Generierung maßgeschneiderter Relationenschemata in Softwareproduktlinien mittels Superimposition

Martin Schäler[1], Thomas Leich[2], Norbert Siegmund[1], Christian Kästner[3], und Gunter Saake[1]

[1] Fakultät für Informatik, Universität Magdeburg, Deutschland
{schaeler, nsiegmun, saake}@cs.uni-magdeburg.de,
[2] METOP Forschungsinstitut, Magdeburg, Deutschland
thomas.leich@metop.de
[3] Fachbereich Mathematik und Informatik, Philipps Universität Marburg, Deutschland
kaestner@informatik.uni-marburg.de

Zusammenfassung Die Erstellung eines individuellen Programms aus einer Softwareproduktlinie (Programmfamilie) erfordert auf Anwendungs- und Datenbankseite einen speziell angepassten und aufeinander abgestimmten Funktionsumfang. Die Modellierung maßgeschneiderter Relationenschemata stellt z.B. aufgrund der großen Anzahl an Programmen, die aus einer Produktlinie erstellt werden können, eine Herausforderung dar. Wir präsentieren einen Lösungsvorschlag zur Modellierung und Generierung von maßgeschneiderten Relationenschemata mittels Superimposition. Wir zeigen anhand einer realen, produktiv eingesetzten Fallstudie welche Vorteile unser Ansatz in den Bereichen Wartung und Weiterentwicklung erzeugt und welche Herausforderungen beispielsweise durch redundant definierte Schemaelemente existieren.

1 Einleitung

Die Entwicklung von langlebigen Datenbank (DB)-Schemata stellt seit jeher eine Herausforderung dar [21]. Moderne Anforderungen an Softwareimplementierungen in Bezug auf Wiederverwendbarkeit und individualisierten Funktionsumfang eröffneten auf Anwendungsseite ein weites Forschungsgebiet und führten zu einer Vielzahl von Lösungsvorschlägen [11]. Hingegen sind die Auswirkungen im DB-Schemabereich weitestgehend unbekannt [26]. Vor allem im Kontext von Softwareproduktlinien [19] (SPL) entstehen neue Herausforderungen für die Entwicklung von DB-Schemata. SPLs werden häufig zur Erstellung individuell auf den Nutzer zugeschnittener Produkte eingesetzt [2]. In ihr repräsentieren *Merkmale* eine für den Kunden sichtbare Funktion der Software [11]. Eine SPL zur Erstellung eines maßgeschneiderten Dokumentenverwaltungssystems kann beispielsweise die Merkmale *Nutzerverwaltung* und *Mehrbenutzersynchronisation* enthalten, die unabhängig als Extrafunktionalität vermarktet werden. Entsprechend der Kundenwünsche wird die Dokumentenverwaltungssoftware mit oder ohne diese Merkmale generiert (Variante der Software). Je nach Funktion des Merkmals ist es auf die Existenz gewisser DB-Schemaelemente zur Speicherung und Verarbeitung der für das Merkmal relevanten Daten angewiesen. Die Nutzerverwaltung des Dokumentenverwaltungssystems benötigt z.B. Relationen in denen die Nutzer mit den zugehörigen Rechten

hinterlegt sind. Häufig wird bisher ein globales Schema unabhängig von der Merkmalauswahl ausgeliefert. Die derzeit verwendeten Vorgehen erzeugen die folgenden Probleme:

- Es wird ein *komplexes, schwer verständliches Gesamtschema* erzeugt, von dem große Teile in vielen Varianten nicht benötigt werden. Dies erschwert die *Wartbarkeit* eines Variantenschemas und die *Weiterentwicklung* der Software.
- Die *Ausdrucksfähigkeit der Modellierung* wird beschränkt. Es können beispielsweise Relationen in eine variable Anzahl von Fragmenten partitioniert werden oder es kann kein Gesamtschema angeben werden, wenn die Modellierung konkurrierende Schemaelemente enthält.
- Es entstehen Probleme in Bezug auf die *Integrität* der enthaltenen Daten.
- Die *Skalierbarkeit* des Ansatzes muss gewährleistet sein, da die Anzahl der generierbaren Varianten exponentiell zur Zahl der Merkmale wächst.

Wir sind hingegen daran interessiert, für jede Variante der Software ein maßgeschneidertes DB-Schema zu generieren. Langfristig beabsichtigen wir die folgenden Zielstellungen zu erreichen:

- Ziel ist es den *Funktionsumfang* des DB-Schemas *individuell* auf den Kunden abstimmen. Dazu gehört u.a. die *Optimierung* auf bestimmte Anfragetypen (Exact Match, Range Queries, etc.) oder die Einbeziehung von *Sicherheitsaspekten*, wie Verschlüsselung der Daten oder Information Hiding.
- Ähnlich zur Anwendungsseite sollen DB-Elemente in Varianten der SPL *wiederverwendet* werden. Das bedeutet, dass Teile der Modellierung auf DB-Schemaebene in ähnlichen Produkten erneut verwendet werden, um den *Aufwand* zur Erstellung von Varianten zu verringern.
- Die *Verständlichkeit* der Modellierung sowie einzelner Variantenschemata der Software soll verbessert werden, um sowohl die *Weiterentwicklung* der Software, als auch die *Wartung* von Varianten zu vereinfachen.

In einem ersten Schritt zur Erfüllung der Ziele wird untersucht, inwiefern sich Varianten eines Schemas für eine reale Fallstudie in einem konkreten Datenmodell erstellen lassen. Weiterhin wird überprüft welche Probleme dabei auftreten, um daraus Erkenntnisse abzuleiten, inwiefern die Komplexität eines solchen Vorgehens einen Einsatz in der Praxis erlaubt. Aufbauend auf den Ergebnissen von SIEGMUND et. al. [26], die auf der BTW 2009 einen ersten Ansatz zur Erstellung maßgeschneiderter ER-Schemata [8] präsentierten, schlagen wir ein solches Vorgehen für das Relationenmodell [10] vor. Hierbei handelt es sich unseres Wissens nach um die erste Arbeit ein maßgeschneidertes DB-Schema für ein konkretes Datenmodell und für eine reale, produktiv eingesetzte Fallstudie zu erzeugen, um die Praxistauglichkeit des Vorgehens besser beurteilen zu können. Wir untersuchen die folgenden Problemstellungen:

1. Wie können variable DB-Schemata für das Relationenmodell im Kontext von SPLs modelliert werden?
2. Wie lässt sich aus der Modellierung ein maßgeschneidertes DB-Schema für eine Variante der SPL erstellen?

3. Inwiefern existieren Interaktionen von Merkmalen auf DB-Schemaebene und welche Auswirkungen haben sie auf die Erzeugung eines Variantenschemas?
4. In welchem Umfang zeigen sich die erkannten Probleme bei der Anwendung unseres Lösungsvorschlags an einer produktiv eingesetzten Fallstudie?

2 Die Fallstudie ViT-Manager

Bei der Fallstudie ViT®-Manager[4] handelt es sich um ein Controlling-Tool mit dessen Hilfe es möglich ist Verbesserungspotentiale in einem Unternehmen zu erfassen, zu klassifizieren und die Ausschöpfung des Potentials nachzuvollziehen. Die Fallstudie verfügt ohne externe Bibliotheken über ca. 50.000 Zeilen Quellcode, das DB-Schema enthält 54 Relationen mit 309 Attributen. Sie wird produktiv sowohl in Mittelstandsunternehemen, als auch in Großkonzernen eingesetzt. Weiterhin sind die einsetzenden Unternehmen in verschiedenen Branchen angesiedelt. Aus den heterogenen Einsatzgebieten ergeben sich unterschiedlichste Anforderungen an die einzelnen Varianten der Fallstudie. In Großkonzernen stehen beispielsweise das Aufdecken von Synergien, umfangreiche Reportingfunktionen sowie die Unterstützung möglichst vieler *Methoden zur Ausschöpfung der vorhandenen Verbesserungspotentiale (MAP)* im Vordergrund. Hingegen legen Mittelstandsunternehmen vor allem Wert auf eine konsequente und nachvollziehbare Ausschöpfung der aufgedeckten Potentiale anhand monetärer Größen, wie z.B. eingesparter Arbeitszeit.

