Ein konfiguratives Metamodellierungswerkzeug

Patrick Delfmann, Sebastian Herwig, Milan Karow, Łukasz Lis

European Research Center for Information Systems
Westfälische Wilhelms-Universität Münster
Leonardo-Campus 3
48149 Münster

[vorname.nachname]@ercis.uni-muenster.de

Abstract: Im vorliegenden Beitrag wird ein Metamodellierungswerkzeug vorgestellt, das in der Literatur vielfach geforderte Funktionalitäten aus den Bereichen des Method Engineering sowie des Modell-Variantenmanagements umsetzt. Aufbauend auf konzeptionellen Überlegungen wird ein Werkzeug konstruiert, das die flexible Spezifikation von beliebigen, untereinander integrierten Modellierungssprachen ermöglicht. Die Anwendung der Sprachen im Rahmen der Modellerstellung wird durch ein redundanzfreies Variantenmanagement auf Sprach- und Modellebene flexibilisiert. Der vorliegende Beitrag beschreibt die Herleitung des Fachkonzepts, sowie die Architektur der entwickelten Software und zeigt deren Funktionsweise exemplarisch.

1 Motivation

Die Informationsmodellierung ist ein essentielles Werkzeug in Informationssystementwicklungs- und Organisationsprojekten [KK84], [Ka88], [DB95], [Da04]. In Organisationsprojekten dienen Modelle als strukturiertes Kommunikationsmittel zwischen den Stakeholdern. Mit Hilfe konkreter Sprachen werden dabei organisationale Informationen wie Geschäftsprozesse, die Aufbauorganisation oder Ressourcen beschrieben. Diese Modelle dienen einerseits als Teil der Anforderungsdokumentation für Softwareentwicklungs- oder Softwareauswahlprozesse, können andererseits jedoch auch (zunächst) unabhängig von IT-Artefakten zur Prozessbeschreibung und -reorganisation von Organisationen in Wirtschaft und Verwaltung eingesetzt werden. In der Gestalt von Referenzmodellen können solche Artefakte über Projektgrenzen hinaus wiederverwendet werden [BDK06]. Beispiele für fachkonzeptuell eingesetzte Modellierungssprachen bzw. Sprachsysteme sind u. a. ARIS [Sc00], BPMN [Ob06], SOM [FS93], und MEMO [Fr99]

In der Softwareentwicklung sind Informationsmodelle ein integraler Bestandteil der Anforderungsbeschreibung und der Entwurfsdokumentation, bzw. nehmen im Rahmen eines modellgetriebenen Systementwicklungsprozesses (z.B. als Umsetzung der Model Driven Architecture, vgl. [Ob03]) die Rolle des zentralen Entwicklungsartefaktes ein. Dabei werden Softwarebestandteile auf unterschiedlichen Abstraktionsebenen durch

Modelle beschrieben und über automatische Transformationsverfahren in spezifischere Modelle oder in Programmcode übersetzt. Im Softwarekontext werden insbesondere die Diagrammtypen der UML eingesetzt [Ob04].

Die o.g. Einsatzszenarien stellen je nach Fokus und Modellierungszweck sehr unterschiedliche Anforderungen an die einzusetzenden Beschreibungsmittel. So ist bspw. im Rahmen eines frühen Fachkonzeptes eine einfache, intuitive Verständlichkeit der Modelle für die Kommunikation mit Fachanwendern und Kunden notwendig. Bei Modellen zur automatischen Quellcodeerzeugung tritt diese Anforderung jedoch vor der Notwendigkeit formaler Präzision in den Hintergrund.

Neben der Einordnung der Modelle in den Entwicklungsprozess spielen Einflussfaktoren wie die fachliche Domäne, Projektziele, eingesetzte Technologien und Produkte, Entwicklungsparadigmen, Wiederverwendung, sowie auch Vorlieben und Expertise der Projektmitarbeiter eine wichtige Rolle bei der Wahl der Beschreibungsmittel. In Gestalt des Method Engineering [Br96], [Ro97] hat sich in diesem Zusammenhang ein Forschungsfeld eröffnet, in dessen Rahmen die Konstruktion spezifischer Modellierungssprachen und Sprachsysteme für individuelle Anwendungen propagiert wird. Arbeiten aus dem Bereich des Modellvariantenmanagements [SGD03], [BDK04], [De06], [DK07], [RA07] betonen darüber hinaus die Notwendigkeit der redundanzfreien Verwaltung von Sprach- und Modellvarianten für unterschiedliche Anwendungsszenarien. Als Beispiele können hier zielgruppenadäquate Perspektiven auf Organisationsmodelle oder plattformabhängige Modellvarianten für Entwurfsmodelle im Rahmen der modellgetriebenen Softwareentwicklung genannt werden. Empirische Studien haben gezeigt, dass in Wissenschaft und Praxis Bedarf nach entsprechenden methodischen Unterstützungen besteht, eine adäquate Werkzeugunterstützung aber bisher fehlt [DK07].

Ziel unserer Forschungsarbeit ist es, diese methodisch-technische Lücke zu schließen und ein integriertes Werkzeug zur Unterstützung des Method Engineering und des Modellvariantenmanagements zu entwickeln. Als Ergebnis dieser Arbeit stellen wir im vorliegenden Beitrag das Modellierungswerkzeug [εm] vor, welches die individuelle Spezifikation von Modellierungssprachen, deren Nutzung zur Erstellung von Modellen, sowie das Variantenmanagement auf Ebene von Sprachen und Modellen ermöglicht. Dazu werden zunächst relevante Vorarbeiten vorgestellt und die angewendete Forschungsmethode erläutert (Abschnitt 2). Nach der konzeptionellen Spezifikation des Modellierungswerkzeugs (Abschnitt 3) wird dessen Architektur und Funktionsweise (Abschnitt 4) erläutert. Abschließend werden aus den Ergebnissen weitere Forschungsperspektiven abgeleitet (Abschnitt 5).

2 Eigene und verwandte Vorarbeiten; Forschungsmethode

2.1 Eigene und verwandte Vorarbeiten

Die Konstruktion des Werkzeugs [εm] baut auf Ergebnissen umfassender Vorarbeiten auf und repräsentiert die technische Umsetzung der *konfigurativen Referenzmodellierung*

– einer methodischen Konzeption zur redundanzfreien Verwaltung von Sprach- und Modellvarianten. Diese erlaubt die automatisierte Anpassung von konzeptionellen Modellen an spezifische Anwendungskontexte und nutzt das Prinzip der Modellprojektion. Für spezifische Anwendungen nicht relevante Modellelemente werden ausgeblendet. Ein solches Verfahren ermöglicht es, die konfigurative Referenzmodellierung auf beliebige konzeptionelle Modellierungssprachen anzuwenden [BDK04], [De06].

