Robuste Systemevolution durch automatische Unterstützung

Boris Stumm
stumm@informatik.uni-kl.de
Technische Universität Kaiserslautern

Zusammenfassung

Unternehmen integrieren ihre Systeme immer stärker miteinander, um den größtmöglichen Nutzen aus den Informationen zu ziehen. Resultat sind hochkomplexe, verteilte Informationssysteme mit einer Vielzahl von Abhängigkeiten zwischen den beteiligten Einzelsystemen.

Informationslandschaften unterliegen ständig verschiedenen Formen von Veränderung durch Umstrukturierungen, Neuanschaffungen usw. Während das bei einzelnen Systemen unproblematisch ist, können sich lokale Änderungen in einer integrierten Informationslandschaft auf andere Systeme auswirken. Das reicht vom Totalausfall über Datenkorruption bis hin zu subtilen Fehlern, die möglicherweise erst viel später entdeckt werden.

Diese negativen Auswirkungen müssen daher auf ein Minimum (sowohl an Zahl als auch an Dauer) reduziert werden. Das ist auf organisatorischer Ebene alleine (z. B. neue Geschäftsprozesse zur Koordination der Verantwortlichen) nicht zu bewältigen. In dieser Arbeit stellen wir einen Ansatz vor, der Entwickler und Systemverwalter einerseits bei Planung und Durchführung von Änderungen unterstützt, und andererseits das Gesamtsystem robust macht gegen ungeplante lokale Systemänderungen.

1 Einleitung

Die heutige Systemlandschaft ist geprägt von zunehmender Komplexität und Heterogenität. Durch Neuanschaffungen und Firmenzusammenschlüsse kommen immer neue Systeme hinzu. Verschiedene Systeme werden integriert, um die vorhandenen Informationen besser verarbeiten zu können. In einer solchen heterogenen Informationslandschaft werden einzelne Systeme immer wieder verändert und weiterentwickelt, sei es, um Fehler zu beheben, oder um neuen Bedürfnissen gerecht zu werden.

Diesen nicht zentral gesteuerten Prozess nennen wir *Systemevolution*. In der Regel sind nur ein einzelnes oder einige wenige Systeme von einem Evolutionsschritt betroffen; aufgrund von Interdependenzen zwischen den Systemen kann sich eine Änderung jedoch auf weitere Systeme auswirken. Bei einer fehlenden Organisation dieses Prozesses werden die Auswirkungen möglicherweise erst viel später erkannt, was die Problemursachen schwer nachvollziehbar macht. Mit einer *Änderung* bezeichnen wir prinzipiell alle Vorgänge in einem System, die sich auf andere Systeme funktional auswirken. Das betrifft Schemaänderungen (Schemaevolution), Konfigurationseinstellungen, Netzwerkverbindung, Qualitätszusagen usw.

Diese Problematik verschärft sich mit der zunehmenden Komplexität und Heterogenität von integrierten Informationssystemen, deshalb wird eine automatisierte Unterstützung des Evolutionsprozesses erforderlich. Wir möchten in dieser Arbeit einen Überblick über unseren Ansatz geben. Er soll ein Integrationssystem robust machen gegen Probleme, die durch lokale Änderungen hervorgerufen werden. Dazu definieren wir einen Prozess und stellen automatisierte Analysewerkzeuge bereit, die auch bei nicht optimalem Prozessablauf Probleme erkennen können.

Die Arbeit ist wie folgt aufgebaut. In Abschnitt 2 grenzen wir die Probleme der Systemevolution in einer heterogenen Umgebung ein und stellen Soll- und Ist-Zustand in verteilten Informationsystemen gegenüber. Unser Ansatz, mit dem wir die Probleme angehen wollen, wird in Abschnitt 3 näher ausgeführt. Eine Abgrenzung zu verwandten und komplementären Arbeiten sowie ein Fazit geben wir in Abschnitt 4.

