Exploratives Modellieren für SAP NetWeaver®

Andreas Tönne, Georg Heeg eK
Januar 2007

In diesem Dokument stellen wir das Prozesskonzept „Explorative Modellierung" für agile Softwareentwicklung vor. Explorative Modellierung verbindet die Tugenden agiler Softwareentwicklung mit einer ausführbaren und damit unmittelbar verifizierbaren Modellrepräsentation in einer besonders geeigneten Programmierumgebung. Als Beispiel zeigen wir den Einsatz in der Modellierung eines heuristischen Rechnungs-Dubletten-Algorithmus in SAP NetWeaver®[1].

Unternehmensspezifische Implementierungen von Standardsoftware

Ein zentraler Erfolgsfaktor für die Implementierung von Individual- und Standardlösungen liegt in der korrekten Abbildung der Geschäftsprozesse des Unternehmens. Hierbei sind besonders die Geschäftsprozesse kritisch, die den Unternehmenszweck und –Individualität besonders berühren, denn diese sind häufig nicht durch ein Standard-System abgedeckt. Hier steht das Unternehmen in dem Widerstreit, seine eigenen Geschäftsprozesse an die Möglichkeiten des Standard-Systems anzupassen oder umgekehrt das Standard-System um diese Geschäftsprozesse zu erweitern. Die Anpassung der Unternehmensprozesse an die Möglichkeiten eines Standardsystems ist mit einem hohen Risiko der Akzeptanz verbunden. Die Kosten für ein Training der Mitarbeiter sowie die sich aus alter Routine und missverstandenen neuen Prozessen ergebenen Fehler können beträchtlich werden. Dies wurde unter anderem durch die im Zuge der Jahr-2000-Problematik oft hastig durchgeführten Einführungen von Standard-Systemen deutlich. Die Implementierung der unternehmenseigenen Prozesse in einem Standardsystem ist daher eine sinnvolle Alternative.

Mit der Enterprise Service-Oriented-Architecture E-SOA auf Basis von Web-Services bietet SAP NetWeaver® basierend auf offenen Standards und innovativen Software-Technologien eine klare Strategie für die Architektur und Implementierung dieser Erweiterungen. Die Herausforderung besteht vor allem darin, ein präzises Verständnis zu entwickeln, was denn nun den zu implementierenden Geschäftsprozess auszeichnet und dieses Verständnis zusammen mit der Fachabteilung des Unternehmens herbeizuführen und zu verifizieren.

Agile Software-Prozesse Innovative Software-Technologien und Service-orientierte Architekturen sind einerseits Faktoren für Exzellenz und Risiko-Kontrolle. Andererseits setzen sie auch ebenfalls innovative Software-(Entwicklungs-)Prozesse voraus, ohne diese die Erfolgsfaktoren nicht einsetzbar sind. Agilität als Oberbegriff bezeichnet derartige innovative Software-Prozesse mit zentral problem-orientierten Prinzipien, die seit einigen Jahren eine Leitfunktion in der Software-Entwicklung haben. Zu den zentralen Werten der Agilität zählen

- Einfachheit
- Innovation
- Kommunikation
- Kundenorientierung

Zum Wesen der Agilität gehört, dass Steuerung, Erfolgkontrolle und Risikomanagement von Softwareentwicklung auf Kommunikation, kontinuierlicher Testbarkeit und Gruppendynamik

[1] SAP® und SAP NetWeaver® sind eingetragene Marken der SAP AG.

(u.a. Verantwortung innerhalb des Teams und des Teams gegenüber dem Unternehmen) basiert. Herkömmliche (Wasserfall-)Software-Prozesse versprechen dagegen eine Pseudo-Sicherheit durch eine umfangreiche formale Steuerung und Dokumentation aller Aspekte der Entwicklung. Derartige Prozesse können die fundamentalen Fehler einer Entwicklung, die aus Nichtwissen oder Kommunikationsproblemen entstehen, nicht vermeiden. Papier ist geduldig! Agile Prozesse dagegen liefern eine unmittelbar für Entwickler und Fachleute einsichtige Gewissheit des Fortschritts.

- Formalistischen, stark bürokratisierten Software-Prozessen werden leichtgewichtige kurze Design- und Entwicklungszyklen entgegengesetzt.
- Dokumenten-lastige getrennte Design- und Entwicklungsschritten werden durch integrierte Entwicklungszyklen ersetzt, in denen das Verständnis und die kontinuierliche Verifikation des Fortschritts an erster Stelle stehen.
- Am Ende jedes Zyklus entsteht eine lauffähige Anwendung, die den aktuellen Stand der implementierten Features darstellt.

Explorative Modellierung

Agile Software-Prozesse haben sich als sehr erfolgreich für das Design und die Implementierung moderner, innovativer Anwendungen erwiesen. Die darin subsumierten Werte und Prinzipien haben sich recht natürlich unter Nutzung der objekt-orientierten Programmierung entwickelt. Beiden ist gemein, dass weniger der technische Implementierungsakt („wie bringe ich es der Maschine bei?“) als der Problem- oder Begriffsaspekt („worüber rede ich eigentlich?“) im Vordergrund steht.

