Interner Bericht

Ähnlichkeit von Prozeßmodellen

Martin Verlage
Horst Hientz

277 / 95

Fachbereich Informatik

Universität Kaiserslautern • Postfach 3049 • D-67653 Kaiserslautern

Ähnlichkeit von Prozeßmodellen

Martin Verlage
Horst Hientz*

277 / 95

* Q–Labs Software Engineering GmbH
Technopark I
Sauerwiesen 2
67661 Kaiserslautern

Herausgeber: AG Software Engineering
Leiter: Prof. Dr. H. Dieter Rombach

Kaiserslautern, Dezember 1995

Kurzfassung

Die Sichten von Projektmitgliedern auf Prozesse von Software-Entwicklungen sollen in der Prozeßmodellierungssprache MVP–L formuliert und anschließend in ein Umfassendes Prozeßmodell integriert werden. Dabei ist die Identifikation ähnlicher Informationen in verschiedenen Sichten von Bedeutung. In dieser Arbeit berichten wir über die Adaption und Synthese verschiedener Ansätze zum Thema Ähnlichkeit aus unterschiedlichen Domänen (Schema–Integration beim Datenbank–Entwurf, Analoges und Fallbasiertes Schließen, Wiederverwendung und System–Spezifikation). Das Ergebnis, die Ähnlichkeitsfunktion *vsim*, wird anhand eines Referenzbeispiels illustriert. Dabei gehen wir insbesondere auf die Eigenschaft der Funktion *vsim* ein und berichten über Erfahrungen im Umgang mit dieser Funktion zur Berechnung der Ähnlichkeit zwischen Prozeßmodellen.

Inhalt

1 Einleitung 1

2 Sichten auf Software–Entwicklungsprozesse 2

2.1 Prozeßmodellierung 2

2.2 Multi–View Modellierung 3

3 Ähnlichkeit von Prozeßmodellen 7

3.1 Eigenschaften der Ähnlichkeit im allgemeinen 7

3.2 Anforderungen an die Ähnlichkeitsfunktion 8

3.3 Der Ähnlichkeitsbegriff in anderen Domänen 10

3.4 Definition der Ähnlichkeit von Sichten 11

4 Validierung anhand des ISPW–Beispiels 16

5 Bisherige Erfahrungen 20

6 Zusammenfassung 22

7 Literatur 23

Verzeichnis der Abbildungen

Abb. 1: Konzepte der Sprache MVP–L 2

Abb. 2: Sicht Testingenieur 3

Abb. 3: Integration einzelner Sichten zu einem Umfassenden Software–Prozeßmodell 4

Abb. 4: Die Schritte bei der Sichten–Integration 5

Abb. 5: Ähnlichkeit von Ausschnitten zweier Sichten (Lösungen des ISPW-Beispiels in MVP-L und EPOS) 13

Abb. 6: Prozeßhierarchien zweier Lösungen zum ISPW–Beispiel 18

Abb. 7: Ähnlichkeitswerte von Validate_design im Vergleich 18

Verzeichnis der Tabellen

Tabelle 1: Levenshtein-Distanz für Modify_design und Modify Design 14

Tabelle 2: Produktflußähnlichkeit zweier Prozeßmodelle 14

Tabelle 3: Ähnlichkeitswerte für alle Sichten-Paare 17

1 Einleitung

Prozeßmodelle sind Repräsentationen von Software-Entwicklungsaktivitäten der realen Welt [2]. Die Modelle entsprechen Sichten von Rollen auf einen Software–Entwicklungsprozeß. Die Sichten werden in der Prozeßmodellierungssprache MVP-L (multi-view process modeling language) beschrieben. Die Ähnlichkeit zwischen getrennt entwickelten Prozeßmodellen (Sichten) wird benötigt, um Sichten einzelner Rollen zu integrieren. Ein Umfassendes Prozeßmodell entsteht, welches als Repräsentation des Projekts alle Rollen unterstützt. Die Ähnlichkeit zwischen Prozeßmodellen ist außerdem nützlich für die Wiederverwendung von Prozeßmodellen [3], die Überprüfung der Einhaltung von Standards (z.B. [4]) oder dem Vergleich von Prozeßmodellen zum besseren Verständnis (z.B. [5]).

In der Informatik wird die Ähnlichkeit bei der Integration von Datenbank–Schemata [6], der parallelen Entwicklung von System–Spezifikationen [7], dem Fallbasierten Schließen [8] dem Analogen Schließen [9,10] und der Wiederverwendung [11,12] erforscht. Die Probleme lassen sich in zwei Klassen unterteilen: a) Zu einem vorgegeben Objekt (Anfrage) möchte man aus einer Menge von Objekten das ähnlichste oder die ähnlichsten Objekte auswählen (Fallbasiertes Schließen, Wiederverwendung, Analoges Schließen), oder b) zu zwei Objekten wird die Ähnlichkeit ermittelt, um Anpassungen vorzunehmen oder Informationen zu vermischen (Schema–Integration, Spezifikation, Analoges Schließen). Alle Teilgebiete haben wichtige Erkenntnisse hervorgebracht, die Eingang in die in diesem Papier beschriebene Arbeit gefunden haben; folgende Ziele sind jedoch in ihrer Gesamtheit von keinem der betrachteten Ansätze erreicht, sodaß wir eine eigene Definition benötigen:

1. Es dürfen nicht nur die Bezeichner von Objekten verglichen; dies würde voraussetzen, daß die Bezeichner das Objekt genau charakterisieren.

2. Die Betrachtung der Ähnlichkeit muß unterschiedlichen Zwecken dienen. Starre Charakterisierungsschemta sind deshalb untauglich.

3. Die betrachteten Objekte besitzen keine einheitliche Größe. Wir nehmen an, daß auch die Ähnlichkeitsbetrachtung zwischen Objekten unterschiedlicher Größe von Nutzen ist.

4. Ähnlichkeit muß partiell definiert werden; die Entdeckung von Ähnlichkeiten auf feineren Abstraktionsebenen wird dadurch ermöglicht.

Die hier definierte Ähnlichkeit beruht nicht auf Bezeichnern oder Attributierungen, sondern auf der Struktur der Prozeßmodelle. Dadurch werden auch Ausschnitte von Prozeßmodellen vergleichbar (z.B. Ähnlichkeit beim Produktzugriff oder der Zuordnung von Ressourcen). Zuerst wollen wir in Abschnitt 2 auf die Domäne eingehen, aus der unsere Problemstellung stammt. Danach diskutieren wir unseren Ähnlichkeitsbegriff in Abschnitt 3 und definieren die Funktion *vsim*. Ergebnisse aus einer Studie illustrieren Eigenschaften unserer Ähnlichkeitsfunktion in Abschnitt 4, bevor wir in Abschnitt 5 über Erfahrungen berichten und zusammenfassen (Abschnitt 6).

2 Sichten auf Software–Entwicklungsprozesse

Software–Entwicklungsprojekte sind schwierig zu beherrschen, da viele spezielle Prozesse zur Entwicklung des Produkts notwendig sind. Um die Software-Entwicklungsprozesse systematisch zu kontrollieren, benötigt man explizite Repräsentationen, also sog. Prozeßmodelle. Zum Beispiel wird im *Capability Maturity Model* verlangt, daß zum Erreichen des *Level 3* explizite Prozeßmodelle vorliegen [13]. Allgemein verfolgt die Prozeßmodellierung folgende Ziele (vgl. [14,15]): *Unterstützung des Verständnisses und der Kommunikation zwischen Menschen, Unterstützung von Prozeßverbesserung, Unterstützung der Projektlenkung, Automatisierte Anleitung im laufenden Projekt, Unterstützung der Erstellung von Prozeßhistorien,* und *Automatisierte Unterstützung der Projektabwicklung.* Prozeßmodelle können in beliebiger Formalität – von informell bis zu formal – beschrieben sein.

2.1 Prozeßmodellierung

Im MVP–Projekt wird zur Prozeßmodellierung die Sprache MVP–L benutzt [2,15]. Abbildung 1 gibt die in MVP–L realisierten Konzepte entsprechend der Notation in [16] wieder.

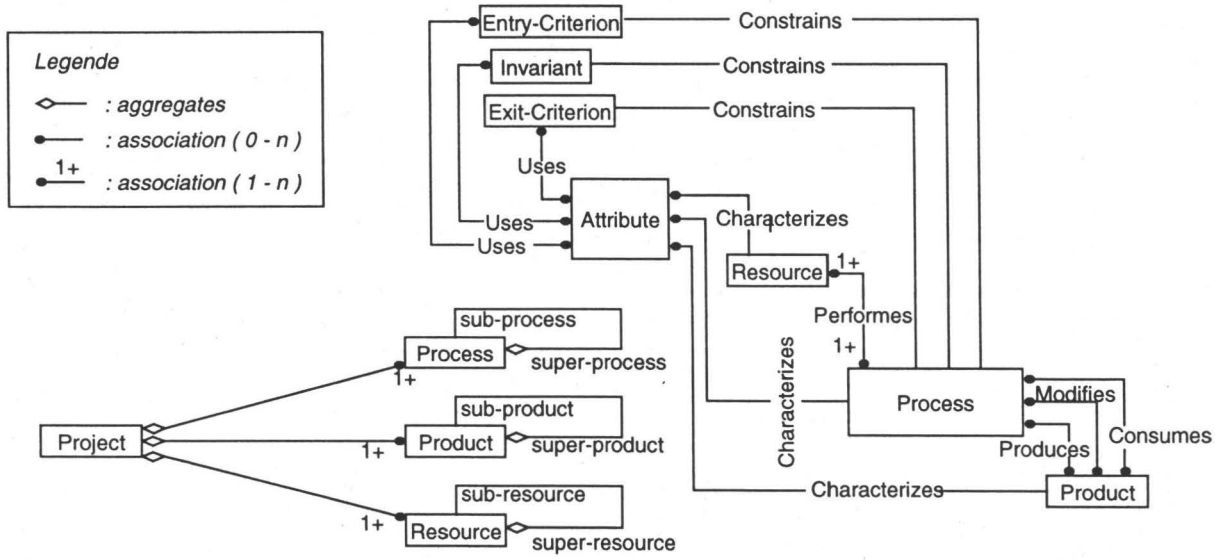

Abb. 1: Konzepte der Sprache MVP–L

In MVP-L sind die wesentlichen Konzepte *Prozesse*, *Produkte* und *Ressourcen*. Sie werden durch Typen (Modelle) beschrieben. Alle Prozesse, Produkte und Ressourcen können Attribute besitzen und in Subobjekte verfeinert werden. Die Prozesse als komplexeste Objekte sind darüberhinaus noch strukturiert in Produktschnittstelle, abwickelnde Ressourcen (Personen oder Werkzeuge), Kontextinformationen und Vor- und Nachbedingungen. Die Beziehungen *produces, modifies,* und *consumes* zwischen Prozessen und Produkten werden unter dem Begriff *Produktfluß* zusammengefaßt. In einem Projektplan kann man die Instanziierungen der Modelle beschreiben und die Objekte mit Parametern versorgen.

MVP-L ist eine Sprache für *process modeling in-the-large*, d.h. es wird stärker auf die Beziehungen zwischen Objekten (Prozessen, Produkten und Ressourcen) eingegangen als auf deren Se-

mantik oder Implementierung. Zu den wesentlichen Beziehungen zwischen Objekten gehören die Anordnung von Prozessen und Produkten in Aggregationshierarchien, die Zuordnung von Ressourcen zu Prozessen, die Produktflußbeziehungen und die Kontrollflußbeziehungen über die Vor- und Nachbedingungen (Exit und Entry Criteria genannt). Die Spezifikation des Kontrollflusses über Bedingungen machen MVP-L zu einer *regelbasierten* Sprache, die jedoch im Unterschied zu anderen regelbasierten Sprachen aufgrund der Prozeßhierarchie auch eine Regelhierarchie besitzt. MVP-L ist vergleichbar mit Sprachen zur Spezifikation von Software-Produkten.

