Proc. Workshop Geschäftsprozessorientierte Architekturen (Informatik '04), LNI P-51, Germany, September 2004, S. 555-560

Repräsentation von Schema- und Instanzobjekten in adaptiven Prozess–Management–Systemen

Markus Lauer, Stefanie Rinderle, Manfred Reichert

Abteilung Datenbanken und Informationssysteme, Universität Ulm
{ml4, rinderle, reichert}@informatik.uni-ulm.de

1 Einführung

Immer mehr Unternehmen setzen zur Steuerung, Verwaltung und Überwachung ihrer betrieblichen Abläufe ein Prozess–Management–System (PMS) ein. Eine wichtige Voraussetzung für den sinnvollen Einsatz eines PMS ist, dass es die Eigenschaft der Adaptivität besitzt [RRD04a, SMO00]: Es muss Änderungen der hinterlegten Prozessvorlagen – wir sprechen hierbei von *Schemaevolution* – und der sich in Ausführung befindlichen Prozesse zur Laufzeit zulassen, damit die Unternehmen in der Lage sind, flexibel und schnell auf neue Anforderungen zu reagieren. Prozessoptimierungen, Änderungen der innerbetrieblichen Organisation, wie z. B. Auslagern der Produktion ins Ausland, oder neue gesetzliche Bestimmungen machen Modifikationen der Prozessvorlage notwendig. Änderungen von Instanzen sind notwendig, um unvorhersehbaren Situationen (z. B. Ausfall von Maschinen) begegnen oder um späte Kundenwünsche noch berücksichtigen zu können.

Adaptivität stellt jedoch hohe Ansprüche an das PMS: Effiziente Algorithmen müssen dafür sorgen, dass die Änderungen nicht zu inkonsistenten Zuständen von *Prozessvorlagen* und -*instanzen* führen. Modifikationen der Prozessvorlagen müssen, wo vom Ausführungszustand her möglich, auch auf die darauf basierenden Prozessinstanzen *propagiert* werden können. Bei *verzerrten Instanzen* — das sind Instanzen, die sich aufgrund von Adhoc–Änderungen in ihrer Ablaufsstruktur gegenüber der Original–Vorlage unterscheiden — müssen zusätzlich Probleme, die sich bei der Migration auf die neue Version aus überlappenden Vorlagen- und Instanzänderungen ergeben, erkannt und gehandhabt werden. Dabei dürfen die anderen, parallel ablaufenden Funktionen des PMS nicht beeinträchtigt werden, selbst wenn in realen Anwendungen tausende von Instanzen migriert werden müssen. Hinzu kommt, dass dem PMS für diese Aufgaben nur eingeschränkt Ressourcen, wie z. B. Speicher, zur Verfügung stehen. Alle diese Anforderungen verlangen nach einer flexiblen und ressourcensparenden Architektur sowie nach einer effizienten Implementierung.

Die auf dem Markt erhältlichen Produkte bieten entweder gar keine oder nur eingeschränkte Änderungsmöglichkeiten zur Laufzeit oder erfüllen die angesprochenen Anforderungen nur unzureichend. Wir entwickeln in unserem Projekt ADEPT ein PMS der nächsten Generation, welches Änderungen zur Laufzeit vollständig unterstützt. Die formalen Grundlagen hierfür haben wir in früheren Veröffentlichungen beschrieben [RRD03, RRD04b]. In diesem Beitrag stellen wir einen ersten Architekturansatz vor, der Änderungen von Instanzen und Prozessvorlagen ermöglicht, wie z. B. das Einfügen und Löschen von Aktivitäten,

Sync-Kanten und Datenelementen, der die Migration effizient unterstützt und der einen geringen Speicherbedarf aufweist. Er wurde von uns in einem neuen Prototyp implementiert, der auf dem Workshop ebenfalls demonstriert werden soll.

2 Schema- und Instanzobjekte in adaptiven PMS

Ein in der Literatur (vgl. [We01]) vorgeschlagener Implementierungsansatz von Prozessvorlagen und -instanzen wird in Abb. 1 illustriert. Dabei wird die *Prozessbeschreibung*

