Vorgehensmodelle für die Entwicklung wissensbasierter Systeme

Jürgen Angele, Dieter Fensel und Rudi Studer

Zusammenfassung

Im Folgenden werden unterschiedliche Vorgehensweisen zur Entwicklung wissensbasierter Systeme (WBS) aufgezeigt. In der Praxis noch weit verbreitet ist der Ansatz des Rapid Prototyping. Beim Rapid Prototyping wird informales Wissen unmittelbar in einer operationalen Wissensrepräsentationsform oder einer Programmiersprache implementiert. Eine andere Vorgehensweise setzt sog. Role Limiting Methoden voraus. Role Limiting Methoden sind ausführbare Werkzeuge, die bereits alles Wissen zur Lösung von Problemen eines speziellen Typs enthalten. Sie müssen nur noch um das bereichsspezifische Wissen angereichert werden. Bei modellbasierte Entwicklungsmethoden werden eine Reihe aufeinanderfolgender Modelle bis hin zum endgültigen System entwickelt, deren Detaillierungs- und Formalisierungsgrad zunimmt. Der Beitrag skizziert alle drei Vorgehensweisen und zeigt deren Vor- und Nachteile und deren Einsatzgebiete auf.

1. Einleitung

E. A. Feigenbaum [Fei77] beschreibt das Knowledge Engineering als „die Kunst ein komplexes Computerprogramm zu entwickeln, das Weltwissen repräsentiert und Schlüsse daraus zieht“. In den letzten 15 Jahren wurden große Fortschritte bei der Entwicklung von Methoden, Techniken und Werkzeugen erzielt, die den Entwicklungsprozess von WBS weg von einer Kunst hin zu einer Ingenieursdisziplin führen. Grundsätzlich lassen sich das Transferparadigma und das Modellierungsparadigma bei der Entwicklung von wissensbasierten Systemen unterscheiden.

In der Zeit, in der Knowledge Engineering als Transfer von Wissen verstanden wurde (Transferparadigma), wurden eine Reihe von Werkzeugen wie z.B. regelbasierte Systeme (vgl. [Hay83]) oder auch Wissensakquisitionssysteme entwickelt, die den schnellen Transfer von menschlichem Wissen direkt in ein WBS unterstützen sollten. Diese Werkzeuge machten die impliziten Annahmen, dass (1) deren Wissensrepräsentationsformen, wie z.B. Regeln, die richtige Art

Wissen zu beschreiben darstellen und dass (2) das benötigte Wissen bereits existiert und nur aufgesammelt werden muss.

Diese Techniken und Werkzeuge bewährten sich hervorragend bei der schnellen Entwicklung von kleinen, prototypischen Systemen und trugen so zu den Anfangserfolgen in diesem Gebiet bei. Allerdings versagten diese Ansätze bei der Entwicklung großer, zuverlässiger und langlebigerer kommerzieller Systeme, bei denen der Wartungsaufwand oft jeden vertretbaren Rahmen überstieg. Als Konsequenz wurde das reine Transferparadigma verworfen und das Modellierungsparadigma entwickelt [WiB84]. Dieses betrachtet die Wissensakquisition nicht mehr nur als reinen Transferprozess von Wissen, sondern als Prozess in dem ein Modell des benötigten Wissens neu entwickelt wird, das die Funktionalität des zu entwickelnden Systems repräsentiert. Ein solches Modell überbrückt die Lücke zwischen informalen Beschreibungen des menschlichen Wissens und dessen Implementierung im WBS.

Ein ähnlicher Paradigmenwandel fand auch im Software Engineering statt. Auch dort wurden Methodiken wie z.B Strukturierte Analyse entwickelt, bei denen abstrakte Zwischenrepräsentationen entwickelt werden, bevor das endgültige System realisiert wird. Die hauptsächlichen Unterschiede zwischen WBS und „traditionellen Softwaresystemen“, die somit auch unterschiedliche Entwicklungsansätze rechtfertigen, stellen sich wie folgt dar:

- In [Par86] wird der Begriff „unvollständig spezifizierte Funktion“ geprägt, um Probleme zu charakterisieren, zu deren Lösung es eines WBS bedarf. Typischerweise lösen WBS Probleme, die entweder sehr komplex sind, oder die noch nicht einmal ganz verstanden sind [ShG92]. Somit kann ein solches System nicht vollständig vorab spezifiziert werden, sondern diese Spezifikation muss in enger Zusammenarbeit mit Experten und Benutzern entwickelt werden.
- Für viele Probleme, die mit WBS gelöst werden sollen sind nur Algorithmen bekannt, deren Komplexität so groß ist, dass das allgemeine Problem nur für kleine Probleminstanzen[1] lösbar ist. Da eine allgemeine Problemlösung somit unmöglich ist, muss das Problem eingeschränkt werden, oder die Lösung kann nur approximativ sein, oder es müssen Heuristiken gesammelt werden, die das Problem in den meisten Fällen zufriedenstellend lösen (vgl. [Neb96], [FeS96]).

[1] Ein Fall mit so wenig Eingaben, dass dieser Fall trotz der Komplexität des Problems noch in

- Der letzte Unterschied ist eher technischer Natur. In WBS wird das Wissen meist explizit repräsentiert. In „traditionellen“ Softwaresystemen ist hingegen ein Großteil des Wissens in die Algorithmen „hineinkompiliert“. In Fachgebieten, in denen sich das Wissen sehr schnell ändert, hat die explizite Repräsentation insbesondere für die Wartung des Systems Vorteile.

Zur Entwicklung von WBS existieren im Wesentlichen drei unterschiedliche Ansätze.

Die klassische Vorgehensweise zur Entwicklung eines WBS ist der Rapid Prototyping Ansatz, der weitgehend dem Transferparadigma verhaftet ist. Hierbei wird das erworbene Wissen direkt in einem ausführbaren Formalismus kodiert. Diese Kodierung stellt dann nach Abschluß des Entwicklungsprozesses direkt das WBS dar.

Für spezielle Aufgaben, z.B. für Diagnoseaufgaben, existieren Werkzeuge, sog. Role-Limiting Methoden. Sie beinhalten bereits alles Wissen zur Lösung von Problemen eines speziellen Typs und müssen nur noch um das bereichsspezifische Wissen, also z.B. um das Wissen um Krankheiten und deren Ursachen angereichert werden.

Das Modellierungsparadigma hatte sog. modellbasierte Ansätze zur Folge. In einem modellbasierten Ansatz werden unterschiedliche Aspekte und unterschiedliche Realisierungsgrade in mehreren Zwischenrepräsentationen dokumentiert, bevor schließlich das endgültige WBS realisiert wird. Modellbasierte Ansätze lehnen sich deshalb an lebenszyklusorientierte Ansätze aus dem Bereich des Software Engineering an.

Im Folgenden sollen diese drei unterschiedlichen Ansätze näher beleuchtet werden.