Verwandte Arbeiten, z. B. die Arbeit von SOFFER, GOLANY und DORI, konstruieren anpassbare Modelle einer speziellen Sprache – sogenannte Object-Process Diagrams – mit dem Ziel, ein modellbasiertes Customizing von ERP-Systemen zu ermöglichen. Dabei wird der jeweilige Anwendungskontext anhand von Attributen spezifiziert. Im Anschluss werden die Modelle abhängig von den Attributausprägungen angepasst [SGD03]. ROSEMANN und VAN DER AALST unterstützen in ihrer Arbeit zur Erweiterung Ereignisgesteuerter Prozessketten (EPK) [KNS92] die Spezifikation von Regeln, mit welchen sich Abhängigkeiten zwischen Modellelementen explizieren lassen. Trifft ein Modellersteller z. B. die Entscheidung, bestimmte Funktionen aus einem Modell zu entfernen, leitet ihn die Regelbasis dabei an, auch von diesen Funktionen abhängige Modellbereiche korrekt anzupassen. Ebenso wird die kontextspezifische Anpassung der Modelle ähnlich wie bei [SGD03] methodisch unterstützt [RA07]. Weiterentwicklungen des Ansatzes, die eine Toolunterstützung adressieren sowie die Spezifikation der Anwendungskontexte verfeinern – liefert bspw. [Ro07]. Im Rahmen der hier vorzustellenden Werkzeugentwicklung wird der Ansatz der konfigurativen Referenzmodellierung bevorzugt, da er nicht an eine konkrete Modellierungssprache gebunden ist.

Softwarewerkzeuge, die grundsätzlich für die Spezifikation von individuellen Modellierungssprachen geeignet erscheinen und gleichzeitig Skriptsprachen zur eventuellen Erweiterung durch ein Sprach- und Modellvariantenmanagement bereitstellen, liegen als Metamodellierungswerkzeuge vor (Eine Übersicht gängiger Werkzeuge findet sich bei [DK07]). Empirische Studien haben jedoch gezeigt, dass entweder die Skriptsprachenfunktionalität zur Umsetzung eines Variantenmanagements nicht ausreicht oder die Ergonomie der Benutzerschnittstelle einen praktischen Einsatz nicht vorsieht [DK07]. Aus diesem Grund wird das zu erstellende Werkzeug als Eigenentwicklung realisiert.

2.2 Forschungsmethode

Sowohl die eigenen Vorarbeiten als auch der vorliegende Beitrag folgen dem Forschungsansatz der Design Science [He04], der im Kern die Konstruktion von *wissenschaftlichen Artefakten*, wie z. B. Methoden, Sprachen und Softwarewerkzeugen zum Inhalt hat. Im Rahmen eines Design Science-Ansatzes ist sicherzustellen, dass die durchgeführten Forschungsarbeiten ein relevantes Thema behandeln. Diese Relevanz ist zu belegen. Darüber hinaus ist sicherzustellen, dass die konstruierten Forschungsartefakte einen *innovativen Beitrag* zur bereits in der Disziplin vorhandenen Wissensbasis darstellen, d. h., dass vergleichbare Lösungen nicht bereits existieren. Die konstruierten Artefakte sind im Rahmen eines Design Science-Ansatzes zu *evaluieren*, um sicherzustellen, dass die gesetzten Forschungsziele tatsächlich erreicht werden. Demgemäß sind die Vorarbeiten insbesondere in den Bereichen der Identifikation der *Relevanz* des The-

mas [DK07], [De06], dem Beleg der *Innovationskraft* [De06] sowie der *Konstruktion von wissenschaftlichen Artefakten* [BDK04], [De06] zuzuordnen. Dem gegenüber ist die Implementierung von [εm] neben ihrer Eigenschaft als *Artefaktkonstruktion* auch der praktischen *Evaluation* des perspektivenbasierten Variantenmanagements zuzuschreiben.

3. [εm]-Fachkonzept

Die konzeptionelle Grundlage von Modellierungswerkzeugen ist in der Regel ein umfassendes Datenmodell, welches die Elemente und Beziehungen der instanziierbaren Modellelemente enthält. Bei Modellierungswerkzeugen, welche dediziert eine vorgegebene Modellierungssprache oder eine definierte Reihe von Modellierungssprachen unterstützen, kann die Typ- bzw. Sprachebene als Datenschema fest implementiert werden. Das Daten- bzw. Klassenmodel entspricht in diesem Fall also konzeptionell genau dem Metamodell der Sprache(n). Beispiele einer solchen Umsetzung sind Instanzen des Java Metadata Interface (JMI), welches Java-Klassenstrukturen für MOF-Modelle bereitstellt und insbesondere im UML-Kontext eingesetzt wird [Su02].

Im Falle von [εm] werden Sprachdefinitionen erst zur Laufzeit im Werkzeug selbst festgelegt, so dass auch die Elemente der Sprachebene wiederum Instanzen einer übergeordneten Typebene sind, welche im Datenmodell abzubilden ist. Die Mechanismen der konfigurativen Referenzmodellierung spezifizieren durch ihr Drei-Ebenen-Metamodellsystem [BDK04], [De06] eine weitere Abstraktionsebene. Die direkte Implementierung dieses Systems ist wenig praktikabel, weshalb es für die Konstruktion eines Softwarewerkzeugs in einem einzigen Datenmodell zu konsolidieren ist. Die entsprechenden Mechanismen des Variantenmanagements können somit direkt implementiert werden. Abbildung 1 zeigt schematisch die Gegenüberstellung des [εm]-Datenmodells und der Struktur der Sprach- und Modelldaten zur Laufzeit des Werkzeugs.

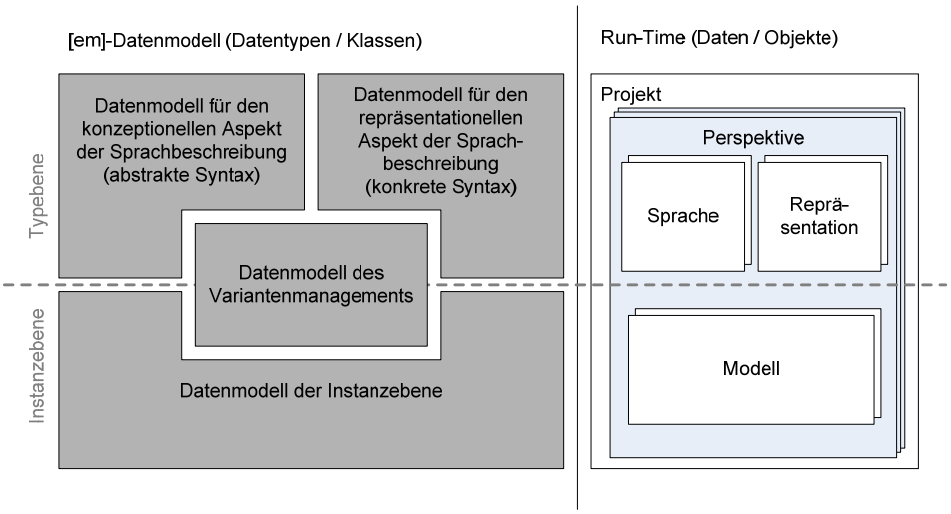

Abbildung 1: Typ-Instanz-Strukturen in [εm]