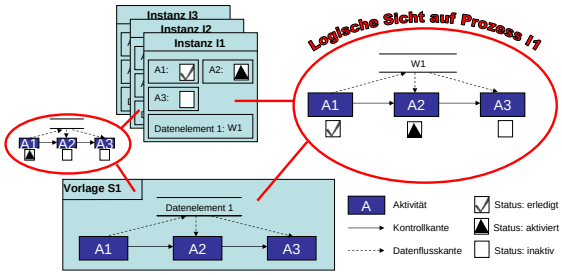

Abbildung 1: Die Repräsentation von Instanzen eines Prozess-Typs

(u. a. Kontroll- und Datenfluss) in einem Objekt, der *Prozessvorlage*, gekapselt, das den *Prozesstyp* repräsentiert. Die *Instanz–Objekte*, welche Prozessinstanzen repräsentieren, enthalten selbst nur Laufzeitinformationen, wie den Ausführungsstatus von Aktivitäten und logisch, nicht unbedingt physisch, den Inhalt der Datenelemente. Die Typzugehörigkeit wird durch eine Referenz auf das entsprechende Prozessvorlagen–Objekt ausgedrückt. Alle Instanzen eines Prozesstyps referenzieren das gleiche Vorlagen–Objekt. Dieser Ansatz soll als Ausgangsbasis dienen. Gegenüber der Variante, für jede Instanz die jeweilige Prozessbeschreibung redundant zu speichern, resultiert ein erheblich geringerer Speicherplatzbedarf bei großer Instanzzahl. Ein weiterer Vorteil ist, dass sich damit auch signifikant Rechenzeit einsparen lässt, da strukturverändernde Operationen bei einer Schemaevolution nur einmal auf die Prozessvorlage angewendet werden müssen, bei der anderen Variante dagegen auf jede einzelne Prozessinstanz.

Bei dieser Implementierung darf allerdings die Schemaänderung nicht direkt auf dem Vorlagen–Objekt ausgeführt werden, welches die aktuelle Version repräsentiert. Die Propagation der Schemaänderungen auf Instanzen würde zwar damit in einem Vorgang erfolgen, aber dann könnten auch Instanzen fälschlicherweise migriert werden, die für diese Änderungen zu weit fortgeschritten sind. In Abb. 2 laufen Instanzen I1 und I2 auf derselben Vorlage S1, wobei I1 weiter fortgeschritten ist als I2. Da die Aktivität A2 bei I1 schon ausgeführt worden ist, ist I1 unverträglich mit der neuen Version von S1, bei der vor A2 die Aktivität A12 eingefügt wird. Wird nun die Änderung direkt auf S1 ausgeführt, so migriert zwar I2 ordnungsgemäß, aber Instanz I1 folgt verbotenerweise ebenfalls dem neuen Schema, da sie weiterhin S1 referenziert. Eine Inkonsistenz ist die Folge: A2 von I1 wurde abgearbeitet, bevor die Vorgängeraktivität A12 ausgeführt worden ist.

Die Koexistenz von Instanzen neuer und alter Form kann sichergestellt werden, indem eine Kopie des Vorlagen–Objekts angelegt wird, daran die Modifikationen ausgeführt werden und dann alle migrierbaren Instanzen auf dieses Objekt umgehängt werden. In diesem Fall

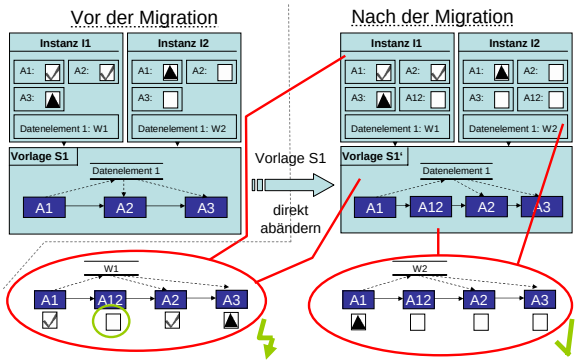

Abbildung 2: Unerlaubte Migration der Instanz I1 bei der direkten Abänderung der Vorlage S1

besteht der Migrationsvorgang aus dem “Umbiegen” der Referenz auf die neue Version.