2 Problematik

Zu den Änderungen, die ein Informationssystem während seiner Lebensdauer erfährt, gehören alle Formen der Schemaevolution und Änderungen der technischen Konfiguration des Systems (Hardware und Software). Die Auswirkungen sind unterschiedlicher Art:

- Bei der „klassischen Schemaevolution“ werden Schemaelemente hinzugefügt, geändert oder gelöscht.
- Reparaturen und Verbesserungen an Systemen sind eigentlich auch der Schemaevolution zuzurechnen. Wir führen sie hier gesondert auf, weil es normalerweise nur „kleine“ Änderungen sind (z. B. Erweiterung eines Datentyps von char(20) auf char(30) oder Änderung des Verhaltens einer Funktion bei Eingaben außerhalb des Definitionsbereichs). Während bei größeren Schema-Umbauten davon ausgegangen werden kann, dass alle betroffenen Systeme mit einbezogen werden, ist das hier nicht unbedingt der Fall.
- Auch Änderungen an der Konfiguration von Systemen sind von Bedeutung. Dies umfasst die Vergabe von Benutzerrechten, periodisch wiederkehrende Abläufe (z.B. Data-Warehouse-Updates), Softwareupdates, Bandbreite der Netzwerkanbindung usw.

Bei eng gekoppelten Systemen, z.B. Anwendung und zugehöriges DBVS oder Data Warehouse und OLAP-System, kann man in der Regel davon ausgehen, dass bei Änderungen alle betroffenen Systeme gleichzeitig betrachtet werden und die Abhängigkeiten klar sind. Im größeren Rahmen einer verteilten Informationslandschaft sind Systeme aber nur noch lose gekoppelt, und nicht jeder Systemverantwortliche ist sich aller Abhängigkeiten bewusst. In einer solchen Konstellation wird eine automatische Verwaltung der Abhängigkeiten und Beziehungen der Informationsysteme zueinander nötig, um Probleme, die in abhängigen Systemen durch lokale Änderungen an einzelnen Systemen oder Systemgruppen entstehen, zu vermeiden oder zumindest frühzeitig zu erkennen. Diese Probleme nennen wir allgemein *Änderungsprobleme*.

Um Änderungsprobleme zu vermeiden, muss dafür gesorgt werden, dass vor der Durchführung von *lokalen* Änderungen die Auswirkungen auf *globaler* Ebene analysiert werden und eventuell auftretende Konflikte zuerst aufgelöst werden.

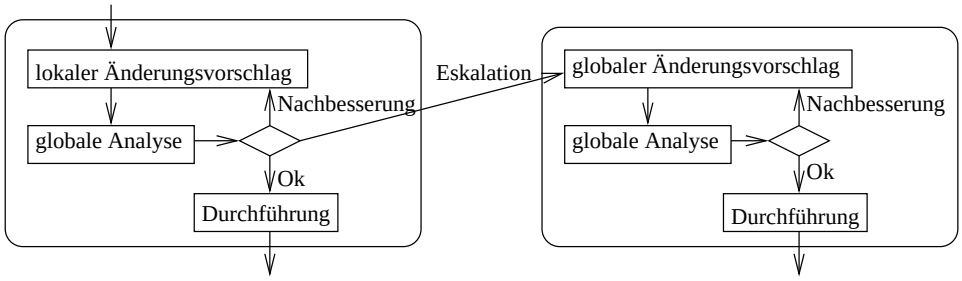

Abbildung 1: Prozess der Systemevolution

Abbildung 1 veranschaulicht diesen Prozess. Er lässt sich in einen lokalen und einen globalen Teil aufteilen, die beide ähnlich ablaufen. Bevor Änderungen an einem System vorgenommen werden wird der Änderungsvorschlag analysiert. Bei Problemen kann nachgebessert oder der Prozess auf alle betroffenen Systeme ausgeweitet werden. Der Kreislauf von Änderungsvorschlag, Analyse und Nachbesserung wiederholt sich dann auf der globalen Ebene, bis alle Konflikte beseitigt sind. Am Ende steht die Durchführung der Änderungen. Die Abfolge soll an einem kleinen Beispielszenario verdeutlicht werden.