2.1 Die ViT-SPL

Die Fallstudie ViT-Manager entstand aus einer monolithischen Anwendung, die aufgrund von neuen Kundenanforderungen durch zusätzliche Funktionen erweitert wurde. Die zunehmende Komplexität und die unterschiedlichen Anforderungen der Kunden ließen daher eine Überführung der monolithischen Anwendung in eine SPL sinnvoll erscheinen. Bei einer SPL handelt es sich um eine Familie von Programmen, die eine Anzahl gleicher und unterschiedlicher Merkmale besitzt [9] und häufig zur Erstellung individueller Software genutzt wird [2]. Ein Merkmal beschreibt dabei eine für den Nutzer wesentliche Funktion der Software [11]. Die Reportingfunktion oder die Nutzerverwaltung des ViT-Managers sind Beispiele für ein solches Merkmal. Die Generierung eines individuellen Produktes erfolgt durch Auswahl der benötigten Merkmale. Das Produkt wird anschließend durch eine zuvor definierte Erstellungsstrategie generiert. In ViT erfolgt dies durch eine Plugin-Architektur [13].

2.2 Das Merkmalmodell der Fallstudie

Die zur Verfügung stehenden Merkmale und deren Beziehungen untereinander werden im Merkmalmodell zusammengefasst und im Merkmaldiagramm visualisiert [14]. Das Modell beschreibt die Variabilität einer SPL. Es können z.B. Beziehungen zwischen Merkmalen und Bedingungen, wie Merkmal *A benötigt B*, definiert werden. Die Variabilität einer SPL besagt, welche Kombinationen von Merkmalen zu einer gültigen

[4] Verbesserung im Team (ViT) ist ein eingetragenes Warenzeichen der Karer Consulting AG.

Variante führen und somit welche Funktionen miteinander kombiniert werden können. Die Anwendung wurde, wie in Abb. 1 gezeigt, in Merkmale zerlegt um Varianten der

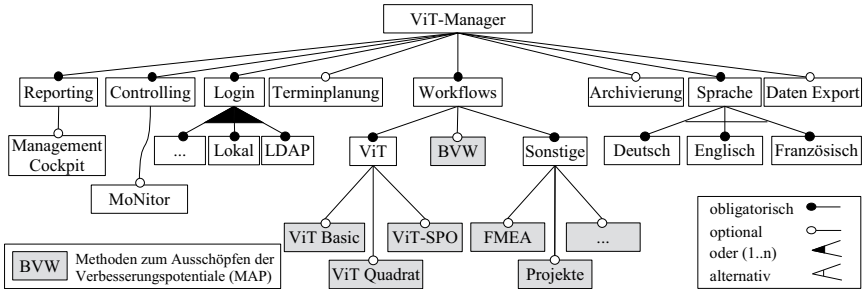

Abbildung 1. Ausschnitt des Merkmaldiagramm des ViT-Managers Version 1.6.9b

SPL generieren zu können. Kern der Anwendung ist die Unterstützung der verschiedenen *MAPs*, die in einem eigenem Merkmal implementiert und als Blätter des Workflowteilbaums zu erkennen sind. Es handelt sich um optionale Merkmale, die nicht in jeder Variante der Software enthalten sein müssen. Weiterhin kann der ViT-Manager entweder auf *Deutsch, Englisch oder Französisch* ausgeliefert werden, unterstützt auf Wunsch ein selbst definierbares *Reporting* und verfügt über weitere optionale Funktionen wie *Archivierungsfunktionen*, bei denen die Daten zusätzlich anonymisiert werden können. Die Anmeldung eines Nutzers an die Software ist auf verschiedenen Wegen, wie *LDAP* oder *Lokaler Anmeldung*, möglich, von denen mehrere gleichzeitig in einer Variante enthalten sein können. Insgesamt können mehrere tausend valide Varianten der SPL erstellt werden.

Bei der Zerlegung der Anwendung in Merkmale wurde festgestellt, dass die Verwendung eines monolithischen DB-Schemas Probleme wie mangelnde Übersichtlichkeit, ungenutzte Schemaelemente und die fehlende Unterstützung alternativer Merkmale mit sich bringt. Die Probleme auf DB-Schemaebene werden nachfolgend allgemeingültig beschrieben, bevor wir in Abschnitt 6 zur Fallstudie zurückkehren, wobei sie solange als durchgängiges Beispiel genutzt wird.

3 Problembeschreibung

Unterschiedliche Anwender haben verschiedene Anforderungen an Softwareprodukte. Diese werden auf Anwendungsseite beispielsweise durch Varianten einer SPL erfüllt. Es existiert eine Vielzahl von Ansätzen, die die Generierung von Varianten einer Software z.B. mittels plugin-basierten Frameworks [13] erlaubt. Hingegen sind die bisherigen Untersuchungen auf DB-Schemaebene nicht ausreichend [26]. Bei DB-Schemata handelt es sich um langlebige Softwareprodukte, die nicht selten länger genutzt werden als die Anwendungen in deren Kontext sie entwickelt wurden [21]. Somit kommt ihrer Entwicklung besondere Bedeutung zu.

3.1 Grenzen derzeitiger Ansätze

Nachfolgend werden die derzeit verwendeten Lösungsansätze vorgestellt und erläutert welche in der Einleitung genannten Probleme bei ihnen auftreten.

Verwendung eines globalen Schemas. Häufig wird bisher für jede Variante einer SPL ein statisches, globales und historisch gewachsenes DB-Schema unabhängig von der Merkmalauswahl ausgeliefert [26]. Es enthält sämtliche Schemaelemente, die jemals in einer der Varianten benötigt werden. Bei entsprechender Komplexität der Software entsteht somit ein *hochkomplexes, schwer verständliches* DB-Schema. Die Auslieferung eines monolithischen Schemas kann dazu führen, dass in einer Variante Schemaelemente enthalten sind, die nicht benötigt werden. Somit entstehen, je nach Merkmalauswahl, ungenutzte Tabellen und Attribute. Dies führt zu *Integritätsproblemen* bei leeren Spalten auf denen beispielsweise Fremdschlüssel, Check oder Not Null Bedingungen definiert sind oder es wird gänzliche auf Schemaelemente zur Integritätssicherung verzichtet. Darüber hinaus erschwert ein solches globales Schema das Verständnis und beeinträchtigt die *Wartung und Weiterentwicklung*. Ein weiterer entscheidender Nachteil besteht darin, dass nicht in jedem Fall ein globales Schema angegeben werden kann. Dies ist der Fall, wenn in einer SPL alternative, sich gegenseitig ausschließende Merkmale existieren. Diese können sich *widersprechende Schemaelemente* definieren, die nicht gleichzeitig in einem globalen Schema enthalten sein können.

Sichtbasierte Ansätze. Das Problem der fehlenden Variabilität auf DB-Schemaebene wurde bereits erkannt und Sichten aus dem globalen Schema generiert, die den Varianten einer SPL entsprechen [7]. Das globale Schema wird dennoch in jeder Variante der SPL ausgeliefert. Die Generierung solch maßgeschneiderter Sichten verringert die Komplexität eines Schemas nicht, sondern verbirgt sie lediglich. Die Sichten *erhöhen die Komplexität* des Schemas für den Entwickler und während der Wartung, da sie zusätzliche Schemaelemente einführen, anstelle nicht benötigte zu entfernen. Somit wird die Verständlichkeit des Gesamtschemas negativ beeinflusst.

Lösung in Frameworks. In Frameworks werden häufig Namensräume verwendet, die verhindern, dass die einzelnen Plugins, welche als Merkmale angesehen werden können, sich gegenseitig beeinflussen. Somit können plugin-übergreifend *keine Integritätsbedingungen* verwendet werden oder es werden zusätzliche Abhängigkeiten wie Plugin A benötigt B erzeugt. Die Sicherung der Konsistenz erfolgt daher meist auf Anwendungsebene [26]. Aufgrund von Fehlern in der Anwendung oder durch direkten Zugriff auf die DB ist es daher dennoch möglich inkonsistente Daten in der DB zu hinterlegen. Weiterhin führt dieses Vorgehen dazu, dass jedes Plugin für eine semantisch identische Relation eine *einzelne Partition* dieser anlegt. Legen beispielsweise drei optionale Merkmale horizontale Partitionen der Relation Mitarbeiter z.B. für verschiedene Unternehmensbereiche an, ist die Ausgabe der Gesamtmitarbeiterzahl des Unternehmens nicht trivial. Ein Reportingmerkmal müsste je nachdem welche Merkmale in der Variante enthalten sind, auf die einzelnen Partitionen zugreifen. Es muss dazu wissen, in welchen Merkmalen solche Partitionen angelegt werden und welche davon in der Variante existieren. Ein solches Vorgehen erzeugt daher einen deutlich erhöhten Aufwand auf Anwendungsseite.