2. Rapid Prototyping von WBS

Beim Rapid Prototyping Ansatz (vgl. [Bra89]) wird Wissen, das aus der Literatur oder durch Befragung von Experten gewonnen wird und in Form von verbalen Äußerungen, Wissensprotokollen oder Diagrammen zur Verfügung steht, unmittelbar in einer operationalen Wissensrepräsentation oder in einer Programmiersprache implementiert. Eine weitverbreitete Repräsentationsform bilden z.B. Produktionsregeln. Das Wissen wird in Form solcher Regeln dargestellt. Ein unabhängiger Inferenzmechanismus sorgt für die Auswertung der Regeln. Das lauffähige Programm liefert dann eine direkte Rückkopplung, um das

unterstützt diese Vorgehensweise direkt die zyklische Natur des Modellierungsprozesses. Diese Rückkopplung bildet die Grundlage zur Revision oder Modifikation des Modells. Das somit entstandene ausführbare System bildet nach Abschluß der Entwicklung das endgültige WBS. Ein konzeptuelles Modell, das das Wissen unabhängig von Realisierungsaspekten beschreibt, wird nicht dokumentiert, sondern wird lediglich im Kopf des Knowledge Engineers gebildet. Die Wartung des Systems wird hierbei als unabhängige Aktivität begriffen.

Die Methode des Rapid Prototyping beinhaltet allerdings auch eine ganze Reihe von Nachteilen:

- Der Knowledge Engineer, der das System realisiert, hat eine Vielzahl von Tätigkeiten gleichzeitig zu leisten. Diese Tätigkeiten umfassen die Akquisition und Interpretation des Wissens, die Strukturierung dieses Wissens, den Entwurf und die Implementierung und Evaluierung des endgültigen Systems. Somit muss die gesamte Komplexität des endgültigen Systems und der Systementwicklung gleichzeitig vom Entwickler betrachtet werden.

- Unterschiedliche Aspekte des Wissens, wie z.B. Aspekte, die die Realisierung betreffen, funktionale Anforderungen an das System usw. werden gleichzeitig betrachtet und werden somit auch im endgültigen System untrennbar vermischt.

- Die Funktionalität entwickelt sich von einer kleinen Funktionalität zu Beginn des Prozesses zu der Funktionalität, die für das endgültige System gefordert ist. In der gleichen Weise entwickelt sich auch die Architektur des Systems, ohne vorher in ihrer Gesamtheit geplant worden zu sein.

- Das lauffähige System ist die einzige Repräsentation des Modells. Es ist also kein konzeptuelles Modell verfügbar, das von Realisierungsaspekten abstrahiert und das auch zur Dokumentation dienen könnte. Insbesondere ist diese Repräsentation und somit das zugrundeliegende Modell für den Experten oft nicht verständlich und nicht mit ihm diskutierbar und nicht durch ihn validierbar.

- Die Repräsentation des Wissens wird bestimmt und eingeschränkt durch den gewählten Implementierungsformalismus (z.B. Produktionsregeln).

Somit sind also die modellierten Gegebenheiten nur sehr schwer aus dem Programm des Systems zu entschlüsseln. Aufgrund der fehlenden Planbarkeit der Architektur und der fehlenden Trennung unterschiedlicher Arten des Wissens führt

Verständlichkeit und Wartbarkeit von größeren Systemen entscheidend beeinträchtigen. Insbesondere wegen der immerwährenden Weiterentwicklung aufgrund des approximativen Charakters eines solchen Modells und wegen der hohen Flexibilität eines solchen Systems, was die Austauschbarkeit des in ihm enthaltenen Wissens betrifft (z.B. Austausch von Regeln), verschlechtert sich die Qualität eines solchen Prototypen mit der Dauer seines Einsatzes zunehmend. Einige solcher Systeme mussten aufgrund der daraus resultierenden Schwierigkeiten bei der Wartung auch bereits abgeschaltet werden, andere wie z.B. XCON wurden vollständig umstrukturiert und neu implementiert (vgl. [CoB89]). Hinzu kommt, dass bei Fluktuation der Entwicklungsmannschaft wegen der meist fehlenden ergänzenden Dokumentation auch das Wissen über den Inhalt des Systems mit der Zeit verloren geht.

Prototyping mit Hilfe von Produktionsregelsystemen bildet auch heute noch die in der Praxis gängigste Vorgehensweise bei der Entwicklung von WBS. Es gibt einige Beispiele erfolgreich eingesetzter Systeme, aber es gibt eben auch eine Reihe von Beispielen, bei denen o.g. Nachteile zu einem unvertretbar hohen Wartungsaufwand führten.

3. Entwicklung von WBS mit Role-Limiting Methoden

Ausgangspunkte der Entwicklung von WBS bilden hierbei eine Menge vordefinierter, sog. Role-Limiting-Methoden (RLMs) (vgl. [Mar88, McD88]). Diese RLMs sind auf dem Rechner ausführbare Werkzeuge, die genau eine fest vorgegebene Problemlösungsmethode realisieren. So realisiert z.B. das Werkzeug COKE [PoP92] die Problemlösungsmethode “Vorschlagen und Vertauschen”. Diese wird zur Lösung von Zuordnungsproblemen benutzt. Dabei werden jeweils Zuordnungen generiert und falls diese vorzugebende Bedingungen verletzen, werden mit vorzugebenden Austauschregeln Zuordnungen variiert, bis eine Lösung gefunden ist. Neben der Problemlösungsmethode ist die Strukturierung des Wissens, wie es von der Problemlösungsmethode verwendet wird, in solchen RLMs bereits vorgegeben (vgl. [KLV90]). Die Vorgehensweise besteht also aus der Auswahl einer RLM und der anschließenden Eingabe anwendungsspezifischen Wissens, z.B. Bedingungen, Austauschregeln und konkrete Fakten. Da RLMs ausführbar sind, kann das anwendungsspezifische Wissen durch Prototyping entwickelt werden.

Die Verwendung solcher RLMs weist die folgenden Vorteile auf:

- Falls eine für den gegebenen Aufgabentyp passende Methode vorhanden ist, muss nur noch das für die aktuelle Anwendung notwendige anwendungsspezifische Wissen eingegeben werden.

- Die gesamte Problemlösungsmethode, wie z.B. “Vorschlagen und Vertauschen”, ist in den Werkzeugen bereits realisiert und somit fester Bestandteil dieser RLMs, so dass der Experte oder der Knowledge Engineer diese nicht im Detail verstehen muss, um ein WBS zu entwickeln, sofern die Problemlösungsmethode für die aktuell zu lösende Aufgabe passt.

- Das Werkzeug, das ja eine Problemlösungsmethode zusammen mit der Strukturierung des Wissens realisiert, ist jederzeit ausführbar. Somit kann das anwendungspezifische Wissen frühzeitig durch Testen validiert werden. Diese Vorgehensweise genügt somit der zyklischen Natur des Modellierungsprozesses.

- Wissensakquisitionswerkzeuge (wie z.B. MOLE [Esh88]), die die Eigenschaften der fest eingebauten Problemlösungsmethode ausnützen, können so spezialisiert werden, dass das benötigte Wissen sehr effizient erfragt werden kann.

Insgesamt wird also die Komplexität des Entwicklungsprozesses entscheidend verringert, weil große Teile des Wissens (die Problemlösungsmethode und die Strukturierung des Wissens) fest vorgegeben sind und weil durch entsprechend spezialisierte Wissensakquisitionswerkzeuge das benötigte Wissen sehr effizient erfragt werden kann.

