in: Datenbanksysteme in Büro, Technik und Wissenschaft, Proc. GI/SI-Fachtagung, Zürich, März 1989, Hrsg. T. Härder, Informatik-Fachberichte 204, Springer-Verlag, 1989, S. 271-276

Ein Geschichts- und Versionsmodell für komplexe Objekte

Wolfgang Käfer
Universität Kaiserslautern

Überblick

Nicht-Standard-Datenbanksysteme (NDBS) werden zur Verwaltung komplex strukturierter Objekte, wie sie beispielsweise in Ingenieur- oder Büroanwendungen vorkommen, entwickelt. Die Integration der Zeit in das Datenmodell ist eine inhärente Forderung, wenn zusätzlich Entwurfsabläufe oder Büroprozeduren unterstützt werden sollen. Die heterogenen Anwendungen führen zu unterschiedlichen Zeitmodellen, die durch Begriffe wie "Version" oder "Geschichte" gekennzeichnet sind.

Das vorgestellte TeMA-Datenmodell erlaubt die Definition und Verarbeitung von zeitbehafteten oder versionsbehafteten Entities. Der gemeinsame Kern der Zeitmodelle und dessen Grundstrukturen und -operationen sowie deren Implementierung als Zusatzebene des NDBS PRIMA /Hä87/ werden aufgezeigt.

1. Das Geschichtsmodell

Das Geschichtsmodell basiert auf den in /Hä84/ und /Kl81/ vorgestellten Überlegungen und ist in /Kä88/ beschrieben. Die Geschichte eines Entities wird als *zeitlich geordnete Folge von charakteristischen Aussagen*, den sog. *Geschichtsausprägungen*, zusammen mit einer Auswertefunktion gesehen. Damit lassen sich sowohl zustandserhaltende (z.B. Kontostand), ereignisorientierte (z.B. Ein-/Auszahlungen auf/von einem Konto) als auch kontinuierliche (z.B. Fieberkurve) Geschichten abbilden. Der Ablauf der Geschichte wird dabei durch die *Gültigkeitszeit* geprägt, die angibt, zu welchem Zeitpunkt die Aussage in der realen Welt zu wirken beginnt. Die *Aufzeichnungszeit* beschreibt den Zeitpunkt, zu dem die Aussage in der Datenbank abgespeichert wurde und wird somit im Gegensatz zur Gültigkeitszeit vom DBS automatisch bestimmt (Revision). Sie ist von der primär interessierenden Gültigkeitszeit unabhängig (z.B. rückwirkende Lohnerhöhungen). Dies hat zur Folge, daß für die Aktualisierung von Geschichten neben den Operationen zum Erzeugen und Anfügen von Aussagen auch das Einfügen und aus praktischen Erwägungen auch das Ändern und Löschen benötigt werden. Weiterhin müssen neben den einfachen Wiedergewinnungsoperationen, die "AS-OF"- und "WALK-THRU-TIME"-Anfragen auf Geschichten realisieren, auch Operationen zur Unterstützung komplexer Anfragen (z.B. zeitbehafteter Verbund) vorgesehen werden. Die Relevanz der Gültigkeitszeit in CIM-Umgebungen wird schon in /BI87, MS83/ aufgezeigt. Sie resultiert aus der Tatsache, daß CIM die Schnittstelle zwischen den "internen" Produktionsunterlagen (Miniwelt) und dem fertigen Produkt (Welt) etabliert.

In CAD-Umgebungen wird die Gültigkeitszeit wenig beachtet, da die Miniwelt stark in sich abgeschlossen ist. Bei der Entwicklung eines "Objektes" werden meist nur Abhängigkeiten innerhalb des Objektes verwaltet.

2. Das Versionsmodell

Das Versionsmodell wurde im wesentlichen aus den in /BK85, DL88, KAC86/ und /KSW86/ vorgestellten Anforderungen entwickelt. Analog zum Geschichtsmodell müssen die Veränderungen der Entities als zunächst *ungeordnete Menge von Versionen* (= unterschiedliche Beschreibung des gleichen Entity) aufgezeichnet werden. Die *Entstehungsreihenfolge* wird meist durch eine fortlaufende Numerierung, die *Abstammungsbeziehungen* (welche Version wurde aus welcher Version entwickelt) meist in Form von azyklischen, gerichteten Graphen dokumentiert. Auch eine Darstellung wichtiger Entwurfsentscheidungen (diese Version beschreibt einen alternativen Lösungsweg) wird gefordert (*Klassifizierung* von Versionen eines Entities, beispielsweise als "Alternativen").