3.2 Modellierung eines maßgeschneiderten Schemas

Ein intuitiver Ansatz zur Modellierung maßgeschneiderter DB-Schemata besteht darin für jede Variante der Software ein eigenes Relationenschema zu modellieren. Der Aufwand eines solchen Vorgehens ist in der Praxis jedoch nicht zu beherrschen. Die Zahl der Varianten wächst exponentiell zur Anzahl der Merkmale. Die Fallstudie ViT verfügt über 26 Merkmale aus denen sich mehrere tausend valide Varianten generieren lassen. Daher ist es nicht möglich für jede Variante der Software ein eigenes Schema zu entwerfen, sondern es müssen Teile der Modellierung wiederverwendet werden können.

Abgrenzung der Betrachtung. In unserem Ansatz wird ein maßgeschneidertes Relationenschema erstellt, hierfür wird ein Ausschnitt des Relationenmodells betrachtet. In der Modellierung sind vorerst die folgenden Elemente der Datendefinitionssprache des SQL:1999 Standards [1] enthalten: *Relationen, Attribute, der Wertebereich eines Attributs und die Definition des Primärschlüssels einer Relation*. Weiterhin wird zur Vereinfachung der Betrachtung davon ausgegangen, dass die Modellierung der ersten Normalform genügt. Es ist geplant zukünftig weitere Schemaelemente zu unterstützen. Wir konzentrieren uns in dieser Arbeit auf das DB-Schema, weitere Themen wie beispielsweise Variabilität auf Datenebene werden nicht betrachtet.

3.3 Generierung eines Variantenschemas

Aus der Modellierung soll das individuell zugeschnittene DB-Schema erzeugt werden. Es muss eine Technik zur Generierung des Variantenschemas gewählt werden, welche die folgenden Anforderungen erfüllt.

Vollständigkeit. Alle Schemaelemente, die von den gewählten Merkmalen benötigt werden, müssen im Schema der Variante enthalten sein.

Komplexitätsreduktion. Es sollen ausschließlich benötigte Schemaelemente im Variantenschema vorhanden sein, um die Wartung dieses zu vereinfachen.

Komponierbarkeit. Ein Merkmal muss die Modellierung eines anderen erweitern können. Insbesondere soll es möglich sein Attribute zu Relationen hinzuzufügen ohne dass bekannt ist, in welchen Merkmalen die Relation definiert wurde, so dass keine Kenntnisse über den Aufbau der anderen Merkmale vorhanden sein müssen. Dies erleichtert sowohl die Weiterentwicklung als auch die Wartung.

Korrektheit. Alle Schemaelemente müssen in der korrekten Ausprägung im Variantenschema vorhanden sein. Das bedeutet, dass alle Attribute eines Merkmals den in der Modellierung definierten Wertebereich erhalten und sie Bestandteil des Primärschlüssels der Relation sind, wenn dies in der Modellierung angegeben wurde.

3.4 Interaktion von Merkmalen auf DB-Schemaebene

Das Problem der Interaktion von Merkmalen ist ein wohlbekanntes Problem auf Anwendungsseite und schränkt die Wiederverwendbarkeit von Softwareartefakten ein [15,20]. Es entsteht, wenn zwischen mehreren im Merkmalmodell unabhängigen Merkmalen, aufgrund der Implementierung, Abhängigkeiten bestehen. Das bedeutet, dass einige laut

Merkmalmodell valide Varianten der SPL durch die zusätzlichen Abhängigkeiten der Implementierung nur dann generiert werden können, wenn die zusätzlichen Abhängigkeiten aufgelöst werden. Bezogen auf das Beispiel des Dokumentenverwaltungssystems benötigt das Merkmal *Nutzerverwaltung*, nur dann wenn ebenfalls das optionale Merkmal *Mehrbenutzersynchronisation* gewählt ist, zusätzliche Funktionen. Es muss beispielsweise feststellen können, welche Nutzer (exklusive) Schreibsperren auf Dokumenten aufheben können. Es wird daher auf folgende Fragestellungen eingegangen:

- Existiert das Problem der Interaktion von Merkmalen auf Modellierungs- oder Implementierungsebene des DB-Schemas in unserem Lösungsansatz?
- Welchen Einfluss hat es auf die Generierung von Variantenschemata?
- Wie und in welchem Umfang äußert sich dieses Problem in der Fallstudie?

4 Modellierung eines maßgeschneiderten DB-Schemas

In unserem Ansatz wird für jedes Merkmal festgehalten, welche Schemaelemente ein Merkmal benötigt und aus dieser Modellierung für jede Variante ein maßgeschneidertes Schema komponiert. Die Idee besteht darin, dass jedes Merkmal dem Variantenschema die zusätzlich von ihm benötigten Schemaelemente hinzufügt und mehrfach definierte Elemente dennoch lediglich einmal im Ergebnis vorhanden sind. Somit muss für jedes Merkmal ein Teilschema modelliert werden und nicht für jede Variante ein komplettes DB-Schema.

4.1 Zerlegung des DB-Schemas in Merkmale

Es existieren zwei Möglichkeiten ein variables Schema zu modellieren. Es kann von Anfang an als variables Schema entworfen werden oder es wird aus einem zuvor verwendeten globalen Schema erzeugt. In der Fallstudie ist bereits ein monolithisches Gesamtschema vorhanden, deshalb konzentrieren wir uns auf diesen Fall. Bei der Restrukturierung der Anwendung in eine SPL muss daher ebenfalls das DB-Schema in Merkmale untergliedert werden. Zu diesem Zweck wird nachfolgend ein allgemeingültiger Zerlegungsprozess vorgestellt. Der von uns vorgeschlagene Prozess untergliedert sich in die folgenden Teilprobleme.

1. Bestimmung der Merkmale in die das Schema zerlegt wird.
2. Definition eines Kriteriums, wann ein Schemaelement einem Merkmal zugeordnet wird.
3. Ausführen der Zerlegung anhand des zuvor bestimmten Kriteriums.

Bestimmung der Merkmale in die das Schema zerlegt wird. Auf DB-Schemaebene werden dieselben Merkmale, wie auf Anwendungsseite verwendet, da für die Anwendung ein maßgeschneidertes Schema erstellt werden soll. Somit enthält ein Merkmal sowohl die Implementierung des Anwendungs-, als auch die des DB-Teils einer für den Nutzer wichtigen Funktionalität. Eine Variante der Software kann wie bisher durch Auswahl der notwendigen Merkmale im Merkmalmodell der SPL erfolgen und muss nicht für Anwendungs- und DB-Teil separat erfolgen [24].

Wann wird ein Schemaelement einem Merkmal zugeordnet? Aus der Anforderung, dass für jede Variante einer Software ein maßgeschneidertes DB-Schema erstellt werden soll, ergibt sich, dass in dem Teilschema eines Merkmals alle Schemaelemente vorhanden sein müssen, die das Merkmal zur korrekten Funktionsweise benötigt.

Definition 1. *Ein Schemaelement wird einem Merkmal genau dann zugeordnet, wenn das Element innerhalb des merkmalspezifischen Quellcodes lesend oder schreibend verwendet wird.*

Das Merkmal ViT Basic trägt beispielsweise die Beschreibung eines Verbesserungspotentials in das Attribut *beschreibung* der Relation *probleme* ein, daher wird die Relation mit dem genannten Attribut in die Modellierung des Merkmals aufgenommen. Vorkommen von *SELECT* *-Anweisungen werden durch die Signatur sämtlicher in der FROM-Klausel aufgeführten Relationen ersetzt. Analog wird bei *INSERT INTO* Statements ohne Attributliste verfahren.