Auf der anderen Seite begründen die Ursachen für die Vorteile der Anwendung von RLMs auch deren Nachteile:

- Entscheidend für die Qualität des zu entwickelnden WBS ist die Auswahl einer geeigneten RLM aus einer Menge vorgegebner RLMs, die zu dem zu lösenden Problem passt. Bislang wurden dazu noch keine anerkannten Kriterien und Methoden entwickelt. Eine gewisse Hilfe bei dieser Auswahl kann die Einteilung der Probleme in verschiedene Problemtypen und deren Charakterisierung durch spezifische Eigenschaften bilden (vgl. [Pup93]).

- Oft passt eine RLM nur in groben Zügen zu der zu lösenden Aufgabe. Durch die fest vorgegebene Problemlösungsmethode und Strukturierung des Wissens ist eine RLM nur in einem sehr eng begrenzten Rahmen (etwa durch Parametrierung) anpassbar. Die Folge davon ist dann oft, dass die zu

- Falls die gestellte Aufgabe nur durch eine Kombination von Problemlösungsmethoden gelöst werden kann, müssen RLMs zur Lösung der Gesamtaufgabe in geeigneter Weise kombinierbar sein. Solche Kombinationen könnten aus der Nacheinanderausführung der Problemlösungsmethoden, aber auch z.B. aus der Benutzung einer Problemlösungsmethode durch eine andere, bestehen. Auch hierbei bilden die fest vorkodierten Strukturen der Problemlösungsmethoden ein ernsthaftes Hindernis zur freien Kombination dieser Methoden.

- Wissensbasierte Systeme werden in Zukunft wohl verstärkt in größere Systemumgebungen integriert werden. Sie werden deshalb oft nur eine Komponente unter anderen darstellen. Die Role-Limiting Methoden liegen bereits fertig implementiert als Werkzeuge vor. Sie sind deshalb entsprechend schwierig in andere Umgebungen integrierbar.

Somit wird klar, dass dieser Ansatz trotz seiner Nützlichkeit in speziellen Bereichen auch Nachteile aufweist, die insbesondere in der schlechten Adaptierbarkeit an die aktuelle Problemstellung begründet sind. In neueren Ansätzen werden dann solche RLMs auch verallgemeinert und damit flexibler (vgl. [Mus92], [PoG93]). Auf der anderen Seite können jedoch die fest eingebauten Strukturen ausgenutzt werden, um Wissensakquisitionswerkzeuge zu entwickeln, die spezifisch für die RLM sind. META-KA [Gap95] ist z.B. ein Werkzeug, der diese Strukturen ausnutzt, um automatisch graphische Editoren zur Wissens-akquisition zu generieren. In ähnlicher Weise wurde im PROTEGE-II Projekt [EPM94] das Werkzeug DASH zur Generierung von Wissensakqusitions-werkzeugen entwickelt. Somit ist es nach der Auswahl, Adaption und Konfiguration solcher RLMs und nach einer anschließenden Generierung eines Wissensakquisitionswerkzeuges oft für den Experten selbst möglich, den variablen Teil des Wissens zu modellieren.

Die Verbreitung von RLMs in der Praxis ist relativ gering. Dies liegt insbesondere daran, dass wenig ausgereifte Produkte am Markt verfügbar sind.

4. Vorgehensweise bei Modellbasierten Ansätzen

4.1 Prinzipien modellbasierter Ansätze

Heute ist das Modellierungsparadigma innerhalb des Knowledge Engineering sehr

Prototyping Ansatzes und die Unflexibilität der Role-Limiting Methoden. Sie sind deshalb insbesondere für die Entwicklung großer, langlebiger Systeme gedacht. Die meisten aktuellen modellbasierten Ansätze realisieren dabei folgende Prinzipien:

- Sie unterscheiden ein Domänenmodell und ein Taskmodell. Ein Taskmodell beschreibt das Problem, das durch das WBS gelöst werden soll. Eine „generic task“ [Cha92] abstrahiert von einem spezifischen Anwendungsgebiet (Domäne) und ist daher auf eine Vielzahl von Anwendungsgebieten anwendbar (also wiederverwendbar). So kann z.B. eine Diagnoseaufgabe unabhängig davon formuliert werden, ob sie nun zur Krebsdiagnose, zur Diagnose von Herzkrankheiten oder auch zur Diagnose von Fehlern in Maschinen eingesetzt wird. In gleicher Weise können so auch Modelle über einen Anwendungsbereich (Domänenmodelle) für eine Reihe von Aufgaben verwendet (also wiederverwendet) werden. Somit kann z.B. das funktionale Modell eines technischen Systems für Entwurfszwecke als auch für Diagnosezwecke verwendet werden. Durch Wiederverwendung solcher Modelle können somit die Kosten einer Neuentwicklung reduziert, als auch die Qualität der Modelle verbessert werden (vgl. [FFR97, HSW97]).
- Die Entwicklung des Problemlösungsprozesses ist bereits Bestandteil eines solchen frühen Modells. Im Gegensatz dazu wird im Software Engineering eine klare Trennung zwischen der Beschreibung der Funktionalität (dem Was) und der Beschreibung wie diese Funktionalität erreicht wird (dem Wie) vorgeschlagen. Die erstere ist die Spezifikation, die zweite der Entwurf bzw. die Implementierung. Aufgrund der Schwierigkeiten bei WBS nur das Was beschreiben zu können (siehe Einleitung), wird hier eine realisierungsunabhängige Beschreibung des Wie entwickelt. Problemlösungsmethoden [BrV94] schaffen die Verbindung zwischen Taskmodellen und Domänenmodellen, indem sie beschreiben, wie das Domänenwissen eingesetzt werden muss, um das Problem effizient zu lösen.

Modellbasierte Ansätze integrieren meist semiformale und formale Spezifikationstechniken:

- Informale und semiformale Modelle bieten eine abstrakte und informale Ebene zur Repräsentation von Wissen. Graphische Darstellungen wie Entity-Relationship-Diagramme, Datenflussdiagramme, Programmflussdiagramme und Zustandsübergangsdiagramme sind für den Experten und den

Benutzer leicht verständlich und unterstützen somit die Kommunikation zwischen Experten, Benutzer und Systementwickler.

- Formale Sprachen erlauben die geforderte Funktionalität und das dafür notwendige Anwendungswissen eindeutig und präzise, aber unter Abstraktion von Realisierungsaspekten, zu beschreiben. Mit Verifikationstechniken oder symbolischer Ausführung kann die Evaluierung des formal beschriebenen Modells unterstützt werden.

Die Entwicklungsaktivitäten sind meist in ein zyklisches Prozessmodell eingebettet, das die inkrementelle, zyklische und stark reversible Natur des Wissensakquisitionsprozesses widerspiegelt:

- Die Entwicklung eine Reihe aufeinanderfolgender Modelle garantiert einen sanften Übergang von semiformalen Beschreibungen, über ein formales Modell, bis hin zum Entwurf des Systems. Dieser sanfte Übergang zusammen mit der gemeinsamen Struktur der Modelle trägt wesentlich zur inkrementellen und reversiblen Entwicklung eines WBS bei.
- Die Ausführbarkeit eines formal beschriebenen Modells ermöglicht dessen Entwicklung durch Prototyping. Dies erlaubt schwache und nicht bekannte Anforderungen zu klären. Zusätzlich kann das jeweilige Modell frühzeitig ohne Berücksichtigung nachfolgender Modelle validiert werden.

