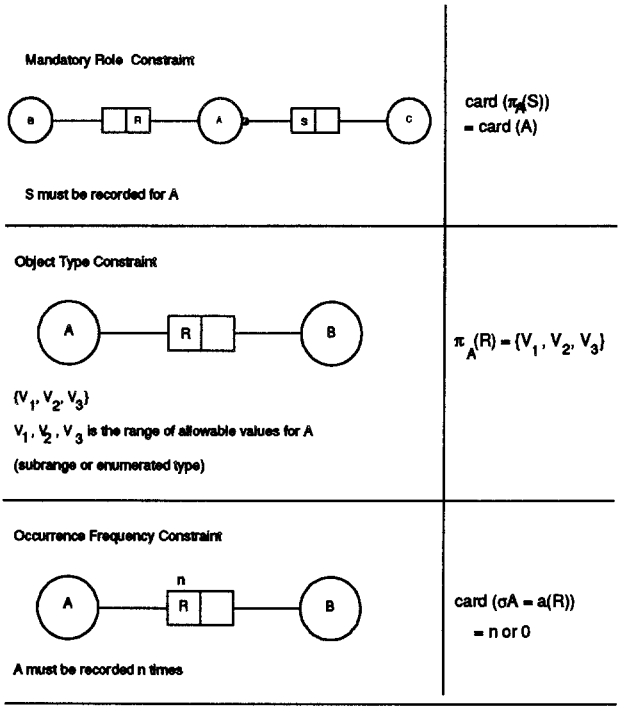

Fig. 2. Other intrafact constraints

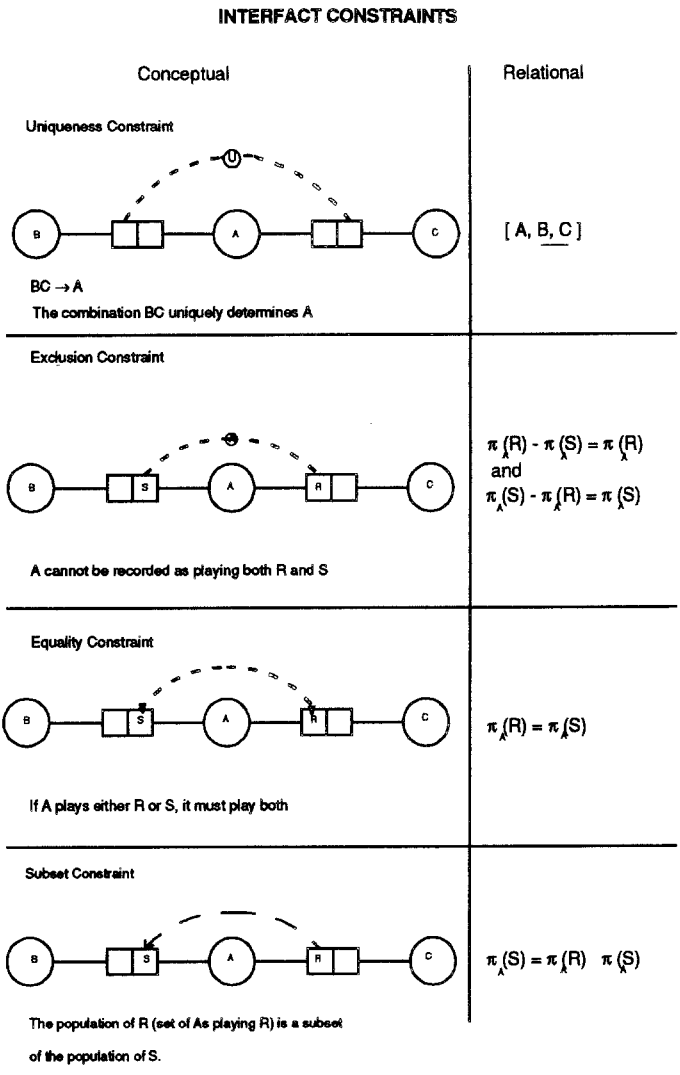

Fig. 3. Interfact Constraints

tinued research. However, an incorrect conceptual schema produces an incorrect relational schema. Failure to identify and remove derivable fact types , for example, could lead to unnormalized relations due to hidden transitive dependencies. Uniqueness constraints in NIAM represent functional dependencies. We have developed relational algebra expressions equivalent to the most frequently used of the other constraints at the conceptual level. Subset, equality and exclusion constraints are more expressive versions of the familiar referential integrity (foreign