Ausführen der Zerlegung. Nach Definition 1 müssen alle DB-Aufrufe der Software identifiziert und den entsprechenden Merkmalen zugeordnet werden. Dies kann entweder über spezielle Eigenschaften der Implementierung auf Anwendungsseite der SPL, die das Auffinden solcher Stellen im Quellcode erleichtern oder durch statische Analyse der Anwendung [25] erfolgen. In der Fallstudie erfolgt jeder DB-Zugriff über das global definierte DB-Objekt, so dass die Zugriffe mittels der vorhandenen Suchfunktion der verwendeten Entwicklungsumgebung identifiziert werden können.

4.2 Modellierung eines Merkmals

Die Modellierung des Schemas eines Merkmals erfolgt mit Hilfe einer einfachen textuellen Syntax. In der Definition 1 wird erläutert, wann ein Schemaelement in die Modellierung eines Merkmals aufgenommen wird. Das Kriterium stellt sicher, dass alle benötigten Schemaelemente in ihr vorhanden sind.

In der Abb. 2 ist die Syntaxrepräsentation der Merkmale *Login* und *Lokale Anmeldung* aus Abb. 1 angegeben. Die Modellierung des Merkmals *Login* enthält eine Relation *login_daten* mit den Attributen *deaktiviert*, *loginname* und *personalnummer* mit den jeweiligen Wertebereichen. Über den Primärschlüssel Personalnummer wird der Loginname eindeutig einem Benutzer der Software zugeordnet. Der eigentliche Passwortabgleich erfolgt jedoch durch das Merkmal *Lokale Anmeldung* indem das für den Loginnamen angegebene Passwort mit dem in der Datenbank hinterlegten verglichen wird. Daher benötigt das Merkmal ebenfalls eine Relation *login_daten* und legt die beiden Attribute *loginname* und *passwort* an. Somit ist z.B. das Attribut *loginname* redundant in den Teilschemata beider Merkmale vorhanden.

Alternativ dazu könnte für jedes Merkmal eine eigene Relation angelegt werden. Dies führt jedoch zu den selben Problemen, wie sie in 3.1 für Frameworks beschrieben wurden.

4.3 Interaktionen auf Modellierungsebene

Interaktionen auf Modellierungsebene existieren in zwei Formen, die nachfolgend näher erläutert werden. Sie treten auf, wenn

Login

```
1 RELATION login_daten(
2   deaktiviert          int(1)          ,
3   loginname            varchar(255)    ,
4   personalnummer       varchar(255) PK
5 );
```

Lokale Anmeldung

```
1 RELATION login_daten(
2   loginname            varchar(255) ,
3   passwort             varchar(255)
4 );
```

Abbildung 2. Modellierung zweier Merkmale mit redundanten Schemaelementen

- semantisch identische Schemaelemente von mehreren Merkmalen redundant definiert werden;
- aufgrund der Kombination der Merkmale die Modellierung eines Merkmals angepasst werden muss, da zusätzliche Schemaelemente benötigt werden.

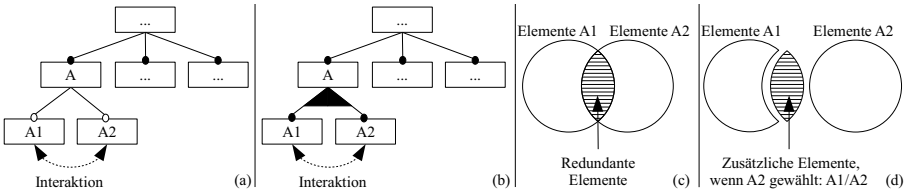

Abbildung 3. Interaktionen auf Modellierungsebene

Redundant definierte Schemaelemente. Redundant definierte Schemaelemente treten auf, wenn beispielsweise zwei zueinander optionale Merkmale (Abb. 3 (a)) oder Merkmale, die über eine oder-Beziehung (Abb. 3 (b)) miteinander verbunden sind, dieselben Schemaelemente benötigen. Da Varianten existieren in denen lediglich eines der beiden Merkmale vorhanden ist, muss jedes Merkmal alle benötigten Schemaelemente in seiner Modellierung enthalten. Redundanzen treten in dem hier betrachteten Ausschnitt des Relationenmodells auf Attributebene auf, wenn zu einer Relation dasselbe Attribut in mehreren Merkmalen definiert wird. Mehrfach definierte Relationen können unterschiedliche Attribute enthalten, so dass dieser Fall nicht als redundante Definition angesehen wird. In ViT existieren solche Interaktionen beispielsweise zwischen den Blättern des Workflowteilbaums aus Abb. 1. Hierbei handelt es sich um die Merkmale, welche die Implementierung der verschiedenen *MAPs* enthalten. Sie sind zueinander optional und enthalten eine große Anzahl an redundanten Schemaelementen (Abb. 3 (c)). Redundante Schemaelemente erzeugen die folgenden Probleme.

- Für jede redundante Definition muss sichergestellt werden, dass jedes Attribut denselben Wertebereich erhält und keine widersprüchlichen Modellierungen entstehen, aus denen kein valides Schema generiert werden kann.
- Die redundanten Schemaelemente erhöhen die Komplexität des Schemas eines Merkmals und beeinträchtigen somit die Verständlichkeit der gesamten Modellierung.

Zusätzlich benötigte Schemaelemente. Bei einer bestimmten Kombination von gewählten Merkmalen ist es, aufgrund der Interaktion dieser Merkmale, notwendig zusätzliche Schemaelemente zum Variantenschema hinzuzufügen. In der Fallstudie existiert beispielsweise eine Funktion zur *Archivierung* der Daten der einzelnen *MAPs*. Je nachdem welche MAPs in der Variante vorhanden sind, müssen die entsprechenden Archivtabellen angelegt werden. Die Lösung alle theoretisch benötigten Schemaelemente in die Modellierung der *Archivierung* aufzunehmen erzeugt die in 3.1 beschriebenen Probleme eines globalen Schemas. In unserem Ansatz werden die zusätzlich benötigten Schemaelemente (siehe Abb. 3 (d)) der Modellierung des Merkmals durch *Derivatives* [17] hinzugefügt. Hierbei handelt es sich um zusätzliche Merkmale, die dem Variantenschema automatisch hinzugefügt werden, wenn die Merkmale auf die sie sich beziehen gleichzeitig in der Variante enthalten sind. Das Derivative *A1/A2* wird dem Variantenschema automatisch hinzugefügt, wenn die Merkmale A1 und A2 in einer Variante vorhanden sind.

5 Generierung eines Variantenschemas

Es wurde bisher festgehalten, dass ein Merkmal auf DB-Schemaebene ein relationales Teilschema enthält. Die Generierung einer Variante der Software erfolgt durch Auswahl der benötigten Merkmale anhand der zuvor festgelegten Erstellungsstrategie. Nachfolgend wird veranschaulicht, wie die Komposition eines Variantenschemas mittels Superimposition (Überlagerung) in unserem Lösungsvorschlag erfolgt. Es wird weiterhin untersucht welche Probleme während der Komposition auftreten können.

5.1 Superimposition - Sprachunabhängiger Kompositionsalgorithmus

Die Superimposition beschreibt einen sprachunabhängigen Kompositionsalgorithmus, bei dem mehrere Merkmale durch Mischen ihrer zugrunde liegenden (Sub) Strukturen vereinigt werden [3]. Diese Technik wird beispielsweise in der Sichtintegration [5] und der merkmal-orientierten Softwareentwicklung [6] verwendet. Für die Superimposition von Merkmalen (•-Operator) existiert von APEL et al. eine Algebraformalisierung die zeigt, dass die Operation assoziativ und im allgemeinen Fall nicht kommutativ ist [4]. Ein Variantenschema (S) wird mittels Superimposition aller gewählten Merkmale (m_i) durch den Operator • erzeugt und repräsentiert selbst wieder ein Merkmal (M):

$$\bullet\colon M \times M \to M \qquad\qquad S = m_n \bullet \ldots \bullet m_2 \bullet m_1 \tag{1}$$

Struktur eines Merkmals auf DB-Schemaebene. Die Komposition zweier Merkmale basiert auf der Vereinigung von *Merkmalstrukturbäumen* (MSB), welche die interne Struktur eines Merkmals repräsentieren und als vereinfachter abstrakter Syntaxbaum angesehen werden können [3]. In der Abbildung 4 ist der MSB des Merkmals *Login* basierend auf der Modellierung aus Abb. 2 abgebildet. Die Wurzel des Baumes verdeutlicht, dass es sich um die Modellierung des DB-Schemas handelt, somit kann das Merkmal sowohl das DB-Schema, als auch die Anwendungsimplementierung enthalten. Die Kinder des Wurzelknotens enthalten alle vom Merkmal benötigten Relationen. Im

angegebenen Beispiel definiert das Merkmal die Relation *login_daten*. Auf der Blattebene befinden sich die Attribute der Relationen. Sie verweisen auf die Implementierung des Schemaelementes. Sie enthält den Wertebereich und die Angabe ob das Attribut Teil des Primärschlüssel der Relation ist. Der Grafik ist zu entnehmen, dass das Merkmal Login die Relation *login_daten* sowie die Attribute *deaktivert*, *loginname* und *personalnummer* enthält. Die Implementierung des Attributs *personalnummer* definiert den Wertebereich als *varchar(255)* und zeigt, dass das Attribut einziger Bestandteil des Primärschlüssels der Relation ist.