4.2 Der Entwicklungsprozess

Im Folgenden wird der Knowledge Engineering Prozess anhand eines Beispiels eines modellbasierten Ansatzes, dem MIKE - Ansatz (Model-based and Incremental Knowledge Engineering) [AFL98] dargestellt. Dazu wird zuerst das zugrundeliegende Modellierungsparadigma skizziert. Dann werden die unterschiedlichen Entwicklungsaktivitäten, die daraus entstehenden Dokumente und die Abfolge der Entwicklungsaktivitäten beschrieben. Schließlich werden die Anwendung von Prototyping und Wiederverwendung im Entwicklungsprozess diskutiert.

4.2.1 Das Modell der Expertise

Ähnlich wie im Software Engineering zwischen der Spezifikation eines Systems und seiner Realisierung unterschieden wird, wird im Knowledge Engineering zwischen einer Repräsentation auf dem Knowledge Level und einer Repräsentation auf dem Symbol Level unterschieden (vgl. [New82]). Ein Knowledge Level Modell

Spezifikation im Software Engineering, sondern darüber hinaus auf abstraktem Niveau auch noch *wie* das System diese Funktionalität erbringt:

- Aufgrund der Komplexität der zu lösenden Probleme (siehe Einleitung) müssen auch anwendungsspezifische Problemeinschränkungen und Heuristiken modelliert werden. Solche Einschränkungen und Heuristiken sind oft genau das Wissen, das einen Experten von einem Laien unterscheidet [Van89]. Diese Einschränkungen und Heuristiken ergänzen die Problemlösungsmethode und beschreiben somit einen Teil dessen, wie das System seine Funktionalität erbringt.
- Typischerweise ist die geforderte Funktionalität nur in Form von prozeduralem Wissen, also Wissen darüber wie das Problem gelöst wird, verfügbar. Somit kann die Funktionalität des WBS nur durch eine abstrakte Beschreibung wie das Problem gelöst wird, modelliert werden. Im Gegensatz dazu wird z.B. im Software Engineering oft davon ausgegangen, dass die Funktionalität, unabhängig davon wie diese erreicht wird, beschrieben werden kann.

Als Folge davon ist es nicht möglich eine Spezifikation (ohne Beschreibung des Wie) zur erstellen (wenn dies überhaupt möglich ist) und diese dann mit „normalem“ Informatik-Know-How zu lösen, sondern anwendungspezifisches Wissen (Domänenmodell) und Wissen über das Problemlösungsverfahren (Taskmodell) sind zwingend notwendig, um eine Lösung zu entwickeln.

Das Knowledge Level Modell wird in MIKE *Modell der Expertise* genannt (vgl. [SWA94]). Dieses Modell unterscheidet unterschiedliche Wissensarten und beschreibt diese auf unterschiedlichen Ebenen:

- Die Domänenebene enthält Wissen über anwendungsspezifische Konzepte, deren Attribute und deren Beziehungen und über Fakten des Anwendungsbereichs. Darüber hinaus werden hier anwendungsspezifische Heuristiken für den Problemlösungsprozess dargestellt.
- Die Inferenzebene beschreibt eine funktionale Sicht auf den Problemlösungsprozess. Hier wird beschrieben, welche Inferenzen gezogen werden müssen, welches Domänenwissen dafür jeweils benötigt wird und welche Datenabhängigkeiten zwischen den Inferenzen bestehen.
- Die Kontrollebene spezifiziert die zu lösende Aufgabe und deren Teilaufgaben. Zudem wird hier Kontrollwissen über die Abfolge der einzelnen Problemlösungsschritte repräsentiert.

Die Inferenz- und Kontrollebene zusammen repräsentieren die Problemlösungs–methode. Die Problemlösungsmethode zusammen mit den Heuristiken auf der Domänenebene beschreiben also, *wie* das System seine Funktionalität erbringt.

Inferenz- und Kontrollebene sind vergleichbar mit den Datenfluss- und Kontrollflussbeschreibungen bei der Strukturierten Analyse im Software Engineering. Beide werden jedoch generisch, also unabhänig von einem konkreten Anwendungsbereich beschrieben. Eine Trennung des anwendungsspezifischen Wissens von dem Problemlösungswissen auf der Domänenebene existiert so nicht in den Modellen des SE, sondern Datenbeschreibungen finden für die einzelnen Stores (jedoch nicht generisch) im Datenflussdiagramm statt.

Die Struktur des Modells der Expertise und deren Beschreibungsprimitive werden während des gesamten Entwicklungsprozesses in MIKE durchgängig verwendet. Somit arbeiten alle Entwicklungsaktivitäten an demselben konzeptuellen Mdell und alle daraus resultierende Entwicklungsdokumente basieren auf diesem konzeptuellen Modell. Während des nachfolgenden Realisierungsprozesses soll die Struktur dieses Modells weitgehend erhalten bleiben und sich schließlich auch in der Implementierung wiederspiegeln. Somit können Teile des realisierten Systems zurückverfolgt werden bis zu korrespondierenden Beschreibungen früherer Entwicklungsdokumente.

4.2.2 Entwicklungsaktivitäten und -dokumente

Die unterschiedlichen Entwicklungsaktivitäten und die Dokumente, die aus diesen Aktivitäten resultieren, werden in Abbildung 1 gezeigt (vgl. [Ang93], [Neu94]). Während des gesamten Entwicklungsprozesses muss eine grosse Distanz zwischen den informalen Anforderungen und dem menschlichen Wissen auf der einen Seite und dem realisierten WBS auf der anderen Seite überwunden werden. Durch Aufteilung dieser Distanz in mehrere Teile kann die Komplexität des Entwicklungsprozesses entscheidend verringert werden, weil in jedem Schritt unterschiedliche Aspekte des Systems, weitgehend unabhängig von anderen Aspekten, betrachtet und entwickelt werden können. Deshalb besteht der MIKE Entwicklungsprozess aus einer Reihe von Einzelaktivitäten: Erhebung, Interpretation, Formalisierung/Operationalisierung, Entwurf und Implementierung. Jede dieser Aktivitäten behandelt einen unterschiedlichen Aspekt der Systementwicklung.