Die Verwaltung der Versionsmengen und der Entstehungsreihenfolge wird durch das Geschichtsmodell in natürlicher Weise unterstützt. Im folgenden wird deshalb auch nicht mehr zwischen Versionen und Aussagen bzw. Versionsmengen und Geschichtsausprägungen unterschieden. Innerhalb der Versionsverwaltung müs-

sen zusätzlich Operationen zur Manipulation von azyklischen, gerichteten Graphen und Klassen bereitgestellt werden.

Weiterhin müssen *Beziehungen zwischen Versionsmengen* (und nicht zwischen Aussagen wie im Geschichtsmodell) behandelt werden können. Diese Forderung ergibt sich aus dem Entwurfsvorgang, bei dem erst in einer späteren Verfeinerung entschieden wird, welche der Versionen eines Entities konkret verwendet werden soll. Diese sog. *generischen Referenzen* sind den Geschichtsmodellen nicht inhärent.

Zusammenfassend läßt sich festhalten, daß in unserem Geschichts- und Versionsmodell ein Entity durch folgende Begriffe beschrieben wird:

- Aussage/Version: dient zur Beschreibung eines Entities zu einem Änderungsstand
- Graph: spezifiziert Beziehungen zwischen Versionen eines Entities
- Klasse: faßt Versionen (mit gleichen Eigenschaften) eines Entities zusammen
- Referenz: wird zur Bildung von generischen und direkten Beziehungen herangezogen

Der folgende Abschnitt befaßt sich mit der Implementierung dieser Objekte im Molekül-Atom-Datenmodell (MAD-Modell, /Mi87, Mi88/). Dabei werden die Objektstrukturen durch das MAD-Modell beschrieben und die Objekte selbst als die Grundeinheiten des Temporalen MAD-Modells (TeMA-Datenmodell) vorgestellt.

3. Integration in das MAD-Modell

3.1 Skizzierung des MAD-Modells

Beim MAD-Modell handelt es sich um eine Relationenmodell-Erweiterung, deren Ziel der konsistente Übergang der Verarbeitung von homogenen zu heterogenen Satzmengen ist. Die Grundelemente des MAD-Modells sind die *Atomtypen*. Sie sind vergleichbar mit den Relationen des Relationenmodells, d.h., sie fassen *Attribute* unterschiedlichen Typs zusammen. Aus diesen Atomtypen können dynamisch *Molekültypen* gebildet werden. Ein Molekültyp ist die Zusammenfassung von Atomtypen, zwischen denen definierte *Beziehungstypen* existieren. Die Spezifikation dieser Beziehungstypen basiert auf einem vom *System unterstützten Primärschlüssel-Fremdschlüssel-Konzept*, welches durch speziell dafür eingeführte Attributtypen (IDENTIFIER und REFERENCE) realisiert wird. Jeder Beziehungstyp wird immer symmetrisch (d.h. als bidirektionale Beziehung) dargestellt, wobei sowohl (1:1) als auch (1:n)- bzw. (n:m)-Beziehungen direkt und als Teil des Atomtyps beschrieben werden können. Damit können auf Atomebene beliebige Netzstrukturen aufgebaut werden. Molekültypen können selbst wieder zum Aufbau noch komplexerer, insbesondere rekursiver Molekültypen benutzt werden.

Die Datenmanipulationssprache des MAD-Modells (MQL, Molecule Query Language) erlaubt die Verarbeitung von Molekülmengen, wobei jedes Molekül wiederum aus einer Menge von Atomen potentiell verschiedener Typen besteht. Der Typ der Moleküle entspricht dabei entweder einem bereits vordefinierten Molekültyp oder er wird dynamisch in der Anweisung festgelegt. MQL ist an SQL angelehnt und besteht analog aus einer FROM-Klausel (zur Spezifikation und Definition der für die Anweisung relevanten Molekültypen), einer WHERE-Klausel (zur Spezifikation der Restriktionen / Qualifikationsbedingungen) und einer SELECT-Klausel (zur Spezifikation der Projektionen).

