Elementary Transactions

Jan Pieter Zwart and Stijn Hoppenbrouwers

Research Group Model-Based Information Systems
Academy of Communication and Information Technology
HAN University of Applied Sciences
Ruitenberglaan 26, 6826 CC Arnhem, The Netherlands
janpieter.zwart@han.nl, stijn.hoppenbrouwers@han.nl

Abstract. Designing the *data* perspective of an information system has benefited greatly from modeling at the conceptual level. From such a model, logical data structures (ERM, Relational, DWH) can be generated automatically. In this paper, we describe how to generate the elementary building blocks of the *process* perspective from the conceptual data level. Our goal is to derive a complete set of elementary transactions from the elementary fact types and constraints at the conceptual level. Definitions of the required concepts and rules for their basic behavior are given.

Keywords: elementary transaction, operator, snapshot, parameter, precondition.

1 Introduction

1.1 Integration of Data and Process Perspective

Data modeling has benefited greatly from considering data structures and constraints at the conceptual level, ignoring all implementation considerations such as table structures (logical level) or access paths (physical level). Especially Fact Oriented Modeling (FOM) techniques that consider *elementary fact types* (attribute-free modeling: no premature grouping of facts), such as FCO-IM [1, 2] and ORM [3], have been shown to lead to reliable, validatable and verifiable data models. From such data models, different logical levels can be easily generated automatically (relational database schemas [1, ch.4], [3: ch.11], ERM schemas, UML class diagrams, DWH star schemas [3, 4, 5] etc.): see figure 1, arrow 1.

Process modeling is often done without a direct connection to a conceptual data model. Nijssen [6] was the first to link fact types from the data model to information flows in the process model at the conceptual level. However, a proper integration between data models and process models remains a problem today.

One of the main topics in our research group Model-Based Information Systems is generating applications from (the metadata of) database structures. Luursema and coworkers have built the engine IMAGine [5] that generates user interfaces from a relational database schema and additional metadata. However, this generator works on the logical level (see figure 1, arrow 2), and it still involves arbitrary choices

Y.T. Demey and H. Panetto (Eds.): OTM 2013 Workshops, LNCS 8186, pp. 513–523, 2013.

between several alternative possibilities. This shows there is still a gap in our understanding how satisfactory user tasks can be generated from metadata. The approach with elementary transactions (arrow 3 in figure 1) in combination with the results of a task analysis (arrow 4) might help to bridge this gap.

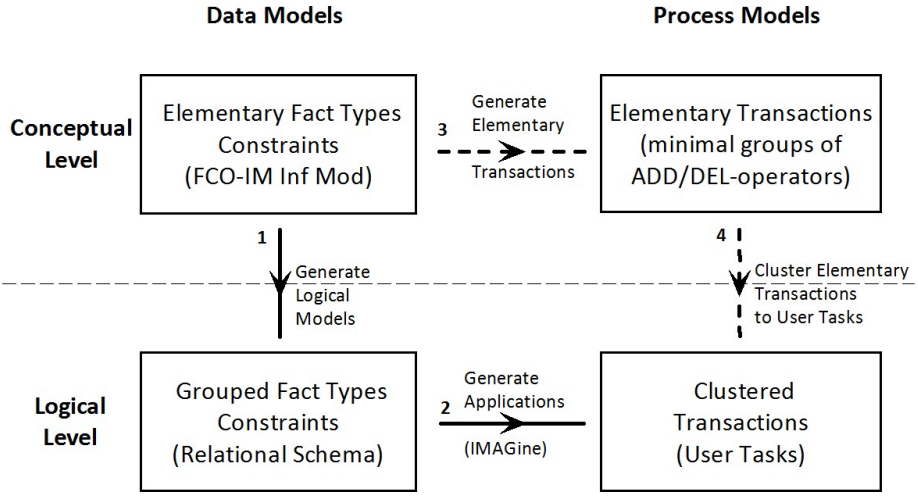

Fig. 1. Data and Process Perspectives

This paper deals only with arrow 3 in figure 1: we will show it is possible at the *conceptual* level to generate the basic building blocks of the process perspective, in the form of elementary transactions, from the metadata in the conceptual information model. These elementary transactions are the basic and indivisible building blocks that must be used for all meaningful processes the information system should provide: all user tasks will consist of clusters of one or more elementary transactions, and all elementary transactions will be a part of at least one user task.