Babylonische Verständnisprobleme Begriffe sind deshalb von besonderer Bedeutung, da ihre korrekte Erfassung und ihre Modellierung von fundamentaler Bedeutung für die Softwareentwicklung sind. Modellierung ist die kritische Phase eines Designs, in der eine Brücke zwischen der Sprache der Fachleute und der Entwickler geschlagen wird. Dies ist die Phase, in der die teuersten Fehler des gesamten Software-Prozesses gemacht werden können! Explorative Modellierung wurde entwickelt, um diese Begriffsfindung und –Verifikation zu fördern.

Neben der unterschiedlichen Sprache und dem unterschiedlichen Begriffsverständnis wirkt bei der Modellierung auch ein höchst unterschiedlicher Grad der Formalisierung und Abstraktion der Begrifflichkeit, was eine Quelle von tiefgreifenden Missverständnissen ist. Während ein Entwickler eine rasche Formalisierung der Begriffe und ihre Einordnung in ein Schema aus Abstraktion und Generalisierung sucht, nutzt der Fachmann andere Wege der Begriffsbeschreibung, die durch Exemplar-Orientierung und Analogieschlüsse geprägt ist.

Paradoxerweise versuchen viele agile und herkömmliche Entwicklungsprozesse diese Kluft der Sprachen durch die Einführung einer weiteren technischen Sprache – UML – zu überbrücken. UML als visuelle Modellierungssprache erfüllt die Forderung nach Formalisierung und Einheitlichkeit der Modellbeschreibung aus Sicht des Entwicklers. UML ist damit sehr geeignet für die Dokumentation von Design und Implementierung. UML ist aber nicht hilfreich für das Verständnis des Modells aus Unternehmenssicht und verletzt damit die agilen Prinzipien der Kundenorientierung und Verständlichkeit. Der Fachmann steht immer noch vor dem Dilemma, sein Verständnis der Anforderungen in der formalen, abstrakten Beschreibung des Informatikers wieder zu finden. Dies ist für viele Fachleute häufig eine Überforderung, mit dem Resultat, dass Fehler der Modellbildung erst spät in den Implementierungszyklen gefunden werden. Fachleute nutzen meist Beispiele für Problembeschreibungen und besitzen keine formale Abstraktion hiervon, die eine Verifikation des UML-Modells erlauben würde.

Die Arbeit mit UML birgt zudem oft das Problem, dass der Übergang von der Modellierung der Fachkonzepte zum Implementierungsdesign unscharf und fließend ist. So kann das UML-Modell zuviel implementierungs-orientiertes Detail enthalten, was das Verständnis des Modells erschwert.

Verifizierbare Modelle statt Diagramme Unser Konzept der explorativen Modellierung setzt an diesem Dilemma an. Anstelle einer abstrakten formalen Beschreibung des Modells setzten wir ausführbare, für den Fachmann unmittelbar erfahrbare und verifizierbare Modelle. Die Modellfindung erfolgt nicht durch Erstellung umfangreicher UML-Diagramme sondern durch agile Modellierungszyklen. Es wird das Modell in einer geeigneten Programmiersprache experimentell implementiert und mit der Fachabteilung abgestimmt, ergänzt und weiterentwickelt.

Den gesamten Prozess aus experimenteller Implementierung, Verifikation und Aktualisierung der Dokumentation nennen wir explorativ. Eine gute technische Einführung in die Prozesse und den sich daraus ergebenden Modellierungsdialogen findet man in „Domain-Driven Design" [1]. Die Schlüsselkonzepte hierbei sind

- Verwendung der fachlichen Begriffe in einer möglichst sprachlich nahen Form
- Keine technische Unterscheidung von Daten, Entities und immateriellen Konzepten wie Services. Alles wird einheitlich dargestellt.
- Abbildung von Prozessen und sprachlich formulierten Abläufen direkt in Programmablauf.
- Wahl der einfachsten möglichen Implementierung, die die Prozesse und Abläufe ausführbar macht.
- Erweiterung der Modell-Implementierung um klar abgegrenzte technische Elemente, die Experimente mit dem Modell ermöglichen (GUI, Schnittstellen zu existierenden Systemen).
- Eine kontinuierliche Feedback-Schleife zwischen Experimenten und aktueller Modelldokumentation.

Gerade der letzte Punkt ist sehr einfach in diesem Konzept, da alle Implementierungsartefakte ohne sprachliche/technische Übersetzung in die Modellbeschreibung zurückfließen können.