key) constraint. A brief review of the most commonly used constraints, on both conceptual and relational schemata, is presented in Figures 1-3. We now present procedures for the addition and removal of fact types from the schema. The close relationship between conceptual and relational schemata is used in an interactive approach to schema evolution. The human designer deals with the semantic aspects, which require creativity and does so mainly on the conceptual schema. It is easier for humans to reason at the conceptual level, while automatic or interactive processes are easier if relational concepts are used. Interactive checking is performed on the relational schema, or on an FD graph formed from the conceptual schema. Since conversion from a relational to a conceptual schemata cannot be totally automatic, most checking is done on the FD graph. From this graph, both conceptual and normalized relational representations can be created. However, certain constraint checks are best performed on the relational schema for efficiency reasons.

3 A procedure for the addition of a fact type

If the following procedure is used to add a fact type to a well-formed conceptual schema, the new schema will also be well formed. It may be used for the design of a new schema, starting with a single fact type or integration of existing schemata. It is also useful for ad hoc modifications to schemata. There may be many correct, equivalent designs. Any conceptual (or relational) schema design is only one representative of its equivalence class, which may contain many other schemata. Some of these are naturally "better" than others. We do not examine the problem of schema evaluation [3] here. However, we aim to produce a design with as few relational tables as possible. It must be noted that design is a special, simpler case of the general integration problem. In many cases, the addition of a fact type is easy and causes no "side effects". However, certain checks must be performed to ensure correctness, since there are cases in which things may go wrong. Structural conflict [15] may occur during integration. In the worst case an entire subschema may conflict, representing the same information in two different ways on the integrated schema. The derivation checking process detects such conflicts.

We now present an informal outline of the procedure, before examining each step in detail. The procedure will then be applied to a simple example schema.

A Procedure for the addition of a fact type

1. *Produce a conceptual schema including the new fact type , and all intrafact constraints which apply to it.*
   Check that the new fact type does not conflict with any global semantic integrity constraint.
2. *Add any interfact constraints or global semantic integrity constraints related to the new information.*
3. *Check for intrafact constraint conflict and fact type redundancy. Update the derivation paths of existing fact types where required.*

4. *Check mandatory role constraints and other intrafact and interfact constraints. All constraints represented on the schema before the addition of the fact type should be represented on the new schema. Some mandatory role constraints which were previously implied may need to be explicitly added.*

The procedure must be performed in the given order for the following reasons :

- Redundancy checking must be performed after the addition of interfact constraints, since the addition of an interfact constraint may cause a fact type to become derivable.
- Constraint checking should be performed after all other changes to the schema, since some changes may affect constraints.

3.1 Step One

Produce a conceptual schema including the new fact type, and all intrafact constraints which apply to it.
First the information to be added should be formulated as an elementary fact type. An elementary fact type is one which cannot be split into two or more fact types. The fact type to be added should be related to (have a common entity with) a fact type that is already on the schema. Intrafact constraints should be added to the fact type. Every n'ary fact type must have a uniqueness constraint spanning at least n-1 roles. If a new fact type involving a primitive entity [2] type which previously played only one role is added a mandatory role constraint needs to be added. The constraint may be on the original role, the new role or a disjunction of the two. This choice should be based on the designer's knowledge of the UoD. If the fact type conflicts with a global semantic integrity constraint, either alter or remove the constraint, or alter the fact type. We do not present any formal specification of Step One, since it is to be performed entirely by the human designer. He/she should be free to use creativity and intelligence to produce the most suitable design for the application. A large degree of responsibility comes with this freedom. Later steps include interactive checking procedures which reduce the burden on the designer. Step One involves using the CSDP (Steps 1 to 7) to produce a subschema consisting only of the fact type to be added. This fact is then added to the conceptual schema diagram.

3.2 Step Two

Add any interfact constraints or global semantic integrity constraints related to the new information. Once again, this step must be performed entirely by the human designer. This basically involves performing Step 8 of the CSDP.