1.2 Transactions

The concept of a transaction has always been a part of relational database systems, and every RDBMS has implemented it: see [7, p.14], [8, p.413], [9, Ch.25.2], [10, p.183], [11, p.137], [12, Ch.7], [13] to name but a few. Dietz uses the word in a broader sense [14]. Bollen [15] uses the term Exchange Rule for a similar concept.

None of these authors however discusses an *elementary* transaction, but in the Fact Oriented Modeling (FOM) world this concept has also been recognized in principle for a long time as well. Nijssen [6, p.533] discusses it briefly. Halpin mentions 'elementary update' and 'compound transaction' [3, p.35]. In an earlier edition of the same book [16, p.685], Halpin briefly discusses elementary transactions, but adds: "this approach is simply too time-consuming for data-intensive systems and is often too unstable, since business processes tend to change far more rapidly than the underlying data." This passage is absent in his more recent book [3] however.

An approach that resembles our ideas was reported by Pepels and Plasmeijer [17]. They describe a "plan to infer from an ORM [Object Role Model] groups of basic operations associated with objects/fact types […]. Such a Group we define to be a *logical unit of work* (an *luw*)". They remark "Real life ORM's with a substantial number of fact types result in an enormous amount of *luw's*.". Furthermore they use three basic operations: add, delete and update. So their approach differs from ours in essential ways: their *luw's* are not necessarily elementary, and an update operator is inconsistent with the concept of elementary facts since it operates at a sub-elementary level. To our knowledge their paper has had no follow-up to date.

Bollen [15] does not discuss the elementarity of the exchange rules he defines, though the examples he gives are indeed elementary.

In the present paper we will show in detail how elementary transactions can be defined, and indicate how they can be derived from an FCO-IM information model.

2 Materials and Methods

The concept of an elementary transaction does not depend on the modeling technique used, as long as that technique supports elementary facts. All FOM techniques can serve, and we will use FCO-IM [1, 2] to illustrate and concretize the theory.

We will develop the theory by defining the concepts and prescribing their opera-tional behavior. The following example will be used to illustrate the results.