Das [εm]-Datenmodell repräsentiert die Verwaltung sämtlicher Modellierungssprachen- bzw. modellbezogenen Daten sowie deren Zuordnung zu etwaigen Varianten. Ein Ziel der Spezifikation ist dabei die möglichst breite Einsetzbarkeit bei der Definition nutzerindividueller Modellierungssprachen. Hierfür ist eine genaue Untersuchung der syntaktischen Dimension sowohl auf konzeptioneller (auch: abstrakte Syntax) als auch auf repräsentationeller Ebene (konkrete Syntax) notwendig. Zur Darstellung der Teilaspekte des Datenmodells wird im Folgenden die (min,max)-Notation des Entity-Relationship-Modells (ERM) nach SCHLAGETER und STUCKY verwendet [SS83].

3.1 Konzeptioneller Aspekt

Sprachen sind nach ihrer semiotischen Definition *Systeme von Zeichen*, wobei das Zeichen in eine Inhaltsebene (das Bezeichnete) und eine Ausdrucksebene (Zeichenträger) zerlegt werden kann [Ec91]. Die Beziehung zwischen diesen beiden Ebenen wird als *semantische Dimension* der Semiotik untersucht [Mo88]. Die Einbeziehung der *Intention* einer sprachlichen Handlung durch einen Zeichennutzer kennzeichnet die *pragmatische Dimension* der Semiotik. Die Beziehungen der Zeichenträger untereinander werden durch die *syntaktische Dimension* der Semiotik beschrieben.

Für die Konzeption eines Modellierungswerkzeuges zur Spezifikation von Modellierungssprachen und ihrer Nutzung sind die Aspekte der *Semantik* und *Pragmatik* im semiotischen Sinne irrelevant, da die *Interpretation* der Modellelemente (Zeichen) nicht innerhalb des formalen Systems der Software stattfindet. Modellierungstools sind entsprechend auf die *syntaktische Dimension* von Sprachhandlungen beschränkt. Die *Syntax* einer Sprache lässt sich entsprechend minimal durch ihre Zeichenträger und die Beziehung zwischen diesen beschreiben. Im Rahmen von Modellierungssprachen sprechen wir hier von *Elementtypen* und *Beziehungen*. Abbildung 2 zeigt diesen generischen Ansatz.

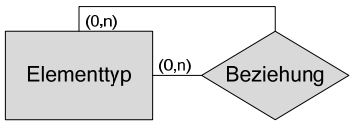

Abbildung 2: Generisches Datenmodell zur Sprachspezifikation

Im abgebildeten Datenmodell werden sämtliche zu instanziierenden Sprachelemente durch Elementtypen dargestellt. Das schließt neben Knoten bspw. Kanten, Attribute etc. ein. Diese generische Sichtweise stellt eine erhebliche Spezifikationskomplexität bereits für vergleichsweise einfache Sprachen dar, da durch die fehlende Spezialisierung sämtliche der Elementtypen über die Beziehungsstruktur zu interpretieren sind. Bei konzeptuellen Modellen im Sinne der Systementwicklung handelt es sich um *Graphen*, so dass *Knoten* und *Kanten* zentrale Sprachelemente darstellen. Abbildung 3 zeigt ein erweitertes Modell, welches die Definition von Kanten sowie deren Richtung und Kardinalität vereinfacht.

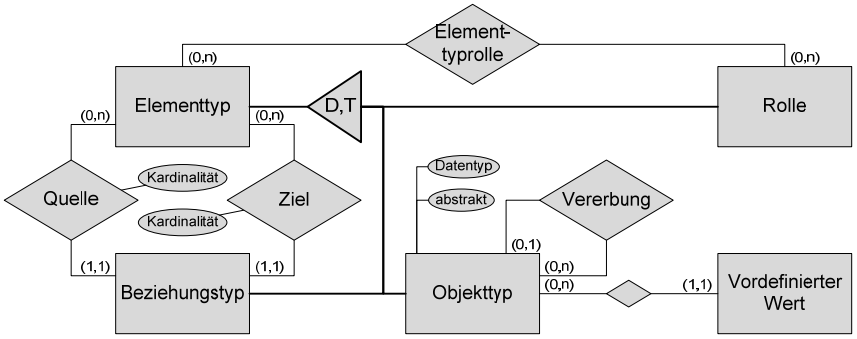

Abbildung 3: Erweitertes Datenmodell zur konzeptuellen Sprachspezifikation

Zentrales Element des Datenmodells ist weiterhin der *Elementtyp.* Hinzugefügt wird der *Beziehungstyp*, der die Spezifikation von zulässigen Beziehungen zwischen Elementen erlaubt, wobei durch *Quelle* und *Ziel* eine Richtung angegeben wird. Die typische Repräsentation einer Instanz des Beziehungstyps ist die *Kante*, jedoch sind Beziehungen nicht auf diese Darstellung beschränkt (vgl. Abschnitt 3.2) Durch die disjunkt-totale Spezialisierung des *Beziehungstyps* vom Elementtyp sind auch Beziehungen zwischen Beziehungen spezifizierbar (z. B. Kanten, welche auf Kanten zeigen). Beziehungen nach diesem Schema sind stets binär; n-äre Beziehungen müssen in mehrere Beziehungstypen zerlegt werden. *Objekttypen* repräsentieren in der Regel existenzunabhängige Elementtypen, die grafisch zumeist – jedoch nicht ausschließlich – als *Knoten* repräsentiert werden.

Objekttypen können ihre Eigenschaften und Beziehungen *vererben* – auf diese Weise ist die Spezifikation *abstrakter* Objekttypen möglich, die eine effiziente Eingabe gemeinsamer Eigenschaften der Subtypen erlaubt. Sämtliche Eigenschaften von Elementtypen, bspw. Attribute, werden wiederum als Objekttypen beschrieben und mit ihren Trägern über entsprechende Beziehungstypen verknüpft. Objekttypen können einen *Datentyp* besitzen, so dass sich als Instanzen Primitive (z. B. Strings, Zahlenwerte etc.) bilden lassen. Besitzt ein Objekttyp den Datentyp „Enumeration“, so werden ihm entsprechend *vordefinierte Werte* zugeordnet. Die *Rolle* ermöglicht die weitere Spezialisierung eines Elementtyps in Relation zu einer Beziehung, die dieser Typ eingeht. So kann bspw. ein Ereignis in einer EPK die Rollen „Vorbedingung“ und „Ergebnis“ annehmen.