3.2 Realisierung der Geschichtsausprägungen bzw. der Versionsmengen

Die Geschichtsausprägungen bilden die Einheit der Verarbeitung der Geschichts- und Versionsverwaltung und entsprechen aus dieser Sicht den Atomen des MAD-Modells. Die Darstellungsmächtigkeit der Atome ist jedoch zu gering, um zeitlich geordnete *Folgen von Aussagen* modellieren zu können. Deshalb werden die Aussagen bzw. Versionen als Atome, die Folge derselben als (generische) Moleküle gebildet. Jeder versionsbehaftete Atomtyp wird durch genau einen Molekültyp realisiert, der aus zwei Atomtypen aufgebaut ist, um zugunsten eines schnellen Zugriffs auf die aktuellen Daten, eine Trennung von aktuellen und historischen Daten zu erreichen. Die Atome eines solchen Molekültyps werden zeitlich geordnet, d.h., beginnend von den

aktuellen Daten wird eine zeitlich sortierte Liste von historischen Daten (mittels Rekursion) aufgebaut. Aus diesen Anforderungen läßt sich das MAD-Schema-Diagramm aus Bild 1 ableiten.

Der Atomtyp **?_topic** umfaßt dabei die aktuellen Daten. Über den Beziehungstyp **hist** können die nächst älteren Daten erreicht werden. Durch den Beziehungstyp **later** wird dieser Vorgang rekursiv in die Vergangenheit fortgesetzt. Nach diesem Prinzip wird jeder zeitbehaftete Atomtyp im Anwenderschema in einen entsprechenden Molekültyp umgesetzt. Bild 2 zeigt dies an einem Beispiel.

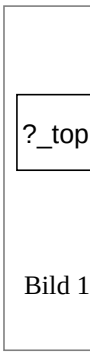

Bild 1

Die Beschreibung des versionsbehafteten Atomtyps **bsp** (mit einem baumartigen Abstammungsgraphen **extraction**, ohne Klassen) im TeMA-Datenmodell wird im MAD-Modell durch eine Molekültypdefinition ersetzt. Die in der DEFINE MOLECULE_TYPE-Anweisung angegebene FROM-Klausel gibt die Struktur des Moleküls an. Es wird eine "lineare Liste" erzeugt, deren Anker durch ein Atom des Typs **bsp_topic** und deren folgende Elemente durch Atome des Typs **bsp_hist** gebildet werden. Die Kardinalitätsrestriktion "(0,1)" besagt, daß 0 oder 1 **bsp_hist**-Atom über die Beziehung **hist** erreicht werden kann. Die RECURSIVE-Klausel bewirkt, daß die Liste solange in die Vergangenheit expandiert wird, bis keine weiteren Elemente mehr erreichbar sind. Die Angabe "(0,*)" hätte eine Struktur in Art eines "Pointer-Array" definiert, dessen Reihenfolgeinformation aber innerhalb der MQL nicht effizient einzusetzen gewesen wäre.

TeMA-Datenmodell

```
CREATE TEMPORAL_ATOM_TYPE  bsp
    (extraction : TREE; NO_CLASSES)
    (bsp_id  : IDENTIFIER,
         .  (* benutzerdefinierte Attribute *)
                .                );
```

Abbildung ===⇒

MAD-Modell

```
DEFINE MOLECULE_TYPE bsp : ALL

      FROM bsp_topic.hist-bsp_hist

      RECURSIVE bsp_hist.later-bsp_hist
```