2.1 Concrete FCO-IM Example

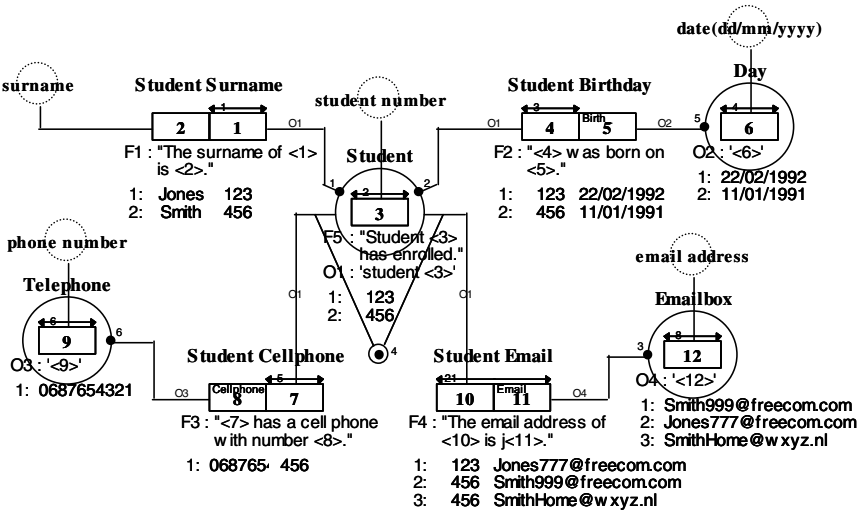

Fig. 2. Example IM

Figure 2 shows a trivially small example FCO-IM information model (see [1] for a detailed explanation of FCO-IM, freely downloadable from www.fco-im.nl). There are eight fact types, shown as named sets of small rectangles, like Student Surname. Each rectangle is called a role. Five fact types have a fact type expression (FTE), numbered F1, F2, … F5, which can be regarded as sentence types (predicates). Four fact types are nominalized [1, ch.2.4.6], (i.e. nested, enclosed in a solid circle), and can be regarded as object types (although in FCO-IM they are technically just nested fact types). All object types have an object type expression (OTE), which can be regarded as parts of sentence types, to be used in fact types attached to the object type, numbered O1, O2, … O4.

Each role can be played by a *label type* (dotted circle), which can be regarded as a source of labels, or an object type. In the latter case the name of the OTE to be used in the corresponding blank of the FTE is written next to the line joining the role and the object type.

When one tuple in the population is properly substituted into a FTE, a verbalization of one concrete elementary fact results, called a fact expression (FE). From F1, tuple 2: "The surname of student 456 is Smith.". Note that O1 has been substituted into the blank <1> in F1.

A small arrow above a role indicates a uniqueness constraints (UC): all values in the population of the role(s) under it must be unique. A big dot indicates a Totality Constraint (TC): every tuple in the population of the object type must play this role.

3 Results

3.1 Definitions

Snapshot
Each fact type can in principle have a population of 0, 1 or more tuples, depending on the constraints and the course of fact registrations.

Def1: A valid snapshot is a population of a data model at one point in time that does not violate any constraint in the model. The notation S1, S2, … will refer to particular snapshots. The population in figure 2 is a valid snapshot.

Def2: The current snapshot is the snapshot that exists just before a transaction is executed.

Operators
The building blocks of transactions are only the operators ADD and DEL. An ADD-operator will add one complete elementary fact, and a DEL-operator will delete one complete elementary fact. Parts of elementary facts cannot be changed, only a whole fact can be added or deleted. An update of a fact is to be seen as a combination of one DEL and one ADD operation, replacing the old fact with a new one.

Def3: ADD(IM, S, FTE, FE) is the operator to add in model IM, with S as the current snapshot, to the fact type with fact type expression FTE the fact expression FE.

Def4: DEL(IM, S, FTE, FE) is the operator to delete in model IM, with S as the current snapshot, from the fact type with fact type expression FTE the fact expres-sion FE.

The parameters IM and S are included to stress that the operators need access to the metadata and to the current snapshot. For brevity, when the context is clear, they can be omitted, so ADD(FTE, FE) etc. will suffice.

Clearly, the action of an operator will depend strongly on the current snapshot: for instance adding FE to FTE might be allowed with S1 as current snapshot, but might violate a constraint (e.g. a uniqueness constraint) with S2 as current snapshot.

Transactions

The reason transactions are needed at all is that the constraints in the IM will in general require combining ADD and/or DEL operators, because the population created by executing only a part of the operators in the transaction will in general yield an invalid snapshot. A common example: if an object type plays two roles (namely R1 in FTx, and R2 in FTy), and both R1 and R2 have a single-role TC and a single-role UC, then any addition of a fact Fx to FTx will also require the addition of a fact Fy to FTy. Note that the two possible sequences of these operations {ADD(FTx, Fx); ADD(FTy, Fy)} and {ADD(FTy, Fy); ADD(FTx, Fx)} both result in an invalid snapshot halfway: after executing just the first operator, the population will violate a totality constraint before the second operation is executed. Therefore it is conceptually essential to consider the transaction as a whole: a single set of operations, without a sequence.

Def5: A transaction is an operation T that transforms a given valid snapshot S1 of an IM into another valid snapshot S2 of the same IM. Notation: T(S1)=S2.