Explorative Modellierung als Prozess setzt keinen eigenen Entwicklungsprozess voraus. Man kann die Modellierung störungsfrei in vorhandene agile Entwicklungsprozesse einbetten. Selbst in nicht-agilen oder „Wasserfall-artigen" Prozessen wird explorative Modellierung durch die Verbesserung der Ausgangsbasis der Entwicklung Vorteile bringen. Es ist kein Widerspruch, dass UML als Modellkommunikationssprache als ungeeignet abgelehnt wird, aber UML durchaus zur Dokumentation des jeweiligen Entwicklungsstandes vorteilhaft genutzt werden kann.

Seine größte Wirkung erzielt explorative Modellierung zu Beginn eines Projekts, wenn die Anwendungsdomäne „auf der grünen Wiese" modelliert werden muss. Aber auch in späteren Iterationen gibt es immer wieder Zeitpunkte, in denen das Modell substantiell konzeptionell erweitert wird und eine explorative Modellierung diese Erweiterung absichert. Ein wichtiges Element des Umgangs mit explorativer Modellierung ist das Experiment. Damit dieses möglich ist, muss das experimentelle Modell stets auch in einem Testkontext integriert sein. Das Modell ist nicht beschränkt auf die formale Repräsentation der Konzepte sondern ist stets auch ausführbares Programm.

Agile Modellierungssprache Explorative Modellierung setzt bewusst die Implementierungssprache und Modellierungssprache gleich! Damit hierbei mehr erreicht werden kann als mit herkömmlichem Prototyping, muss die eingesetzte Implementierungssprache selber den agilen Prinzipien folgen. Übersetzt auf Programmiersprachen heißen diese

- Barrierefrei (Untechnisch)
- Meta-programmierbar
- Interaktiv
- Konzept-orientiert

Durch eine untechnische, auf die Darstellung von Konzepten orientierte Sprache erreicht man eine hohe Verständlichkeit der Modellimplementierungen. Es gibt nur geringe Brüche zwischen den Implementierungsartefakten und den Modellbegriffen. Der Fachmann findet „seine Konzept" unmittelbar im Modell vor und kann mit seiner Implementierung Experimente machen.

Interaktivität der Programmierumgebung ist wesentlich für die Akzeptanz des Konzepts der explorativen Modellierung. Größere Verzögerungen durch aufwändige Compile-Link-Test-Zyklen stören die Kommunikation mit der Fachabteilung und das Durchführen von Experimenten.

Meta-Programmierbarkeit ist sehr wichtig, damit die Modell-Experimente möglichst genau den Fachkonzepten nachgebildet werden können. Der „Abstand" zwischen den Fachkonzepten und der experimentellen Implementierung muss so gering wie möglich gehalten werden. Es soll nicht notwendig sein, dass ein Fachkonzept aus technischen Gründen in ein implementierungs-orientiertes Konzept übersetzt werden muss.

Es gibt in der aktuellen Software-Landschaft eine kleine Zahl von hierfür geeigneten Programmiersprachen. Ihnen allen ist gemein, dass ein dynamisches Typsystem besitzen, rein objekt-orientiert und mit mächtigen Meta-Programmierkonzepten ausgestattet sind und eine abgestimmte leistungsfähige Programmierumgebung mitbringen. Am nächsten kommt dem Ideal einer Programmiersprache für exploratives Modellieren die Sprache Smalltalk, gefolgt von Ruby.

Smalltalk ist deshalb so attraktiv für exploratives Modellieren, weil die Sprache den geringsten technischen Ballast aller aktuellen Sprachen hat und außerordentlich kurze Entwicklungszyklen kennt. Der geringe technische Ballast drückt sich darin aus, dass nur ein minimaler Aufwand der Deklaration von Programmstrukturen getrieben werden muss und die Sprache auf wenigen einfachen und mächtigen Konzepten ruht. Entwicklungszyklen (Edit-Compile-Link) gibt es in dieser Form überhaupt nicht. Stattdessen wird inkrementell das Programm Methode für Methode aufgebaut und zu jedem Zeitpunkt ist das Programm ausführbar. Das ist außerordentlich förderlich für die gemeinsam mit dem Fachmann durchgeführte Modellierung.

Exploratives Modellieren in der Praxis mit SAP NetWeaver®

Experimente mit den Modellen ist ein Schlüsselkonzept explorativer Modellierung. Fachleute brauchen Beispiele! Fachleute brauchen Anschauung! Kein Modell erfüllt diese Bedürfnisse, wenn es nicht in einer realistischen Testumgebung ausprobiert und erfahren werden kann.

Hierbei ist zu sehen, dass die experimentellen Modelle nicht Selbstzweck und vollständig in sich lauffähig sind. Modelle bestehen in der Regel nicht nur aus neu modellierten Konzepten sondern stehen im Bezug zu einer existierenden fachlichen Umgebung. Wollen wir mit einem Modell Experimente machen, so ist diese fachliche Umgebung ebenfalls in das experimentelle Modell aufzunehmen. Hierbei kann man zum Beispiel Mockups der fachlichen Umgebung erstellen oder auch ihre in der Testumgebung existierende Implementierung über eine Fassade nutzbar machen.