Definition der den Molekültyp konstituierenden Atomtypen im MAD-Modell:

```
CREATE   ATOM_TYPE bsp_topic
(bsp_id            : IDENTIFIER,
valid              : TIME,
alive              : BOOLEAN,
extraction_succ    : REFERENCE (bsp_topic.extraction_pred) (0,*),
extraction_pred    : REFERENCE (bsp_topic.extraction_succ) (0,1),
hist               : REFERENCE (bsp_hist.top) (0,1),
                   :
                   : (* benutzerdefinierte Attribute *)
                   :                );
```

```
CREATE ATOM_TYPE bsp_hist
(bsp_v_id  : IDENTIFIER,
valid      : TIME,
top        : REFERENCE (bsp_topic.hist) (0,1),
later      : REFERENCE (bsp_hist.earlier) (0,1),
earlier    : REFERENCE (bsp_hist.later) (0,1),
           :
           :
           : (* benutzerdefinierte Attribute *)
           :                );
```

Bild 2: Definition eines versionsbehafteten Atomtyps

Die vordefinierten Molekültypen, die als Makros gesehen werden können, sind innerhalb der MQL wie Atomtypen verwendbar. Bei der Übersetzung der MQL-Anweisungen werden zunächst die Makros expandiert und anschließend die Projektions- und Selektionsklausel angewendet. In /Kä88/ wird diese Vorgehensweise auf die Selektionsklausel übertragen: es werden spezielle Prädikate zur Formulierung zeitbehafteter Anfragen bereitgestellt, die zur Übersetzungszeit auf Selektions- und Projektionsbedingungen auf den konstituierenden Atomen der Geschichtsausprägungen expandiert werden. Analog können spezielle Operatoren zur Verwaltung der Graphen und Klassen gebildet werden. In Bild 3 wird die Expansion für den Fall auf-

gezeigt, daß der Zustand von Geschichtsausprägungen zu einem bestimmten Zeitpunkt (in der Vergangenheit) ermittelt werden soll (der Zugriff auf eine bestimmte Version ist ein Sonderfall dieser Anfrage). Durch die UNTIL-Klausel wird die Evaluierung des Rekursivmoleküls bei Erreichung der benötigten Aussage abgebrochen. PREVIOUS bewirkt, daß diese Aussage noch zum Ergebnismolekül hinzugenommen wird. Von diesem gebildeten Rekursivmolekül wird nur die "letzte" Ausprägung (LAST) zur Bildung des Ergebnisses herangezogen (zustandserhaltende Geschichte).