A transaction T is a set of ADD and/or DEL operators. The notation T1, T2 … will refer to particular transactions. Note that all transactions are to be defined in such a way that if T(S1)=S2 and S1 is valid, then S2 must be valid as well.

Since the execution of a single operator can involve constraints that do not apply directly to the FTE concerned, as the example with the two totality constraints above illustrates, operators cannot be executed without being embedded in a transaction, even if such a transaction consists of just this one operator.

Any two transactions T1 and T2 can be combined into a composite transaction T12, by simply performing them both. Particularly interesting therefore are transactions that contain just a minimal set of operators: leaving out at least one of these operators would fail to generate a valid new snapshot.

Def6: An elementary transaction is a transaction that contains a minimal set of ADD and/or DEL operators.

As an example, one of the elementary transactions for the IM in figure 2 would be:

```
TRANSACTION T1
    ADD(F1, FE1)                        ADD(F2, FE2)
  [ADD(F3, FE3) | ADD(F4, FE4)]         ADD(F5, FE5)
END TRANSACTION
```

Here the notation [x|y] means: choose either x or y, but not both.

This transaction would add a new student, identified by a student number (F5), surname (F1 because of TC 1), birthday (F2 because of TC 2) and either an email address or a cellphone number (F3 or F4 because of TC 4 on roles 7 and 10).

3.2 Variables and Input Parameters

In transaction T1, the same student number must be entered into roles 1, 3, 4 and [7|10]. This can be made clear by using variables and input parameters (see [15] for a similar approach, which does not specify recursive structures however):

```
TRANSACTION T1(V1(O1),V2(surname),V3(O2),V4(O3),V5(O4))
      ADD(F1, {1:V1, 2:V2})                    ADD(F2, {4:V1, 5:V3})
     [ADD(F3, {7:V1, 8:V4}) | ADD(F4, {10:V1, 11:V5})]
      ADD(F5, 3:V1[O1<3[student number]>])
END TRANSACTION
```

Explanation of notation: V1(O1) means: variable V1 must contain an OE of type O1; V2(surname) means: V2 must contain a label of type surname; 5:V3 means: variable V3 must be substituted into role 5; V1[O1<3[student number]>] means: the student number that is in blank <3> that is in OTE O1 that is in variable V1. This way of designating a constituent part of a variable, by recursively going as deeply into the nested structure as needed, is often necessary.

3.3 Preconditions

Every transaction can have one or more preconditions that specify when it can be executed (see [15] for a similar approach). Transaction T1 must concern a new student: the student number entered in the OE that goes into V1 must not yet exist in the population of object type Student. This can be expressed as V1(O1<3[student number]>) NOT IN F5<3[student number]>. We will abbreviate this to NEW(V1) below. Similar preconditions with IN (must already exist in the population, abbreviated as OLD(V1)), COUNT and other functions can be specified as well. T1 now reads:

```
TRANSACTION T1(V1(O1),V2(surname),V3(O2),V4(O3),V5(O4))
      PRECONDITION NEW(V1)
      ADD(F1, {1:V1, 2:V2})                    ADD(F2, {4:V1, 5:V3})
     [ADD(F3, {7:V1, 8:V4}) | ADD(F4, {10:V1, 11:V5})]
      ADD(F5, 3:V1[O1<3[student number]>])
END TRANSACTION
```

3.4 Working Mechanisms

Transactions

The working of a transaction T1(S1)=S2 can be seen as a simultaneous joint application of all its constituent operators to S1, resulting in S2 appearing without any intermediate stages. Of course, any actual implementation would have to use a particular

sequence of operations, even if 4GL techniques are used, but that is not of concern here. A conceptual working mechanism can be formulated in pseudo code as:

```
START TRANSACTION (T1(S1)=S2)
   Disable all constraints. Perform all operations in T1 (in any order).
   Check S2 against all constraints. IF at least one constraint is violated
   THEN give error message and rollback ELSE re-activate all constraints
END TRANSACTION
```

DEL Operators

An ADD operator needs input parameters for all the new values. A DEL operator can do with fewer parameters: if all facts about a student are to be deleted, only the student number is required. So we have chosen to minimize the number of parameters for DEL operators.