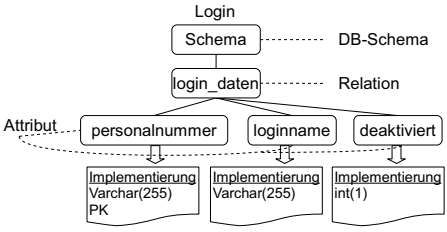

Abbildung 4. Merkmalstrukturbaum des Login-Merkmals

5.2 Vereinigung zweier Merkmale

Die Superimposition zweier Merkmale erfolgt, wenn beide Merkmale ausgewählt wurden und daher Bestandteile der zu erstellenden Variante sind. Der Kompositionsprozess beginnt an den Wurzeln der zu vereinigenden MSBs und erfolgt dann rekursiv im gesamten Baum. Zwei Knoten werden miteinander verschmolzen, wenn ihre Elternknoten miteinander verschmolzen wurden bzw. es sich um Wurzelknoten handelt und sowohl Name als auch Typ übereinstimmen. In der Abb. 5 wird das Verschmelzen zweier Merkmale in Gleichung (2) am Beispiel der bereits bekannten Merkmale *Login* und *Lokale Anmeldung* veranschaulicht, welche als Teil einer zu erstellenden Variante ausgewählt wurden.

$$m_{Lokale\ Anmeldung} \bullet m_{Login} = m_{Ergebnis} \tag{2}$$

Das Ergebnis besteht aus dem Teilschema des Merkmals *Login*, welches um das zusätzliche Attribut *passwort* des Merkmals *Lokale Anmeldung* ergänzt wurde. Das redundant definierte Attribut *loginname* ist im Ergebnis, wie beabsichtigt, einmal enthalten. Problematisch ist der Fall, wenn auf Blattebene ein Attribut mehrfach definiert ist. Dieser Fall ist im Beispiel 5 durch das Attribut *loginname* gegeben, es repräsentiert das Problem der Interaktion von Merkmalen auf Implementierungsebene (siehe 3.4).

5.3 Interaktionen auf Implementierungsebene: Verschmelzung von Blattknoten

Das Problem der Interaktion von Merkmalen tritt auf Implementierungsebene in unserem Ansatz auf, wenn Attribute redundant definiert wurden (siehe 4.3). Für das Problem der

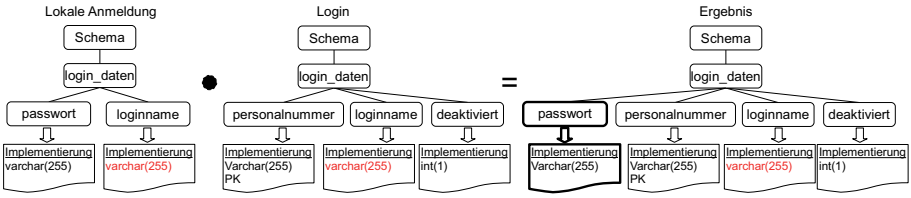

Abbildung 5. Superimposition: Beispiel für Merkmale mit Redundanzen

Verschmelzung von Blattknoten besteht die Möglichkeit entweder die Komposition von Blattknoten gänzlich zu verbieten oder Kompositionsregeln für diese anzugeben [4]. Das Beispiel der Merkmale *Login* und *Lokale Anmeldung* zeigt, dass Interaktionen zwischen den Merkmalen in der Praxis benötigt werden, so dass ein Verbot der Verschmelzung von Blattknoten zu restriktiv ist und nachfolgend Kompositionsregeln angegeben werden.

Kompositionsregeln zur Verschmelzung von Attributen. Im untersuchten Ausschnitt des relationalen Datenmodells befinden sich auf Blattebene die Attribute einer Relation. Sie enthalten den Wertebereich und die Definition ob ein Attribut Teil des Primärschlüssels der Relation ist. Die Kompositionsregel für Primärschlüssel besagt, sobald das Attribut in mindestens einem der beiden Merkmale als Schlüssel deklariert wurde, bleibt es Bestandteil des Primärschlüssels der Relation. Durch den von uns in 4.1 vorgestellten Zerlegungsalgorithmus eines globalen Schemas können keine Probleme durch die Erweiterung des Primärschlüssels auftreten. Wird das Schema von Beginn an als variables Schema entworfen, kann es vorkommen, dass durch die Erweiterung des Schlüssels durch ein Merkmal in einem anderen beispielsweise ein *INSERT* Statement nicht mehr funktionieren, weil es keine Kenntnis über das zusätzliche Schlüsselattribut hat und daher einen NULL-Eintrag für dieses erzeugt. In solchen Fällen wird empfohlen Surrogatschlüssel zu verwenden. Ist dies nicht sinnvoll, muss das Merkmal welches den Schlüssel erweitert die DB-Zugriffe aller weiteren Merkmale entsprechend der neuen Primärschlüsseldefinition anpassen. Bei der Komposition von Wertebereichen werden zwei Fälle unterschieden.

Fall 1: Attribute mit identischem Wertebereich. Im ersten Fall wird der Wertebereich in das Ergebnis übernommen. Somit ist die Implementierung des Wertebereichs aller Knoten identisch. Dieser Fall tritt in Abb. 5 auf. In beiden Merkmalen ist für das Attribut *loginname* der Wertebereich *varchar(255)* angegeben und wird daher in das Ergebnis übernommen.

Fall 2: Attribute mit unterschiedlichem Wertebereich. Für den Fall, dass die Wertebereiche unterschiedlich sind, wird davon ausgegangen, dass es sich hierbei um einen Fehler handelt. Eine Superimposition der beiden Merkmale in Abb. 6 ist beispielsweise nicht möglich, da für das Attribut *loginname* unterschiedliche Wertebereiche angegeben sind. Alternativ könnte eine Regel angeben werden, so dass eine der beiden Modellierungen übernommen wird indem sie eine vorherige überschreibt. Die Erfahrungen zeigen je-

doch, dass es sich hierbei (bisher) immer um eine Inkonsistenz der Modellierung eines Merkmals gehandelt hat, die behoben werden musste.

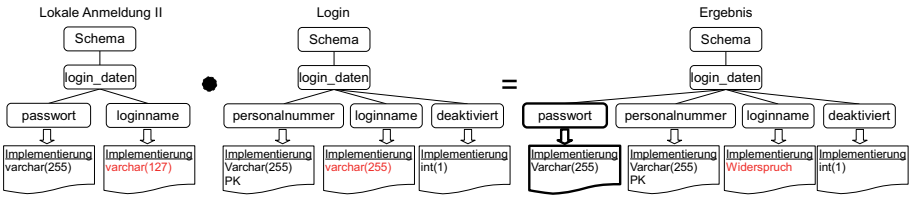

Abbildung 6. Beispiel für Merkmale mit widersprüchlichen Definitionen

6 Anwendung des Ansatzes auf das DB-Schema der Fallstudie ViT

Wir haben bisher gezeigt, wie mittels Superimposition aus den verschiedenen Teilschemata der Merkmale ein Variantenschema generiert wird und welche Probleme dabei auftreten können. In diesem Abschnitt wird das Vorgehen an einer realen, nicht trivialen Fallstudie angewendet, um zu ermitteln welche Vorteile das Verfahren mit sich bringt und in welchem Umfang Probleme durch Interaktionen der Merkmale in der Fallstudie auftreten.