Wenn wir exploratives Modellieren mit SAP NetWeaver® betreiben, dann besteht die Implementierung der fachlichen Umgebung aus RFC-Modulen oder Web-Services. Das Programmiersystem muss die Integration und Import dieser Module und Services in das explorative Modell möglichst leichtgewichtig unterstützen. Für Cincom Smalltalk wurde hierfür eine dedizierte Integrationskomponente geschaffen, die alle agilen Anforderungen unterstützt.

Die Integration des experimentellen Modells in eine Testumgebung erfolgt hauptsächlich zur Durchführung von Experimenten und der Verifikation der Modellbegriffe. In der Praxis wird man hierfür weitere Implementierungen für die Testumgebung erstellen, zum Beispiel Testoberflächen, Datenexporte- und –Importe oder Analysewerkzeuge. Es ist aber ebenso denkbar, dass das experimentelle Modell direkt in eine reale Anwendung integriert wird und unmittelbar in der laufenden Anwendung geprüft wird. Zum Beispiel, indem man das experimentelle Modell selber als Services bereit stellt.

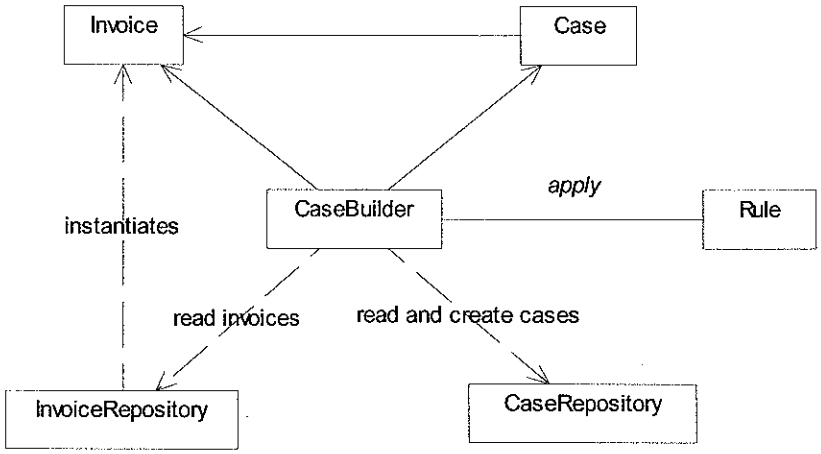
Duplicate-Analyzer – Ein Fallbeispiel

Das Beispiel Duplicate-Analyzer zeigt, welche hochwertigen Modellierungsergebnisse man in kurzer Zeit mit explorativer Modellierung erzielen kann. Es handelt sich hierbei um ein Projekt zusammen mit dem Business Process Renovation-Team der SAP Labs, das mit Cincom Smalltalk als Modellierungssprache durchgeführt wurde. Es wurden neue Konzepte für eine bereits bestehende, unbefriedigende Lösung gefunden und verifiziert. Die hier beschriebenen Arbeiten der explorativen Modellierung fanden in einem Team von zwei SAP Entwicklern und zwei Beratern von Georg Heeg eK in einem Zeitraum von sieben Tagen statt.

Aufgabenstellung eines Duplicate-Analyzer ist das Auffinden von Rechnungs-Dubletten im Datenbestand von FI (Finanzbuchhaltung). Diese Aufgabe teilte sich in eine Heuristik zur Findung potentieller Dubletten (Verdachtsfälle) und eines Business-Prozesses zur Verarbeitung dieser Verdachtsfälle durch einen Sachbearbeiter auf. Die Modellierungsaufgabe bestand darin, den sogenannten Case-Builder zu definieren, der auf Basis von Heuristiken plausible, in ihrem Umfang handhabbare Verdachtsfälle aufbaut. Die bisherige Lösung hatte diese Aufgabe nur unvollständig gelöst. Das gewählte Modell wurde durch eine vorhandene Implementierungskomponente für Volltextsuche (TREX) und insbesondere durch seine Suchalgorithmen bestimmt und damit beschränkt. Die gewünschte Flexibilität der Algorithmen war durch die Implementierung getriebene Modellierung unmöglich geworden.

Die interessante Fragen, die natürlich auch gleich zu Anfang gestellt wurden, sind: Wie entstehen eigentlich doppelte Rechnungen und woran erkennt man sie? Letzteres ist eine typische unklare Problemstellung, wobei die Antwort einem Sachbearbeiter einfach fällt. Er vergleicht zwei verdächtige Rechnungen im Detail und schließt mithilfe seiner Erfahrung und Kenntnisse der in dem Unternehmen typischen Fehlerquellen, ob er eine Dublette gefunden hat. Diese Dubletten können dann vom Leistungsempfänger zurückgefordert werden. Hierin liegt der wirtschaftliche Nutzen der Anwendung.