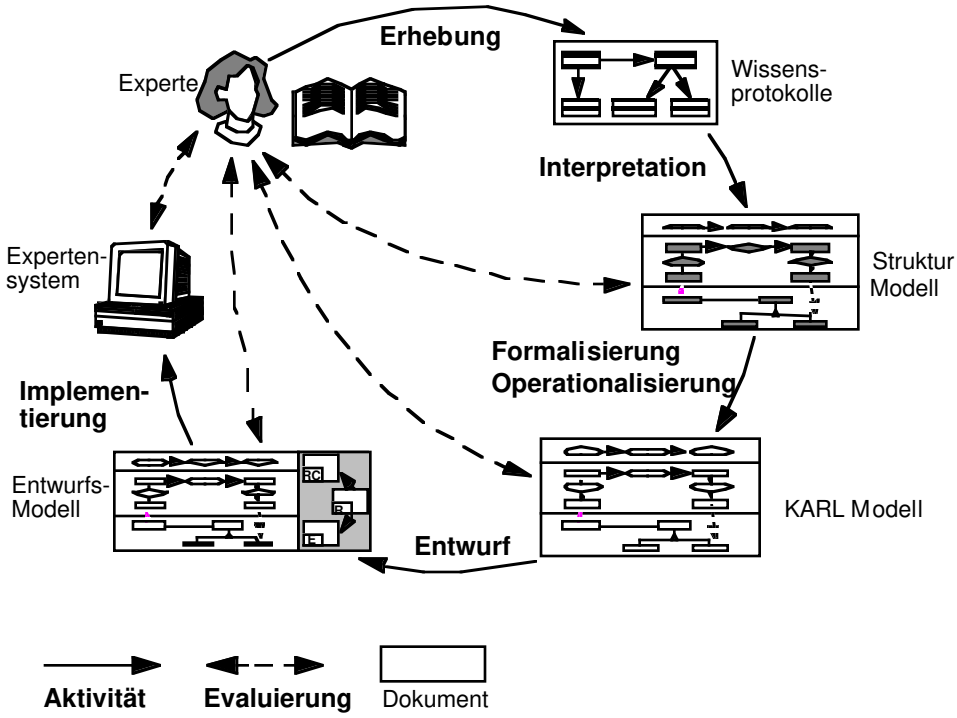

Abbildung 1: Aktivitäten und Dokumente des Entwicklungsprozesses von MIKE

Der Wissensakquisitionsprozess beginnt mit der Erhebungsphase (siehe Abb. 1). Darin wird das Wissen des Experten über das Problem, über den Anwendungsbereich und über mögliche Problemlösungsstrategien erfasst. Hierbei finden Methoden wie das strukturierte Interview, Beobachtungen von Experten bei der Problemlösung, Strukturierungstechniken u.ä. (siehe [Eri92]) Anwendung. Das daraus resultierende Wissen liegt in Form von natürlichsprachlichem Text oder auch Tabellen vor und wird in sog. Wissensprotokollen abgelegt.

Während der Interpretationsphase werden innerhalb der Wissensprotokolle Strukturen gesucht und in einer semiformalen Variante des Modells der Expertise repräsentiert: dem Strukturmodell ([Neu93], [Neu94]). Die Strukturierungsinformation, wie z.B. die Datenabhängigkeiten einzelner Problemlösungsschritte, werden in einer festen, eingeschränkten Sprache beschrieben. Hingegen bestehen die Basisbausteine dieses Modells, also z.B. die Beschreibung eines einzelnen elementaren Problemlösungsschrittes immer noch aus dem natürlichsprachlichen Text, wie er aus den Wissensprotokollen entstammt. Dieses Modell bildet somit eine erste Strukturierung des Wissens. Es hat eine vermittelnde Rolle zwischen den unstrukturierten natürlichsprachlichen Beschreibungen und der nachfolgenden formalen Repräsentation des Wissens. Eine solche Beschreibung bietet dabei den Vorteil, dass sie für den Experten

vereinfacht. Der Experte kann so in den Entwicklungsprozess mit einbezogen werden und er kann so frühzeitig die entstandenden Strukturen validieren. Darüber hinaus bildet eine solche Repräsentation die Basis zur Dokumentation des Systems und kann vom WBS selbst in seiner Erklärungskomponente genutzt werden. Dies erleichtert auch die spätere Wartung des Systems.

Das semiformale Modell der Expertise bildet die Basis für den Formalisierungs-/ Operationalisierungsschritt. Das Ergebnis dieses Schrittes bildet das formale Modell der Expertise: das KARL Modell. Dieses Modell besitzt dieselbe Struktur wie das semiformale Modell der Expertise. Die Basisbausteine sind darin jedoch nicht mehr informal, sondern in der formalen Sprache KARL [FAS98] beschrieben. Diese Beschreibung hat eine eindeutige Semantik und vermeidet einen Interpretationsspielraum, wie er in natürlichsprachlichen Beschreibungen sehr wohl innewohnt, und beschreibt somit das Wissen eindeutig und präzise. Zur Entwicklung des KARL Modells müssen die natürlichsprachlichen Beschreibungen also interpretiert und präzisiert werden, was zu einem besseren Verständis des gesamten Problemlösungsprozesses führt. Weil die Sprache KARL bis auf wenige Ausnahmen ausführbar ist (siehe [Ang93]), kann das KARL Modell direkt auf eine ausführbare Repräsentation abgebildet werden. Somit kann das KARL Modell durch Testen validiert werden.

Das KARL Modell ist das Ergebnis der Wissensakquisitionsphase, die die Schritte Erhebung, Interpretation und Formalisierung/Operationalisierung umfasst. Dieses Modell beschreibt alle funktionalen Anforderungen an das zu entwickelnde System. Während der nachfolgenden Entwurfsphase werden darüber hinaus zusätzlich nicht-funktionale Anforderungen betrachtet [LaS95]. Dies umfasst z.B. Anforderungen wie die Effizienz des Systems, seine Wartbarkeit, seine Robustheit, seine Portabilität usw. Zusätzlich werden die Einschränkungen und Bedingungen, die aus der Zielhard- und Software entstehen, berücksichtigt. Der Effizienzaspekt wird bereits innerhalb der Wissensakquisitionsphase, soweit er den konzeptuellen Entwurf der Problemlösungsmethode betrifft, betrachtet. Insoferen ist die funktionale Dekomposition bereits ein Bestandteil der Wissensakquisitionsphase. Die Entwurfsphase von MIKE korrespondiert deshalb mit dem Feinentwurf und dem Modulentwurf im Software Engineering. Das Entwurfsmodell, das während der Entwurfsphase entwickelt wird, wird in der Sprache DesignKARL ([Lan94], [Lan95]) einer Erweiterung von KARL formuliert. DesignKARL enthält zusätzlich zu den aus KARL stammenden Primitiven Sprachkonstrukte zur Strukturierung des Modells und zur Beschreibung von Algorithmen und Datenstrukturen. Das Entwurfsmodell kann,

Das Entwurfsmodell deckt alle funktionalen und nicht-funktionalen Anforderungen ab. Während der Implementierungsphase wird dieses Modell dann in der Zielhard- und Softwareumgebung implementiert.

Das Ergebnis dieses gesamten Entwicklungsprozesses ist eine Menge von Verfeinerungen des Modells der Expertise. In allen diesen Verfeinerungen sind korrespondierende Teile explizit zueinander in Beziehung gesetzt. Das Wissen des Strukturmodells wird mit sog. *Erhebungslinks* zu dem entsprechenden Wissen in den Wissensprotokollen verknüpft. Konzepte und Inferenzaktionen stehen in Bezug zu Protokollknoten, in denen sie natürlichsprachlich beschrieben sind. Das Entwurfsmodell verfeinert das KARL Modell, indem einzelnen Inferenzen Algorithmen zugeordnet werden und durch die Einführung zusätzlicher Datenstrukturen. Diese Teile des Entwurfsmodells sind durch Querverweise verbunden mit den entsprechenden Inferenzen des KARL Modells und sind so auch verbunden mit den entsprechenden Protokollknoten ([Lan94], [Lan95]). Diese expliziten Beziehungen zusammen mit dem Bestreben die Struktur des Modells der Expertise während des Entwurfs beizubehalten, garantieren die Rückverfolgbarkeit von Modellteilen bis hin zu frühen informalen Beschreibungen der funktionalen und nicht-funktionalen Anforderungen.