6.1 Probleme des globalen Schemas in der Fallstudie

Laut der Befragung des METOP ViT-Entwicklerteams ergeben sich aus der Verwendung eines globalen Schemas in den einzelnen Varianten des ViT-Managers folgende Probleme, die durch die Verwendung unseres Verfahrens gelöst werden sollen:

Umständliche Wartung einer Variante. Die Wartung der produktiv eingesetzten Varianten wird im First-Level vom Kunden selbst übernommen. Die Auslieferung des globalen Schemas erschwert die Verständlichkeit desselben. Falls Daten beispielsweise versehentlich gelöscht wurden, muss zur Wiederherstellung aller Daten in einem konsistenten Zustand das komplette DB-Schema verstanden werden. Momentan ist dies ein komplexer Prozess, da nicht benötigte Schemaelemente vorhanden sind und in dem historisch gewachsenen Schema nicht sofort ersichtlich ist, auf welche Relationen sich die gesuchten Daten verteilen.

Aufwendige Weiterentwicklung der Software. Die Weiterentwicklung der Software erzeugt Probleme, da die Identifizierung welche Merkmale ein Schemaelement benötigen äußerst aufwendig ist. Wird ein Schemaelement während der Weiterentwicklung der Software modifiziert, kann dies Seiteneffekte auf den Anwendungsteil anderer Merkmale ausüben. Es mussten bisher für alle Varianten der Anwendung überprüft werden, inwiefern sie mit dem globalen Schema nach dessen Modifikation kompatibel waren.

Inkonsistente Daten. Erste Versuche Teile des Schemas aus Varianten zu entfernen, führten zu Problemen mit Integritätsbedingungen. In einigen Varianten konnten in einzelnen Relationen keine Daten eingefügt werden, weil der entsprechende referenzierte Datensatz nicht vorhanden war oder Null-Werte für Attribute übertragen wurden, auf denen Not-Null Bedingungen definiert waren. Das Löschen dieser Integritätsbedingung im globalen Schema führte anschließend zu inkonsistenten Daten, die zuvor einen Abbruch der Transaktion erzeugt hätten.

6.2 Das variable DB-Schema

Das monolithische DB-Schema des ViT-Managers wurde, wie in 4.1 beschrieben in Merkmale zerlegt, so dass jedes Merkmal die von ihm benötigten Schemaelemente enthält. In der Tabelle 1 wird die Größe der Modellierung der einzelnen Merkmale angegeben. Die Modellierung des *ViT-Merkmals* enthält beispielsweise 88 Attribute, die sich auf 16 Relationen verteilen. Die Syntaxdarstellung des *ViT-Merkmals* (siehe 4.2) benötigt 135 Zeilen Quellcode.

Tabelle 1. Größe der Merkmalmodellierung

Merkmal	Attribute	Relationen	Codezeilen
ViT	88	16	135
ViT-SPO	80	14	121
BVW	75	12	110
ViT Quadrat	65	12	100
ViT Basic	63	12	98
Terminplanung	35	7	61
…			
LDAP-Login	1	1	3

Aus der Modellierung können maßgeschneiderte Variantenschemata für die einzelnen Varianten des ViT-Managers erstellt werden. In der Abb. 7 sind die Größen der Variantenschemata visualisiert. In der Maximalkonfiguration, in der alle zur Verfügung stehenden optionalen Merkmale ausgewählt wurden, verfügt das DB-Schema über 309 Attribute. Wird das Merkmal *BVW* nicht gewählt, sind im Variantenschema 234 Attribute vorhanden, wird zusätzlich zum *BVW ViT-SPO* nicht gewählt 210 usw. Sind die Merkmale *BVW, ViT-SPO, ViT Quadrat, ViT Basic und Terminplanung* nicht in der Variante enthalten, wird die Anzahl der Attribute nahezu halbiert. In der Minimalkonfiguration, in der keines der optionalen Merkmale vorhanden ist, sind 146 Attribute enthalten.

6.3 Evaluierung der Ergebnisse der Zerlegung des DB-Schemas

Bei Anwendung des Ansatzes an der Fallstudie stehen Verbesserungen im Bereich Wartung von Varianten und Weiterentwicklung der Software im Vordergrund. Weiterhin soll die Existenz inkonsistenter Daten in Varianten verhindert werden. Eine quantitative Analyse, etwa die Messung der durchschnittlichen Bearbeitungszeit der Wartungsaufträge, ist aufgrund der unterschiedlichen Komplexität der einzelnen Aufträge derzeit

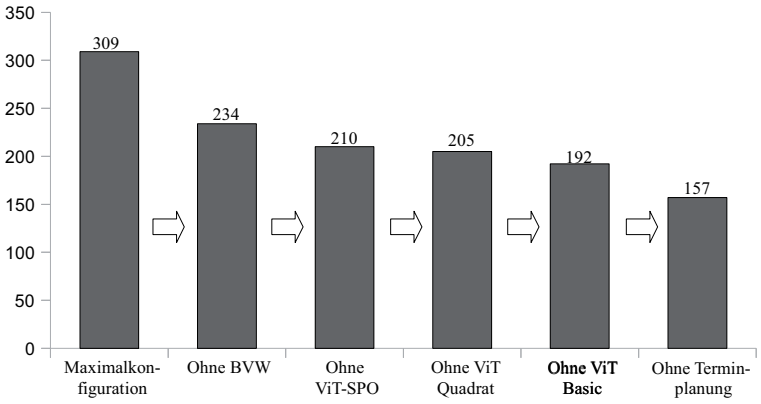

Abbildung 7. Anzahl der Attribute in den Variantenschemata

nicht möglich. Wir nutzen erneut Erkenntnisse aus einer Befragung des METOP ViT-Entwicklerteams.

Wartung einer Variante. Die Wartbarkeit von Variantenschemata konnte deutlich verbessert werden, indem die Komplexität der Variantenschemata reduziert wurde. In Abb. 7 wird veranschaulicht, dass die Größe des Variantenschemas je nach Merkmalauswahl halbiert werden kann. Zusätzlich ist bekannt welche Schemaelemente ein Merkmal verwenden. Somit können z.B. die betroffenen Relationen eines versehentlich gelöschten *MAP-Datensatzes* durch den First-Level Support beim Kunden schneller aufgefunden und durch Umsetzen des Löschbits wiederhergestellt werden.

Weiterentwicklung der Software. Die Weiterentwicklung der Software wurde ebenfalls vereinfacht. Das optionale Merkmal zur *Archivierung* der *MAP-Daten* wurde nach der Zerlegung des DB-Schemas in Merkmale implementiert. Durch die Zerlegung war bekannt, welche Daten der *MAPs* archiviert werden müssen, so dass die Implementierung des DB-Schemas des Merkmals und insbesondere der Derivatives, die die zusätzlich benötigten Schemaelemente enthalten, spürbar vereinfacht wurde. Ohne die Zerlegung des Schemas in Merkmale hätten die entsprechenden *MAP* Schemaelemente für jedes Derivative mühsam im Quellcode des *MAP-Merkmals* identifiziert werden müssen. Ähnliche positive Effekte wurden bei der Implementierung des *Daten-Exports* beobachtet.

Inkonsistente Daten. Auf die Wiedereinführung der zuvor gelöschten Integritätsbedingungen wurde bislang verzichtet. Jedoch ist geplant zukünftig Integritätsbedingungen, wie Fremdschlüssel, Not-Null, etc. in die Betrachtung mit einzubeziehen. Wir erhoffen uns ähnlich gute Ergebnisse wie bei den anderen beiden Punkten.

6.4 Interaktionen der Merkmale

Die nach 4.1 erzeugte Zerlegung des monolithischen DB-Schemas erlaubt die Erstellung von Variantenschemata. Nachfolgend wird untersucht in welchem Grad Interaktionen

unter den Merkmalen auftreten. Somit ist es möglich den Einfluss des Problems beispielsweise in Bezug auf Konsistenz und Verständlichkeit der Modellierung besser Einschätzen zu können.