Wir betrachten ein föderiertes DBVS (FDBVS), das eine Personendatenbank verwaltet. Eines der Quellsysteme speichert Name und Vorname von Personen konkateniert in der Spalte NAME der Tabelle PERSON. Im Quellsystem soll diese Spalte aufgeteilt werden in VORNAME und NACHNAME. Dieser Vorschlag wird analysiert, und es wird festgestellt, dass ein Problem auftritt, da das FDBVS auf die

Namensspalte zugreift. Es bietet sich jetzt die Möglichkeit, im Nachbesserungsschritt die Spalte NAME redundant im Schema zu belassen, oder durch die Verwendung von Sichten abwärtskompatibel zu bleiben. Damit sind die Konflikte aufgelöst, und die Änderung kann vollzogen werden. Wenn nicht nachgebessert wird, muss ein globaler Änderungsvorschlag eingebracht werden, der auch das FDBVS in die Änderungen mit einbezieht. Dadurch können weitere Systeme beeinflusst werden, weshalb eine iterative Ausweitung der in die Planung einbezogenen Systeme möglich ist.

Dieser Prozess funktioniert nur dann zufriedenstellend, wenn *alle* Änderungen wie beschrieben analysiert werden. Aufgrund der normalerweise relativ autonomen Einzelsysteme und verteilten Verantwortlichkeiten kann das im allgemeinen nicht garantiert werden. Deshalb ist es uns wichtig, in unserem Ansatz einen geordneten Evolutionsprozess zwar zu unterstützen, aber keinesfalls vorauszusetzen. Damit ist die Robustheit des Gesamtsystems auch unter widrigen Bedingungen gewährleistet (siehe Abschnitt 3).

Je größer die Informationslandschaft wird, desto unwahrscheinlicher ist es auch, das alle Informationen zur automatischen Analyse vollständig vorliegen. Das hat verschiedene Gründe:

- Änderungsvorschläge sind nicht immer genau und exakt beschrieben, es können z.B. semantische Informationen fehlen. Beispiel: In einer Spalte NAME, die vorher Vor- und Nachnamen einer Person enthielt, ist nach dem Einfügen der Spalte VORNAME nur noch der Nachname gespeichert. In diesem trivialen Fall vermutlich unproblematisch, kann dies in größerem Rahmen jedoch zu Problemen führen.
- Die Abhängigkeiten zwischen verschiedenen Systemen sind nicht immer vollständig beschrieben, weil semantische Informationen fehlen oder die Abhängigkeiten lediglich prozedural beschrieben sind (z.B. ETL-Skripte).
- Um eine globale Analyse mit vertretbarem Aufwand durchführen zu können, müssen die Metadaten in einem homogenen Format vorliegen, wodurch ein Informationsverlust im allgemeinen nicht vermeidbar ist.

Unser Ziel ist es, auch mit unvollständiger Information noch nützliche Ergebnisse zu liefern. Prinzipiell lässt sich das mit pessimistischen Abschätzungen bei fehlender Information erreichen.

3 Ansatz

Die für uns relevanten Aufgaben im Systemevolutionsprozess bestehen in der Überwachung der Informationssysteme, der Analyse von Änderungsvorschlägen (bzw. vollzogenen Änderungen) und der Ergreifung von Maßnahmen im Konfliktfall. Dafür stellt unser Ansatz drei Komponenten bereit: Änderungsmanager, Metadaten-Repository (MDR) und Metadaten-Agenten (MDAs). Im Metadaten-Repository sind die zur Analyse benötigten Metadaten gespeichert. Der Änderungsmanager nutzt die dort gespeicherte Information, um Änderungsanalysen durchzuführen. Für jedes Informationssystem gibt es einen MDA, welcher die Vermittlerrolle zwischen Änderungsmanager, MDR und Systemen übernimmt. Die MDAs sind zuständig für die Synchronisierung der Daten im MDR und den betreuten Systemen, das Erkennen von Änderungen, und zur Maßnahmenergreifung bei Problemen. Abbildung 2 gibt einen Überblick über die Zusammenhänge zwischen den Komponenten.