[2] A primitive entity type is any entity type that is not composite or nested

3.3 Step Three

Check for intrafact constraint conflict and fact type redundancy.
An added fact type may be derivable. Also, the addition of a fact type (or an interfact uniqueness constraint) may make it possible to derive another fact type or
constraint. [3] The addition of a fact type may, in turn, lead to the addition of an interfact uniqueness constraint, which could then affect the derivability of existing fact types. Figures 4-6 show an example in which this occurs. The derivation paths of fact types already known to be derivable may be affected by the addition of a fact type , in which case they must be updated. If the human designer has not detected and resolved constraint conflicts at an earlier stage, the schema will include multiple fact types representing the same information, but each having a different constraint pattern.

Orlowska and Zhang [13] have developed an algorithm to detect derivable fact types on a schema. An FD graph (G=(V,E) where E is a non-empty set of edges, and V a non-empty set of nodes) is formed from the NIAM schema and is searched for redundant FDs. We extend the conversion step to handle the case where an interfact uniqueness constraint forms (part of) the derivation path.

The conversion is done as follows :
(1) For every simple entity type A, there is a simple node labeled A.
(2) For every nested entity type $X(A_1, A_2..A_n)$*, there is a compound node labeled X and dotted arcs from X to* A_1, A_2, A_n
(3) For every n'ary fact type Y (A_1, A_2, A_n) *whose uniqueness constraint covers whole roles, there is a compound node labeled Y, and dotted arcs from Y to* A_1, A_2, A_n *and a full arc from Y to* θ *(*θ *is a new attribute)*
(4) For every n'ary fact type whose uniqueness constraint covers n- 1 roles $(A_1, A_2..A_{n-1})$*, there is a compound node (simple node if* $n = 2$ *) labeled X* $= A_1, A_2, A_{n-1}$ *, and a full arc from X to* A_n
(5) For every set of entities $A_1, A_2..A_n$ *where each* A_i *plays a role in a fact type with a particular entity E, if the roles played by* $A_1, A_2..A_n$ *are linked by an interfact uniqueness constraint, there is a compound node labeled X, a full arc from X to E and dotted arcs from X to* $A_1, A_2..A_n$

The DFT algorithm [13] then uses the synthesis approach to identify redundant FDs in the graph, which may correspond to derivable fact types on the NIAM schema. Since the fact types carry more semantic information than the FDs, the algorithm presents the suspected derivation path to the designer, and asks for confirmation that the fact type is derivable. For example, in our university database the functional dependency $P \rightarrow U$ may mean Person studies at university or Person received their first degree from university. If this FD is repeated, it appears to be redundant (on the FD graph). However, both the above fact types may need to be recorded. In addition, the algorithm may be enhanced

[3] Other changes to constraints may also lead (directly or indirectly) to the appearance of derivable fact types . Constraint updates themselves lead to new problems and are the topic of further research. They will not be discussed in detail here.

to support other constraint checking processes. To simplify interactive checking of constraints, we store the FD graph at the end of each run (before returning the conceptual schema). Constraint checking procedures can then access both this previous graph and the current FD graph, and compare them. However, some derivable fact types are not detectable by this method, since they do not involve redundant FDs. This problem occurs when a relationship that exists in the real world is not represented on the schema. The designer should check for such relationships [4], and include them on the schema. DFT will then locate any redundancy. Detection and resolution of such problems is the subject of further research.

Identify and resolve any constraint conflicts. Nodes on the FD graph which have multiple outgoing arcs, each going to the same node are likely to have been added because of constraint conflict. The algorithm should report these to the designer and allow him/her to resolve the conflict (and remove the unneeded fact types). We propose a four dimensional representation of constraints. This constraint table should be added to the target schema. Any constraints on "new" fact types which conflict with those on the target schema are added to the constraint table. Thus, no user will have to accept changes in constraints as a result of integration. In case of uniqueness constraint conflict, the weakest constraint should always be recorded on the target schema itself. Any stronger conflicting constraints should be entered in the table. It is relatively easy to enforce these additional FDs for transactions which access the particular schema involved. The constraint may be visualized as a relation with four columns - Schema Identification,Role or Roles,Constraint, Value. The schema identification consists of a number or name for the schema. This could be a character string. The role or roles involved must then be listed. The value varies with the type of constraint. The following should be recorded :

- uniqueness constraint conflict
  Assuming that the weakest of the conflicting constraints is shown on the schema, the relational design will not reflect the other constraint patterns. The additional FDs which should be enforced in all cases which conflict must be recorded, so that they can be enforced at the local schema level. These should be determined by interaction with the human designer.
- occurrence frequency constraint conflict
  The cardinality or range of cardinalities must be specified.
- mandatory role constraint conflict
  Specify if the role is to be mandatory. If no value is entered, it is assumed to be optional.
- object type constraint
  The range of values allowable for this schema must be specified.