Interaktionen durch redundante Schemaelemente. Redundanzen existieren ausschließlich auf Attributebene, da sie sich bei der Superimposition zweier MSB auf der Blattebene befinden und dieses Problem ausschließlich auf Blattebene auftritt (siehe 5.2). An dieser Stelle wird untersucht ob redundant definierte Attribute existieren und für jedes festgehalten, wie oft es definiert wurde. Die Ergebnisse der Analyse sind in Abb. 8 visualisiert. Es sind insgesamt 309 verschiedene Attribute vorhanden. Die Merkmale enthalten jedoch 550 Attributdefinitionen, somit existieren 241 redundante Definitionen, dies entspricht nahezu 44 Prozent der Gesamtmodellierung. Einige Schemaelemente sind in bis zu sechs verschiedenen Merkmaldefinitionen vorhanden. Die Ergebnisse bestätigen, dass das Problem der redundant definierten Schemaelemente in der Fallstudie und somit in der Praxis auftritt. Es erzeugt in der Fallstudie die folgenden Probleme.

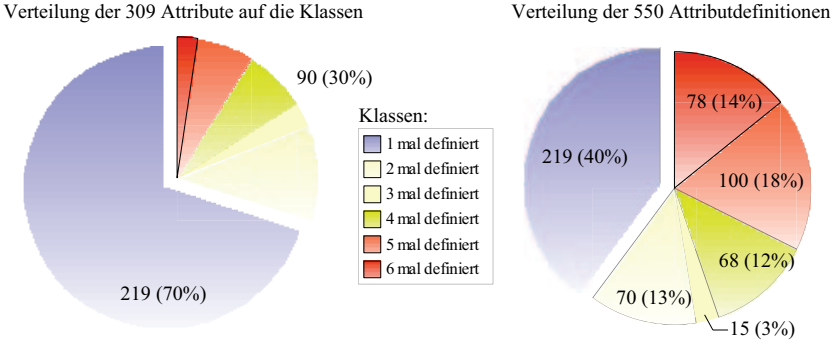

Abbildung 8. Interaktionen durch redundant definierte Schemaelemente

Unübersichtliche Größe der Modellierung. Die redundant definierten Schemaelemente, die 44 Prozent aller Attributdefinitionen ausmachen, erhöhen sowohl die Größe der gesamten Modellierung, als auch die Modellierung der einzelnen Merkmale und lassen diese schnell unübersichtlich werden. In der Fallstudie entstehen Merkmale mit knapp 90 Attributen (siehe Tabelle 1). Dies entspricht fast einem Drittel aller im DB-Schema vorhandener Attribute. Somit wird die Verständlichkeit der Merkmalschemata negativ beeinflusst.

Wahrung der Konsistenz der Modellierung. Es muss sichergestellt werden, dass jedes semantisch identische Attribut den selben Wertebereich erhält. Aufgrund der Generierung der Teilschemata aus einem globalen Schema sind momentan keine widersprüchlichen

Attributdefinitionen vorhanden. Bei der Modifikation eines Schemaelementes während der Weiterentwicklung muss jedoch darauf geachtet werden, dass jede der bis zu sechs redundanten Definitionen des Elementes angepasst wird. Andernfalls entstehen widersprüchliche Definitionen, die die Generierung eines validen Variantenschemas verhindern (siehe 5.3).

Interaktionen durch zusätzlich benötigte Schemaelemente. Das Problem (siehe 4.3), dass aufgrund der Merkmalauswahl zusätzliche Schemaelemente benötigt werden, tritt in der Fallstudie zwischen dem Merkmal *Archivierung* und den *MAPs* sowie zwischen den *MAPs* und dem Datenexport auf. In der Abbildung 9(a) wird gezeigt, dass das Merkmal *Archivierung* je nachdem welche *MAPs* gewählt wurden zusätzliche Schemaelemente zum Archivieren der MAP-Daten benötigt. Hierbei handelt es sich um eine Kopie der MAP-Relationen, in denen die Daten dauerhaft gespeichert werden, um z.B. die Größe der operativen Relation zu verringern. Die zusätzlich benötigten Schemaelemente werden je nach Merkmalauswahl der Basismodellierung durch *Derivatives* hinzugefügt (Abb. 9(b)) in denen die zusätzlich benötigten Elemente enthalten sind. Der Abbildung 9(a) ist weiterhin zu entnehmen, dass das Problem der redundanten Definition ebenfalls an den Überschneidungen der Zusätze, die durch rote Schraffierung gekennzeichnet sind, existiert. Es muss daher ebenfalls sichergestellt werden, dass keine widersprüchlichen Definitionen in den Derivatives vorhanden sind.

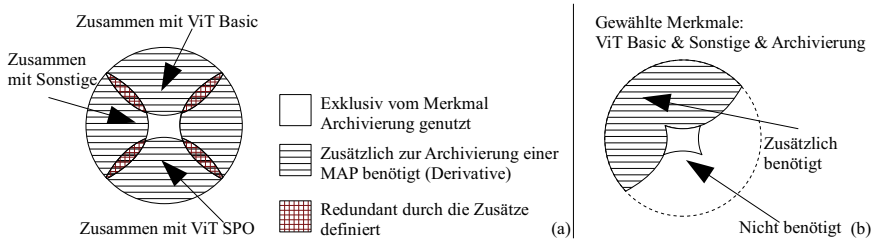

Abbildung 9. Modellierung des Merkmals Archivierung

Insgesamt konnten wir feststellen, dass sich unser Ansatz gut zur Erstellung von maßgeschneiderten DB-Schemata für die Fallstudie eignet. Die Probleme durch Interaktionen der Merkmale sind beherrschbar und es wurden deutliche positive Effekte in den Bereichen *Wartbarkeit und Weiterentwicklung* der Software erzielt.

7 Verwandte Arbeiten

Zur Generierung variabler DB-Schemata existieren bislang wenige Arbeiten. In seiner Dissertation [18] verwendet MAHNKE einen komponenten-basierten Ansatz aus dem Variantenschemata zusammengesetzt werden. Er konzentriert sich jedoch auf das objektrelationale Datenmodell, schließt Interaktionen explizit aus und schlägt zur Anwendung

seines Ansatzes ABLE-SQL eine Spracherweiterung des SQL:1999 Standards [1] vor. Eine solche Modifikation des Standards oder des relationalen Datenmodells [10] wird von dem hier vorgestellten Ansatz nicht benötigt. SIEGMUND et al. präsentieren in [26] eine Anwendung der virtuellen und physischen Trennung der Belange auf das ER-Modell [8]. Wir gehen jedoch einen Schritt weiter und konzentrieren uns auf das relationale Modell. Dies erlaubt die Verwendung des Ansatzes in realen Fallstudien, was wir ebenfalls in dieser Arbeit gezeigt haben. Zusätzliche konnten wir erstmals Aussagen darüber treffen, welche Probleme bei der Generierung maßgeschneiderter DB-Schemata durch Interaktionen der Merkmale auftreten. Weitere Arbeiten zur Anpassung eines DB-Schemas an spezielle Nutzeranforderungen basieren auf Sichten eines globalen DB-Schemas. Dazu gehören Arbeiten zum Thema Sichtintegration [5,22,27] und zur Generierung maßgeschneiderter Sichten [7]. Bei sicht-basierten Ansätzen wird jedoch immer das gesamte Schema ausgeliefert und die Komplexität desselben lediglich vor dem Nutzer durch zusätzliche Sichten versteckt. Die zusätzlichen Sichten erhöhen somit sogar die Komplexität des Schemas, anstatt die Komplexität durch Entfernen der nicht benötigten Elemente zu reduzieren. Weiterhin ist die Erstellung eines Gesamtschemas ausschließlich dann möglich, wenn keine alternativen Merkmale existieren. Mit der Komposition von individuellen Softwarelösungen aus physisch getrennten Fragmenten beschäftigen sich ebenfalls HERMANN et al. [12] und SABETZADEH et al. [23], jedoch liegt ihr Fokus auf der abstrakten Ebene und sie betrachten nicht das relationale Datenmodell oder spezifische Probleme das DB-Schema betreffend.

8 Zusammenfassung

Wir haben gezeigt, wie variable DB-Schemata für das Relationenmodell im Kontext von SPLs modelliert werden können. Für jedes Merkmal wird festgehalten, welche Schemaelemente es benötigt. Somit muss nicht für jede Variante der Software ein DB-Schema modelliert werden, sondern lediglich für jedes Merkmal. Die Modellierung enthält einen Ausschnitt der DDL des SQL:1999 Standards. Es wurde hierfür ein Vorgehen angegeben, mit dem ein zuvor verwendetes globales Schema in Merkmale zerlegt werden kann. Die Generierung eines Variantenschemas erfolgt mittels Superimposition, einem sprachunabhängigen Kompositionsalgorithmus. Hierbei fügen die gewählten Merkmale dem Variantenschema alle von ihnen benötigten Schemaelemente hinzu.