2.1 Repräsentationsvariante für verzerrte Instanzen

Wie lassen sich mit obigen Implementierungsvorschlag instanzbezogene Adhoc–Modifikationen abbilden? Eine Möglichkeit ist es, eine Kopie des entsprechenden Vorlagen–Objekts anzufertigen, die Änderungen darauf anzuwenden und dann die Instanz auf dieses Schema umzubiegen. Von Nachteil ist, dass damit die ursprüngliche Prozesstyp–Zugehörigkeit verloren geht: Die verzerrte Instanz referenziert nicht mehr das ursprüngliche Vorlagen-Objekt S1, obwohl sie noch von dem durch dieses Objekt beschriebenen Prozesstyp abstammt. Ohne zusätzliche Vorkehrungen wird die Instanz bei einer späteren Schemaevolution nicht mehr berücksichtigt. Hinzu kommt, dass dieser Ansatz zu der anfangs erwähnten Lösung, bei der in jeder Instanz der gesamte Prozessablauf hinterlegt ist, degeneriert, wenn mehr und mehr Instanzen individuell abgeändert werden. Berücksichtigt man aber, dass bei Instanzänderungen i. A. nur ein kleiner Teil des ursprünglichen Prozessschemas angepasst wird, so lässt sich eine alternative Strategie zur Repräsentation verzerrter Instanzen anwenden: Man speichert bei jeder modifizierten Instanz nur die Abweichungen vom ursprünglichen Schema. Diese können auf zweierlei Arten repräsentiert werden, entweder durch die Änderungsoperationen selbst oder durch eine Delta–Schicht. Aus Platzgründen wenden wir uns gleich dem aus unserer Sicht besseren Konzept zu: der Delta-Schicht.

Abb. 3 verdeutlicht das Konzept der *Delta–Schicht*. Sie wird durch ein Objekt repräsentiert, das die gleichen Schnittstellen wie das Prozessvorlagen–Objekt besitzt und daher nach außen hin die gleichen Operationen anbietet. Der Unterschied zum eigentlichen Vorlagen–Objekt besteht darin, dass es nicht den gesamten Prozess–Graphen wiedergibt, sondern nur die Ausschnitte, die durch instanzbezogene Modifikationen geändert wurden. Zusammen mit dem Original–Vorlagen–Objekt, das den Prozesstyp der Instanz bestimmt, repräsentiert das Delta–Schicht–Objekt das gegenüber der Prozessvorlage abgeänderte Laufzeitschema der modifizierten Instanz. Das Instanz–Objekt, das die geänderte Instanz repräsentiert, referenziert dann nicht mehr, wie in Abb. 3 anhand von I1 gezeigt, das entsprechende Vorlagen–Objekt, sondern das Delta–Schicht–Objekt. Erst das Delta–Schicht–Objekt setzt auf dem Original–Vorlagen–Objekt auf und bewahrt auf diesem Weg u. a. die

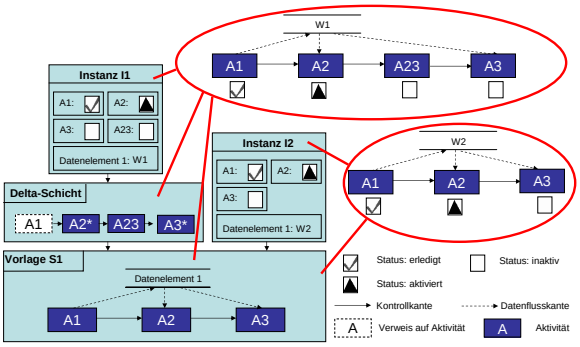

Abbildung 3: Das Konzept der Delta-Schicht

Typzugehörigkeit von I1 zum Prozesstyp S1. Die unveränderten Instanzen (z. B. Instanz I2) referenzieren das originale Prozessvorlagen–Objekt weiterhin direkt. Bei modifizierten Instanzen werden deshalb Anfragen, wie „Gib mir alle direkten Nachfolgeaktivitäten von Aktivität Z!“, zuerst an die Zwischenschicht gerichtet, bei unveränderten dagegen direkt an das originale Vorlagen–Objekt.

Wie die Delta–Schicht realisiert werden kann, hängt von der Repräsentation der Knoten und Kanten des Prozess–Graphen ab. In unserer Implementierung existieren keine Kanten–Objekte. Wir speichern zu jeder Aktivität die IDs der Vorgänger- und Nachfolgeraktivitäten und stellen dadurch implizit den Kontrollfluss her. Vor einer dynamischen Änderung werden alle Aktivitäten–Objekte automatisch in die Delta–Schicht kopiert, die von der Änderung betroffen sind. Diese wird dann auf den Kopien durchgeführt. Um z. B. die Aktivität A23 sequentiell zwischen die Nachbaraktivitäten A2 und A3 einzufügen, werden zuerst die Aktivitäten–Objekte A2 und A3 komplett, inklusive der ID, kopiert und als A2* und A3* in die Delta–Schicht eingefügt[1]. A2 und A3 müssen kopiert werden, da sich ihre Nachfolger- bzw. Vorgängermengen ändern. Danach wird das Aktivitäten–Objekt A23 in der Delta–Schicht angelegt. Schließlich wird A23 sequentiell zwischen A2* und A3* eingereiht, indem die Vorgänger- und Nachfolgerlisten von A2*, A3* und A23 entsprechend angepasst werden. In einer Implementierung mit Kanten–Objekten müssen für die Einfüge–Operation nicht A2 und A3 in die Delta–Schicht kopiert werden. Es werden nur A23 und die Kantenobjekte für die Verbindungen A2→A23 und A23→A3 angelegt. Zusätzlich muss A2→A3 noch als gelöscht markiert werden.