In addition, to minimize the number of required preconditions, we assume the following general behavior for a DEL operator. Suppose a DEL(FTEx, FEy) operation is to be carried out. If FEy exists in the current population of FTEx, the operator deletes the fact. If FEy does not exist in the population, it does nothing. This choice simplifies the formulation of elementary transactions considerably, because it eliminates the need to check for every DEL operation whether or not FEy exists in the current snapshot. Here is an example transaction with a DEL operator for the IM in figure 2:

```
TRANSACTION T5(V1(O1))
      PRECONDITION COUNT(F4(10)=V1)>0
      DEL(F3, 10=V1)
END TRANSACTION
```

The complete set of elementary transactions for the information model in figure 2 is given in the appendix to this paper.

4 Discussion

4.1 Population of Object Types without FTE

ADD and DEL operators only deal with one complete elementary fact at a time. Therefore only fact types with a fact type expression (FTE) have to be considered. We assume that the population of fact types without a FTE, which in FCO-IM can only be nominalized fact types (object types) [1, ch.2.5.2], is handled automatically by any system that implements this theory. Indeed CaseTalk [2], a software tool that supports creating FCO-IM information models, does this. It is a crucial property of FCO-IM that 'real-world objects' are not modeled separately, but are present only in the form of 'object expressions' in verbalizations of elementary facts.

4.2 Arbitrary Choices

We have made a few arbitrary choices, aimed at reducing the number of conditions or transactions, which could also be made differently.

The possibility to choose exactly one item from a list of alternative operators, e.g.:

```
TRANSACTION T1
    ADD(F1, FE1)                    ADD(F2, FE2)
  [ADD(F3, FE3) | ADD(F4, FE4)]     ADD(F5, FE5)
END TRANSACTION
```

The alternative would be to define two separate transactions:

```
TRANSACTION T1a                     TRANSACTION T1b
  ADD(F1, FE1)   ADD(F2, FE2)         ADD(F1, FE1)   ADD(F2, FE2)
  ADD(F3, FE3)   ADD(F5, FE5)         ADD(F4, FE4)   ADD(F5, FE5)
  END TRANSACTION                   END TRANSACTION
```

We prefer the single-transaction version, because it complies with the definition of elementary transactions (Def6): since exactly one operator must be chosen in any concrete instance of the transaction, no operator can be left out in this instance. This greatly reduces the number of elementary transactions if several multiple-role TC's are involved (a situation that is fortunately not very common in practice).

We chose to minimize the number of parameters for DEL operators. The alternative would be to define input parameters for all values concerned. This would however be unnecessary and inconvenient (see also section 4.3 below).

4.3 Update Transactions

An update of an elementary fact can be viewed as replacing the old fact with a new one, i.e.: of a DEL operation followed by an ADD operation of a complete elementary fact. Therefore an update operator is not needed. Manipulating a *part* of an elementary fact (replacing part of a fact keeping the rest the same) would mean going below the elementary fact level, and we choose to not consider sub-elementary operations. We can call a transaction that combines exactly one ADD and DEL operation on the same fact type an *update transaction.*

It can be argued that update transactions are not even needed: suppose we have an update transaction UT, which performs an update on snapshot S1: UT(S1)=S2. The same snapshot S2 can be created from S1 by performing several other elementary transactions. In the example in figure 2: to update a surname of a student: either use transaction T7 (see the appendix), or delete everything about this student using transaction T4 followed by adding all facts about this student anew using transactions T1, and possibly T2 and/or T3 as well.