Die Konzepte von MVP–L sind ausreichend für die Modellierung von realen Entwicklungsprozessen [17]. Sie werden im Sprachbericht beschrieben [15]. Eine Abgrenzung zu anderen Prozeßmodellierungssprachen findet sich in [18]. Möchte man MVP–L, oder eine andere Prozeßmodellierungssprache einsetzen, so müssen die Prozeßmodelle die Realität widerspiegeln. Dies ist eine Anforderung nicht nur an den zur Dokumentation verwendeten Formalismus, sondern auch an das Vorgehen bei der Modellierung.

2.2 Multi–View Modellierung

Nimmt ein Projektmitglied eine bestimmte Rolle an (z.B. Tester) und führt darin einen Prozeß aus (z.B. Modifizieren der Testpläne), so bezieht er sich auf eine Teilmenge aller Informationen im Projekt. Die Rolle hat eine Sicht (engl. *view*) auf Software–Entwicklungsprozesse. Jede Sicht stellt ein eigenes, im allgemeinen in mehrere Subprozesse verfeinertes Prozeßmodell dar, welches durch das MVP-L-Konstrukt *project_plan* instanziiert wird. Ein Beispiel für eine solche Sicht ist in Abbildung 2 gegeben. Die grafische Darstellung erfolgt gemäß [19].

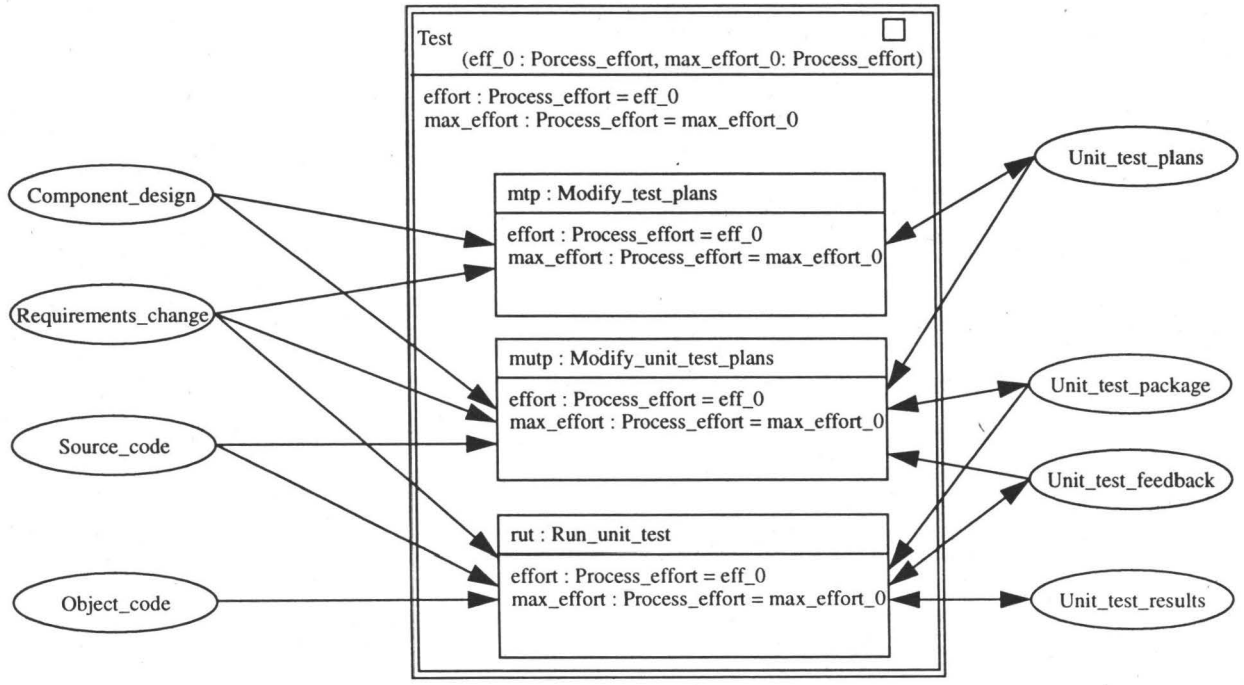

Abb. 2: Sicht *Testingenieur*

Eine *Sicht* ist ein instanziiertes Prozeßmodell in MVP-L. Die Sicht beschreibt einen Zustand (gegeben durch die Menge der instanziierten Objekte) und Prozesse, wodurch beschrieben ist, wie

das Projekt aus dem entsprechenden Blickwinkel fortzuführen ist. Da in einem Projekt viele unterschiedliche Rollen existieren, die ein gemeinsames Ziel verfolgen, stehen auch ihre Sichten in Beziehungen zueinander. Zum Beispiel 'sehen' ein Anforderungsingenieur und ein Entwurfsingenieur dasselbe Dokument *Anforderungsbeschreibung*. Der Projektmanager kann z.B. am Aufwand für die von beiden Ingenieuren abgewickelten Prozesse interessiert sein. Das Ziel der Multi–View Modellierung ist, die Sichten der einzelnen Projektmitglieder zu einem Umfassenden Prozeßmodell zu integrieren. Das Umfassende Prozeßmodell dient dann als Repräsentation des gesamten Projekts und wird zur Koordinierung der Aktivitäten der Projektmitglieder genutzt. Abbildung 3 zeigt abstrakt das Vorgehen bei der Multi–View Modellierung. Erfahrungen bei der Schema–Integration während des konzeptuellen Datenbankentwurfs, zu dem gewisse Analogien existieren, haben gezeigt, daß diese Art der Modellierung i.allg. problematisch durchzuführen ist. Wir wollen jedoch zuerst die Vorteile einer solchen Vorgehensweise diskutieren, bevor die Probleme einzeln angesprochen werden.

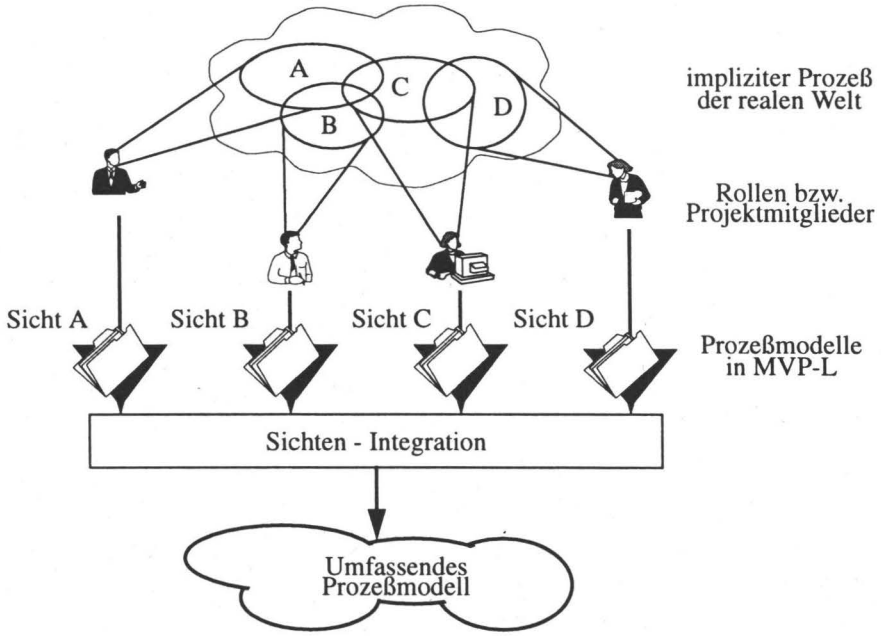

Abb. 3: Integration einzelner Sichten zu einem Umfassenden Software–Prozeßmodell

Es bestehen im wesentlichen drei Vorteile bei der Multi–View–Modellierung [20]:

- *Entdecken von Inkonsistenzen zwischen Sichten.* Während der Integration der einzelnen Sichten können Inkonsistenzen zwischen ihnen aufgedeckt werden. Ein Teil dieser Inkonsistenzen kann auf unterschiedlichem Verständnis derselben Prozeßinformation (z.B. Produkte, Vorbedingungen von Prozessen) beruhen. Die Multi–View–Modellierung hilft diese Inkonsistenzen zu beheben

- *Identifikation von Prozeßmodellen zur Wiederverwendung.* Prozesse sind großteils aufgebaut auf Erfahrung unterschiedlicher Personen. Man kann sich diese Erfahrung zunutze machen, indem man das Prozeßwissen wiederverwendet [3]. Eine Form kann die Ablage solchen Wissens in Form von *Rollenbeschreibungen* sein, die für die Erledigung bestimmter Aufgaben zugeschnitten sind. Jede Rollenbeschreibung ist ein Sicht auf ein Projekt. Die Multi–View–Model-

lierung unterstützt die Wiederverwendung von Rollenbeschreibungen, indem diese einzeln für jede Aufgabe identifiziert und dann integriert werden.

- *Vergleichen mit Standardmodellen.* Standards für Software-Entwicklungsprojekte (z.B. die DIN/ISO 9000 Familie [21]) stellen Sichten dar, die erfüllt werden müssen. Vergleicht man die einzelnen Sichten der Projektmitglieder und die Standard–Sichten und kommt zu einem schlechten Ergebnis (die Sichten ähneln sich nicht), so läßt dies den Schluß zu, daß die Standards nicht erfüllt sind.

Die drei Vorteile können durch dasselbe Vorgehen erreicht werden. Dieses Vorgehen (siehe Abbildung 4) weist Parallelen zur Schema–Integration beim konzeptuellen Datenbankentwurf auf [22]. Wir wählten eine binäre Leiterstrategie, da aus dem Datenbankentwurf bekannt ist, daß diese Strategie den Aufwand der Integration gering hält [6]. Eine Integration von mehr als zwei Sichten würde diesen Schritt komplexer machen, ohne daß weitere wesentlichen Problemstellungen entstünden. Bei der binären Leiterstrategie werden jeweils zwei Sichten integriert und ergeben ein Zwischenresultat (wiederum eine Sicht). Dieser Prozeß kann dann mit weiteren originären Sichten weitergeführt werden. Bei der Entdeckung von Inkonsistenzen stammen die Sichten (z.B. Sicht A und Sicht B in Abbildung 4) von Projektmitgliedern. Bei der Wiederverwendung ist mindestens eine der Sichten ein wiederverwendetes Prozeßmodell. Beim Vergleich mit Standards handelt es sich bei einer der Sichten um ein Umfassendes Prozeßmodell, erstellt aus den Sichten der Projektmitglieder, wogegen die andere Sicht ein Standard–Prozeßmodell ist.