Zur Beantwortung der obigen Anfrage „Gib mir alle direkten Nachfolgeaktivitäten von Aktivität Z!“ sieht die Delta–Schicht bei der Implementierung ohne Kanten–Objekte zuerst nach, ob sie ein Aktivitäten–Objekt mit entsprechender ID gespeichert hat. Wenn ja, so gibt sie dessen Nachfolgerliste zurück. Ansonsten leitet sie die Anfrage an das referenzierte Prozessvorlagen–Objekt weiter. Im Falle der Implementierung mit Kanten–Objekten holt sich die Delta–Schicht erst einmal alle Kanten–Objekte von der Prozessvorlage, in denen die Aktivität Z als Quelle verzeichnet ist. Dann entfernt sie aus dieser Menge alle als gelöscht markierten Kanten und vereinigt sie mit der Menge der durch die Adhoc–Modifikationen neu hinzugekommenen Kanten mit Z als Quelle. Die Menge enthält nun

[1] Der Stern soll nur zum besseren Verständnis beitragen.

2.2 Migration verzerrter Instanzen

Wie lässt sich nun eine verzerrte Instanz auf eine neue Version der zugrunde liegenden Vorlage migrieren, wenn sich die Schema- und Instanz-Änderungen nicht überlappen? Ein Ansatz ist es, die Referenz auf das Vorlagen–Objekt in der Delta–Schicht auf die neue Version “umzubiegen”. Dies kann aber bei der Implementierungsvariante ohne Kanten–Objekte zu Problemen führen: A1, A2 und A3 seien drei sequentielle Aktivitäten. Das Einfügen eines Knoten A23 zwischen A2 und A3 auf Instanz–Ebene führt dazu, das eine Kopie A2* von A2 in der Delta-Schicht angelegt und darin die ID von A23 anstelle der ID von A3 in die Nachfolgerliste aufgenommen wird. Durch Einfügen der Aktivität A12 zwischen A1 und A2 auf Schema–Ebene wird die ID von A1 durch die ID von A12 in der Vorgängerliste von A2 ersetzt. Die Anfrage „Gib mir die direkten Vorgänger von A2“ liefert jedoch weiterhin A1 als Vorgänger, da die Delta–Schicht die Vorgängerliste von A2* zurückgibt, die aber durch die Schemaevolution nicht angepasst wurde und deshalb immer noch die ID von A1 enthält (vgl. Abb. 4).

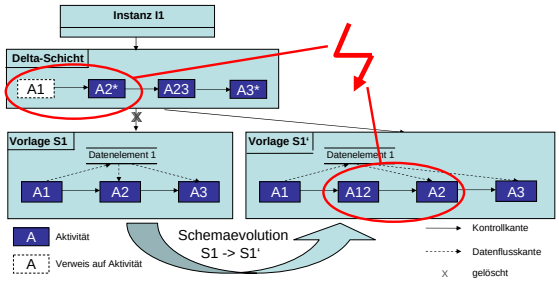

Abbildung 4: Migration von verzerrten Instanzen bei Implementierungen ohne Kantenobjekte

Eine Lösung dieses Problems besteht darin, auf das Vorlagen–Objekt, das die neue Version des Schemas repräsentiert, ein leeres Delta–Schicht–Objekt aufzusetzen, darauf dann die gleichen dynamischen Änderungen anzuwenden und es schließlich von dem entsprechenden Instanz–Objekt zu referenzieren. Da bei dieser Vorgehensweise das Einfügen von A12 vor dem Einfügen von A23 erfolgt, weist A2 schon vor dem Kopieren in die Delta–Schicht A12 als seinen Vorgänger aus. Nach dem Kopieren gilt daher das Gleiche für die Kopie A2* von A2. Die Anfrage wird somit korrekt beantwortet. Die Abfolge der auf Instanz- und Typebene vorgenommenen Änderungen zu vertauschen war zulässig, da vorausgesetzt wurde, dass sich die Änderungen nicht überlappen.