Iteration 1 Die Schlüsselfrage der ersten Iteration war für uns, was verdächtig ähnliche Rechnungen darstellt und wie wir sie in Verdachtsfällen für den Sachbearbeiter zusammenfassen. Ganz im Sinne von explorativer Modellierung haben wir gemeinsam die zu dem Zeitpunkt bekannten Modellbegriffe ausprogrammiert. Das entstehende Modell war recht einfach und tat nichts anderes, als alle geprüften Rechnungen als verdächtig ähnlich zu allen anderen Rechnungen zu sehen.



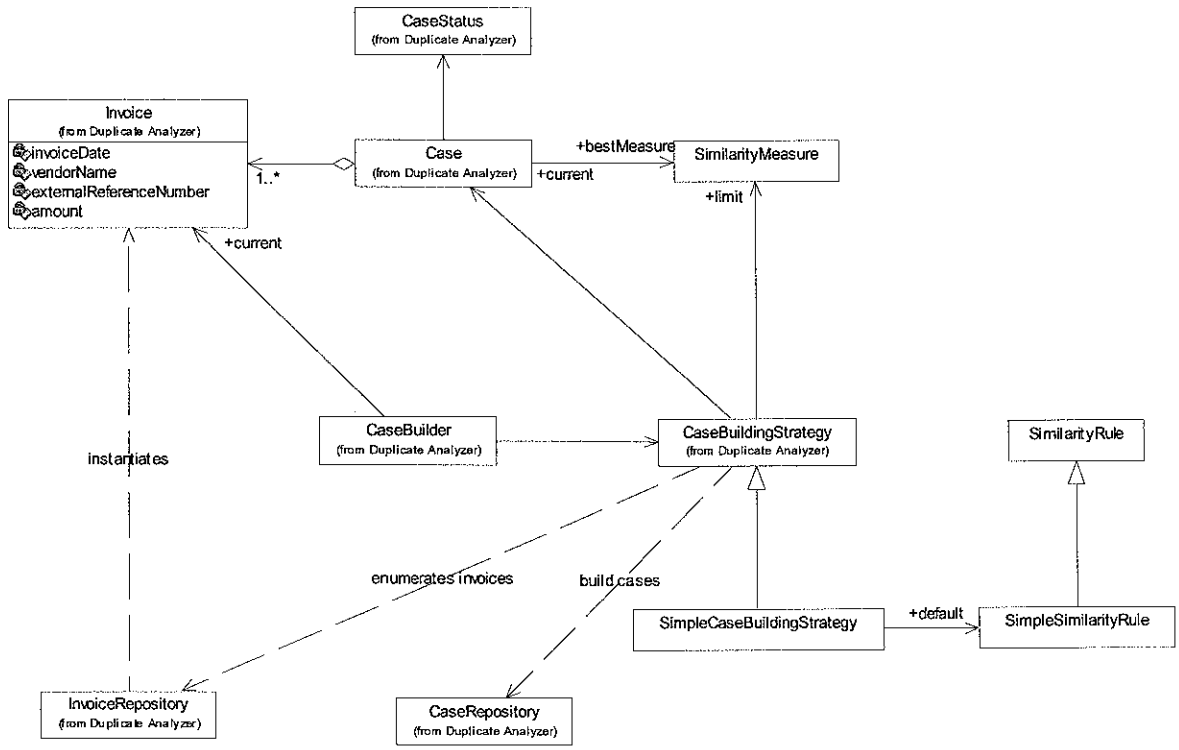
Rechnungen sind hier Modell der aus dem FI-System importierten Rechnungen mit den Attributen Referenzschlüssel (Kundeneingabe), Rechnungsdatum, Name des Rechnungsstellers und Betrag. Die Definition der Ähnlichkeit zweier Rechnungen steckt in den Regeln und ist auf den Attributen der Rechnungen definiert. Damit war eine Basis für erste Experimente geschaffen.

Dieses simple Modell wurde dazu um erste, offensichtliche Regeln erweitert und soweit komplettiert, dass Rechnungen importierbar und analysierbar wurden. So zum Beispiel wurde eine Regel, die Namen auf Tippfehler und Beträge auf prozentuale Abweichungen vergleicht eingeführt. Die Anwendung wurde mit realen Daten von circa 120.000 Rechnungen ausgeführt und die erhaltenen Ergebnisse studiert.

Innerhalb von vier Tagen haben wir eine vollständig lauffähige Anwendung erstellt, die das derzeitige Wissen um Rechnungsdubletten umfasst und konkrete Ergebnisse lieferte. Ein Testlauf mit den 120.000 Datensätzen dauerte etwa eine Stunde und lieferte die erwarteten Ergebnisse: Viel zu viele verdächtig unverdächtige Verdachtsfälle. Die offensichtlichen Regeln für Ähnlichkeiten und die Heuristiken für die Fallbildung waren nicht stark genug, da sie die praktischen Gründe für Rechnungsdubletten nicht berücksichtigt haben.