Derivable fact types should then be eliminated. We describe such fact types as "replaced fact types" since they are replaced by an alternative representation of the same information. We present a procedure for the replacement of a fact type ;

1. **Decide which representation of the fact type to keep on the schema**
   The choice should take into account design objectives, and user requirements. Replace all other representations and update any derivation paths affected. The replacement of a fact type does not cause any problems with the derivation paths of other fact types, as long as derivation paths are correctly updated. Informally, a correct update may be defined as one which replaces the old (replaced) fact type in the derivation path with the new representation of the information. If a fact type or constraint is marked as derivable by the DFT algorithm, some FD path by which the fact type may be derived must exist. Unless this condition holds, a fact type (or constraint) will never be replaced by DFT. If the replaced fact type does not form part of the derivation path of any other derived fact type , problems cannot occur. If the fact type is involved in the derivation path of some other derived fact type , the derivation path is no longer valid. However, as discussed earlier, some other representation of the information must be stored. It may take the form of a fact type, or an interfact uniqueness constraint. Since each fact type or interfact uniqueness constraint corresponds to an FD, there is no practical difference between the case where the new representation is a fact type, and the case where it is an interfact uniqueness constraint. Thus, no additional fact types need to be added to the schema. The derivation path must be updated, replacing the old representation with the new one. In no case can a cycle arise.
2. *Check that all entities can be identified*

   We need to ensure that every entity recorded as playing a role in the database can be identified. For reasons of query response time and entity integrity enforcement, we do not permit the primary identifier of an entity to be derived. The exception to this rule is compound identifiers, where an entity is is identified by some combination of roles, are permitted. The only identifier of an entity which still plays some role must not be removed or replaced. On the relational schema no component of the primary key of a relation which has other attributes should be removed. However, what is originally a primary key may be removed if an alternate key exists and is first made the primary key . A primary key may also "shrink". If a n'ary p.k becomes an n-1 ary p.k the nth attribute may then be removed. Changes to intrafact uniqueness constraints are not examined in detail here, since the case discussed above is the only one that directly affects the addition and removal of fact types. This rule could be enforced by the system.
3. **Check interfact and intrafact constraints**
   Generally, constraints which involve replaced fact types need to be modified and shifted. However, there is a special case in which the constraint is no longer needed, and must be removed completely. This occurs when the interfact constraint in question runs between the role played by an entity E in a fact type replaced by an interfact uniqueness constraint, and the only other role played by E on the schema.

These problems are probably best dealt with by an interactive checking process performed on the relational schema since automated checks at the conceptual level are not feasible. Alternatively, the human designer could be expected to detect the problems at the conceptual level. Cases of interfact constraints involving replaced fact types should be reported to the designer, who should then decide where to place the constraint. Such constraints will be obvious to the system since the relational expressions which represent them will contain attributes which no longer exist in the table in which the constraint was to be enforced. Once again, information in the FD graphs may be used to give the designer some assistance. The derivation path could be presented, since the constraint must be represented on this path. Problems with intrafact constraints may also occur. Alternate methods of representing occurrence frequencies and other intrafact constraints on replaced fact types may be required.

4. **Check for lazy [4] entity types and implied mandatory roles**
   A check should be made for entities which play no roles other than those which identify them (relational tables containing only the identifying attributes of an entity). These represent lazy entity types which the designer may choose to remove. Some mandatory role constraints may have become implied. Although this is not a serious problem, they should be removed to avoid a cluttered conceptual schema diagram and unnecessary computational overheads. Information from the current and previous FD graphs may be used to help the designer locate any such entities. This process is very similar to the mandatory role checking procedure described below. Nodes on the FD graph, which previously had outgoing arcs (apart from those which represent identifiers) but now do not, are likely to represent lazy entity types. These should be reported to the designer for checking.