Bei der Implementierung mit Kantenobjekten tritt das Problem des “vergessenen” Vorgängers nicht auf: Die Delta–Schicht enthält hier durch das dynamische Einfügen von A23 — abgesehen vom Aktivitäten–Objekt für A23 — nur die neuen Kanten A2→A23 und A23→A3. A2→A3 wird dagegen als aufgehoben markiert. Auf Schema–Ebene werden beim Einfügen von A12 die Kante A1→A2 komplett entfernt und dafür A1→A12 und A12→A2 eingefügt. Zur Bearbeitung der obigen Anfrage holt sich die Delta–Schicht erst alle Kanten mit A2 als Ziel von der Prozessvorlage. Das Vorlagen–Objekt gibt in diesem Beispiel nur A12→A2 an die Delta–Schicht zurück. Da die Kante in der Delta–Schicht

nicht als gelöscht markiert ist und dort keine weiteren Kanten mit A2 als Ziel vermerkt sind, liefert die Delta–Schicht schließlich A12 als Vorgängeraktivität von A2. Die Variante mit den Kanten–Objekten hat jedoch den Nachteil, dass immer ein Zugriff auf das zugrundeliegende Vorlagen–Objekt notwendig ist, um die ein- oder ausgehenden Kanten einer Aktivität zu bestimmen. Bei der anderen Variante entfällt dieser Zugriff auf das Vorlagen–Objekt, wenn die entsprechende Aktivität bereits in der Delta–Schicht vorhanden ist.

3 Zusammenfassung und Ausblicke

Mit dem skizzierten Ansatz liegt eine übersichtliche interne Repräsentation von Prozessvorlagen und Instanzen vor, mit der sich die Migration von unveränderten und sogar dynamisch abgeänderten Instanzen einfach realisieren lässt. Der Migrationsvorgang ist auch schnell und effizient, da er bei diesem Ansatz meistens nur aus dem Umhängen von Referenzen besteht. Außerdem konnte der Speicherbedarf gegenüber anderen Ansätzen stark reduziert werden, da auf Wiederverwendung gesetzt wurde — dasselbe Prozessvorlagen–Objekt wird z. B. von mehren Instanzen referenziert — und da die Delta-Schicht nur abgeänderte Abschnitte des Prozessgraphen speichert. Diese Eigenschaften sprechen für einen Einsatz in der Praxis. Der Delta-Schicht-Ansatz zur Repräsentation der Abweichung eines Instanzschemas gegenüber der Vorlage und der vorgestellte Migrationsablauf können prinzipiell uneingeschränkt auf andere Workflow-Modelle übertragen werden. Da aber nur Instanzen migriert werden dürfen, bei denen die Änderungen mit dem bisherigen Prozessablauf vereinbar sind, müssen Informationen sowohl über den momentanen Ausführungsstand als auch über den bisherigen Verlauf vorliegen. Damit ist die Übertragung nur auf Workflow-Modelle sinnvoll, die entsprechende Informationen bereitstellen (siehe z. B. [RRD04a]). Der präsentierte Ansatz muss jedoch noch erweitert werden, um weitere Anforderungen, die ein Einsatz in Produktivsystemen mit sich bringt, zu erfüllen. Es müssen u. a. darauf abgestimmte Sperrverfahren entwickelt werden, um den nebenläufigen Zugriff sowohl auf Prozessvorlagen als auch auf Instanzen zu koordinieren. Beispielsweise muss abgefangen werden, dass ein Benutzer eine Instanz zu modifizieren beginnt, die gerade im Begriff ist zu migrieren. Auch muss untersucht werden, wie eine Migration durchgeführt wird, wenn eine Instanz partitioniert von mehreren Prozess–Engines ausgeführt wird. Außerdem haben wir bisher bei der Migration von verzerrten Instanzen die Einschränkung gemacht, dass sich die Schema- und die Instanzänderungen nicht überlappen dürfen. An diesen und weiteren Aspekten arbeiten wir derzeit sehr intensiv.

Literatur

[RRD03] Reichert, M.; Rinderle, S.; Dadam, P.: On the Common Support of Workflow Type and Instance Changes Under Correctness Constraints. Proc. CoopIS ’03, Catania, 2003

[RRD04a] Rinderle, S.; Reichert, M.; Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems - A Survey. Data and Knowledge Engineering, 50(1):9-34(2004)

[RRD04b] Rinderle, S.; Reichert, M.; Dadam, P.: Flexible Support of Team Processes by Adaptive Workflow Systems. Distributed and Parallel Databases, 16(1):91-116(2004)

[SMO00] Sadiq, S.; Marjanovic, O.; Orlowska, M.E.: Managing Change and Time in Dynamic Workflow Processes. IJCIS, 9(1&2):1-25(2000)

[We01] Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. Proc. 34th Hawaii Int’l Conf. on System Sciences, 2001