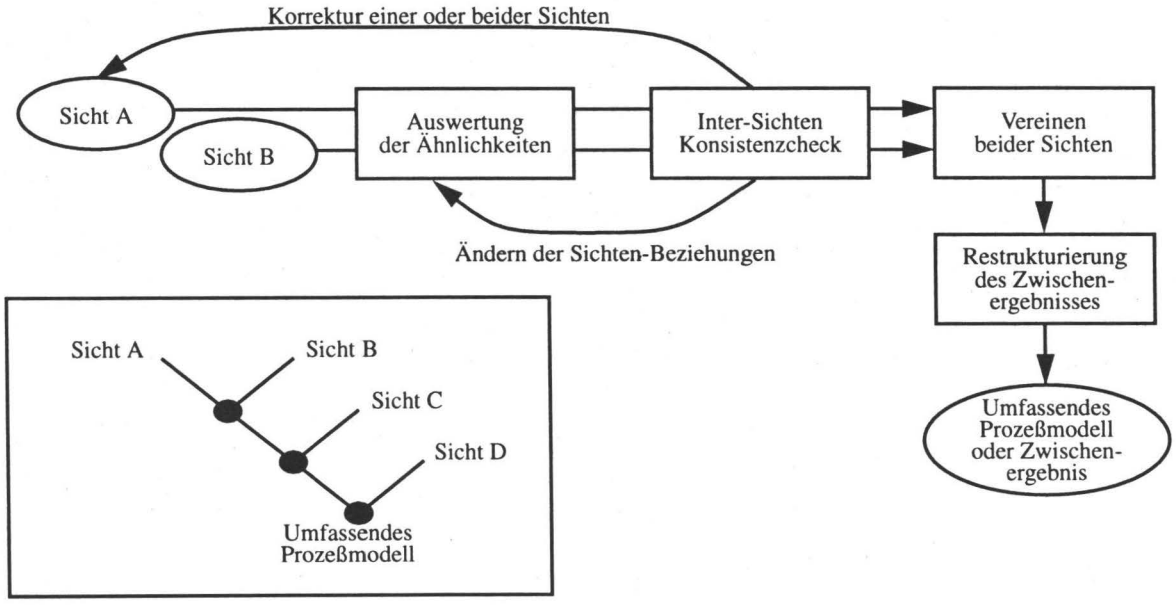

Abb. 4: Die Schritte bei der Sichten–Integration

Es ist wichtig, daß bei der Integration mehrerer Sichten auf Ausschnitte der Sichten Bezug genommen werden muß. Zwei Sichten beschreiben weder immer vollständig dasselbe, noch werden sie immer disjunkt sein. Fast immer stimmen sie teilweise überein (d.h. sie überlappen sich). Die Ausschnitte von Sichten entsprechen Konzepten von MVP–L. Die Modelle (und demzufolge auch die Objekte) sind also strukturiert. Jedes Objekt besitzt Strukturelemente, die einen Typ besitzen.

So kann zum Beispiel ein Prozeß das Strukturelement *anforderungen.status = 'complete'* vom Typ *Vorbedingung* besitzen. Ausschnitte von Prozeßmodellen können entweder aspektbezogen (z.B. wird der Aspekt *Kontrollfluß* durch *Attribute* und *Vor– und Nachbedingung* realisiert) oder Subobjekte sein. Es kann auch eine Kombination existieren (z.B. Produktfluß zwischen Subprozessen). Man beachte, daß ein Ausschnitt nicht unbedingt eine Menge syntaktisch korrekter Produkt-, Prozeß- oder Ressourcenmodelle in MVP–L sein muß.

In den einzelnen Phasen der Sichten–Integration wurden bisher die folgenden Probleme identifiziert, von denen in diesem Bericht die Ähnlichkeit von Prozeßmodellen behandelt wird:

1. *Ähnlichkeit.* Um Sichten zu integrieren müssen Ähnlichkeiten (z.B. Prozesse, Produkte, Vorbedingungen, Attribute, Produktflußbeziehungen) in unterschiedlichen Sichten identifiziert werden. Synonyme (dasselbe Objekt in zwei Sichten besitzt unterschiedliche Bezeichner) und Homonyme (unterschiedliche Objekte in zwei Sichten besitzen denselben Bezeichner) schließen automatische Ermittlungen der Ähnlichkeit aus. Eine geeignete Definition der Ähnlichkeit kann jedoch Fehler vermeiden und wesentliche Vorarbeit leisten.
2. *Konsistenz.* Zwei Sichten mit ermittelten Ähnlichkeiten können integriert werden, falls sie keine inkonsistenten Informationen beinhalten. Werden Inkonsistenzen hier nicht entdeckt, so kann dies später zu unnötigen Iterationen bei der Integration führen. Sind die Konsistenz–Bedingungen zu restriktiv formuliert, so kann dies zu einer Abweisung eigentlich integrierbarer Sichten führen. Eine geeignete Definition muß gefunden werden.
3. *Wissensbasierte Unterstützung.* Sowohl bei der Bestätigung von Ähnlichkeitsannahmen als auch bei der Entdeckung und Auflösung von Inkonsistenzen ist Expertenwissen erforderlich. Es stellt sich die Frage inwiefern Systeme unterstützen können. Es läßt sich aufgrund unserer bisherigen Erfahrungen sagen, daß dieses Wissen wichtig für die Integration ist, was sich mit Erfahrungen aus dem Datenbankentwurf deckt [22,6]. Ein Integrationsingenieur muß die berechneten Ähnlichkeitswerte bestätigen oder verwerfen. Aus Ähnlichkeitsannahmen werden Ähnlichkeitstatsachen. Diese Informationen können während der Integration revidiert werden (in Abb. 4 Pfeil 'Ändern der Sichten–Beziehungen').

3 Ähnlichkeit von Prozeßmodellen

In diesem Abschnitt wird ein Ähnlichkeitsbegriff für Prozeßmodelle in MVP-L entwickelt. Diese Entwicklung ist jedoch unserer Meinung nach auf beliebige strukturierte Objekte, also z.B. in einer Programmiersprache geschriebene Funktionen oder Inhalte einer Datenbank, entsprechend übertragbar.

Zwischen Objekten können verschiedene Beziehungen gelten, die ausschließlich über ihre Merkmale - eine Menge von Merkmalen hatten wir als Ausschnitt bezeichnet - definiert sind. Die *Identität* verlangt, daß zwei Objekte bezüglich einer festgelegten Menge von Merkmalen vollständig übereinstimmen. Die *Gleichheit* kann man auffassen als die Gleichheit aller Merkmale, das Problem wird verfeinert: die Gleichheit der Objekte ist abhängig von der Gleichheit ihrer Merkmale. Zwischen zwei Merkmalen gelten nur die Beziehungen *gleich* und *ungleich*. Die *Ähnlichkeit* zweier Objekte bezieht sich auf eine Teilmenge von Merkmalen und auf einer Ähnlichkeitsdefinition zwischen Merkmalsausprägungen. Die Ähnlichkeit kann durch eine Funktion repräsentiert werden, die Ähnlichkeiten der Merkmalsausprägungen miteinander verknüpft. Diese Verknüpfung ist üblicherweise eine Linearkombination, versehen mit Gewichtungsfaktoren, welche die Bedeutung der einzelnen Merkmale widerspiegeln. Die Ähnlichkeit ist flexibler ist als Identität und Gleichheit, da sich betrachtete Merkmale, die Ähnlichkeiten der Merkmalsausprägungen und die Ähnlichkeitsfunktion vorgegebenen Problemen anpassen lassen. Anders gesagt, sind Identität und Gleichheit Spezialfälle der Ähnlichkeit. Die Freiheit bei der Definition der Ähnlichkeit macht es aber schwierig, eine geeignete Definition zu finden.

In diesem Abschnitt wollen wir die Definition einer Ähnlichkeit für Prozeßmodelle vorstellen. Die Merkmale sind dabei Informationen im Prozeßmodell (z.B. Produktfluß oder die Attribute). Wir stellen die Eigenschaften der definierten Ähnlichkeit in den Vordergrund und diskutieren diese im Vergleich zu bereits bekannten Ansätzen.

3.1 Eigenschaften der Ähnlichkeit im allgemeinen

Die Ähnlichkeit zweier Objekte kann im allgemeinen durch einen Abstand (Distanzmaß) zwischen ihnen definiert werden. Die Objekte sind sich um so ähnlicher, je näher sie beieinander liegen.

Distanzmaß:
Eine Abbildung $d : M \times M \rightarrow R^+$ über einer Menge M heißt Distanzmaß, falls $\forall\, x, y \in M$ gilt:

(i)	$d(x,y)$	=	$d(y,x)$	(Symmetrie)
(ii)	$d(x,y)$	=	$0 \Leftrightarrow x = y$	(Gleichheit)

Zwei Objekte sind bezüglich der betrachteten Merkmale gleich, falls sie einen Abstand von 0 haben. Damit man die gewohnte Interpretation von Distanzen in 2– oder 3–dimensionalen Räumen anwenden kann, muß ein Distanzmaß zusätzlich zu obigen Bedingungen noch die folgende Eigenschaft besitzen:

Metrik:
Ein Distanzmaß d auf einer Menge M ist eine Metrik, falls $\forall\, x, y, z \in M$ gilt:

(iii)	$d(x,y) + d(y,z) \geq d(x,z)$	(erfüllte Dreiecksungleichung)

Die Distanz zweier Objekte liegt im halboffenen Intervall [0, ∞ [. Aus Gründen des Rechnens mit ermittelten Abständen ist es sinnvoll, für die Ähnlichkeit eine Normierung vorzunehmen. Üblicherweise wählt man eine Abbildung auf das Intervall [0,1], wobei der Wert 1 der Distanz 0 entspricht (d.h. die beiden Objekte sind gleich). Somit erhält man analog zu Distanzmaßen:

Ähnlichkeitsmaß:
Eine Abbildung $s : M \times M \rightarrow [0,1]$ über einer Menge M heißt Ähnlichkeitsmaß, falls $\forall\, x, y \in M$ gilt, daß s symmetrisch ist und aus $s(x,y) = 1$ die Gleichheit von $x = y$ folgert. Die Definition als Metrik erfolgt analog zur obigen Definition für Distanzmaße.

Die Definition des Ähnlichkeitsmaßes läßt den Wert 1 nur für gleiche Objekte zu. Es kann jedoch erwünscht sein, daß dieser Wert auch für den Vergleich zweier Objekte erzielt wird, die nicht gleich sind, sondern sich nur bez. einer Untermenge von Eigenschaften gleichen (z.B. zwei Autos besitzen dieselbe Farbe, sind aber von unterschiedlichen Herstellern). Deshalb muß die Bedingung (ii) bei Ähnlichkeitsmaßen abgeschwächt werden, zu einer Bedingung, die hinreichend für den höchsten Ähnlichkeitswert ist. Man erhält eine schwächere Klasse von Abbildungen:

Ähnlichkeitsfunktion:
Eine Abbildung $sim : M \times M \rightarrow [0,1]$ über einer Menge M heißt Ähnlichkeitsfunktion, falls $\forall\, x, y \in M$ gilt:

(i)	$sim(x,x)$	=	1	(Reflexivität)
(ii)	$s(x,y)$	=	$s(y,x)$	(Symmetrie)

Unter Umständen kann es auch sinnvoll sein, die Forderung nach Symmetrie (Punkt *(ii)* bei Ähnlichkeitsmaßen) einzuschränken. Anschaulich ist dies dann der Fall, wenn der Abstand zwischen zwei Orten unterschiedlich ist, weil etwa je nach Fahrtrichtung Einbahnstraßen unterschiedliche Fahrtstrecken verursachen. Ist die Ähnlichkeitsfunktion asymmetrisch, so verdoppelt sich der Berechnungsaufwand, da für jedes Objektpaar die Funktion zweimal angewendet werden muß.