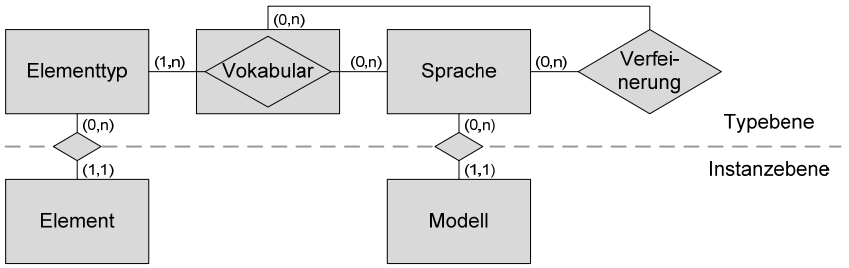

Abbildung 4: Sprachintegration, Modellhierarchien und Instanziierung

Der Aufbau einer methodenorientierten Systemarchitektur erfordert in der Regel die Integration verschiedener Modellierungssprachen (vgl. Abschnitt 1). Die Integration der Sprachen muss dabei über die Sprachspezifikation gewährleistet werden (vgl. Abbildung 4). Elementtypen, die in das *Vokabular* mehrerer Sprachen aufgenommen werden, dienen für diese Sprachen als Schnittstellen und ermöglichen so die horizontale Integration von Modellierungssichten. Zur vertikalen Integration werden *Verfeinerungen* eingesetzt. Auf Typebene wird dabei beschrieben, mit welcher Sprache die nähere Beschreibung der Instanzen eines Elementtyps erfolgen kann (z. B. EPKs für EPK-Funktionen). Elementtypen bzw. Sprachen als *Elemente* bzw. *Modelle* instanziiert. Jedes Modell (bspw. eine konkrete EPK) ist somit eindeutig einer Sprache zugeordnet. Analog gehört jedes Element (bspw. ein konkretes EPK-Ereignis) eindeutig einem Elementtyp an.

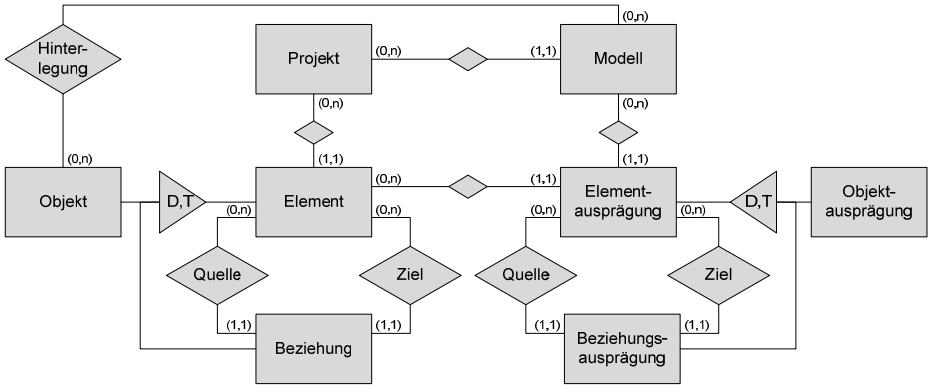

Abbildung 5: Instanzebene des [εm]-Datenmodells

Abbildung 5 spezifiziert die Instanzebene des [εm]-Datenmodells, auf der die Modelldaten verwaltet werden. Ein Modellierungsvorhaben wird als *Projekt* bezeichnet und fasst alle im Rahmen des Vorhabens erstellten *Modelle* zusammen. Strukturanalog zur Typebene werden *Elemente* in *Objekte* und *Beziehungen* spezialisiert und können untereinander Beziehungen eingehen. Objekte sind mithilfe von *Hinterlegungen* als Modelle verfeinerbar. Elemente sind im gesamten Projekt als Definition einmalig. Von diesen Elementen können *Ausprägungen* erzeugt werden, die Mehrfachverwendungen von gleichen Elementen in Modellen repräsentieren. Jede Elementausprägung ist eindeutig einem Element und eindeutig dem Modell, in dem sie verwendet wird, zugeordnet.

3.2 Repräsentationeller Aspekt

Die Zuordnung der konzeptionellen Typen zu ihren Darstellungsobjekten zeigt der Modellausschnitt in Abbildung 6. *Diagrammtypen* fassen unterschiedliche Darstellungsformen für Sprachen zusammen (für das ERM bspw. (min,max)- vs. Krähenfußnotation). Die Zuordnung der Elementtypen zu einem Diagrammtyp erfolgt über die *Elementtyprepräsentation*. Dem Elementtyp wird so in Abhängigkeit des Diagrammtyps eindeutig eine *Darstellung* zugewiesen. Analog wird einem Objekt, für das eine Verfeinerung angelegt ist, eine Markierung in Form eines *Verfeinerungssymbols* annotiert, das vom Objekt- und Diagrammtyp abhängig ist. Als Visualisierungskonzept für die Elementtyp-

rolle werden auf Darstellungsebene so genannte *Ankerpunkte* eingesetzt. Dabei handelt es sich um Koordinatenpositionen relativ zum grafischen Darstellungsobjekt des Elementtyps, an denen Kanten angeknüpft werden können. Ein Ankerpunkt repräsentiert dabei eine oder mehrere Rollen, die ein Element einnehmen kann. Für jede Rolle können beliebig viele Ankerpunkte definiert werden. Im o. g. EPK-Bespiel würden bspw. für den Objekttyp „Ereignis“ Ankerpunkte der Rolle „Vorbedingung“ an der Unterseite für ausgehende Kontrollflüsse und solche der Rolle „Ergebnis“ an der Oberseite für eingehende Kontrollflüsse angeordnet.