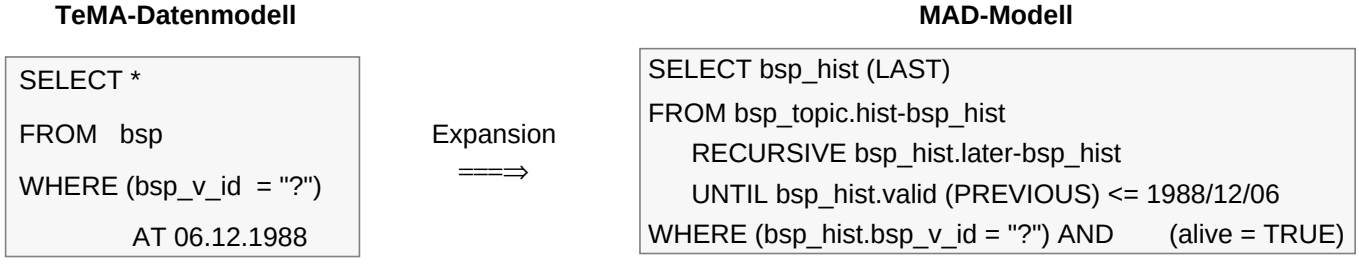

Bild 3: Expandierung einer Anfrage zur Bestimmung einer bestimmten Version

3.3 Realisierung direkter und generischer Referenzen

Die Bildung von Molekültypen geschieht im MAD-Modell durch die Definition von Beziehungstypen zwischen Atomtypen. Die Erweiterung der Atomtypen zu versionsbehafteten Atomtypen, die als Molekültypen realisiert sind, bewirkt, daß sich auch die Qualität der Referenzen ändert: an Stelle von Atomen müssen jetzt Moleküle (in ihrer Gesamtheit) referenziert werden. Eine Realisierung dieser Molekülreferenzen als Atomreferenzen auf das Wurzelatom des Moleküls genügt nicht. Bild 4 zeigt einen versionsbehafteten Molekültyp **ABC**, der durch drei versionsbehaftete Atomtypen **A**, **B** und **C** gebildet wird. Jeder versionsbehaftete Atomtyp wird gemäß Kapitel 3.2 durch einen entsprechenden Molekültyp realisiert, der aus je einem **topic**- und einem **hist**-Atomtyp gebildet wird. Bild 4 zeigt die Abbildung sowie eine mögliche Ausprägung, die nur aktuelle Daten enthält. Wegen der besseren Übersichtlichkeit werden die bidirektionalen Referenzen im Gegensatz zum MAD-Schema-Diagramm aus Bild 1 im folgenden durch ungerichtete Kanten dargestellt.

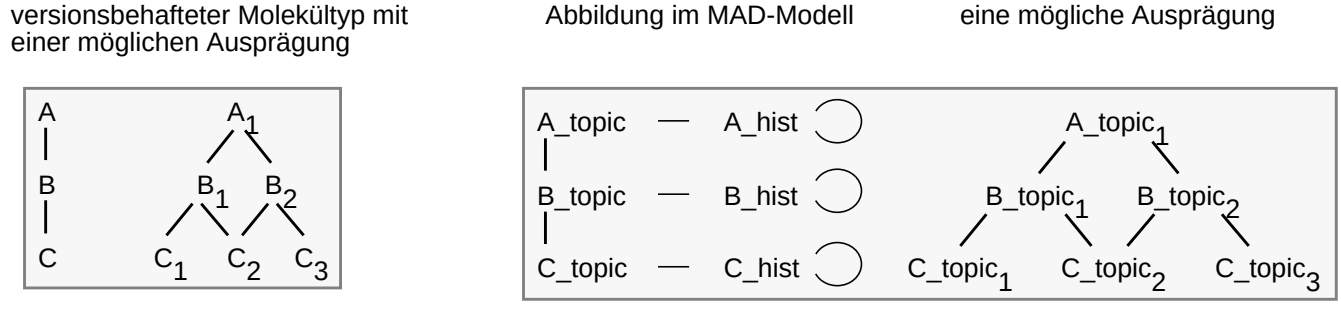

Bild 4: Beispiel eines versionsbehafteten Molekültyps zusammen mit einer Ausprägung

Werden nicht-strukturbildende Attribute beispielsweise in $\mathbf{B\_topic}_2$ geändert, so ergibt sich die in Bild 5a dargestellte Änderung der Molekülausprägung. Wird allerdings ein strukturbildendes Attributpaar geändert, so daß beispielsweise die Beziehung zwischen $\mathbf{C\_topic}_2$ und $\mathbf{B\_topic}_2$ aufgelöst wird, so zeigt sich, daß die Referenzen auf die Wurzelatome nicht ausreichend sind. Im vor der Änderung aktuellen Atom $\mathbf{B\_hist}_{21}$ müssen Referenzen auf $\mathbf{C}_2$ und $\mathbf{C}_3$ enthalten sein, die auf die jeweiligen Wurzelatome $\mathbf{C\_topic}_2$ und $\mathbf{C\_topic}_3$ abgebildet werden. Wegen der Symmetrie der Referenzen müssen somit auch Referenzen von **C_topic**-Atomen auf **B_hist**-Atome möglich sein. Damit ergibt sich die Darstellung aus Bild 5b. Bei dieser Art der Referenzen handelt es sich um generische Referenzen. Soll eine konkrete Version verwendet werden, so muß sie noch innerhalb der Versionsmenge bestimmt werden, beispielsweise durch die Versionsnummer, durch eine bestimmte Eigenschaft usw. Zur Realisierung der direkten Referenzen wird eine weitere Beziehung