Zusammenfassend ist die allgemeine Darstellung einer Ähnlichkeitsfunktion *sim* also:

Für jeweils zwei Objekte $x, y \in M$ mit Merkmalen $m_1, m_2, \ldots, m_n$, Merkmalsausprägungen $m_i(x)$ bzw. $m_i(y)$, bekannten Ähnlichkeiten $sim_i\,(\, m_i(x), m_i(y)\,)$ und Gewichtsfaktoren g_i mit $\sum_{i=1}^{n} g_i = 1$ ist die Ähnlichkeit definiert als:

$$sim(x,y) = \sum_{i=1}^{n} g_i \times sim_i(m_i(x), m_i(y))$$

3.2 Anforderungen an die Ähnlichkeitsfunktion

Der Abschnitt 3.1 hat gezeigt, daß verschiedene Möglichkeiten bestehen, den Begriff der Ähnlichkeit in eine mathematische Definition zu fassen. Wir werden deshalb im folgenden auf einzelne Anforderungen an die Ähnlichkeitsfunktion für Sichten auf Software-Prozesse kurz eingehen, um die später folgende Definition der Funktion *vsim* (von engl. View Similarity) plausibel zu machen.

Eine wesentliche Anforderung ist die Gültigkeit der Interpretation der Funktionsergebnisse. Eine Interpretation der Werte auf mehrere Nachkommastellen ist nicht sinnvoll. Die relativen Werte der Ähnlichkeiten sind aussagekräftiger als die absoluten Werte. Man kann jedoch zwei wesentliche Fehler herausstellen:

- α–Fehler: Eine bestehende Ähnlichkeit zwischen Objekten wird nicht erkannt, d.h. das Ergebnis der Ähnlichkeitsfunktion ist von 1 relativ weit entfernt.

- β–Fehler: Objekte werden durch die Funktion als ähnlich klassifiziert, obwohl sie in Wirklichkeit unähnlich sind.

Diese Anforderung ist eine Eigenschaft der Funktion in Bezug zur Realität. Diese Forderung ist nicht unbedingt erfüllt, da sich die Ermittlung der Ähnlichkeit wie weiter unten beschrieben auf Heuristiken begründet. Validierungen und Anpassungen sind notwendig. Anforderungen, die sich nur auf die Funktion selbst beziehen und ihre Definition begründen, sind im folgenden kurz dargestellt:

A1 *vsim* muß partiell anwendbar sein. Die Ausschnitte der Sichten sollen dabei frei wählbar sein entsprechend den Konzepten von MVP-L. Es können unterschiedliche Beziehungen zwischen den Sichten oder Ausschnitten gelten (z.B. Überlappung, Teilmenge).

A2 *vsim* muß die Bedeutung syntaktischer Konstrukte berücksichtigen. MVP–L besitzt verschiedene Konstrukte, die für die Ähnlichkeit unterschiedliche Bedeutung haben. Zum Beispiel sind Prozeßbezeichner für die Ähnlichkeit weniger interessant als bereits entdeckte ähnliche Beziehungen von Subprozessen. Außerdem bestehen gewisse Ähnlichkeiten bereits zwischen Konstrukten. Zum Beispiel sind sich die Zugriffsarten *produce* und *modify* von Prozessen auf Produkte ähnlicher als *produce* und *consume*.

A3 *vsim* muß mit Gewichtungsparametern für einzelne Merkmale anpaßbar sein. Die Bedeutung einzelner Merkmale kann variieren. Zum Beispiel sind die Beziehungen von Prozessen und Produkten gerade in den Fällen besonders wichtig, bei denen die Sichten angrenzende Phasen beschreiben. Hier sind nur ähnliche Produkte enthalten, aber keine ähnlichen Prozesse. Dies muß mittels Gewichtsparametern eingestellt werden können.

A4 *vsim* muß reflexiv sein. Eine Sicht sollte immer am ähnlichsten zu sich selbst sein.

A5 *vsim* muß symmetrisch sein. Die Reihenfolge der Integration darf nicht entscheidend sein. Es muß unerheblich sein, ob Sicht A mit Sicht B verglichen wird oder umgekehrt. Ähnlichkeiten in den Sichten sind zu identifizieren, unabhängig von der Reihenfolge der Betrachtung.

A6 *vsim* darf keine Metrik sein. Ähnlichkeiten in unterschiedlichen Merkmalen können zur Verletzung der Dreiecksungleichung führen.

A7 *vsim* darf kein Ähnlichkeitsmaß sein. Durch die Gewichtung kann eine Unähnlichkeit zwischen Merkmalen zweier Objekte unterdrückt werden (z.B. soll der Name keine Rolle spielen). Der Ähnlichkeitswert ist dann trotzdem 1, obwohl sich die Objekte nicht in allen Merkmalen gleichen.

3.3 Der Ähnlichkeitsbegriff in anderen Domänen

Bei der Definition der Ähnlichkeit sind Erfahrungen aus anderen Domänen übernommen worden. Es handelt sich hier um die Schemaintegration bei Datenbanken, das Analoge Schließen (*Analogy Reasoning*) und das Fallbasierte Schließen (*Case–Based Reasoning*).

Bei der Integration von Datenbank–Schemata werden mehrere Schemata integriert. Ergebnisse verschiedener Entwurfsaufgaben oder bereits existierende Datenbanken werden zusammengeführt. Die Forschungen, insbesondere bezüglich des Integrationsprozesses, finden schon seit mehreren Jahren statt [6]. Die Bedeutung dieser Ergebnisse für unsere Arbeit haben wir schon in Abschnitt 2.2 beschrieben. Jedoch ist die Ähnlichkeit von Schemata in den uns bekannten Arbeiten aus dieser Domäne nicht detailliert ausgeprägt, da die verwendeten Datenmodelle konzept-arm sind (meist Entity–Relationship–Ansatz oder Erweiterungen). Die Identifizierung von gleichen oder ähnlichen Bausteinen findet fast nur aufgrund von Bezeichnern statt. Strukturelle Merkmale werden bei den Betrachtungen im allgemeinen nicht durchgeführt.

Das Analoge Schließen versucht Beweise über Terme wiederzuverwenden, indem durch Substitutionen zu beweisende Terme auf (Teil–) Terme bereits bewiesener Aussagen abgebildet werden. Die Übereinstimmungen in beiden Termen werden als *match* bezeichnet. Der Beweis wird entsprechend des matches modifiziert. Die Ähnlichkeit zwischen Termen wird aus zwei Gründen benötigt. Erst müssen geeignete Terme aus der Menge der bereits bewiesenen Terme ausgesucht werden [10]; dann muß festgestellt werden, welches match die beste Ausgangsbasis für eine Modifikation des Beweises ergibt [9]. Die Techniken des Analogen Schließens versuchen einen perfekten match zwischen Termen herzustellen. Dies ist bei der Integration von Sichten nicht unbedingt erforderlich, da Sichten nur in bestimmten Ausschnitten ähnlich sein können (z.B. sind in den Sichten *Anforderungsingenieur* und *Designer* nur die Produkte ähnlich, aber nicht die Prozesse).

Beim Fallbasierten Schließen werden Lösungen abgearbeiteter Fälle wiederverwendet (z.B. in medizinischen Diagnosesystemen). Man geht davon aus, daß wenn sich die Problemstellungen ähneln, die Lösung des alten Falls für den neuen Fall verwenden läßt. Die Ähnlichkeit von Fällen wird über Merkmalsausprägungen der Problemstellungen hergeleitet. Es ist jedoch zu beachten, daß die Ähnlichkeiten lediglich zwischen Attributierungen der Objekte, die durch ein gemeinsames Schemata definiert sind, berechnet werden. Alle Fälle werden ähnlich abgelegt und behandelt. Hierbei können auch Informationen verwandt werden, die nicht im Objekt selbst enthalten sind. Auch kann dadurch auf der Struktur der Objekte selbst keine Betrachtung der Ähnlichkeit stattfinden.

Die Wiederverwendung von Wissen jeglicher Art (also nicht nur Software–Bausteinen) verlangt eine Identifizierung der geeignetsten Kandidaten aus einer Menge bereits existierenden Wissens [3]. Die Berechnung der *geeigneten* Kandidaten erfolgt ähnlich wie beim Fallbasierten Schließen über Charakterisierungsschemata [11]. Dies verlangt eine Abstraktion der wiederzuverwendenden Objekte [23]. Also auch bei der Wiederverwendung erfolgt die Betrachtung der Ähnlichkeit getrennt vom Objekt, es kann nicht auf die Struktur zugegriffen werden. Dies ist vor allem dann notwendig, wenn in großen Datenmengen effizient gesucht werden muß. Bei der Ähnlichkeit

von Prozeßmodellen ist dies ja nicht der Fall, da immer nur schrittweise die Ähnlichkeit zwischen zwei Prozeßmodellen verglichen wird.

Für unsere Arbeiten haben wir Ideen und Erfahrungen aus allen diesen Gebieten verwendet. Für die Ähnlichkeit zwischen Prozeßmodellen waren jedoch in erster Linie die Arbeiten aus dem Analogen Schließen relevant, die starken Einfluß auf die Definition der Ähnlichkeit zwischen Prozeßmodellen gehabt haben.

3.4 Definition der Ähnlichkeit von Sichten

Bei der Definition der Ähnlichkeit von Sichten haben wir insgesamt sechs Heuristiken aus dem *Analogen Schließen* übernommen (siehe auch Abschnitt 3.3) [9]. Die für Terme definierten Heuristiken mußten von uns auf unser Modell von MVP–L angepaßt werden (eine detaillierte Beschreibung ist in [1] zu finden):

- *PH – Partieller Homomorphismus:* Ähnlichkeitsbetrachtungen zwischen zwei Sichten müssen strukturverträglich sein. Es werden Ausschnitte an denselben 'Stellen' miteinander verglichen. Darauf aufbauend kann die Definition der Ähnlichkeit verfeinert werden, indem auf die Ähnlichkeit zwischen gleichen Strukturelementen zurückgeführt wird.

- *SyT – Syntaktische Typen:* Es dürfen nur syntaktisch gleiche Elemente miteinander verglichen werden (z.B. kein Vergleich zwischen Prozeßmodellen und Produktmodellen). Syntaktisch gleiche Ausschnitte von Sichten sind per Definition ähnlich. Im Fall von MVP–L wird aufgrund der PH–Heuristik eine grobe Zuordnung der Strukturelemente durchgeführt. Nur an den Stellen, an denen in der Syntax Alternativen spezifiziert sind, ist die SyT-Heuristik anwendbar.

- *AT – Argumenttypen:* Die Argumente einer Funktion müssen vom selben Typ sein. Im Fall von MVP–L sind dies die Schnittstellenparameter von Prozeßmodellen, also im wesentlichen Produkte auf die zugegriffen wird, Ressourcen (Werkzeuge oder Projektmitglieder) und Subprozesse bei aggregierten Prozeßmodellen. Da in MVP–L die Typen für Parameter zum Teil durch die Struktur eingeschränkt sind, ist diese Heuristik nur von relativem Nutzen.

- *IS – Identische Symbole:* Es wird überprüft, ob die aufeinander abzubildenden Ausschnitte durch dasselbe Symbol repräsentiert sind. In MVP–L sind dies die Bezeichner der Prozeß–, Produkt– und Ressourcenmodelle, bzw. der davon abgeleiteten Objektbezeichner. Im Gegensatz zum Analogen Schließen wird hier davon ausgegangen, daß *ähnliche* Symbole für *ähnliche* Objekte vergeben werden. Diese Forderung ist schwächer als die Identität, erfaßt also mehr potentielle Beziehungen. Probleme ergeben sich durch Synonyme und Homonyme. Die IS–Heuristik kann auch stärker werden, indem man eine Regel für den Abstand zweier Symbole definiert.