4.2.3 Prototyping in MIKE

Der Entwicklungsprozess von MIKE integriert unterschiedliche Arten des Prototyping ([Ang93], [AFL98]). Der gesamte Entwicklungsprozess, also die Sequenz der Entwicklungsschritte Wissensakquisition, Entwurf und Implementierung, werden zyklisch wie beim Spiralmodell [Boe88] durchlaufen (siehe große Spirale in Abb. 2). Jeder durchlaufende Zyklus hat einen Prototypen des WBS zum Ergebnis, der durch Testen in der realen Zielumgebung validiert werden kann. Die Ergebnisse dieser Validierung gehen in den nächsten Entwicklungszyklus ein und führen zur Korrektur, zur Modifikation oder zur Erweiterung dieses Prototyps. Dieses Vorgehen wird solange fortgesetzt bis alle Anforderungen an das WBS erfüllt sind. Im Gegensatz zu einem linearen Prozessmodell können dadurch sogar sich während der Entwicklungszeit ändernde Anforderungen berücksichtigt werden. Diese zyklische Vorgehensweise integriert darüber hinaus die Wartung des Systems in den Entwicklungsprozess. Die Wartung erfolgt, indem das aktuelle System evaluiert wird und nachfolgend eine Reihe von Entwicklungszyklen durchgeführt werden, in denen das System korrigiert oder an neue Anforderungen angepasst wird.

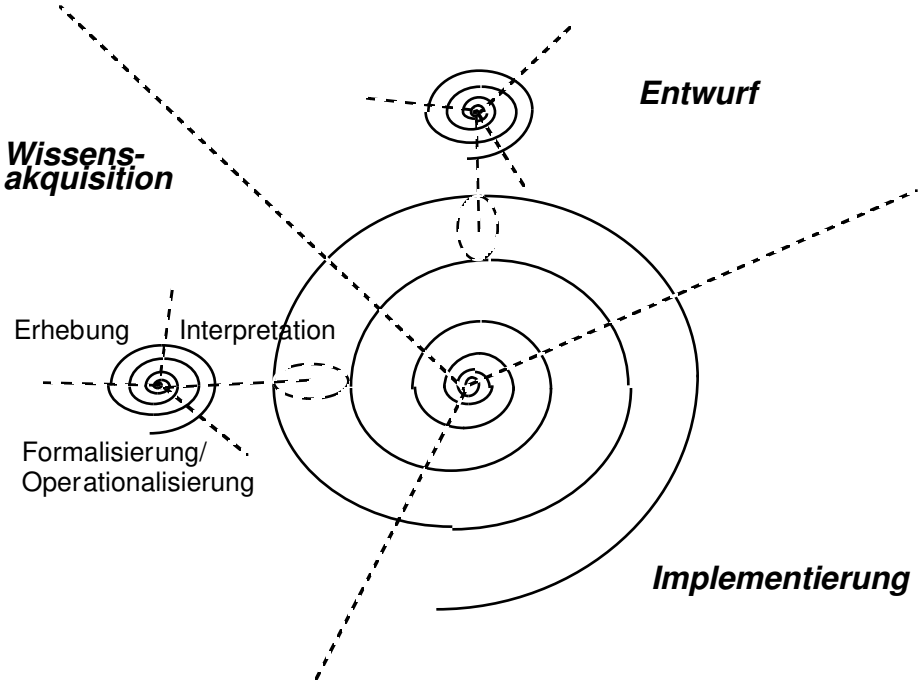

Abbildung 2: Prototyping im MIKE-Entwicklungsprozess

Durch die Ausführbarkeit der Sprache KARL kann das KARL Modell durch exploratives Prototyping [Flo84] entwickelt werden. So werden die verschiedenen Entwicklungsaktivitäten bei der Wissensakquisition zyklisch durchlaufen (siehe kleinen Zyklus bei der Wissensakquisition in Abb. 2) bis der Prototyp (das KARL Modell) die angestrebte Funktionalität erreicht hat. Dadurch, dass das KARL Modell die gesamte geforderte Funktionalität umfasst und dies auch durch den Experten validiert werden konnte, stellt dieses Entwicklungsdokument eine fundierte Basis für die nachfolgende Realisierung des Systems dar. Durch die Rückkopplung können Sackgassen frühzeitig erkannt und vermieden werden.

Ganz ähnlich wie bei der Wissensakquisition werden die nicht-funktionalen Anforderungen in der Entwurfsphase durch experimentelles Prototyping [Flo84] angegangen. Für alle geforderten nicht-funktionalen Anforderungen werden entsprechende Entwurfsentscheidungen, wie z.B. der Einsatz von bestimmten Algorithmen und Datenstrukturen, in DesignKARL formuliert. Das so entstandene Entwurfsmodell wird dann durch Testen validiert. Dies wird solange fortgeführt bis alle nicht-funktionalen Anforderungen erfüllt sind (siehe kleinen Zyklus in der Entwurfsphase in Abb. 2).

Durch die Validierbarkeit von funktionalen Anforderungen in der Wissensakquisitionsphase und von nicht-funktionalen Anforderungen in der Entwurfsphase können diese Entwicklungsaktivitäten getrennt und weitgehend unabhängig voneinander ausgeführt werden. Somit können sich die Entwicklungsaktivitäten jeweils unabhängig voneinander auf unterschiedliche

Aspekte konzentrieren: diese divide-und-conquer Strategie reduziert beträchtlich die Komplexität des gesamten Entwicklungsprozesses.

4.2.4 Wiederverwendung

Der beschriebene Entwicklungsprozess von MIKE kann zusätzlich durch Wiederverwendung von Modellteilen unterstützt werden, die in früheren Projekten für ähnliche Problemstellungen und/oder ähnliche Anwendungsbereiche entwickelt wurden.

Die Problemlösungsmethode (PLM) wird auf der Kontroll- und der Inferenzebene des Modells der Expertise repräsentiert [BrV94], [Pup93]. Eine PLM zerlegt die gestellte Aufgabe in eine Hierarchie von Teilaufgaben. Jede dieser Teilaufgaben wird wiederum eine PLM zugeordnet, die diese Teilaufgabe zu lösen in der Lage ist. Jede dieser PLMs kann wiederum neue Teilaufgaben beinhalten bis zuletzt eine elementare Ebene erreicht ist. Eine PLM beschreibt in generischer Weise die unterschiedlichen Rollen innerhalb des Problemlösungsprozesses, die von unterschiedlichen Teilen des Anwendungswissens eingenommen werden. Dies kann z.B. auch dazu ausgenutzt werden, die Problemlösungsmethode als Richtlinie zu benutzen um anwendungsspezifisches Wissen und Heuristiken zu erheben. Eine PLM kann unabhängig von der Anwendung repräsentiert werden. So kann z.B. die PLM „Hierarchische Klassifikation“, die ein Objekt in eine Hierarchie von Objekten einordnet, dazu benutzt werden, Autos zu klassifizieren, oder auch dazu, Tiere in eine entsprechende Hierarchie einzuordnen. Damit kann diese PLM für ein anderes Problem, das durch hierarchische Klassifikation gelöst werden kann, wiederverwendet werden. Wiederverwendung und Adaption einer PLM basieren auf den Anforderungen, die sie an das Anwendungswissen stellt und auf den Einschränkungen der zu lösenden Aufgaben [BFS96]. In [Fen97] wird diskutiert, wie eine Bibliothek von PLMs organisiert werden könnte und wie die Entwicklung von PLMs durch schrittweise Verfeinerung erfolgen kann.