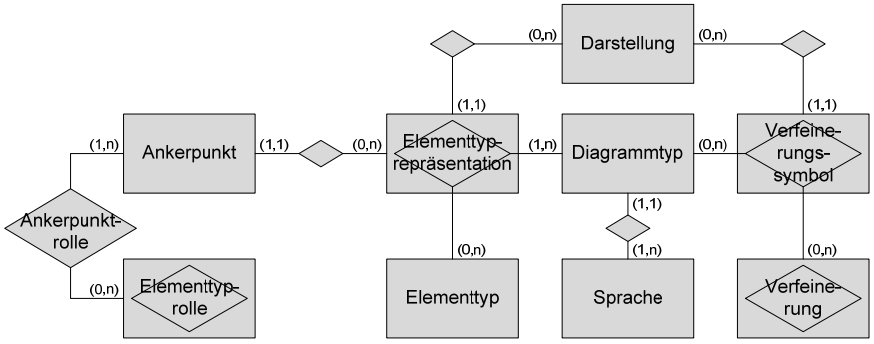

Abbildung 6: Repräsentation konzeptioneller Sprachelemente

Das Konzept der *Darstellung* ist zentral für den repräsentationellen Aspekt der Modellierungssprache. *Objekttypen* werden – sofern ihnen eine sichtbare Darstellung zugeordnet wird – in der Regel durch Knoten dargestellt. Für die Visualisierung von *Beziehungstypen* lassen sich drei verschiedene Muster identifizieren (vgl. Abbildung 7):

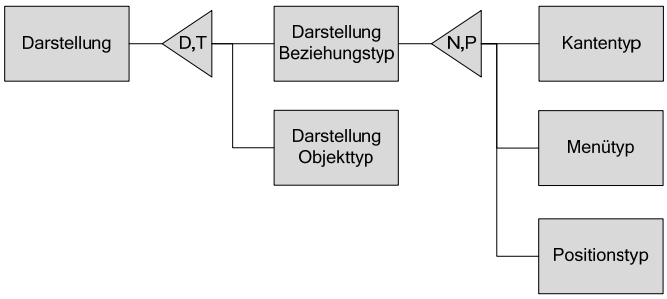

Abbildung 7: Darstellungsarten

- *Kantentyp*: linienartige, grafische Verbinder, die zwei Knoten visuell in Beziehung setzen

- *Positionstyp*: Elemente werden durch die räumliche Anordnung zueinander in Beziehung gesetzt; häufigstes Muster ist dabei die Inklusion (z. B. Subprozesse in BPMN); ein weiteres Muster ist die unmittelbare Verbindung zweier (oder mehrerer) Knoten durch „Aneinanderkleben“.

- *Menütyp*: Das in Bezug genommene Element wird nicht auf dem Zeichenblatt, sondern ausschließlich über ein gesondertes Eigenschaftsfenster oder Kontextmenü des Modellierungswerkzeugs sichtbar. Der Menütyp wird bspw. für Attributierungsbeziehungen eingesetzt.

3.3 Modellvariantenmanagement durch Perspektivenbildung

Die Spezifikation und Umsetzung von Perspektiven ist die konzeptionelle Grundlage für das Sprach- und Modellvariantenmanagement. Perspektiven sind anwendungsspezifische Sichten auf ein Modell, die als Projektion dieses Modells erzeugt werden und nach ihrer Erzeugung eine spezifische Modellvariante darstellen. Abbildung 8 zeigt die Beziehungen, die die Modellierungselemente Perspektiven zuweisen. Jede dieser Beziehungen repräsentiert die datenmäßige Implementierung eines Konfigurationsmechanismus der konfigurativen Referenzmodellierung [BDK04], [De06].

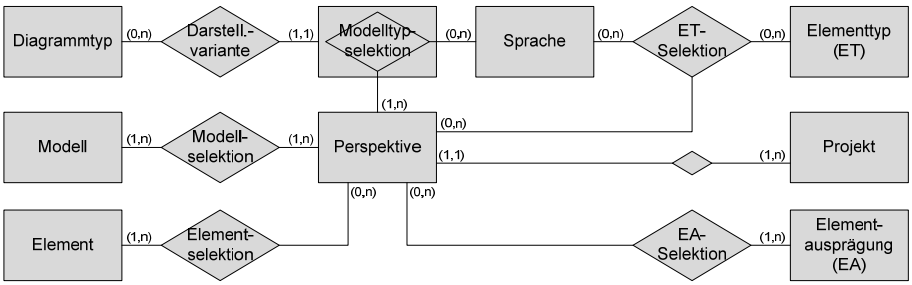

Abbildung 8:Perspektivenmanagement

Jedes Projekt besitzt initial mindestens eine Perspektive. Diese so genannte Standardperspektive ist die Basis für die Generierung weiterer Varianten eines Modellierungsprojekts. Innerhalb der (zusätzlichen) Perspektiven können Modelle modifiziert und erweitert werden. Dabei stehen unterschiedliche Selektions- und Variationsformen zur Verfügung, die determinieren, welche Sprachen mit welchen Elementtypen und Darstellungsformen, welche Modelle, welche Elemente und welche Elementausprägungen in welchen Perspektiven zur Ansicht und Modellierung zur Verfügung stehen. Konkret werden dem Fachkonzept entsprechend die folgenden Konfigurationsmechanismen unterstützt (vgl. zu Konfigurationsmechanismen ausführlich [BDK04], [De06]):

- Mithilfe der *Modelltypselektion* werden die zur Verfügung stehenden Modellierungssprachen je nach Perspektive eingeschränkt. Dies hat zur Folge, dass alle Modelle, die mit einer für die jeweilige Perspektive deselektierte Modellierungssprache erstellt wurden, ausgeblendet werden. Des Weiteren steht die deselektierte Modellierungssprache auch für die Konstruktion neuer Modelle in der entsprechenden Perspektive nicht mehr zur Verfügung.

- *Elementtypselektionen* erlauben die perspektivenabhängige Ein- oder Ausblendung von Sprachbestandteilen. Eine Sprache kann also mithilfe der Elementtypselektion perspektivenabhängig reduziert werden. Als Beispiel sei die Beschneidung der EPK

um organisatorische Objekte genannt, die an Funktionen annotiert werden. Folge dieser Ausblendung ist analog zur Modelltypselektion, dass einerseits alle Modelle in der betrachteten Perspektive um Elemente reduziert werden, die den ausgeblendeten Elementtypen angehören. Andererseits steht die reduzierte Sprache auch zur Modellierung in der Perspektive nunmehr reduziert zur Verfügung.

- Die *Darstellungsvariation* erlaubt die perspektivenspezifische Modifikation des repräsentationellen Aspekts von Modellierungssprachen. Für die Repräsentationsformen der Objekte und Beziehungen – meist Kanten und Symbole – können Varianten angelegt und perspektivenspezifisch ausgewählt werden. Mit der Darstellungsvariation werden insbesondere individuelle Präferenzen der Modellierer adressiert, die sich bspw. aus methodischer oder fachlicher Ausrichtung ergeben können.

- Die Verwaltung von inhaltlichen Modellvarianten wird durch die *Elementselektion* vorgenommen. Einzelne Modellelemente werden als zu einer Perspektive zugehörig oder nicht zugehörig deklariert und werden folglich angezeigt oder ausgeblendet. Die Elementselektion wird dann verwendet, wenn unabhängig von den Ausprägungen des Elements darüber entschieden werden kann, ob es für die Perspektive relevant ist oder nicht.