The paragraph above shows that a set of elementary transactions is not necessarily a minimal set of transactions. Strictly theoretically, update transactions are not needed, but would be very convenient in practice. We choose to allow all transactions that comply with definition Def6: they must all be elementary, i.e. if at least one operation is removed, then the resulting snapshot will not be valid. Note that this is the reason that there is no update transaction for F4 in the appendix: it would not be elementary. The same result can be obtained with T3 followed by T6. In practice of course several non-elementary transactions will be defined as combinations of elementary transactions, and then an update transaction for F4 can be added, but here we are concerned only with elementary transactions.

Finally consider the update transaction below:

```
TRANSACTION T7(V1(O1), V2(surname))
      PRECONDITION OLD(V1)
      DEL(F1, 1=V1)          ADD(F1, {1:V1, 2:V2})
END TRANSACTION
```

It would seem that here a definite *order* for executing the two operators is called for: if the ADD is performed first and the DEL afterward, the result is that both facts are deleted. The reason is our choice for a minimal set of parameters for DEL operators (see section 4.2 above): if the DEL operator had two parameters, with the second one containing the old surname, there would be no problem. However we can easily preserve the conceptual orderlessness by letting an implementing system find all the 'missing' values for the delete operations in the current snapshot first, and then execute fully parameterized versions of the DEL operators with these values. So this is an implementation matter, not a conceptual one since all the necessary information is specified in the transaction as it is defined above.

4.4 Algorithm

Presently we are working on an algorithm to generate all elementary transactions from any given FCO-IM information model. We have built a proof-of-concept implementation that generates all transactions with only ADD operations for information models with a limited set of types of constraints, and are continuing with transactions with only DEL operations, and will tackle the mixed transactions with both ADD and DEL operations last.

We recognize that such an algorithm cannot be complete, unless it takes all possible (types of) constraints into account, which is impossible. We do however think it can be made complete for a specific subset of all types of constraints. We trust it will be possible that a transaction generator can generate the majority of the elementary transactions correctly, leaving only some handwork to incorporate exotic constraints.

4.5 Number of Transactions

Halpin [16] and Pepels [17] have expressed their concern that the number of elementary transactions will be very large (see the quotes in section 1.2 above). That is not our experience so far. The number of elementary transactions seems to be of the order of the number of fact types (see the small example in this paper, and preliminary results with larger information models give the same indication). Although Bollen [15] does not give all the exchange rules, he too arrives at a number of that order. Elementary transactions cannot change with varying user processes; so their number and composition depend only on the constraints in the information model.

4.6 User Interface

If a list of all elementary transactions can be generated from an information model, it will be relatively easy to generate a user interface for executing these transactions

(without grouping them into user tasks yet). This interface could for example show a list of transactions with their purpose (add a new student, update the cellphone number of a student, …). If the user chooses a transaction, the interface could present the facts to be supplied in the form of sentences with blank fields to be filled in. Such an interface would amount to a full DML at the conceptual level.

5 Conclusion

We have shown in this paper how elementary transactions can be defined in terms of sets of operators on elementary facts, with input parameters and preconditions. We have also indicated several behavioral aspects of these operators and transactions. The number of such transactions is of the order of the number of fact types.

We are developing an algorithm for generating all elementary transactions for a given FCO-IM information model, with a limited set of types of constraints. If successful, we will have generated a DML at the conceptual level.

Whether these elementary transactions will help in generating user interfaces (arrow 4 in figure 1) would be an interesting topic for further research.