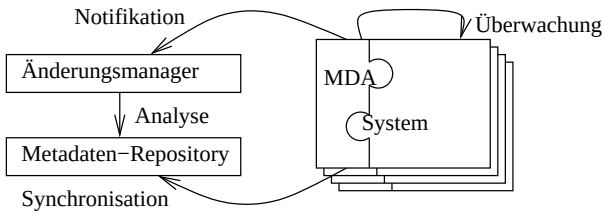

Abbildung 2: Architektur

Bevor eine Analyse stattfinden kann, müssen alle relevanten Metadaten in das MDR geladen werden (Synchronisation). In unserem Fall sind das unter anderem das föderierte Schema und die Quellschemas. Das geschieht teilautomatisiert mit Unterstützung durch die Metadatenagenten, welche proprietäres und MDR-Datenformat gleichermaßen verstehen. Bei einer *geplanten* Änderung wird zuerst beim Änderungsmanager ein Vorschlag eingereicht, welcher in der Regel gleich der neuen Systembeschreibung[1] ist. Der Änderungsmanager analysiert die Auswirkungen des Vorschlags auf andere Systeme (Analyse). Da es (vor allem, wenn mehrere Systeme beteiligt sind) im allgemeinen keine atomare Durchführung gibt, sind während der Durchführungsphase die betroffenen Systeme im MDR entsprechend markiert, und die Metadaten-Agenten können den Zugriff auf Systeme von außen einschränken[2]. Im Falle einer *ungeplanten* Änderung, wenn also im Quell-DBVS die Tabelle PERSON ohne Abstimmung mit dem FDBVS geändert wird, erkennt der zuständige Metadaten-Agent das, alarmiert die zuständigen Personen und den Änderungsmanager (Notifikation) und führt zusammen mit diesem ähnliche Aktionen aus wie auch bei einem geplanten Evolutionsschritt.

Ein wichtiger Punkt unseres Ansatzes ist das Metadatenmodell. Es muss prinzipiell beliebige Datenmodelle ausdrücken können, und auch mit unvollständigen Informationen zurechtkommen. Da wir uns auf die Analyse von möglichen Problemen beschränken ohne tatsächlich Schemaintegration zu betreiben, ist ein solches „universelles“ Datenmodell in unserem Fall durchaus realistisch. Prinzipiell gibt es in unserem Metamodell drei Ebenen. Auf Ebene 1 werden verschiedene *Aussagen* gemacht, die z.B. ein SQL-Schema modellieren. In Ebene 2 machen die Systeme einerseits *Zusagen*, z.B. zur Veröffentlichung ihres Schemas, und haben andererseits *Erwartungen* an Aussagen anderer Systeme. Auf der dritten und obersten Ebene schließlich werden Abhängigkeiten zwischen den Erwartungen und Zusagen modelliert sowie das Verhalten im Konfliktfall festgelegt. Wir prüfen derzeit die Verwendbarkeit von RDF [1] und das Konzept der Reifikation zur Modellierung und Verbindung der Ebenen. Aus Platzgründen kann hier jedoch nicht näher darauf eingegangen werden. Als Grundlage des Datenmodells kann evtl. CWM [2] Verwendung finden, welches in ein RDF-Format umgesetzt wird (z.B. [3]).

4 Abgrenzung und Fazit

Im Bereich der Informationsintegration gibt es viele Ansätze, die sich mit unterschiedlichen Facetten beschäftigen. Wir gehen bei uns davon aus, dass Dinge wie Schema-Matching [4, 5] und Schemaintegration [6] unserem Ansatz vorgeschaltet sind. Einige Arbeiten befassen sich mit ähnlichen Problemstellungen, wie z.B. View Maintenance [7] und View Synchronization [8, 9]. Das Kantana-System [10] definiert den Begriff der Mapping Adoption [11], und ist unserem Ansatz in der Zielsetzung ähnlich. Alle diese Arbeiten konzentrieren sich jedoch hauptsächlich auf die tatsächliche Anpassung der Systeme nach einer Änderung, was ihre Anwendbarkeit einschränkt. Unser Ansatz hingegen verfolgt das Ziel, ein integriertes Informationssystem ganzheitlich zu erfassen, Möglichkeiten zur prozessunterstützten Verwaltung zu geben und auftretende Probleme automatisch zu erkennen und zu analysieren. Auch beschränken wir uns nicht nur auf Schemainformationen, sondern lassen beliebige Metadaten zu. Der RADES-Ansatz [12] verfolgt ähnliche Ziele, setzt jedoch einen Schwerpunkt auf die dort zwingend erforderlichen Prozesse. Ein weiterer ähnlicher, aber beschränkterer Ansatz findet sich in [13]. Auch wenn sich teilweise Überlappungen zeigen, ist unser Ansatz im wesentlichen komplementär zu den vorgenannten. Wir sehen in unserem Ansatz einen Schritt in Richtung des Autonomic Computing [14] auf einer verteilten Ebene.