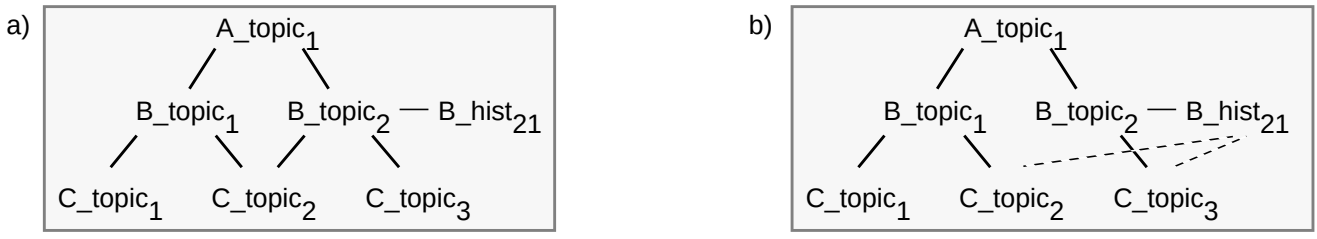

Bild 5: Änderung eines nicht-strukturbildenden und eines strukturbildenden Attributs in B_topic_2

zwischen den **hist**-Atomtypen benötigt. Damit ergibt sich das in Bild 6 dargestellte MAD-Schema-Diagramm.

Die molekülbildenden Referenzpaare zwischen **A**, **B** und **C** bzw. zwischen **A_topic**, **B_topic** und **C_topic** realisieren Beziehungen zwischen Entities, d.h. Mengen von Versionen und damit also generische Referenzen. Die Referenzpaare zwischen den **topic**- und den **hist**-Atomtypen sowie die rekursive Beziehung auf den **hist**-Atomtypen werden zur Bildung der versionsbehafteten Atomtypen benötigt. Das Referenzpaar zwischen den **hist**-Atomtypen realisiert direkte Referenzen.

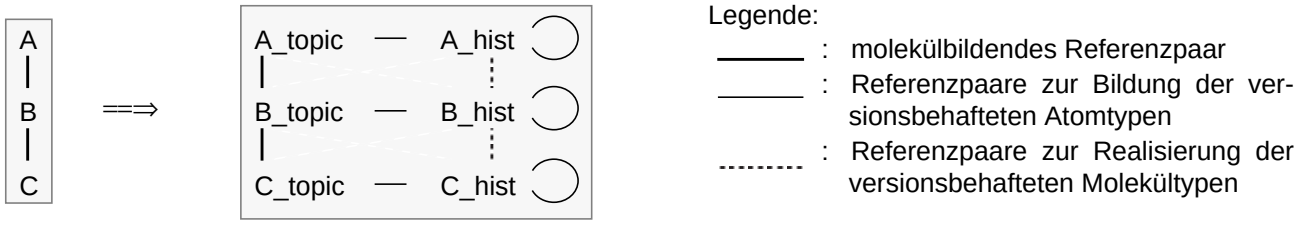

Bild 6: Realisierung von Molekülreferenzen

Anfragen auf versionsbehafteten Molekültypen werden analog den Anfragen auf versionsbehafteten Atomtypen behandelt:

1. Expansion der Molekültypdefinitionen (rekursiv).
2. Expansion der Prädikate (rekursiv) und Zuordnung zu den entsprechenden Atomtypen.
3. Die expandierte Anfrage entspricht einer herkömmlichen MQL-Anweisung. Das Ergebnis muß ggf. entsprechend den Vorgaben des Geschichts- oder Versionsmodells aufbereitet werden.

Eine exakte Beschreibung der Implementierung der Graphen und Klassen muß aus Platzgründen unterbleiben. Beide Konzepte können jedoch leicht mit dem Beziehungskonzept des MAD-Modells ausgedrückt werden. Ähnlich dem Rekursivmolekültyp zur Bildung der versionsbehafteten Atomtypen kann ein Rekursivmolekültyp für Graphen angegeben werden. Das Auffinden aller Vorgänger oder Nachfolger eines Graphknotens läßt sich auf eine elementare MQL-Anweisung zur Selektion des Referenzattributes (vgl. **extraction_pred** bzw. **extraction_succ** in Bild 2) abbilden. Klassen können analog behandelt werden.