3.4 Check mandatory roles and other intrafact and interfact constraints

Ensure that all constraints represented on the source schema are represented on the target schema. In particular, search for previously implied mandatory role constraints which need to be shown on the schema. These occur when a fact type involving a primitive entity type which previously played only one role (apart from those which identify it) is added. Comparison between the previous and current states of the FD graph is used to draw the attention of the designer to possible cases. Nodes which corresponded to entities with implied mandatory role constraints on the previous FD graph [5] and which now have an additional outgoing, full arc should be reported. The mandatory role constraint can then be added where required, since it is no longer implied. Interfact constraints that were between a fact type replaced by a constraint and the only remaining role played by the entity involved are no longer required. These should be removed.

[4] A lazy entity type plays no roles apart from those which identify it

[5] had only one outgoing full arc apart from those which represent their identifiers

All other cases of interfact constraints involving replaced fact types should be reported to the designer for checking. Such constraints can easily be detected, as the corresponding relational expression contains attributes that no longer exist in the table on which the expression was to be enforced. Update the conceptual and relational schemata to reflect the changes made.

Steps 3 and 4 perform a similar function to Step 9 of the CSDP. However, in schema integration this function is crucial - many of the problems detected in these steps cannot arise in the simpler, special case of conceptual schema design.

Figures 4-6 illustrate the process of adding a fact type to a schema which deals with timetabling of university classes. We need to record the rooms in which subjects are held, and the times at which they are held. We add a fact type Subject is held at Time to the original schema. Note that the derivation checking procedure would not find the redundant fact type if performed before the addition of the interfact constraints. The intermediate schema produced by steps one and two of the procedure has different semantics to the final schema. Formally, the two schemata are equivalent, since the same FDs are represented by each. It is the responsibility of the human designer to decide if a fact type highlighted by DFT as potentially redundant is actually required because of semantic considerations. For the purpose of this example, we assume that the designer decides to regard them as equivalent.

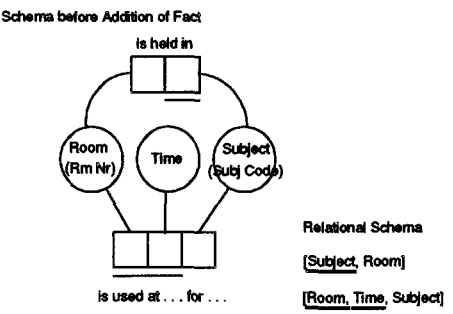

Fig. 4. The schema before addition

4 A Procedure for Removal of a Fact Type

We present a procedure which, when used to remove a fact type from a well-formed conceptual schema, ensures that the resulting schema is well-formed. This procedure should only be used when the human designer has decided to delete a fact type permanently from the schema. Thus, it is more useful for ad hoc modifications to schemata than for design or integration. The procedure for replacement of a fact type is similar, but handles the case where a fact type is

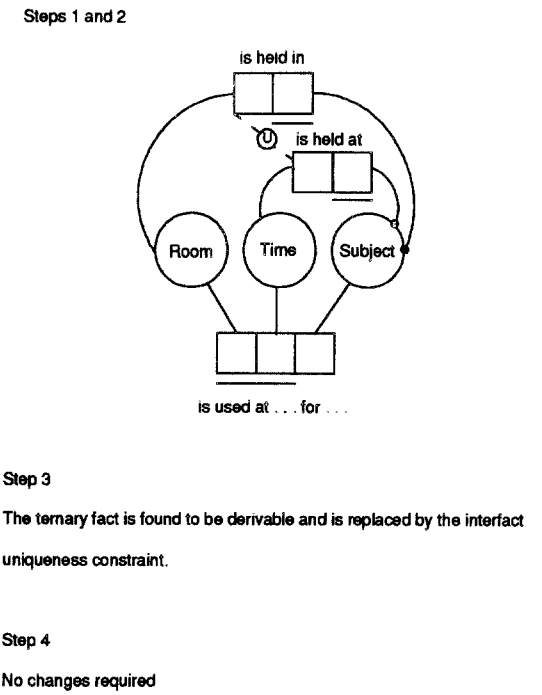

Fig. 5. The addition process

deleted because it is derivable and replaced by some other representation of the information. Therefore when a fact type is removed, some previously derivable fact types may need to be stored. The removal of a fact type which is one of two participating in an interfact uniqueness constraint may also cause this to occur. Once the fact type is removed, the interfact uniqueness constraint must be removed. Then, a new representation of the formerly derivable fact type has to be added. The procedure is as follows :

1. **Produce a conceptual schema without the fact type (and any intrafact constraints which were on it)**
   Apart from the fact type in question, this schema should be semantically equivalent to the original schema. This step should be performed by the human designer.
2. **Remove any interfact constraints that are no longer needed.**
   An interfact constraint between the deleted fact type and the only other role played by the entity concerned must be removed. In all other cases, the designer must use knowledge of the UoD to decide if the constraint is needed. A major difference from replacement of fact types is that constraints never need to be relocated.