- *ST – Semantische Typen:* Diese Heuristik besagt, daß semantische Beziehungen, die nicht durch die Struktur abgedeckt werden, zusätzlich Eingang in die Ähnlichkeitsfunktion haben müssen. Hierbei sind jedoch nur solche Beziehungen sinnvoll, die aus der Struktur der Sichten abgeleitet werden können. Zum Beispiel ist die Ähnlichkeit zwischen modifizierten und produzierten Produkten größer als zwischen produzierten und konsumierten Produkten.

- *CT – Konsistente Zuordnung (engl. consistent translation):* Es werden nur 1:1-Zuordnungen zwischen Elementen zweier Sichten vorgenommen. Es brauchen nicht alle Elemente einer Sicht anderen Elementen zugeordnet werden. Eine konsistente Zuordnung ist also nicht notwendiger-

weise injektiv, aber eine Funktion. Es ist zu beachten, daß im Gegensatz zum Analogen Schließen die Abbildung durch den Ähnlichkeitswert gewichtet werden. Selbstverständlich steigern konsistente Abbildungen das Vertrauen in die gefundenen Ähnlichkeiten. Aber es sollte zum Beispiel möglich sein, ein Prozeßmodell *Komponententest* in einer Sicht auf Prozeßmodelle *funktionales Testen* und *strukturelles Testen* in einer anderen Sicht abzubilden.

Die Bedeutung der einzelnen Heuristiken für die Ähnlichkeit ist gegenüber dem Analogen Schließen verschoben. Aufgrund der semantisch einfachen Modelle zur Bildung von Termen (i.allg. als Struktur von Funktionen, Symbolen und Prädikaten) bedarf es verschiedener Heuristiken, um eine *gute* Definition der Ähnlichkeitsfunktion zu gewährleisten. Im Falle von MVP–L scheinen sich die Heuristiken zu überlappen. Dies gilt insbesondere für die Heuristiken PH, SyT und AT. Durch die Syntax der Sprache werden bestimmte Eigenschaften der potentiell ähnlichen Ausschnitte bereits sichergestellt. Das Auftreten von Teilinformation an bestimmten Stellen ist durch Typisierung beschränkt. Zum Beispiel werden nur Strukturteile unter *product_flow* miteinander verglichen (PH), wobei sichergestellt ist, daß dort nur Produkte auftauchen (SyT); durch das Auftreten an einer bestimmten Stelle der Prozeßschnittstelle ergibt sich deshalb auch sofort eine partielle Gleichheit der Argumenttypen (AT).

Die sechs adaptierten Heuristiken haben unterschiedliche Aufgaben in *vsim*:

- *Einschränken des Suchraums.* Es werden nur Paare betrachtet, die miteinander in Relation stehen oder zu einer Klasse gehören (z.B. alle Produkte). Die PH– und SyT–Heuristiken bilden den Kern von *vsim* und unterstützen die Gliederung der Prozeßmodelle in einzelne Merkmale (siehe auch Anforderungen A1 und A2). Durch diese beiden Heuristiken wird im wesentlichen angegeben, was man sinnvoll miteinander vergleichen kann.
- *Herstellen von Assoziationen.* Die IS–Heuristik wird zur Herstellung einer initialen Assoziationen aufgrund von Bezeichnern benutzt. Handelt es sich bei den assoziierten Modellen um Aggregate, so werden aufgrund der AT–Heuristik auch die 'feineren' Modelle miteinander assoziiert. Die Heuristiken IS und AT sind relativ einfach und werden in *vsim* benutzt, um relativ schnell Assoziationen zwischen Prozeßmodellen herzustellen. Erfahrungen im Umgang mit diesen Heuristiken haben aber gezeigt, daß sie allein im allgemeinen nicht zu guten Ergebnissen für die Ähnlichkeitsbetrachtug führen [9].
- *Bewerten von Assoziationen.* Eine genauere Analyse der Ähnlichkeiten zwischen Objekten lassen die beiden Heuristiken ST und CT zu. Die entsprechend der ST–Heuristiken formulierten Teile von *vsim* sind berechnungsaufwendiger. Deswegen werden die Regeln aufgrund der ST–Heuristik nur auf solche Paare angewendet, die aufgrund der IS– und AT–Heuristiken bereits ausgewählt wurden. Konsistente Abbildungen erhöhen Ähnlichkeiten.

Insgesamt tragen zum Ähnlichkeitswert Berechnungen aufgrund der Heuristiken IS, AT, CT und ST bei. Anschaulich gilt, daß bei IS und AT die Assoziation gewichtet wird und bei CT und ST zusätzliche Analysen einfließen [1]. Die Realisierung von *vsim* besteht aus einer Anzahl von Funktionen, die entsprechend der syntaktischen Struktur der Prozeßmodelle in MVP–L aufgeteilt sind. Insgesamt besteht *vsim* aus einer Menge von 16 einzelnen Funktionen, die frei miteinander kombiniert werden können.[1] Die Anwendung von *vsim* auf Prozeßmodelle soll anhand der in Abbildung 5 dargestellten zwei Prozeßmodelle verdeutlicht werden. Die MVP-L-Beispiele sind

modifizierten Lösungen zu einem Referenzbeispiel entnommen und werden später in Abschnitt 4 detailliert erläutert. Die nachfolgenden Berechnungen sind bis auf mehrere Nachkommastellen angegeben, um die Nachvollziehbarkeit der Berechnung zu gewährleisten. Es macht natürlich nicht viel Sinn, die Ähnlichkeitswerte bis auf diese Genauigkeit zu interpretieren. Man sollte daher Ähnlichkeitswerte immer als ein Mittel für den Vergleich mehrerer Alternativen ansehen und nicht als detaillierte Charakterisierung eines Objektpaares (siehe auch Abschnitt 4).

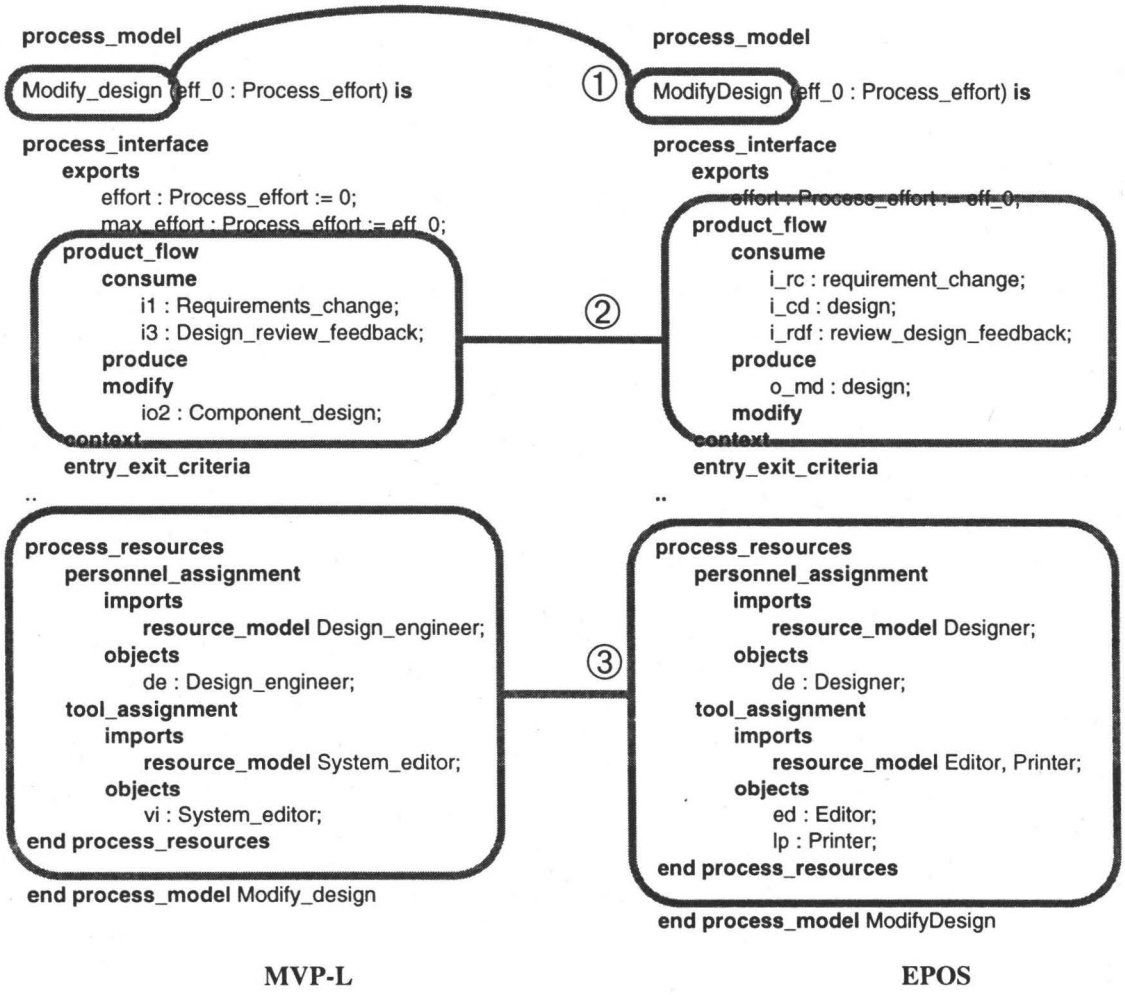

Abb. 5: Ähnlichkeit von Ausschnitten zweier Sichten (Lösungen des ISPW-Beispiels in MVP-L und EPOS)

- Bezeichner ①: Der Vergleich von Bezeichnern beruht auf der Annahme, daß die Namen von Objekten beschreibend sind. Sind die Bezeichner zueinander ähnlich, so kann man daraus schließen, daß auch zwischen den bezeichneten Objekten eine Ähnlichkeit besteht (siehe IS-Heuristik). Die Ähnlichkeit von Bezeichnern wird in *vsim* durch die *weighted Levenshtein distance* [24] ermittelt. Die Levenshtein-Distanz ist ein Maß für die notwendigen Ein- und Ausfügungen bzw. Ersetzungen, um einen Bezeichner in einen anderen zu überführen. Diese drei

1. Dies sind Funktionen zur Berechnung der Ähnlichkeiten von *Produktstruktur*, *Produktfluß*, *Kontrollfluß–Produkten*, *Ressourcenstruktur*, *Ressourcentyp*, *Ressourcen–Prozeß–Beziehungen*, *Ressourcen–Produkt–Beziehungen*, *Prozeßhierarchie*, *Kontrollfluß*, *Produktattribute*, *Ressourcenattribute*, *Prozeßattribute*, *Produktaggregation*, *Bezeichner*, *Aggregationsstufe*.

Operationen können mit unterschiedlichen Gewichtungen (Kosten) versehen werden. Für das Beispiel ergeben sich die Werte aus Tabelle 1.
Es ist ein Buchstabe zu löschen, ein zweiter zu ersetzen. Unter der Annahme, daß alle Operationen gleich viel kosten (in diesem Fall ist die Levenshtein-Distanz symmetrisch), ergibt sich als Ähnlichkeitswert 0.846154 (1 - Kosten / Maximum beider Wortlängen). Wird nicht zwischen Groß- und Kleinschreibung unterschieden, ergibt sich eine Ähnlichkeit von 0.923077.