Zusammenfassung

Die Integration der Zeit in ein Datenmodell für Non-Standard-Datenbanksysteme stellt einen wichtigen Schritt zur Unterstützung der Handhabung von Entwurfsobjekten dar. Dazu wurde ein Konzept vorgestellt, das die auf den ersten Blick heterogenen Bereiche "Zeit in Datenbanksystemen" und "Versionsmodelle zur Unterstützung von Ingenieuranwendungen" gleichermaßen befriedigt. Das vorgestellte Geschichts- und Versionsmodell, das auf einer gemeinsamen Struktur zur Verwaltung zeitbehafteter Daten beruht, wurde in das MAD-Modell, das die Beschreibung komplex strukturierter Objekte erlaubt, integriert.

Danksagung:

Ich möchte mich bei Herrn Prof. Dr. Härder für die Anregung bedanken, mich mit dieser Thematik zu befassen.

Literatur

Ar86: Ariav, G.: A Temporally Oriented Data Modell, ACM TODS, Vol. 11, No. 4, Dec. 1986, Pages 499-527

BI87: Blanken, H., Ijbema, A.: Storage of Versioned Objects in a CIM Environment, Int. Conf. on Data and Knowledge Systems for Manufactoring and Engineering, Hartford, Connecticut, 1987, Pages 65-74

BK85: Batory, D.-S., Kim, W.: Modeling Concepts for VLSI CAD Objects, ACM TODS, Vol. 10, No. 3, Sept. 1985, Pages 322-346

DL88: Dittrich, K.-R., Lorie, R.-A.: Version Support for Engineering Database Systems, IEEE TOSE, Vol. 14, No. 4, April 1988, Pages 429-437

Hä84: Härder, Th.: Überlegungen zur Modellierung und Integration der Zeit in temporalen Datenbanksystemen, Forschungsbericht Nr. 19/84, SFB 124, Universität Kaiserslautern, 1984

Hä87: Härder, Th., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA A DBMS Prototype Supporting Engineering Applications, Proc. of the 13th Int. Conf. on VLDB, Brighton, Great Britain, 1987, Pages 433-442

Kä88: Käfer, W.: Ein Modell zur Integration der Zeit in relationalen Datenbanksystemen, SFB-Bericht Nr. 27/88, SFB 124, Universität Kaiserslautern, 1988

KAC86: Katz, R.-H., Anwarrudin, M., Chang, E.:A Version Server for Computer-Aided Design Data, ACM IEEE 23rd Design Automation Conf., Las Vegas, June 1986, Pages 27-33

KCB86: Katz, R.-H., Chang, E., Bhateja, R.: Version Modeling Concepts for Computer-Aided Design Databases, Int. Conf. on Management of Data, ACM SIGMOD RECORD, Vol. 15, No. 2, Washington, Pages. 379-386

Kl81: Klopprogge, M.-R.: TERM An Approach to Include the Time Dimension in the Entity-Relationship Model, Proceedings of the 2nd Int. Conf. on Entity-Relationship Approach, 1981, Pages 477-512

KSW86: Klahold, P., Schlageter, G., Wilkes, W.: A General Model for Version Management in Databases, Informatik Berichte Nr. 58, Fern-Universität, Hagen, März 1986

Lu84: Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H., Woodfill, J.: Designing DBMS Support for the Temporal Dimension, Proc. of the SIGMOD Int. Conf. on Management of Data, 1984, Pages 115-130

Mi87: Mitschang, B.: MAD - ein Datenmodell für den Kern eines Non-Standard-Datenbanksystems, GI-Fachtagung "Datenbanksysteme für Büro, Technik und Wissenschaft", Darmstadt, IFB Nr. 136, Springer-Verlag, Berlin, Heidelberg, 1987, S. 180-195

Mi88: Mitschang, B.: Das Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf, Implementierung -, Diss., Universität Kaiserslautern, IFB Nr. 185, Springer-Verlag, Berlin, Heidelberg, März 1988

MS83: Müller, Th., Steinbauer, D.: Eine Sprachschnittstelle zur Versionenkontrolle in CAM Datenbanken, in: Schmidt, J.-W.: Sprachen für Datenbanken, IFB Nr. 72, Springer-Verlag, Berlin, Heidelberg, 1983, S. 76-95

Sn87: Snodgrass, R.: The Temporal Query Language TQuel, ACM TODS, Vol. 12, No. 2, June 1987, Pages 247-298