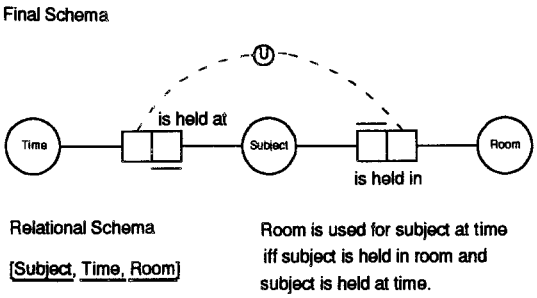

Fig. 6. The final schema - after addition

3. **Check the derivation paths of other derivable fact types.**
   There are two cases in which a previously derived fact type may need to be stored :

   - Removal of a fact type that is directly part of the derivation chain of another derived fact type

   - Removal of a fact type which takes part in an interfact uniqueness constraint (u.c), where the constraint involves only two facts. In this case the constraint itself must be removed. As with a fact type , the interfact u.c may form part of one or more derivation chains. Similar observations apply to the removal of an interfact u.c without any related changes to fact types.

   In either case, the designer must add one or more fact types or interfact uniqueness constraints to represent the information that was previously derivable. This task clearly requires creativity and knowledge of the UoD. However, an enhanced version of the DFT algorithm may be used to locate possible problems and report them to the designer. A simple comparison determines if all FDs in the previous FD graph are also in the current graph. Of course, FDs corresponding to fact types removed by the human designer should be ignored, since they are not intended to appear.
   Before returning the NIAM schema, the DFT algorithm could do the following :

   Compare the current FD graph with the one stored from the previous run : *For every FD (full arc <X, Y>) found in the stored graph that is not in the current graph, and which does not correspond to a fact type actually removed by the designer, warn the designer of possible missing information and give him/her the chance to add any fact types or constraints needed.*

Once again, the order in which the steps of the procedure are performed is important. As discussed above, constraints should be altered before derivation

checking, and final constraint checking should be performed after derivation checking.

Figure 7 shows a simple example, in which the fact types added in the previous example (Subject is held at Time) is removed. The ternary fact type then has to be added to represent perviously derivable information. Note that the schema produced is exactly the same as the schema before the fact type was added (Figure 4). .

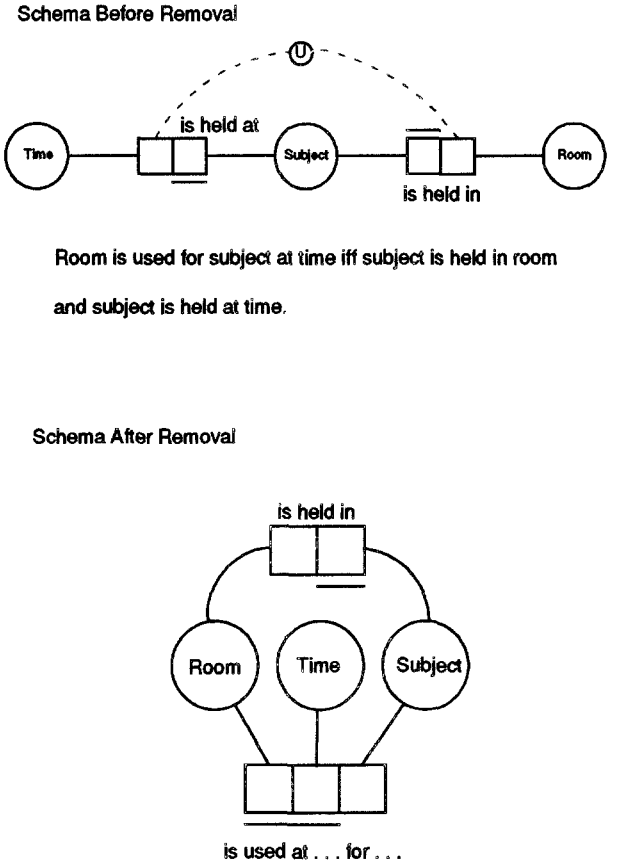

Fig. 7. Removal of a fact type

5 Conclusion

We have presented procedures for the safe addition and removal of information (fact types). These are intended as tools for the design, integration and evolution