Appendix: All Elementary Transactions for the Example IM

```
TRANSACTION T1(V1(O1),V2(surname),V3(O2),[V4(O3)|V5(O4)])
   PRECONDITION NEW(V1)   ADD(F1, {1:V1, 2:V2})  ADD(F2, {4:V1, 5:V3})
   [ADD(F3, {7:V1, 8:V4}) | ADD(F4, {10:V1, 11:V5})]
   ADD(F5, 3:V1[O1<3[student number]>])  END TRANSACTION

TRANSACTION T2(V1(O1), V2(O3)  PRECONDITION OLD(V1) AND
   COUNT(F3(7)=V1)=0     ADD(F3, {7:V1, 8:V3})   END TRANSACTION

TRANSACTION T3(V1(O1), V2(O4))
   PRECONDITION OLD(V1)  ADD(F4, {10:V1, 11:V2}) END TRANSACTION

TRANSACTION T4(V1(O1))  DEL(F1, 1=V1)  DEL(F2, 4=V1)
   DEL(F3, 7=V1)  DEL(F4, 10=V1) END TRANSACTION

TRANSACTION T5(V1(O1))
   PRECONDITION COUNT(F4(10)=V1)>0   DEL(F3, 10=V1) END TRANSACTION

TRANSACTION T6(V1(O1), V2(O4))    PRECONDITION COUNT(F4(10)=V1)>1 OR
   COUNT(F3(7)=V1)>0) DEL(F4, 10=V1 AND 11=V2)   END TRANSACTION

TRANSACTION T7(V1(O1), V2(surname)) PRECONDITION OLD(V1)
   DEL(F1, 1=V1)  ADD(F1, {1:V1, 2:V2}) END TRANSACTION

TRANSACTION T8(V1(O1), V2(O2)) PRECONDITION OLD(V1)
   DEL(F2, 4=V1)  ADD(F2, {4:V1, 5:V2}) END TRANSACTION

TRANSACTION T9(V1(O1), V2(O3)) PRECONDITION OLD(V1)
   DEL(F3, 7=V1)  ADD(F3, {7:V1, 8:V2}) END TRANSACTION
```

References

1. Bakema, G.P., Zwart, J.P.C., van der Lek, H.: Fully Communication Oriented Information Modeling, FCO-IM (2002), http://www.fco-im.nl
2. Wobben, M.: CaseTalk, FCO-IM modeling tool, http://www.CaseTalk.com
3. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann Publishers (2008) ISBN-13: 978-0-12-373568-3
4. Bakema, G.P., Manoku, E., et al.: FCO-IM Bridgetoolset, a software package for repository based information model transformations
5. Luursema, E.D., van Bers, A.C., Nabben, M.T.J.O.: IMAGine. In: Conference Proceedings of Information Systems, the Next Generation. HAN University (2007)
6. Nijssen, G.M.: Universele Informatiekunde. PNA Publishing bv (1993) ISBN 90-5540-001-7
7. Codd, E.F.: The Relational Model for Database Management, version 2. Addison Wesley Publishing Company (1990) ISBN 0-201-14192-2
8. Date, C.J.: An Introduction to Database Sytems, 4th edn. Addison Wesley Publishing Company (1986) ISBN 0-201-14201-5
9. Van der Lans, R.: Het SQL Leerboek, 6th edn. Academic Service (2006) ISBN 90-395-2302-9
10. De Brock, E.O.: De Grondslagen van Semantische Databases. Academic Service (1989) ISBN 90-6233-333-8
11. Ter Bekke, J.H.: Database Ontwerp. Stenfert Kroese b.v. (1988) ISBN 90-207-1659x
12. Ter Hofstede, A.H.M.: Information Modelling in Data Intensive Domains. PhD-thesis, Radboud University Nijmegen, the Netherlands (1993) ISBN 90-9006263-X
13. Proper, H.A.: A theory for Conceptual Modelling of Evolving Application Domains. PhD Thesis, Radboud University Nijmegen, the Netherlands (1994) ISBN 90-9006849-X
14. Dietz, J.L.G.: Introductie tot DEMO. Samsom Bedrijfsinformatie (1996) ISBN 90-14-05327-4
15. Bollen, P.: Fact-Oriented Modeling in the Data-, Process- and Event Perspectives. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, pp. 591–602. Springer, Heidelberg (2007)
16. Halpin, T.: Information Modeling and Relational Databases. Morgan Kaufmann Publishers (2001) ISBN-13: 978-1-55860-672-2
17. Pepels, B., Plasmeijer, R.: Generating Applications from Object Role Models. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 656–665. Springer, Heidelberg (2005)