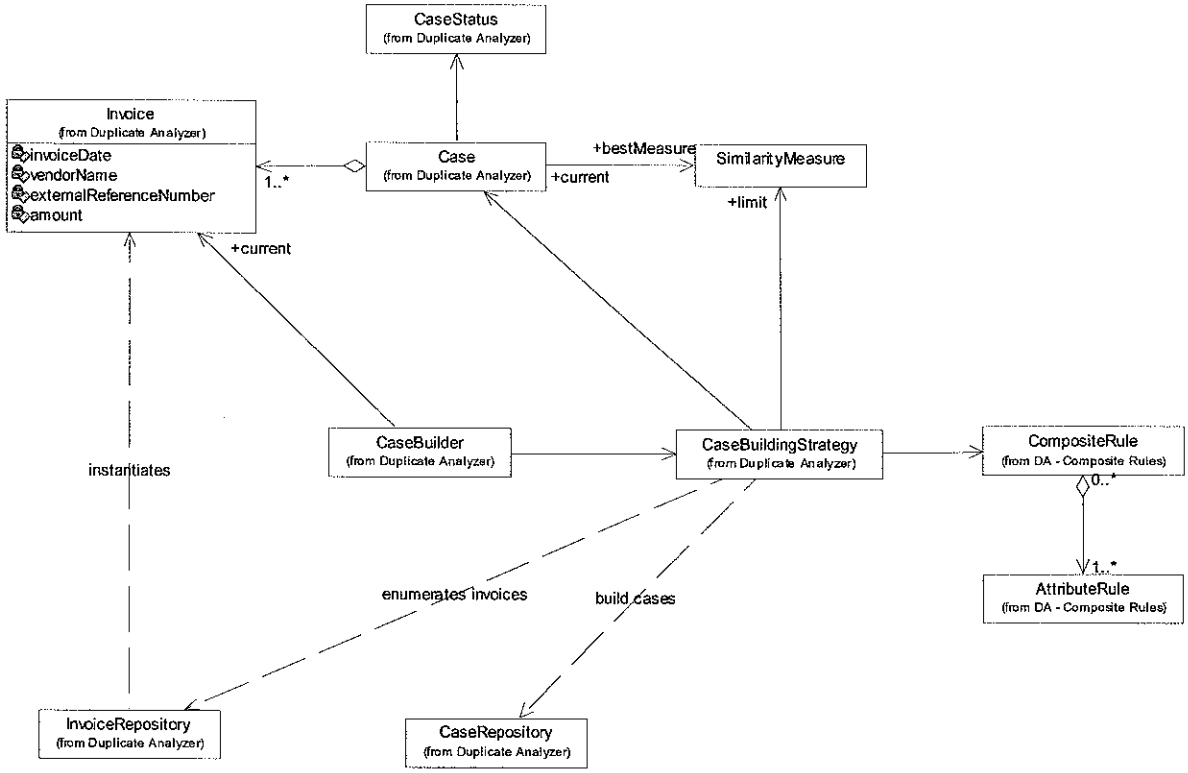
Iteration 2 Ausgehend von diesen ersten Experimenten begann eine Diskussionsphase um alternative Regeln und die Gründe für Rechnungsdubletten. So kann eine Rechnung über verschiedene Sachbearbeiter (d.h. verschiedene Postwege) doppelt erfasst werden und identisch bis auf den Schlüssel sein. Oder eine Rechnung wird, wie im internationalen Verkehr nicht unüblich, unter vertauschtem Monat und Tag erfasst. Oder eine Rechnung ist angemahnt, kann wegen falscher Kundennamen nicht gefunden werden und wird als Dublette erneut erfasst. Neben den Gründen für Rechnungsdubletten wurden auch die Ähnlichkeitsbegriffe intensiv diskutiert: Ist es besser einen Rechnungsbetrag auf prozentuale Abweichung oder auf Tippfehler zu vergleichen? Macht es Sinn, beide Vergleiche zugleich einzusetzen oder verschlechtert man damit den Vergleich?

Ausgehend hiervon wurde das Modell flexibler gestaltet und die Konfigurationsmöglichkeiten stark erweitert.