- Die *Elementausprägungsselektion* kommt zur Anwendung, wenn die Entscheidung, ob ein Element relevant ist oder nicht, von seiner konkreten Ausprägung im Modell abhängig ist. So kann das gleiche Element (bspw. ein Entitytyp „Kunde“) für die betrachtete Perspektive ggf. in einem Modell relevant sein, in einem anderen aber nicht.

4. [εm]-Architektur und -Funktionsweise

4.1. [εm]-Softwarearchitektur

Die bereits dargestellten fachkonzeptionellen Anforderungen wurden mit [εm] in Form eines Softwarewerkzeugs implementiert. Dabei wurde insbesondere die gewünschte Praxistauglichkeit sichergestellt, indem auf Benutzerfreundlichkeit bei gleichzeitig hoher Performance geachtet wurde. Die Software basiert auf dem Microsoft .NET-Framework in der aktuellen 3.0-Version und wurde in C# implementiert. Um eine räumlich und zeitlich verteilte Modellierung zu ermöglichen wurde eine mehrschichtige Client/Server-Architektur mit der klassischen Schichtenaufteilung in Datenhaltung, Anwendungsslogik und Präsentation zugrunde gelegt. Diese ist um eine vierte Synchronisierungsschicht ergänzt, die für die Client/Server-Kommunikation zuständig ist. Aus Performancegründen ist die Anwendungslogik größtenteils Client-seitig verortet. Teile der eigentlichen Anwendungslogik, die für die Benutzerverwaltung und das Rechtemanagement zuständig sind, sind aus Gründen der Sicherheit Server-seitig positioniert (Abbildung 9).

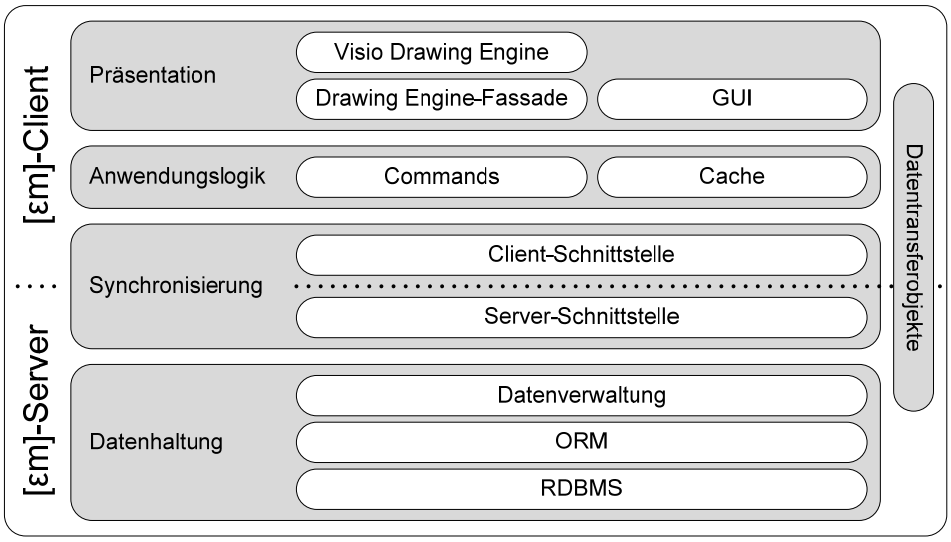

Abbildung 9: [εm]-Softwarearchitektur

Die [εm]-Architektur folgt grundsätzlich dem Prinzip einer linearen Schichtenordnung, in der von einer Schicht lediglich auf die nächstniedrigere zugegriffen werden kann [Ba00]. Eine Ausnahme hiervon bilden die Datentransferobjekte (DTO), die als Datenaustauschmedium im Rahmen des gesamten Werkzeugs fungieren. Um ihren inneren Aufbau zu kapseln und eine Abstraktion der DTO von den restlichen Komponenten zu erreichen, erfolgen die DTO-Zugriffe ausschließlich über Schnittstellen, die in den jeweiligen Schichten definiert werden. Solange die DTO diese Interfaces implementieren, kann ihre innere Funktionsweise verborgen bleiben, wodurch die Schichtentrennung im Wesentlichen nicht verletzt wird.

Die *Datenhaltungsschicht* basiert auf einem relationalen Datenbankmanagementsystem (RDBMS), welches über das *NHibernate*-Framework angesprochen wird. *NHibernate* sorgt hierbei für das objektrelationale Mapping (ORM), was für zusätzliche Flexibilität bei der Auswahl von RDBMS sorgt und den Implementierungsaufwand verringert. Das zugrunde liegende Datenmodell entspricht dabei dem oben dargestellten Fachkonzept. Die über das ORM übermittelten Daten werden innerhalb der Datenverwaltungskomponente auf die DTO übertragen und gelangen über die *Synchronisierungsschicht* auf den Client. Da die Datenhaltungsschicht für die Benutzerverwaltung und das Rechtemanagement zuständig ist, werden an den Client nur solche Daten übermittelt, für die der aktuell angemeldete Benutzer eine Berechtigung besitzt. Um Performanceeinbußen zu vermeiden, wird bei der Synchronisierung das *Lazy-Loading-Entwurfsmuster* eingesetzt, welches Daten erst bei einem tatsächlichen Bedarf lädt [Fo02]. In der Synchronisierungsschicht wird die *Windows Communication Foundation (WCF)* eingesetzt, wodurch die gesamte Client/Server-Kommunikation über eine einheitliche Service-orientierte Schnittstelle erfolgen kann.

In der Client-seitigen *Anwendungslogikschicht* wird das *Command-Entwurfsmuster* eingesetzt, infolge dessen alle einzelnen Modellierungsaktivitäten des Benutzers, wie

z. B. das Anlegen eines neuen Modellelements, in sogenannten Command-Objekten in Form einer Warteschlange zwischengespeichert werden. Dies erlaubt eine zeitliche Trennung der Spezifikation, Registrierung und Ausführung der Modelländerungen [Ga95]. Die Command-Warteschlange stellt die vollständige Modellierungshistorie dar und ermöglicht es, zu einem früheren Zustand des Modells zurückzukehren bzw. rückgängig gemachte Schritte wiederherzustellen. Die Benutzeraktivitäten werden dabei solange zwischengespeichert bis der Benutzer mit dem erreichten Modellzustand zufrieden ist und sich für das Speichern der Änderungen entscheidet. In diesem Fall wird die Command-Warteschlange zu einer Transaktion zusammengefasst, über die Synchronisierungsschicht an den Server übermittelt und dort atomar persistiert. Somit werden im RDBMS nur gültige, konsistente Modelle gespeichert.