M	o	d	i	f	y	_	d	e	s	i	g	n
M	o	d	i	f	y		D	e	s	i	g	n
0	0	0	0	0	0	1	1	0	0	0	0	0

Tabelle 1: Levenshtein-Distanz für *Modify_design* und *Modify Design*

- Produktfluß ②: Der Produktfluß zwischen Prozessen ist vergleichbar mit dem Datenfluß zwischen Funktionen. Die Annahme ist, daß Prozesse mit gleichartigem Zugriff auf gleiche oder ähnliche Daten auch ähnlich sind. Die Ähnlichkeit der Produkttypen geht in die Bewertung der Produktflußähnlichkeit mit ein. Ihre Bedeutung wird mittels Gewichtungsparametern definiert. Die Berechnung erfolgt später dann analog für die Gesamtähnlichkeit der Prozesse.
  Die Aussagen bez. der Ähnlichkeit von Produkten können unverändert oder von *vsim* interpretiert für die Berechnung der Produktflußähnlichkeiten herangezogen werden. Da sehr ähnliche Produkte darauf schließen lassen, daß sie tatsächlich übereinstimmen, kann durch einen Schwellenwert eine Ähnlichkeitstatsache automatisch generiert werden. Zum Beispiel liefert die Berechnung der Produkte *Requirements_change* und *requirement_change* einen Wert von 0.914947. Dieser für eine von *vsim* ermittelte Ähnlichkeitsannahme sehr gute Wert wird deshalb für weitere Berechnungen zu 1.0 (Ähnlichkeitstatsache) verstärkt. In *vsim* ist ein Grenzwert von 0.75 voreingestellt, ab dem Ähnlichkeitstatsachen automatisch ermittelt werden. Dieser Wert kann jedoch verändert werden.
  Entsprechend der PH-Heuristik können alle drei Spezifikationsteile des Produktflusses (*consume*, *produce* und *modify*) miteinander verglichen werden. Die nachfolgende Berechnung geht von einer Gleichgewichtung aller Teile aus:

MVP-L	EPOS	Ähnlichkeit	Gewichtung (1. Stufe)	Gewichtung (2. Stufe)	Produktfluß-ähnlichkeit
consume	consume	0.666667	0.33	0.33	0.47
produce	produce	0	0.33		
modify	modify	0	0.34		
consume	modify	0	0.25	0.33	
produce	modify	1	0.25		
modify	consume	0	0.25		
modify	produce	0	0.25		
consume	produce	0	0.5	0.34	
produce	consume	0	0.5		

Tabelle 2: Produktflußähnlichkeit zweier Prozeßmodelle

Die Gewichtung sollte speziellen Bedürfnissen angepaßt werden. Will man z.B. auf Gleichheit der Prozeßmodelle überprüfen, so sollte man in der zweiten Gewichtungsstufe eine Verteilung von (1 : 0 : 0) wählen. Möchte man dagegen Produktflußbeziehungen untersuchen (Welche Prozesse bearbeiten die von einem Prozeß produzierten Produkte?), so empfiehlt sich eine Gewichtung von ((0:0:0), (0:1:0:0), (0:1)) auf der ersten Stufe und (0 : 0.5 : 0.5) auf der zweiten Stufe. Im Tabelle 2 profitiert die Berechnung der (*consume*, *consume*)-Ähnlichkeit von den automatisch ermittelten Ähnlichkeitstatsachen für (*Requirements_change*, *requirement_change*) und (*Design_review_feedback*, *review_design_feedback*). Bestätigt man per Hand noch zusätzlich die Beziehungen zwischen dem Objekt io2 im MVP-L-Modell und den Objekten i_cd und o_md als Ähnlichkeitstatsache bestätigt, so steigt die Produktflußähnlichkeit auf 0.803333.
Die Berechnung der (*produce*, *modify*)-Ähnlichkeit verdeutlicht, daß hier eine optimistische Strategie verfolgt wird. Man kann auch den Standpunkt vertreten, daß leere Spezifikationsteile nur geringe Aussagen über die Ähnlichkeit zulassen und deshalb der Ähnlichkeitswert an dieser Stelle schwächer sein sollte.

- Ressourcen–Prozeß–Beziehungen ③: Werden zu zwei Prozessen ähnliche Rollen zugeordnet, so bestärkt dies die Annahme, daß die Prozesse im ganzen ähnlich sind. Wie bereits bei der Ähnlichkeitsberechnung für den Produkt- und Kontrollfluß wird auch hier auf die Ähnlichkeit der Ressourcen zurückgegriffen.
  Die Berechnung der Ressourcenähnlichkeit erfolgt vergleichbar zur Berechnung der Produktflußähnlichkeit. Es werden jedoch weniger Kreuzvergleiche durchgeführt, da es keinen Sinn macht, Werkzeuge und Menschen zu vergleichen. Nach automatischer Generierung von Ähnlichkeitstatsachen für die Paare *Design_engineer* und *Designer* (Wert: 0.813333) bzw. *System_editor* und *Editor* (Wert: 0.784615) ergibt sich ein Ähnlichkeitswert von 0.666667 für die Ressourcen–Prozeß–Beziehung.

An dieser Stelle sei noch einmal ausdrücklich darauf hingewiesen, daß Benutzerinteraktionen bei der Ermittlung von Ähnlichkeiten notwendig sind. Die automatische Berechnung einer initialen Menge von Ähnlichkeiten durch Anwendung von *vsim* sind Vorschläge, die vom Benutzer zu bestätigen oder zu verwerfen sind, sie können verstärkt oder abgeschwächt werden. Aus Ähnlichkeitsannahmen werden so Ähnlichkeitstatsachen, die jedoch später wieder revidiert werden können.

Die Anwendung von *vsim* auf das Beispiel in Abbildung 5 illustriert nur den Fall, daß durch die PH-Heuristik bedingt nur strukturell gleiche Ausschnitte in den Prozeßmodellen zur Ähnlichkeitsbetrachtung herangezogen werden können. *vsim* läßt jedoch weitere Betrachtungen zu, die PH-Heuristik wurde also so realisiert, daß auch 'verwandte' Strukturen miteinander verglichen werden können. So macht es durchaus Sinn machen, die Exit-Criteria eines Prozesses mit den Entry-Criteria aller übrigen Prozesse zu vergleichen, um die Kontrollflußbeziehungen zwischen den Prozessen zu analysieren.

4 Validierung anhand des ISPW–Beispiels

Das *ISPW–Beispiel* wurde für den sechsten *International Software Process Workshop* als ein Referenzprozeß entwickelt. Im wesentlichen ist ein Änderungsprozeß beschrieben, der eine neue Anforderung in eine bestehende Komponente einarbeiten soll und diese anschließend validiert. Für den sechsten Workshop wurden insgesamt 18 verschiedene Beschreibungen eingereicht. Im Prinzip entspricht dies 18 unterschiedlichen Sichten auf ein und denselben Prozeß aus unterschiedlichen Perspektiven, da ja jede Lösung in einer eigenen Sprache, die eine bestimmte Menge von Aspekten beinhaltet, beschrieben wurde.

Da die einzelnen Lösungen in jeweils einer eigenen Sprache beschrieben sind, *vsim* aber nur für eine Sprache definiert wurde, mußte eine Transformation nach MVP–L durchgeführt werden. Es wurden drei Lösungen ausgewählt, die sich am einfachsten in eine MVP–L–Repräsentation überführen ließen, da in den wesentlichen Konzepte kaum größere Unterschiede zwischen den Sprachen existieren, bzw. sich andere Konzepte leicht abbilden lassen[2]. Obwohl die Transformation der Beispiele nicht formal durchgeführt wurde, erachten wir die Prozeßmodelle als ausreichend für den Zweck der Validation. Bei den transformierten Beispielen ist ein breiter Bereich von Zielsetzungen für die Prozeßmodellierung abgedeckt:

- EPOS [25] ist eine Plattform zur automatisierten Abwicklung von Prozeßmodellen. Aspekte der Verwaltung von Projektdaten und Integration von Werkzeugen stehen dabei im Vordergrund.

- Statemate [26] ist ein Werkzeug zur Entwicklung von eingebetteten Systemen, wurde jedoch am *Software Engineering Institute* der Carnegie Mellon University zur Erstellung von Prozeßmodellen benutzt. Dabei ging es um die Untersuchung der Beschreibung von Prozessen aus unterschiedlichen Perspektiven und um die Simulation von Prozessen.

- WSH ist eine strukturierte, informelle Lösung des ISPW–Beispiels durch Watts S. Humphrey. Die wichtigsten Relationen zwischen Prozessen und Produkten sind enthalten, aber insgesamt ist diese Lösung vergleichbar mit informellen Ansätzen zur Anforderungsbeschreibung. Nichtsdestotrotz dokumentiert auch diese Lösung eine Sicht auf den Beispielprozeß.

Zusätzlich wurde noch eine Version der Lösung in MVP–L erstellt (MVP–L^{syn}), die synonym zur MVP–L–Lösung ist (d.h. nur Modell– und Objektbezeichner wurden geändert). Dies sollte uns ermöglichen, die Ähnlichkeit aufgrund von Struktur und Typen zu ermitteln, statt über die Ähnlichkeit der Bezeichner. Die Originallösungen befinden sich in einem Zusatz zu [27]. Bei der Übersetzung der Beispiele nach MVP-L wurde nicht auf vollständig semantische Umsetzung der in den Modellen beschriebenen Informationen geachtet. Dies dürfte auch ohne formale Beschreibung von Transformationen schlecht möglich sein. Die Beispiele sollten vier unterschiedliche Sichten liefern, anhand derer die Funktion *vsim* validiert werden kann; eine eventuelle Verifikation bedarf sicherlich genauerer Verfahren.

2. So kann z.B. ein durch die Nutzung prozeduraler Konstrukte spezifizierter Kontrollfluß auf Produktattribute und die Entry-Bedingungen der Prozeßmodelle abgebildet werden.

Beispielhaft seien hier Prozeßhierarchien der Lösungen in MVP–L und EPOS angegeben. Die textuelle Darstellung eines Prozesses aus beiden Lösungen ist teilweise in Abbildung 5 gegeben. Die Hierarchien sollen eine grobe Vorstellung von den Gemeinsamkeiten und Unterschieden in den beiden Lösungen geben. Eine erste Berechnung der Ähnlichkeiten zwischen den einzelnen Sichten bezüglich des obersten Prozesses *technical steps* der Lösungen ergeben die Werte von Tabelle 3 in der oberen rechten Dreiecksmatrix.[3] Die fett gedruckten Werte in der unteren linken Dreiecksmatrix geben die Ähnlichkeiten wieder, die nach der Angabe der Ähnlichkeitstatsachen berechnet wurden. Hier zeigen sich leichte (z.B. zwischen MVP-L_{WSH} und MVP-L_{EPOS}) und große Unterschiede (z.B. zwischen MVP–L^{syn} und MVP_L_{EPOS}). Erstere deuten darauf hin, daß Annahmen von *vsim* bestätigt wurden; zweitere lassen ein Verwerfen anfänglicher Annahmen erkennen. Es sei noch einmal darauf hingewiesen, daß *vsim* zwar einen Wert mit mehreren Nachkommastellen ermittelt, dies aber keineswegs eine präzise Definition der Ähnlichkeit ist. Die Werte müssen in ihrem Kontext interpretiert werden und lassen eher relative Aussagen als absolute Aussagen zu.