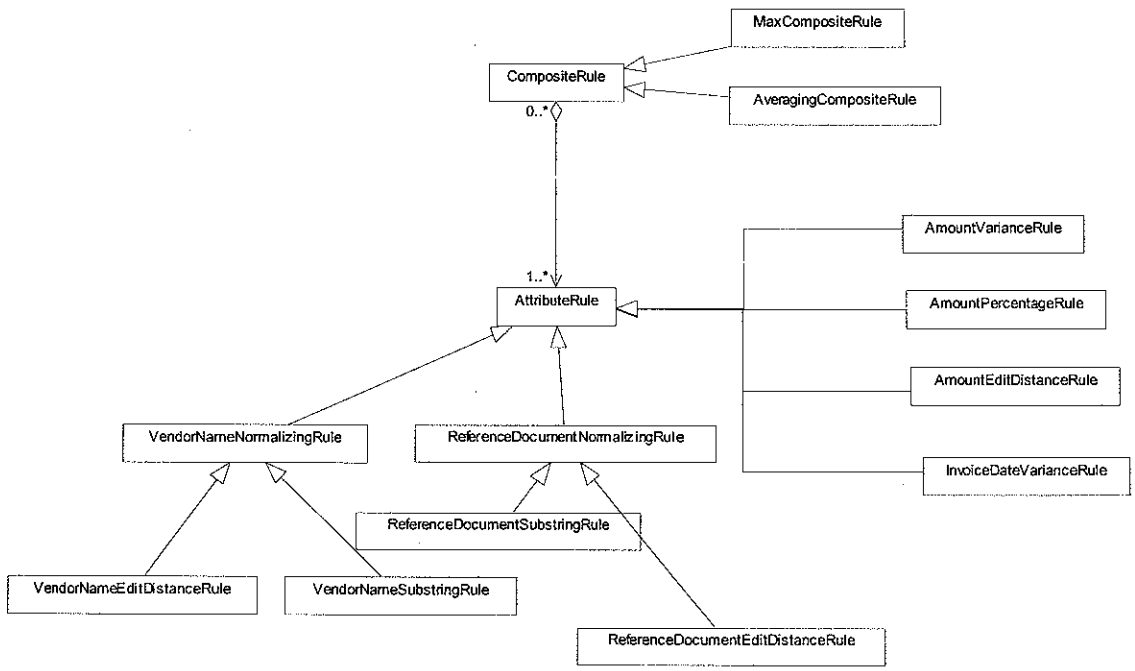


Mit diesem neuen Modell wurden erneut Experimente durchgeführt, wobei eine Vielzahl von Strategien und Regeln entworfen wurden. Der Ablauf des Experiments war annähernd ein Brainstorming mit sehr kurzen Zyklen. Die Modelländerungen für die zweite Iteration und die Experimente haben zusammen etwa zwei Tage gedauert. Das Ergebnis dieser Iteration war neben einem verbesserten Modell und damit verbesserten Verständnis der Problem-Domäne eine praktische Erkenntnis: Weiteren Fortschritt bei der Verbesserung der Heuristiken können wir nur dann erzielen, wenn wir den Aufwand für das Erstellen von neuen Regeln sowie die Kombinationen von Regeln in Heuristiken weiter vereinfachen und damit beschleunigen. Eine weitere Erkenntnis, die wir unter anderem durch Konsultation von kaufmännischen Spezialisten der SAP gewonnen haben ist die, dass die Heuristiken für verschiedene SAP-Kunden sehr unterschiedlich sein können. Insbesondere weil die Gründe für die Entstehung von Dubletten von Unternehmen zu Unternehmen unterschiedlich ausfallen.

Iteration 3 In einer letzten großen Iteration haben wir diese Erkenntnisse in eine weitere Flexibilisierung sowie in passende Experimentierwerkzeuge umgesetzt. Die zentrale Erweiterung des experimentellen Modells war die Verlagerung der Ähnlichkeitsalgorithmen der Rechnungen von den Regeln hin zu den individuellen Attributen. Eine spezifische Regel konnte damit per Konfiguration aus passenden Ähnlichkeitsalgorithmen für die Rechnungsattribute zusammengestellt werden.