Die *Präsentationsschicht* stellt eine mittels der *Windows Presentation Foundation (WPF)* implementierte Benutzeroberfläche (GUI) zur Verfügung, in die eine Drawing-Engine eingebettet ist. Durch die Verwendung von WPF wurde eine moderne, benutzerfreundliche und performante GUI geschaffen, die für einen praktischen Einsatz von [εm] einen kritischen Erfolgsfaktor darstellt. Die *Drawing-Engine* ist für die grafische Darstellung der Modelle während des Modellierens zuständig und wird nach dem Fassaden-Entwurfsmuster über eine eigenständig entwickelte Schnittstelle angesprochen. Somit ist die innere Funktionsweise von der eigentlichen Anwendung gekapselt, was einen eventuellen Austausch der Drawing-Engine erleichtert [Ga95]. Aktuell wird in [εm] die Zeichenkomponente des Visualisierungswerkzeugs *Microsoft Visio* als Drawing-Engine verwendet. Durch die weite Verbreitung dieser Software in der wirtschaftlichen Praxis und den hohen Bekanntheitsgrad bei den Endanwendern wird die Einarbeitungszeit für die Modellierer gesenkt und die Bedienfreundlichkeit des Werkzeugs erhöht. Dabei ist Microsoft Visio in die eigentliche GUI nahtlos integriert, so dass der Eindruck einer einheitlichen Modellierungsoberfläche gegeben wird.

4.2. [εm]-Funktionsweise

Der Bedienung von [εm] liegt ein dialoggesteuertes Konzept zu Grunde, welches sich in Anlehnung an das in Abschnitt 3 dargelegte Datenmodell in die folgenden Nutzerumgebungen gliedert: Die *Navigationsumgebung* stellt das zentrale Bedienelement zur Verwaltung von Projekten sowie zum perspektivenspezifischen Erzeugen und Anzeigen von Modellen dar (vgl. Abbildung 10). Modelle werden hierbei sowohl projekt- als auch perspektivenbezogen verwaltet. In Abhängigkeit eines Projekts und der ausgewählten Perspektive werden in [εm] nur die entsprechend zur Perspektive zugehörigen Modelle bereitgestellt. Die Modelle beinhalten hierbei ausschließlich die Modellelemente, die der entsprechenden Perspektive angehören. Ferner wird durch die perspektivenspezifische Selektion von Modellierungssprachen ebenfalls das Anlegen neuer Modelle auf die verfügbaren Sprachen bzw. Modelltypen eingeschränkt. Die Regelung des generellen Zugriffs auf Projekte, Perspektiven und Sprachdefinitionen erfolgt über eine Rechteverwaltung. Hierbei wird zwischen schreibenden und lesenden Zugriffsrestriktionen unterschieden.

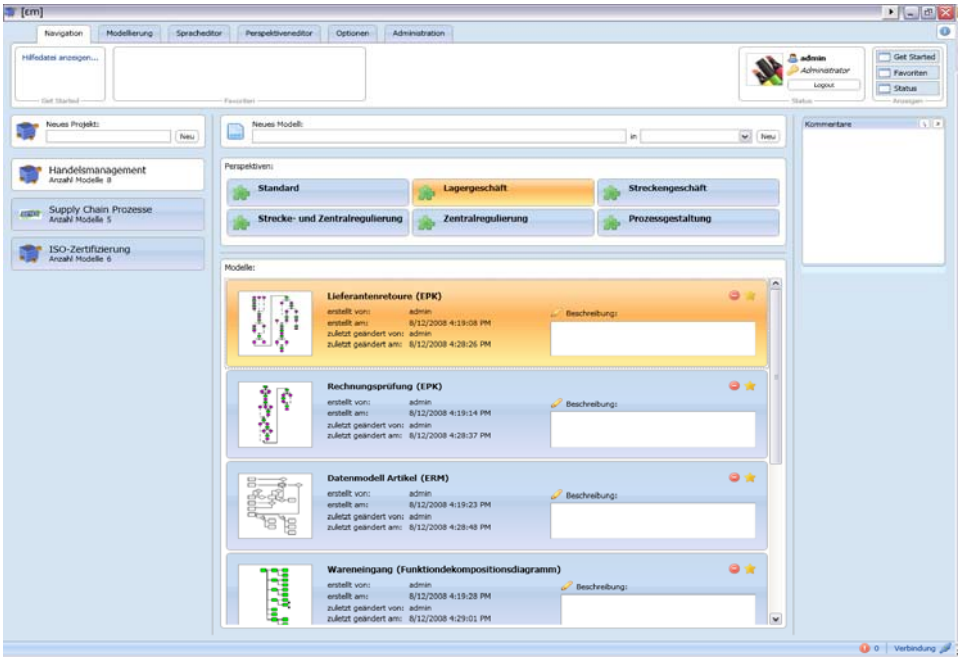

Abbildung 10: Projekt- und Modellverwaltung in [εm]

Mit dem Spracheditor ist eine dialoggesteuerte Metamodellierungsumgebung gegeben, welche die Spezifikation von Modellierungssprachen ermöglicht (vgl. Abbildung 11). Die Spezifikation der konzeptionellen Bestandteile einer Sprache stellt den Ausgangspunkt der Sprachkonstruktion dar. Entsprechend der Typebene des Datenmodells sind dazu ein einem ersten Schritt die in einer Sprache zu verwendenden Objekttypen zu spezifizieren (1). Im Weiteren sind Rollen zu definieren und den jeweiligen Elementtypen (d. h. sowohl Objekt- als auch Beziehungstypen) zuzuordnen (2). Die Definition von erlaubten Beziehungen zwischen Elementtypen erfolgt in einem nächsten Schritt in Form vom Beziehungstypen (3). Zu diesem Zweck werden in [εm] die den Elementtypen zugeordneten Rollen untereinander in Beziehung gesetzt. Ferner sind für Beziehungstypen Kardinalitäten zu spezifizieren, die sie hinsichtlich der zu verbindenden Elemente einnehmen, und anzugeben ob sie gerichtet sind. Weiterhin sind Beziehungstypen einer Darstellungsart zuzuordnen (vgl. Abbildung 6). Neben konzeptionellen Bestandteilen gilt es ebenfalls den repräsentationellen Aspekt der Modellierungssprachen zu definierten und den jeweiligen Elementtypen zuzuweisen. (4) Allen Elementtypen, die der Darstellungsart Knoten- oder Kantentyp folgen, sind hierzu eine grafische Repräsentation zuzuordnen. Entsprechende Repräsentationssymbole werden mit jeweiligen Ankerpunkten in Visio erstellt und stehen nach einmaligem Import in einer zentralen Symbolbibliothek zur Verfügung. Abschließend sind die Ankerpunkte zugewiesener Symbole mit Rollen zu belegen (5). Dadurch wird vorgegeben, über welche Punkte Elemente mit anderen Elementen wie bspw. Kanten zu verknüpfen sind.