Unser Ansatz bietet die Infrastruktur für eine ganzheitliche Verwaltung eines integrierten Informationssystems. Geordnete Prozesse zur Systemevolution werden unterstützt, aber auch in der Abwesenheit dieser kann das System arbeiten. Kernpunkt ist die globale Analyse von lokalen Änderungen und die Fähigkeit, selbst bei unvollständigen Informationen nach dem Prinzip der „graceful degradation“ weiter zu funktionieren. Ein weiterer Schritt ist, sich nicht auf das automatische Erkennen von Änderungen

[1] Wir verwenden hier absichtlich nicht das Wort „Schemabeschreibung“, da die Schemainformationen nur einen Teil der gesamten Metadaten ausmachen.

[2] Hierzu ist möglicherweise nötig, Wrapper einzusetzen, durch die die Zugriffe geleitet werden.

und Änderungsproblemen sowie einigen defensiven Aktionen zu beschränken, sondern aktiv und automatisch Probleme aufzulösen. Es gibt hier im Moment noch viele offene Fragen, auch wenn die Basis der Infrastruktur und des Metamodells klar ist. Im Verbund mit anderen Werkzeugen und Systemen zur (automatischen) Informationsintegration nähert man sich jedoch weiter dem Idealbild eines sich selbst verwaltenden Systems, welches nur in „Notfällen“ noch menschliche Unterstützung benötigt.

Literatur

[1] World Wide Web Consortium (W3C). *Resource Description Framework (RDF): Concepts and Abstract Syntax*, February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[2] Object Management Group (OMG). *Common Warehouse Metamodel (CWM) Specification Version 1.1*, March 2003. http://www.omg.org/cwm/.

[3] Sergey Melnik. Representing UML in RDF, 2000. http://www-db.stanford.edu/ melnik/rdf/uml/.

[4] Sergey Melnik, Erhard Rahm, and Phillip A. Bernstein. A survey of approaches to automatic schema matching. *Journal of Web Semantics*, 1/1, 2004.

[5] Erhard Rahm and Phillip A. Bernstein. Developing Metadata-Intensive Appliacitons with Rondo. *VLDB Journal*, 10, 2001.

[6] C. Parent and S. Spaccapietra. Issues and approaches of database integration. *CACM 41/5*, 1998.

[7] S. Ceri and J. Widom. Deriving Production Rules for Incremental View Maintenance. *Proceedings of the 29th VLDB Conference, Barcelona, Spain*, 1991.

[8] X. Zhang and E. A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. Technical report, 1998.

[9] E. A. Rundensteiner, A. Koeller, X. Zhang, A. J. Lee, and A. Nica. Evolvable View Environment EVE: A Data Warehous System Handling Schema and Data Changes of Distributed Sources. *International Database Engineering and Application Symposium (IDEAS)*, April 1999.

[10] Periklis Andritsos, Ariel Fuxman, Anastasios Kementsietsidis, Renee J. Miller, and Yannis Velegrakis. Kanata: Adaptation and Evolution in Data Sharing Systems. *SIGMOD Record*, 33/4, December 2004.

[11] Yannis Velegrakis, Renee J. Miller, and Lucian Popa. Mapping Adoption under Evolving Schemas. *Proceedings of the 29th VLDB Conference, Berlin, Germany*, 2003.

[12] Clemens Dorda, Hans-Peter Steiert, and Jürgen Sellentin. Modellbasierter Ansatz zur Anwendungsintegration. *Information Technology*, 4/2004, August 2004.

[13] L. Deruelle, M. Bouneffa, G. Goncalves, and J. C. Nicolas. Local and Federated Database Schemas Evolution: An Impact Propagation Model. *LNCS: Proc. of the 10th International Conference on Database and Expert Systems Applications (DEXA'99), Florence, Italy*, September 1999.

[14] John J. Ritsko and Alfred G. Davis, editors. *IBM Systems Journal: Autonomic Computing*, volume 42/1. IBM, 2003. http://www.research.ibm.com/journal/sj42-1.html.