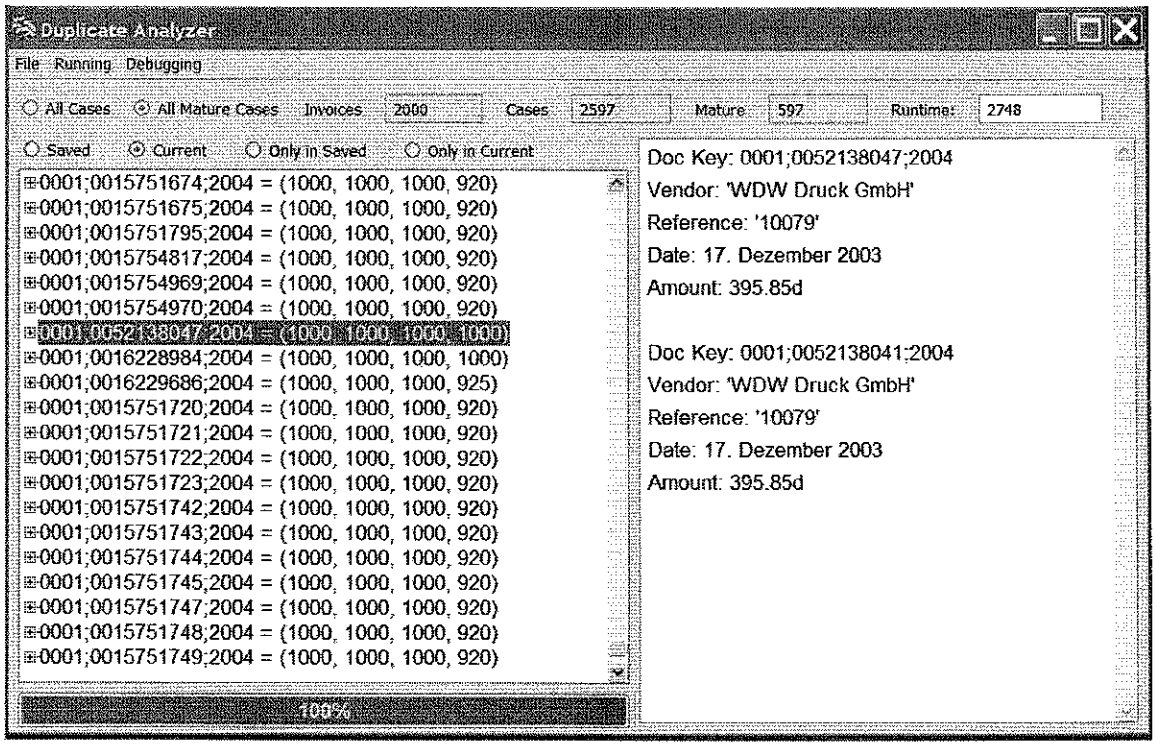
Es wurden vielfältige experimentelle Attribut-Regeln erzeugt und ihre Kombinationen gegen den Testdatenbestand verifiziert. So zum Beispiel Varianten der Ähnlichkeiten von Namen (Substring, Tippfehler) oder Ähnlichkeiten von Beträgen (Abweichungen Absolut oder Prozentual, Tippfehler).



Für die Durchführung der Experimente wurde ein Experimental-GUI entwickelt, mit dem die Fachabteilung „was wäre wenn" Experimente durchführen konnte. Es erlaubte leicht individuelle Regeln und Parameter einzustellen und auch verschiedene Läufe des CaseBuilder zu vergleichen. Läufe der Anwendung mit den Testdaten ergaben erheblich bessere Ergebnisse

in der Qualität und Zahl der gefundenen Verdachtsfälle bei einer Verdoppelung der Laufzeit auf 2 Stunden. Es wurden auch neue Dubletten gefunden, die für die Testdaten vorher nicht bekannt waren.

Aus einem starren Algorithmus wurde ein hochflexibles Modell eines konfigurierbaren Regelsystems für Rechnungsvergleich, das es dem Kunden ermöglicht, eine bestimmte Heuristik experimentell für eine bestimmte Datenbasis zu optimieren.



Fazit

Explorative Modellierung ist ein leistungsfähiger Ansatz zur interaktiven Findung, Kommunikation und experimentellen Verifikation von Modellen. Er basiert auf einer experimentellen Implementierung der Modellbegriffe in einer besonders geeigneten Programmiersprache sowie der systematischen Durchführung von Experimenten mit diesen ausführbaren Modellen.

Explorative Modellierung verbindet die Prinzipien agiler Softwareentwicklung wie Kundenorientierung, Kommunikation und Einfachheit mit einem für den Fachmann anschaulichen und damit verständlichen Ergebnis. Die notwendige formale Stringenz des Modells wird durch die Verifikation über die Implementierung besser gewährleistet als durch ein statisches UML-Diagramm. Die Ergebnisse der explorativen Modellierung können in vielfältigen Gesamt-Entwicklungsprozessen verwendet werden.

Wir haben gezeigt, dass explorative Modellierung speziell bei unklaren, schwach formalisierten und auch bei sich rasch ändernden Anforderungen in kürzester Zeit sehr gute und vor allem tragfähige Resultate liefern kann.

Literatur:

[1] „Domain-Driven Design“, Eric Evans, Adison-Wesley, ISBN 978-0321125217

Über den Autor

Dipl. Inform. Andreas Tönne ist Prokurist bei Georg Heeg eK. Zu seinen Aufgabenbereichen gehören Projektmanagement, Produktentwicklung und Leitung des Standorts Dortmund.

Über Georg Heeg eK

Georg Heeg eK ist der führende Anbieter für Produktentwicklung, Beratung und Schulung für Smalltalk im deutschsprachigen Raum. Georg Heeg eK entwickelt in enger Kooperation mit dem Smalltalk-Hersteller Cincom Systems neue Technologien und Werkzeuge für Smalltalk. Standorte sind Dortmund, Köthen (Anhalt) und Zürich (Schweiz).

Kontakt

Georg Heeg eK
Baroper Strasse 337
D-44227 Dortmund

Email: info@heeg.de
atoenne@heeg.de
Web: http://www.heeg.de
Tel.: +49 231 9 75 99 0
Fax: +49 231 9 75 99 20