Prozeß: *technical steps*	MVP–L	MVP-$L_{Statemate}$	MVP-L_{EPOS}	MVP-L_{WSH}	MVP–L^{syn}
MVP–L	1.0	0.33	0.36	0.26	0.46
MVP-$L_{Statemate}$	**0.30**	1.0	0.53	0.47	0.30
MVP-L_{EPOS}	**0.56**	**0.50**	1.0	0.72	0.15
MVP-L_{WSH}	**0.18**	**0.42**	**0.77**	1.0	0.06
MVP–L^{syn}	**1.0**	**0.30**	**0.56**	**0.18**	1.0

Tabelle 3: Ähnlichkeitswerte für alle Sichten-Paare

Diese anfänglich ermittelten und später durch die Angabe von Ähnlichkeitstatsachen bestätigten Werte lassen auf dieser hohen Abstraktionsebene nicht erkennen, worin die Ähnlichkeit – oder Unähnlichkeit – begründet ist. Die Prozeßhierarchien von MVP–L und MVP-L_{EPOS} (Abbildung 6) würden z.B. eine größere Ähnlichkeit als nur 0.36 bzw. 0.56 begründen. Da aber die Ähnlichkeit der Prozeßhierarchien nur einen Teil der Gesamtähnlichkeit ausmacht, muß es zwischen MVP-L_{EPOS} und MVP–L Unterschiede in anderen Bereichen der Prozeßmodelle geben. Tatsächlich unterscheiden sich die beiden Lösungen erheblich in der Modellierung der Produkte, was die berechneten (Un–) Ähnlichkeitswert rechtfertigt (siehe auch Auszüge ② in Abb. 5). Blendet man dagegen durch geeignete Gewichtungsparameter die Bedeutung von Ressourcen und Produkten aus, so wird für die *technical steps* ein Ähnlichkeitswert von 0.75 berechnet, ohne irgendeine Annahme zu bestätigen. Dies ist für eine initiale Berechnung ohne Benutzerinteraktion ein sehr hoher Wert, was dem intuitiven Empfinden beim Betrachten der Prozeßhierarchien von Abbildung 6 entspricht. Dies macht jedoch auch deutlich, worin die Stärken einer Ähnlichkeitsbetrachtung basierend auf strukturellen Merkmalen liegen kann: Die Unterschiede zwischen den betrachteten Prozeßmodellen lassen sich so relativ einfach ermitteln, indem man beim Vergleich auf feinere Ergebnisse der Ähnlichkeitsberechnungen zurückgreift. Dies steuert gewissermaßen den Verglei-

3. Da *vsim* symmetrisch ist (siehe auch Anforderung A5) reicht die Angabe einer Dreiecksmatrix aus.

chenden bei seinen Untersuchungen und erlaubt ihm, sich auf bestimmte Merkmale der Prozeßmodelle zu konzentrieren.

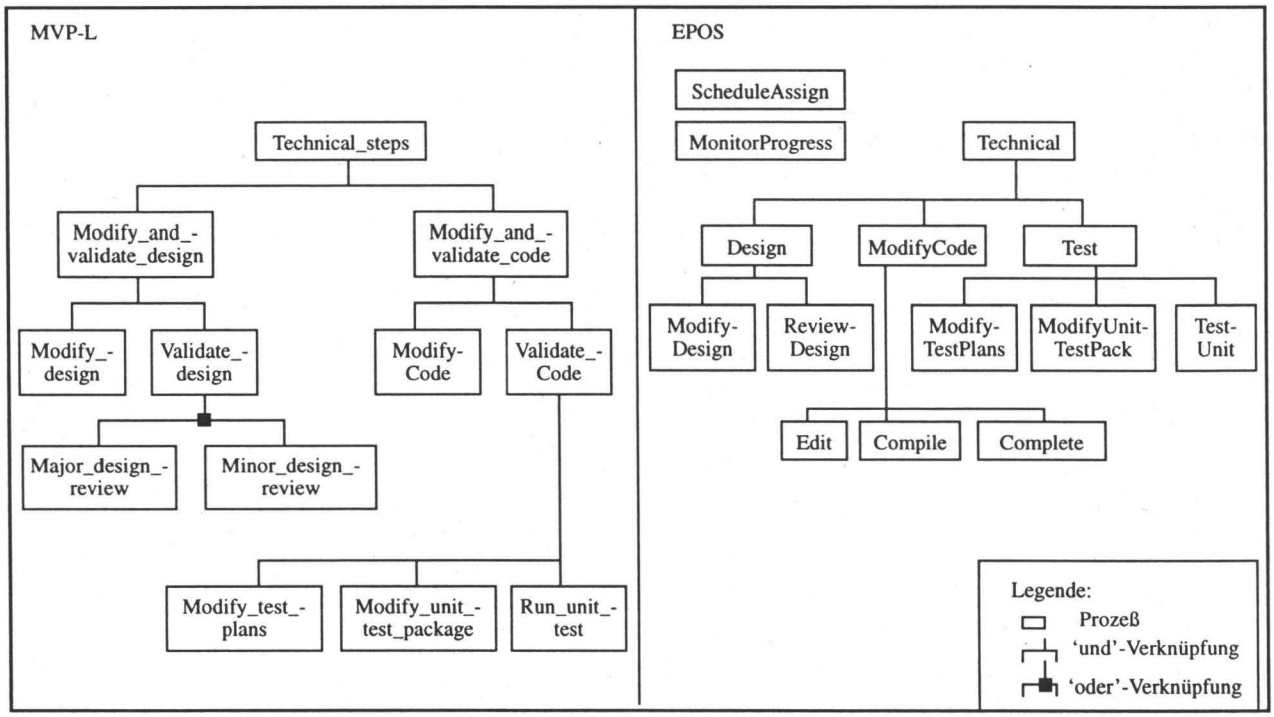

Abb. 6: Prozeßhierarchien zweier Lösungen zum ISPW–Beispiel

Auf der Diagonalen existieren wegen der Reflexivität von *vsim* nur 1.0–Werte. Es ergeben sich unterschiedliche Werte für die Ähnlichkeiten zwischen den Sichten. Der geringste Wert ergibt sich mit 0.06 für die Ähnlichkeit zwischen MVP–L und MVP-L_{WSH}. Der höchste Wert ist für MVP-L_{EPOS} und MVP-L_{WSH} berechnet worden (0.72). Mit dem Wert für MVP–L und MVP-L_{EPOS} läßt sich damit illustrieren, daß *vsim* keine Metrik sein kann (die Dreiecksungleichung ist verletzt, da nicht gilt $0.06 + 0.36 \geq 0.72$).

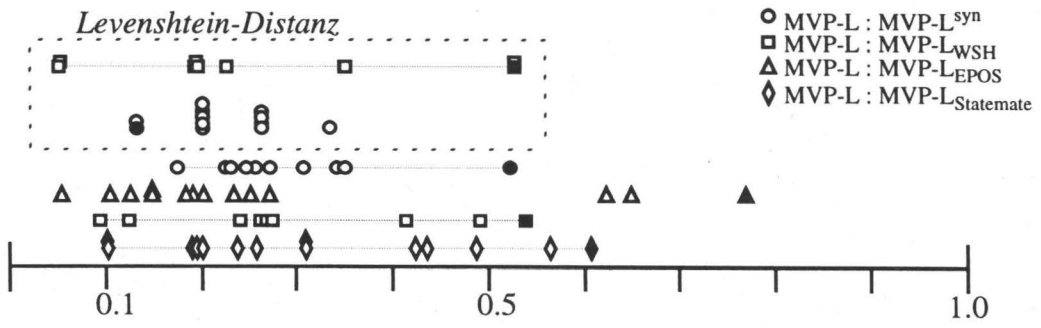

Abb. 7: Ähnlichkeitswerte von *Validate_design* im Vergleich

Die in Abbildung 7 dargestellten Werte zeigen die Ähnlichkeitswerte für ein Prozeßmodell in der originalen MVP–L–Beschreibung (*Validate_design*). Die Berechnungen wurden automatisch, d.h. ohne Bestätigung von Ähnlichkeitsannahmen durchgeführt. Mit Bestätigungen werden die Aussagen im Laufe der Sitzung genauer im Sinne einer Bestimmung der tatsächlichen Ähnlichkeit.

Die schwarz markierten Symbole in Abbildung 7 deuten an, welchen berechneten Ähnlichkeitswert die beste Ähnlichkeitsbeziehung zwischen *Validate_design* und einem Prozeßmodell in den vier anderen Sichten hatte. Es zeigt sich, daß die Berechnung von *vsim* in allen Fällen die tatsächliche Ähnlichkeit auch mit dem höchsten Wert versah. In der gesamten Ähnlichkeitsanalyse der ISPW–Beispiele sind die Unterschiede zwischen den Ähnlichkeitswerten die von *vsim* und der Levenshtein–Distanz berechnet werden nicht groß, wenn man sie relativ betrachtet.[4] Dies ist nicht verwunderlich, wenn man beachtet, daß die Autoren in den Lösungen die in der Problembeschreibung verwendeten Bezeichner ähnlich geschrieben haben. Im obigen Beispiel sind dies: *Validate_design*, *ReviewDesign*, *MajDesRev*, *REVIEW_DESIGN* und *Review_Design*. Wie jedoch eine Unähnlichkeit sich in einem α–Fehler auswirken kann zeigt die Ähnlichkeitsberechnung mittels der Levenshtein–Distanz im Falle des Vergleiches zwischen der MVP–L–Sicht und dem MVP–L^{syn}–Sicht. Hier befindet sich das ähnlichste Prozeßmodell am Ende, es wird als am unähnlichsten eingestuft. Trotz der Übereinstimmung der Bezeichner wird für das Beispiel in Tabelle 3 kein Wert größer als 0.533 erreicht (die Ähnlichkeit der Bezeichner von MVP–L zu EPOS uns Statemate ist nicht in der Abbildung gezeigt, sie verhält sich jedoch vergleichbar zu MVP–L und WSH). Hierbei zeigt sich eine Stärke von *vsim*, da strukturelle Merkmale eine wesentliche Bedeutung haben. So erlaubt der Wert 0.773 bei MVP–L zu EPOS eine gesichertere Aussage, als nur über die Bezeichner. Insbesondere für komplexere Prozeßmodelle ist dies wichtig, da hier die Bedeutung der Ähnlichkeit der Bezeichner immer weiter relativiert wird.

Die Anwendung von *vsim* auf die Lösungen zum ISPW–Beispiel haben gezeigt, daß die Ähnlichkeitsfunktion die Anforderungen erfüllt. Damit besitzen wir ein Werkzeug zur Bestimmung der Ähnlichkeit von Prozeßmodellen. Dies unterstützt eine detaillierte Analyse von Prozeßmodellen und erlaubt schnellere Vergleiche unterschiedlicher Sichten von Projektmitgliedern.

Die Lösungen zum ISPW-Beispiel sind jedoch stark überlappend. Inwiefern dies eine günstige Einschränkung und damit eine Beschränkung der Nützlichkeit von *vsim* darstellt, bleibt zu untersuchen. Interessant wären zum Beispiel angrenzende Sichten, die wenig gemeinsame Informationen besitzen. In solchen Fällen kann sich *vsim* bei der Berechnung der Ähnlichkeiten auf nur wenige Ähnlichkeitstatsachen stützen. Nichtsdestotrotz wird bei geeigneter Auswahl der Gewichtungsparameter die berechnete Ähnlichkeit besser sein, als eine nur auf Bezeichnern beruhende Ermittlung der Ähnlichkeit.

---

4. Es wurde bei der Berechnung der Levenshtein–Distanz nicht zwischen Groß– und Kleinschreibung unterschieden, da STATEMATE für Bezeichner nur Großbuchstaben zuläßt. Diese Entscheidung hatte aber in allen Fällen nur geringen positiven Einfluß auf die Distanz–Werte.