Ähnlich der Wiederverwendung bei PLMs kann die Entwicklung der Domänenebene durch wiederverwendbare Ontologien [Gru93] unterstützt werden:

- Sog. Commonsense Ontologien definieren eine Sammlung sehr allgemeingültiger Konzepte und Beziehungen zwischen diesen Konzepten. Diese Konzepte sind unabhängig von einer konkreten Anwendung. Um solche Ontologien handhaben zu können werden sie typischerweise in unterschiedliche Cluster, wie z.B. Cluster für Zeit, für Ereignisse, oder für den Raum (vgl. [LeG90, KnL94]), unterteilt.

- Domänenontologien stellen Konzeptualisierungen eines konkreten Anwendungsbereichs dar. Sie beschreiben die Struktur und die Inhalte eines speziellen Anwendungsbereichs und definieren für diesen Anwendungsbereich das Vokabular [vHe97]. Domänenontologien zielen darauf ab das Anwendungswissen für unterschiedliche Aufgaben nutzbar und damit wiederverwendbar zu machen. In [FFR97] wird der Ontolingua Server beschrieben, der die Entwicklung komplexer Ontologien unter Verwendung von Ontologien aus einer Bibliothek unterstützt.

Sowohl die Wiederverwendung wie auch die Adaption und die Konfiguration von Ontologien müssen in geeigneter Weise methodisch unterstützt werden. In [PiS94] wird der Ansatz KARO beschrieben, der eine Menge formaler, lexikalischer und graphischer Methoden bereitstellt um Ontologien wiederzuverwenden und sie an eine neue Aufgabe zu adaptieren. In [PiS97] wird diskutiert, wie die Wiederverwendung von PLMs und die Wiederverwendung von Ontologien zur Konstruktion des Modells der Expertise kombiniert werden können.

Modellbasierte Ansätze sind relativ neu. Bislang existieren dazu nur wenige Werkzeuge am Markt. Allerdings wurde die modellbasierte Methodik KADS [SWA94] in einer Reihe von öffentlich geförderten Projekten mit dem Ziel eines europaweiten Standards zum Bau von WBS entwickelt. Es bleibt aber abzuwarten inwieweit sich diese Methodik tatsächlich etabliert. Dies gilt insbesondere für den amerikanischen Raum, in dem immer noch weitgehend Rapid Prototyping mit Produktionsregelsystemen oder Expertensystemshells zur Anwendung kommt.

5. Fazit

Zur Entwicklung von wissensbasierten Systemen existieren im Wesentlichen die Anätze: Rapid Prototyping, Role-Limiting Methoden und modellbasierte Ansätze.

Die Anwendung von Rapid Prototyping im Bereich wissensbasierter Systeme ist wegen der Unterstützung einer zyklischen Vorgehensweise gut für Probleme mit unstrukturierten oder vagen Anforderungen geeignet. Allerdings entstehen hierbei meist unstrukturierte und schlecht dokumentierte Lösungen. Rapid Prototyping ist deshalb nur dann geeignet, wenn der Anwendungsbereich klein und übersichtlich ist und nur eine kurzzeitige Systembenutzung vorgesehen ist.

Role-Limiting Methoden realisieren jeweils genau eine Problemlösungsmethode. Sie sind allerdings sehr unflexibel, falls sie an die zu lösende Aufgabe angepasst werden müssen. Zudem sind Role-Limiting schlecht kombinierbar und schlecht in ein Gesamtsystem integrierbar. Falls also für die gestellte Aufgabe eine genau

benutzt werden soll, also nicht die Notwendigkeit besteht, es in eine größere Umgebung integrieren zu müssen, ist die Entwicklung mit Hilfe von Role-Limiting Methoden mit dem geringsten Aufwand verbunden.

Modellbasierte Methoden sind aus den Problemen der vorgenannten Ansätze heraus entwickelt worden. Durch den schrittweisen Übergang von informalen Beschreibungen des Wissens über formale Beschreibungen bis hin zum endgültigen System wird die Komplexität der Entwicklung zu einem bestimmten Zeitpunkt wesentlich reduziert. Die Kaskade von Beschreibungen unterstützt die Dokumentation und damit die Wartbarkeit des Systems. Ausführbare und formale Modelle erlauben die Validierung und Verifikation dieser Modelle ohne nachfolgende Modelle berücksichtigen zu müssen. Die Wiederverwendung von Modellteilen reduziert den Entwicklungsaufwand und verbessert die Qualität der entstehenden Modelle. Modellbasierte Ansätze sind deshalb auch für größere und langlebigere Systeme geeignet. Sie sind wesentlich flexibler als z.B. Role-Limiting Methoden und unterstützen damit auch besser die Entwicklung wissensbasierter Komponenten größerer Gesamtsysteme.

In Zukunft werden WBS wohl nicht mehr nur als isolierte Systeme betrachtet werden können, sondern in größere Gesamtsysteme integrierbar sein. Dies bedeutet jedoch, dass Methoden aus dem Software Engineering und Methoden aus dem Knowledge Engineering in geeigneter Weise kombiniert werden müssen, um zu Vorgehensweisen für die Entwicklung solcher hybrider Systeme zu gelangen.

Danksagung

Wir danken Susanne Neubert und Dieter Landes, die vielfach zu den Ideen in diesem Beitrag beisteuerten.

6. Literatur

[AFL98] J. Angele, D. Fensel, D. Landes und R. Studer: Developing Knowledge-Based Systems with MIKE, erscheint in *International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering*.

[Ang93] J. Angele: *Operationalisierung des Modells der Expertise mit KARL.* Dissertationen zur Künstlichen Intelligenz, Band 53, infix, St. Augustin, 1993.

[BFS96] R. Benjamins, D. Fensel und R. Straatman (1996): Assumptions of Problem-Solving Methods and Their Role in Knowledge Engineering. In

(ECAI-96), Budapest, August 12-16, 1996.

[Boe88] B.W. Boehm (1988): A Spiral Model of Software Development and Enhancement, *IEEE Computer,* Mai 1988, 61-72.

[Bra89] I. Bratko: Fast prototyping of expert systems using Prolog. In G. Guida et. al. (Eds.): *Topics in Expert Systems Design*, Elsevier Science Publisher B. V., North-Holland, 1989, 69-86.

[BrV94] J. Breuker und W. Van de Velde (eds.) (1994*): The CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, Holland, 1994.