Ziel der Anwendung unseres Ansatzes an der Fallstudie war es zu überprüfen, welche Verbesserungen sich in den Bereichen Wartung, Weiterentwicklung der Software und Vermeidung von Inkonsistenzen ergeben. Durch die Verwendung unseres Ansatzes ließen sich deutliche Verbesserungen im Bereich Wartbarkeit und Vereinfachung der Weiterentwicklung der Software erzielen (siehe 6.3). Somit wurden zwei der drei Ziele erfüllt. Es wurden weiterhin vorbereitende Maßnahmen getroffen, die die Wiedereinführung zuvor entfernter Integritätsbedingungen erlauben. Die Wiedereinführung ist momentan nicht möglich, da wir die entsprechenden Schemaelemente in dieser ersten Arbeit zur Erstellung von Variantenschemata für das relationale Datenmodell nicht betrachtet haben (vgl. 3.2). Hier zeigen sich Verbesserungsmöglichkeiten unseres Ansatzes indem zukünftig weitere Schemaelemente integriert werden müssen.

Weitere Probleme entstehen durch die Verwendung von SELECT * und INSERT INTO TABLE (SELECT ...) Anweisungen, bei denen die Reihenfolge der Attribute in der Relation von Bedeutung ist (vgl. 4.1). Wir haben die entsprechenden Vorkommen durch die Attribute der in der FROM Klausel genutzten Relationen ersetzt. Somit konnten diese Probleme für die Fallstudie gelöst werden, jedoch mag ein solches Vorgehen in weiteren Fallstudien nicht in jedem Fall realisierbar sein. Probleme entstehen weiterhin, wenn Merkmale den Primärschlüssel einer Relation durch zusätzliche Attribute erweitern (vgl. 5.3). Dies kann dazu führen, dass INSERT Statements, die z.B. einen NULL Wert für das neue Schlüsselattribut liefern, keine Daten in die Relation einfügen können. Durch unseren Zerlegungsalgorithmus existieren solche Vorkommen nicht, da die Merkmale aus einem globalen Schema erzeugt werden. Wird ein Schema jedoch von Anfang an variabel entworfen, ist ein solcher Fall durchaus denkbar.

Insgesamt konnten gute Ergebnisse vor allem in den Bereichen Wartung und Weiterentwicklung, selbst mit der momentan begrenzten Zahl an unterstützten Schemaelementen, erzielt werden. Somit zeigt sich, welches Potential der von uns vorgestellte Ansatz besitzt. Die Probleme des Ansatzes konnten jedoch nicht vollständig gelöst werden, so dass weiterer Forschungsbedarf auf diesem Gebiet besteht. Weiterhin müssen zusätzliche Schemaelemente in die Betrachtung mit aufgenommen und die Ergebnisse an weiteren Fallstudien verifiziert werden.

9 Danksagung

Teile dieser Veröffentlichung entstanden aus dem Forschungsvorhaben „Digitale Fingerspuren (Digi-Dak)“ mit der Projektnummer FKZ:13N10817, welches vom Bundesministerium für Bildung und Forschung (BMBF) gefördert wird. Norbert Siegmund wird unterstützt durch das Bundesministerium für Bildung und Forschung (BMBF), Projektnummer 01IM08003C. Die Arbeit ist Teil des ViERforES Projekts. Christian Kästners Arbeit wird durch den Europäischen Forschungsrat (ScalPL #203099) unterstützt. Abschließend danken wir dem METOP ViT-Entwicklerteam.

Literatur

1. ANSI/ISO/IEC 9075:1999: International Standard - Database Language SQL (1999)
2. Apel, S., Kästner, C., Lengauer, C.: Vergleich und Integration von Komposition und Annotation zur Implementierung von Produktlinien. In: Software Engineering 2009. Lecture Notes in Informatics, vol. P-143, pp. 101–112. Gesellschaft für Informatik (GI) (2009)
3. Apel, S., Lengauer, C.: Superimposition: A Language-Independent Approach to Software Composition. In: Proc. Int’l Softw. Compos. Symp. pp. 20–35. Springer (2008)
4. Apel, S., Lengauer, C., Möller, B., Kästner, C.: An Algebra for Features and Feature Composition. In: Proc. Int’l Conf. on Algebraic Methodology and Software Technology. LNCS, vol. 5140, pp. 36–50. Springer (2008)
5. Batini, C., Lenzerini, M., Navathe, S.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18, pp. 323–364 (1986)
6. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering 30(6), pp. 355–371 (2004)

7. Bolchini, C., Quintarelli, E., Rossato, R.: Relational data tailoring through view composition. In: Proc. Int'l Conf. on Conceptual Modeling. LNCS, vol. 4801, pp. 149–164. Springer (2007)
8. Chen, P.: The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems 1(1), pp. 9–36 (1976)
9. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)
10. Codd, E.: A relational model of data for large shared data banks. Communications of the ACM 13(6), pp. 377–387 (1970)
11. Czarnecki, K., Eisenecker, U.: Generative programming: methods, tools, and applications. Addison-Wesley (2000)
12. Herrmann, C., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: An algebraic view on the semantics of model composition. In: Proc. Europ. Conf. on Model driven architecture-foundations and applications. pp. 99–113. Springer (2007)
13. Johnson, R., Foote, B.: Designing Reusable Classes. Journal of Object-Oriented Programming 1(5), pp. 22–35 (1988)
14. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
15. Kästner, C., Apel, S., ur Rahman, S., Rosenmüller, M., Batory, D., Saake, G.: On the impact of the optional feature problem: analysis and case studies. In: Proc. Int'l Conf. on Software Product Line. pp. 181–190. Carnegie Mellon University (2009)
16. Kästner, C., Apel, S.: Virtual Separation of Concerns – A Second Chance for Preprocessors. Journal of Object Technology 8(6), pp. 59–78 (2009)
17. Liu, J., Batory, D., Lengauer, C.: Feature-Oriented Refactoring of Legacy Applications. In: Proc. Int'l Conf. on Software Engineering. pp. 112–121. ACM (2006)
18. Mahnke, W.: Komponentenbasierter Schemaentwurf. Ph.D. thesis, Technische Universität Kaiserslautern, Kaiserslautern, Deutschland (2004)
19. Pohl, K., Böckle, G., Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer (2005)
20. Prehofer, C.: Feature-Oriented Programming: A Fresh Look at Objects. In: Proc. Europ. Conf. on Object-Oriented Programming. LNCS, vol. 1241, pp. 419–443. Springer (1997)
21. Saake, G., Heuer, A., Sattler, K.: Datenbanken Konzepte und Sprachen. mitp Verlag (2008)
22. Sabetzadeh, M., Easterbrook, S.: View merging in the presence of incompleteness and inconsistency. Requir. Eng. 11(3), pp. 174–193 (2006)
23. Sabetzadeh, M., Nejati, S., Liaskos, S., Easterbrook, S., Chechik, M.: Consistency Checking of Conceptual Models via Model Merging. In: Proc. Int'l Conf. on Requirements Engineering. pp. 221–230. Springer (2007)
24. Schäler, M.: Produktlinientechnologien für den Entwurf variabler DB-Schemata unter Berücksichtigung evolutionärer Änderungen. Master thesis (diplomarbeit), University of Magdeburg, Germany (2010)
25. Schirmeier, H., Spinczyk, O.: Tailoring Infrastructure Software Product Lines by Static Application Analysis. In: SPLC '07: Proceedings of the 11th International Software Product Line Conference. pp. 255–260. IEEE Computer Society (2007)
26. Siegmund, N., Kästner, C., Rosenmüller, M., Heidenreich, F., Apel, S., Saake, G.: Bridging the Gap between Variability in Client Application and Database Schema. In: Proc. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web. pp. 297–306. Gesellschaft für Informatik (2009)
27. Spaccapietra, S., Parent, C.: View Integration: A Step Forward in Solving Structural Conflicts. IEEE Trans. on Knowl. and Data Eng. 6(2), pp. 258–274 (1994)