5 Bisherige Erfahrungen

Die bisherigen Anwendungen haben die Eignung von *vsim* für die Definition der Ähnlichkeit von Prozeßmodellen gezeigt. Eine vollständige Unterstützung des Sichten–Integrationsprozesses muß noch erfolgen. Dabei lassen sich jedoch Erkenntnisse aus der Benutzung von *vsim* einarbeiten. Die bisher wichtigsten Erfahrungen mit *vsim* sind:

- Die berechnete Ähnlichkeiten zwischen Sichten entsprechen der Intuition. Diese Übereinstimmung ist letztendlich der entscheidende Faktor bei der Brauchbarkeit dieses Ansatzes. Bei unseren bisherigen Untersuchungen wurden die zu entdeckenden Ähnlichkeitsbeziehungen von *vsim* berechnet.
- Benutzerinteraktion (Eingabe von Ähnlichkeitstatsachen) ist notwendig, da zusätzliches Wissen über die Prozeßmodelle benötigt wird. Besonders deutlich wird dies im Falle von Synonymen und Homonymen.
- Es hat sich als sinnvoll herausgestellt, zuerst die Ähnlichkeiten zwischen Produkten bzw. Ressourcen zu bestätigen, um aussagekräftigere Ähnlichkeitswerte zwischen den Prozessen zu bekommen. Dies ist dadurch erklärbar, daß Produkte und Ressourcen weniger Beziehungen zu anderen Ausschnitten derselben Sicht haben, eine Bestätigung also weniger Unsicherheiten enthält als bei Prozessen. Diese Strategie scheint auch sinnvoll für die nächsten Schritte der Sichten–Integration zu sein.
- Es existiert keine Eindeutigkeit berechneter Ähnlichkeiten. Jede erneute Berechnung der Ähnlichkeitswerte nach einer Bestätigung durch den Benutzer produziert mehrere 'gute' Paare, bei denen ein Prozeß mehrmals vertreten sein kann. Dies macht es notwendig, nicht nur das Paar mit dem höchsten Wert zu betrachten, sondern eine Menge muß genauer untersucht werden. Die Berechnungen werden jedoch mit steigender Anzahl von Bestätigungen eindeutiger.
- Die Auswahl von Sichten bei der Integration kann durch die Ähnlichkeitsbetrachtungen unterstützt werden. Die Integration ist um so leichter, je ähnlicher sich zwei Sichten sind. *vsim* unterstützt die Auswahl der Sichten, indem paarweise die Ähnlichkeit zwischen den einzelnen Sichten berechnet wird, und das Paar mit dem höchsten Ähnlichkeitswert ausgewählt wird.
- Das Verstehen von Beziehungen zwischen Sichten wird erleichtert. Die bisherigen Untersuchungen mit *vsim* haben gezeigt, daß ein einzelner aggregierter Wert aus Ähnlichkeitsbetrachtungen von Merkmalen nur für oberflächliche Analysen geeignet ist. Antworten zu der Fragestellung warum sich zwei Sichten ähneln oder nicht kann nur durch eine detaillierte Betrachtung der Einzelähnlichkeiten abgeleitet werden. Außerdem wird durch die Strukturierung der Ähnlichkeitsfunktion die Aufmerksamkeit auf verschiedene Aspekte der Prozeßmodelle gelenkt.
- Nachfolgende Intergrationsaktivitäten werden unterstützt. Der entsprechend der syntaktischen Struktur der Prozeßmodelle abgeleitete Aufbau von *vsim* erlaubt es, die Konsistenzüberprüfung und das Vereinen von Sichten zu unterstützen. Beziehungen zwischen Ausschnitten von Sichten können unabhängig voneinander betrachtet werden.
- Einstellung der Gewichtungsparameter. Die Werte für die einzelnen Gewichtungsparameter müssen abhängig von der Aufgabe gewählt werden. Identifikationen gleicher Prozesse in verschiedenen Sichten (z.B. Integration von zwei Sichten auf denselben Prozeß) führt natürlich zu

einer anderen Gewichtung, als etwa die Analyse nach gleichen Produkten (z.B. bei Integration zweier angrenzender Sichten).

6 Zusammenfassung

Die Ähnlichkeit verschiedener Objekten wird in verschiedenen Gebieten der Informatik untersucht. Der Vergleich von Kandidaten der Wiederverwendung, die Verbindung einzelner System–Spezifikationen, das Fallbasierte Schließen, das Analoge Schließen oder die Integration von Datenbank–Schemata sind von der Entdeckung ähnlicher Informationsteile in verschiedenen Objekten abhängig. Im Kontext des MVP–Projekts wird im speziellen eine Ähnlichkeit zwischen Prozeßmodellen benötigt.

In dieser Arbeit haben wir die Definition einer Ähnlichkeitsfunktion für Prozeßmodelle (*vsim*) vorgestellt. Dabei haben wir ausgehend von einer speziellen Problemstellung demonstriert, wie eine Ähnlichkeitsfunktion auf der Basis struktureller Merkmale definiert werden kann. Es wurden Erfahrungen aus verschiedenen Bereichen der Informatik miteinander verknüpft. Eigenschaften der Ähnlichkeitsfunktion wurden anhand eines Referenzbeispiels der Prozeßmodellierung illustriert. Die von uns definierte Ähnlichkeitsfunktion *vsim* erlaubt eine vergleichende Betrachtung von Prozeßmodellen. Damit ist man in der Lage, Sichten einzelner Mitglieder eines Software–Projekts, vom Entwickler bis hin zum Projektmanager oder Kunden miteinander zu verknüpfen. Dadurch ist der erste Schritt in Richtung einer umfassenden, in sich konsistenten Projektrepräsentation getan. Diese Technik läßt sich auch verwenden bei der Wiederverwendung bzw. Auswahl von Prozeßmodellen und beim Vergleich mit definierten Standards der Softwareentwicklung. Aber auch jetzt schon läßt sich *vsim* verwenden, um Sichten von Projektmitgliedern zu vergleichen. Dies liefert Rückschlüsse auf das unterschiedliche Verständnis von Projektmitgliedern basierend auf ihren Perspektiven. *vsim* wurde validiert anhand einiger Beispiele eines Referenzmodells zur Evaluation von Prozeßmodellierungssprachen. Die Validation hat gezeigt, daß die in *vsim* realisierten Definitionen ein geeignetes Werkzeug für die Ermittlung der Ähnlichkeit von Prozeßmodellen sind.

Es handelt sich bei *vsim* um ein Beispiel für die Entwicklung und Implementierung eines Ähnlichkeitsbegriffs für strukturierte Objekte einer gemeinsamen Menge (definiert mithilfe einer Sprache, hier MVP-L). Der Vorgang der Definition der Ähnlichkeit ist unserer Meinung nach auch auf andere Problemstellungen übertragbar. Dies gilt nicht nur für andere Prozeßmodellierungssprachen (für eine Übersicht siehe z.B. [18]), sondern auch für beliebige andere Domänen. Voraussetzung ist allerdings eine Struktur innerhalb der verglichenen Objekte. Damit erhält man dann ein mächtigereres Instrument zum Vergleich von Objekten, als die bisher verwendeten Ansätze (siehe Abschnitt 3.3). Inwieweit diese Klasse von Ähnlichkeitsfunktionen allerdings tatsächlich von Bedeutung ist, müssen noch zukünftige Studien in anderen Domänen zeigen. Die bisher gemachten Erfahrungen mit *vsim* sind jedoch auch auf andere Bereiche als die Prozeßmodellierung übertragbar.

7 Literatur

[1] H. Hientz. Ähnlichkeiten von Prozeßmodellen in MVP–L. Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern, 67653 Kaiserslautern, 1994.

[2] H. D. Rombach, A. Bröckers, C. M. Lott, und M. Verlage. Entwicklungsumgebungen zur Unterstützung qualitätsorientierter Projektpläne. *Softwaretechnik-Trends: Mitteilungen der GI-Fachgruppen 'Software-Engineering' und 'Requirements-Engineering'*, 13(3):1–8, August 1993.

[3] R. Prieto-Diaz. Status report: Software reusability. *IEEE Software*, 10(3):61–66, May 1993.

[4] International Organization for Standardization. *ISO 9000: Quality management and quality assurance standards; Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*. Geneva, Switzerland, 1991.

[5] K. Inoue, A. Watanabe, H. Iida, und K. Torii. Modeling method for management process and its application to CMM and ISO 9000–3. In D. E. Perry, editor, *Proceedings of the Third International Conference on the Software Process*, pages 85–98. IEEE Computer Society Press, October 1994.

[6] C. Batini, M. Lenzerini, und S. B. Navathe. A comparative analysis of methodology for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[7] J. S. P. Leite und P. A. Freeman. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, 17(12), December 1991.

[8] B. P. Allen. Case-based reasoning: Business applications. *Communications of the ACM*, 37(3):40–42, March 1994.

[9] S. Owen. *Analogy for Automated Reasoning*. Academic Press, 1990.

[10] M. M. Veloso. *Planning and Learning by Analogy Reasoning*. Springer–Verlag, 1994.

[11] E. J. Ostertag. *A Classification System for Software Reuse*. Dissertation, University of Maryland, 1992.

[12] T. J. Biggerstaff und A. J. Perlis. *Software Reusability – Applications and Experience*, volume II. ACM Press, 1989.

[13] W. S. Humphrey. *Managing the Software Process*. Addison Wesley, Reading, Massachusetts, 1989.

[14] B. Curtis, M. I. Kellner, und J. Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[15] A. Bröckers, C. M. Lott, H. D. Rombach, and M. Verlage. MVP–L language report version 2. Technical Report 265/95, Fachbereich Informatik, Universität Kaiserslautern, D–67653 Kaiserslautern, 1995.

[16] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, und W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[17] C. D. Klingler, M. Neviaser, A. Marmor-Squires, C. M. Lott, und H. D. Rombach. A case study in process representation using MVP–L. In *Proceedings of the Seventh Annual Conference on Computer Assurance (COMPASS 92)*, pages 137–146, June 1992.

[18] H. D. Rombach und M. Verlage. Directions in software process research. In M. V. Zelkowitz, editor, *Advances in Computers, vol. 41*, pages 1–63. Academic Press, 1995.

[19] A. Bröckers, C. Differding, B. Hoisl, F. Kollnischko, C. M. Lott, J. Münch, M. Verlage, und S. Vorwieger. A graphical representation schema for the software process modeling language MVP–L. Technical Report 270/95, Department of Computer Science, Universität Kaiserslautern, 67653 Kaiserslautern, Germany, June 1995.

[20] M. Verlage. Multi–view modeling of software processes. In B. C. Warboys, editor, *Proceedings of the Third European Workshop on Software Process Technology*, pages 123–127, Grenoble, France, 1994. Nr. 772, Springer–Verlag.

[21] International Organization for Standardization. *ISO 9000: 1987 – Quality management and quality assurance standards – Guidelines for selection and use*. Geneva, Switzerland, 1987.

[22] W. Gotthard, P. C. Lockemann, und A. Neufeld. System-guided view integration for object-oriented databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(1):1–22, February 1992.

[23] G. Canfora, A. Cimitile, M. Tortorella, und M. Munro. A precise method for identifying reuseable abstract data types in code. In H. A. Müller and M. Georges, editors, *Proceedings of the International Conference on Software Maintenance*, pages 404–413. IEEE Computer Society Press, September 1994.

[24] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, 163(4):845–848, 1965.

[25] M. L. Jaccheri und R. Conradi. Techniques for process model evolution in EPOS. *IEEE Transactions on Software Engineering*, 19(12):1145–1156, December 1993.

[26] W. S. Humphrey und M. Kellner. Software process modeling: Principles of entity process models. In *Proceedings of the Eleventh International Conference on Software Engineering*, pages 331–342, May 1989.