of fact-based conceptual schemata. In fact, the procedure for addition of a fact type may be seen as a procedure for schema integration. As we have mentioned earlier, the design and integration problems may be viewed as special cases of the general schema evolution problem.

Directions for further research include the development of other evolution operators. In particular, procedures for the update of fact types, and update, addition and removal of various types of constraints need to be developed. The ultimate goal of this reserach is to develop a schema algebra, and an interactive integration operation catering for constraint integration as well as fact type integration. Futher improvements to the ONF mapping algorithm are also currently underway, as is work on formal methods for the detection of derivable fact types. Schema evaluation, and how to produce a "good" design should also be examined.

6 Acknowledgements

The authors would like to thank Dr. Yanchun Zhang for his helpful comments and suggestions. We have had many fruitful discussions with him while preparing this paper. We would also like to thank Prof. Robert Meersman and some anonymous referees for their comments, which have helped to improve the quality of this paper.

References

1. Michel Andany, José Léonard and Carole Palisser. Management of schema evolution in databases. In *Proc. 17th International Conference on Very Large Data Bases (VLDB)*, pages 161–170, 1991.
2. Bogdan Czejdo, Marek Rusinkiewicz, and David Embley. An approach to schema integration and query formulation in federated database systems. *Proceedings of the Third International Conference on Data Engineering*, pages 477–484, 1987.
3. Christoph Eick. Methodology for the design and transformation of conceptual schemas. In *Proc. 17th International Conference on Very Large Data Bases (VLDB)*, pages 25–34, 1991.
4. Terence A. Halpin and Maria E. Orlowska. Fact-oriented modelling for data analysis. *Information Systems*, 2(2):97–119, April 1992.
5. William Kent. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language. In *Proc. 17th International Conference on Very Large Data Bases (VLDB)*, pages 147–160, 1991.
6. J. Larson, S. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989.
7. Victor M. Markowitz and Johann A. Makowsky. Incremental reorganization of relational databases. In *Proc. 13th VLDB Conference*, pages 127–135, 1987.
8. S.B Navathe and S.G Gadgil. A methodology for view integration in logical database design. In *Proceedings of the Eighth International Conference on Very Large Data Bases, Mexico City*, pages 142–155, 1982.

9. S.B. Navathe, T. Sashidhar, and R. Elmasri. Relationship merging in schema integration. In *Proceedings of the Tenth International Conference on Very Large Data Bases, Singapore*, pages 78–90, 1984.
10. G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice Hall, Englewood Cliffs,New Jersey, 1989.
11. Maria E. Orlowska and C.A Ewald. Meta-level updates : The evolution of fact-based schemata. Technical Report 211, Key Centre for Software Technology,Department of Computer Science, University of Queensland, 1991.
12. Maria E. Orlowska and C.A Ewald. Schema evolution - the design and integration of fact-based schemata. In *Proc. Databases '92 : Australian Database Conference*, 1992.
13. Maria E. Orlowska and Yanchun Zhang. On enhancements of semantic methodologies for relational database design. In *Databases in the 1990s : Proceedings of the Australian Database Research Conference*, pages 97–108, 1990.
14. J. Rybnik, Z. Brzeziński, J. Getta, and W. Stępniewski. UNIBASE - an integrated access to databases. In *Proceedings of the Tenth International Conference on Very Large Data Bases, Singapore*, pages 388–396, 1984.
15. Peretz Shoval and Sara Zohn. Binary-relationship integration methodolgy. *Data and Knowledge Engineering*, pages 225–250, 1991.
16. Michael Siegel and Stuart E. Madnick. A metadata approach to resolving semantic conflicts. In *Proc. 17th International Conference on Very Large Data Bases (VLDB)*, pages 133–145, 1991.
17. K. Tanaka and Y. Kambayashi. Logical integration of locally independent relational databases into a distributed database. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, pages 131–141, 1981.