[Cha92] B. Chandrasekaran, T.R. Johnson und J. W. Smith (1992): Task Structure Analysis for Knowledge Modeling*, Communications of the ACM*, 35(9), 1992, 124—137.

[CoB89] W. Coy, L. Bonsiepen: *Erfahrung und Berechnung*. Informatik Fachberichte, Nr. 229, 1989.

[EPM94] H. Eriksson, A. R. Puerta und M. A. Musen: Generation of Knowledge Acquisition Tools from Domain Ontologies, *Int. J. Human-Computer Studies 41,* 1994, 425-453.

[Eri92] H. Eriksson: A Survey of Knowledge Acquisition Techniques and Tools and their Relationship to Software Engineering*, Journal of Systems and Software* ,19: 97-107, 1992.

[Esh88] Eshelman, L.: MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In [Mar88], 1988, 37-80.

[FAS98] D. Fensel, J. Angele und R. Studer: The Knowledge Acquisition and Representation Language KARL, erscheint in *IEEE Transactions on Knowledge and Data Engineering.*

[FFR97] A. Farquhar, R. Fikes und J. Rice: The Ontolingua Server (1997): a Tool for Collaborative Ontology Construction*, International Journal of Human-Computer Studies (IJHCS),* 46(6), 1997, 707—728.

[Fei77] E. A. Feigenbaum: The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering. In *Proceedings of the International Joint Conference on AI (IJCAI-77),* 1977, 1014—1029.

[Fen97] D. Fensel: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In E.Plaza u.a. (Hrsg.), Knowledge Acquisition, Modeling and Management, LNAI, 1319, Springer, 1997.

[FeS96] D. Fensel und R. Straatman: The Essence of Problem-Solving Methods: Making Assumptions for Efficiency Reasons. In N. Shadbolt et al. (eds.): *Advances in Knowledge Acquisition,* LNAI 1076, Springer-Verlag, 1996.

[Flo84] C. Floyd: A Systematic Look at Prototyping. In: R. Budde et al. (eds.)

[Gap95] U. Gappa: *Grafische Wissensakquisitionssysteme und ihre Generierung*. Dissertationen zür Künstlichen Intelligenz, Band 100, infix, St. Augustin, 1995.

[Gru93] T.R. Gruber: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition 5(2):*199-221, 1993.

[Hay83] F. Hayes-Roth, D. A. Waterman und D. B. Lenat (eds.) (1983): *Building Expert Systems*. Addison-Wesley Publisher, 1983.

[KLV90] Karbach, W.; Linster, M.; Voß, A.: Models of Problem-Solving: One label - One idea? In *Proceedings of the 4th European Knowledge Acquisition Workshop EKAW'90* , Amsterdam, 1990.

[KnL94] K. Knight und S. Luk (1994): Building a Large Knowledge Base for Machine Translation. In *Proceedings of the American Ass. for Artificial Intelligence Conference (AAAI'94),* Seattle, 1994.

[Lan94] D. Landes: DesignKARL - A Language for the Design of Knowledge-Based Systems. In *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering SEKE'94*, Jurmala, Latvia, Juni 1994.

[Lan95] D. Landes: *Die Entwurfsphase in MIKE - Methode und Beschreibungssprache.* Dissertationen zur Künstlichen Intelligenz, Band 84, infix, St. Augustin, 1995.

[LaS95] D. Landes und R. Studer (1995): The Treatment of Non-Functional Requirements in MIKE. In W. Schäfer, P. Botella (eds.): *Proceedings of the 5th European Software Engineering Conference (ESEC´95),* Sitges, Spain, September 1995), Springer, Berlin, Lecture Notes in Computer Science 989, 1995.

[LeG90] D.B. Lenat und R.V. Guha (1990): *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley Publ. Co., Inc., Reading, Massachusetts, 1990.

[Mar88] S. Marcus (Ed.): *Automating Knowledge Acquisition for Experts Systems*. Kluwer Academic Publisher, Boston, 1988.

[McD88] J. McDermott: *Preliminary Steps toward a Taxonomy of Problem-solving Methods*. In [Mar88] 225-255.

[Mus89] M. A. Musen: *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Morgan Kaufmann Publisher, San Mateo, Kalifornien, 1989.

[Mus92] M. A. Musen: Overcoming the limitations of role-limiting methods. *Knowledge Acquisition*, Vol. 4, Nr. 2, Juni 1992, 165-170.

[Neb96] B. Nebel: Artificial Intelligence: A Computational Perspective. In G.

publications, Studies in Logic, Language and Information, Stanford, 1996.

[Neu93] S. Neubert: Model Construction in MIKE (Model-Based and Incremental Knowledge Engineering). In N. Aussenac et al. (eds.), *Knowledge Acquisition for Knowledge-Based Systems, Proceedings of the 7th European Workshop (EKAW´93),* Toulouse, Frankreich, 1993, LNAI 723, Springer-Verlag, 1993.

[Neu94] S. Neubert: *Modellkonstruktion in MIKE - Methoden und Werkzeuge.* Dissertationen zur Künstlichen Intelligenz, Band 60, infix, St. Augustin, 1994.

[New82] A. Newell: The Knowledge Level, *Artificial Intelligence*, 18:87-127, 1982.

[Par86] D. Partridge: *Artificial Intelligence. Application In The Future Of Software Engineering.* Ellis Horwood Limited, Großbritanien, 1986.

[PiS97] Th. Pirlein und R. Studer (1997): Integrating the Reuse of Commonsense Ontologies and Problem-Solving Methods, Universität Karlsruhe, Institut AIFB, Forschungsbericht 354, Mai 1997.

[PoG93] K. Poeck und U. Gappa: Making Role-Limiting Shells More Flexible. In: N. Aussenac et al. (eds): *Knowledge Acquisition for Knowledge-Based Systems, Proc. 7th European Knowledge Acquisition Workshop (EKAW`´93) Toulouse*, September 1993, Lecture Notes in Artificial Intelligence 723, Springer, 1993.

[PoP92] K. Poeck und F. Puppe: COKE: Efficient solving of complex assignment problems with the propose-and-exchange method. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence,* Arlington, Virginia, 10.-13. November 1992.

[Pup93] F. Puppe: *Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods.* Springer, 1993.

[SWA94] A. TH. Schreiber, B. Wielinga, J. M. Akkermans, W. Van De Velde und R. de Hoog (1994): CommonKADS. A Comprehensive Methodology for KBS Development, *IEEE Expert*, 9(6), 1994, 28—37.

[ShG92] M.L.G. Shaw und B.R. Gaines: The Synthesis of Knowledge Engineering and Software Engineering. In: P. Loucopoulos (ed.): *Advanced Information Systems Engineering*, LNCS, 593, Springer, 1992, 208-220.

[HSW97] G. van Heijst, A. T. Schreiber und B. J. Wielinga (1997): Using Explicit Ontologies in Knowledge-Based System Development, *International Journal of Human-Computer Interaction (IJHCI),* 46(6), 1997.

Posner (ed.): *Foundations of Cognitive Science,* The MIT Press, Cambridge, 1989.

[WiB84] B. J. Wielinga und J. A. Breuker (1984): Interpretation Models for Knowledge Acquisition. In *Proceedings of the European Conference on AI (ECAI-84),* Pisa, 1984.