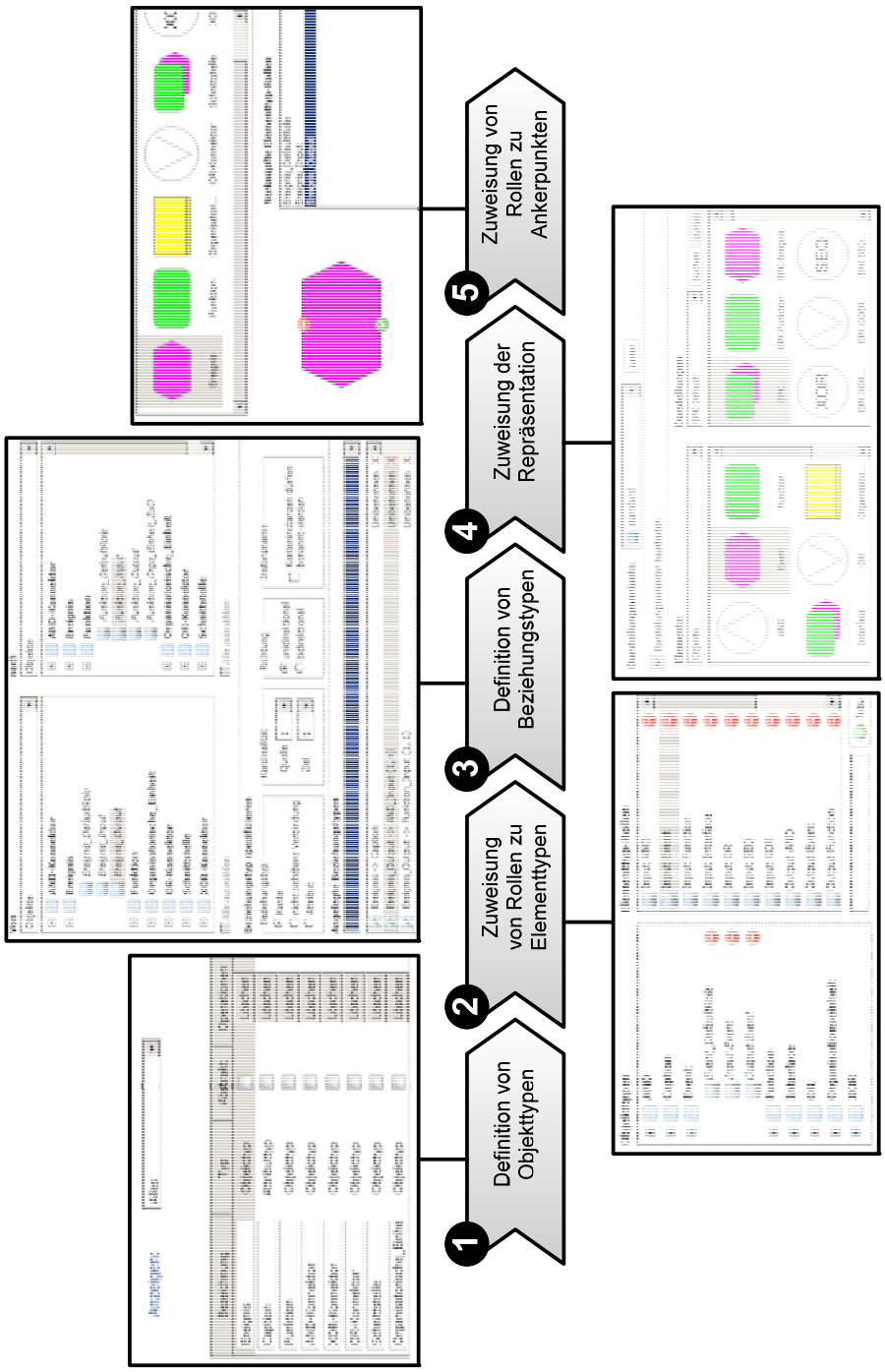

Abbildung 11: Spezifikation von Modellierungssprachen in [εm]

Das entsprechend Abschnitt 3.3 zu Grunde liegende Konzept des Variantenmanagements wird über den *Perspektiveneditor* umgesetzt. Dieser dient dem Erstellen und der Spezifikation von Perspektiven. Durch die Zuweisung von *Sprachen* und damit verbundenen *Elementtypen*, *Darstellungsvarianten*, *Modellen* sowie darin enthaltenen *Elementen* und *Elementausprägungen* zu Perspektiven, wird spezifiziert, welche Aspekte in Bezug auf konkrete Modelle in einer Perspektive dargestellt werden bzw. zur Modellierung zur Verfügung stehen (vgl. Abbildung 12). Hierbei werden einer Perspektive nicht zugeordnete Aspekte bei Auswahl dieser ausgeblendet. Ausgehend von der initialen Definition erfolgen weitergehende Modifikationen von Perspektiven ebenfalls über den Perspektiveneditor. Eine Ausnahme hiervon stellt die Selektion von Elementausprägungen dar. Dahingehende Anpassungen können ebenfalls über den Modelleditor direkt im Modell – d. h. auf dem Zeichenblatt – vorgenommen werden. Während der Modellierung kann – abhängig von den jeweiligen Rechten des Modellierers – beliebig zwischen unterschiedlichen Perspektiven hin und her geschaltet werden. Modifikationen an Modellen wirken sich nur auf die aktive Perspektive aus. Eine Ausnahme bildet die Standardperspektive. Änderungen in der Standardperspektive wirken sich auf sämtliche andere Perspektiven aus.

Abbildung 12: Der [εm]-Perspektiveneditor

Die Erstellung von Modellen – die eigentliche Modellierung – erfolgt über einen *Modelleditor*. Dieser stellt eine Modellierungsoberfläche zur Verfügung, wie sie in üblichen Modellierungswerkzeugen zu finden ist. Die Erstellung von Modellen im Modelleditor erfolgt grundsätzlich in einer Perspektive. Demgemäß haben Änderungen an einem Modell ausschließlich Konsequenzen für die jeweils selektierte Perspektive. Zur Modellierung werden abhängig von der jeweiligen Perspektive die entsprechend selektierten Objekt- und Beziehungstypen der zugrundeliegenden Sprache über eine Symbolleiste bereitgestellt und können auf dem Zeichenblatt ausgeprägt werden. Die Einhaltung der vorgegebenen Sprachsyntax wird hierbei direkt während der Modellierung überprüft und somit die Erstellung syntaktisch inkorrekter Modelle vermieden. Die perspektivenbezogene Modellierung im Modelleditor wird in Abbildung 13 anhand der Darstellung eines Modells in verschiedenen Perspektiven verdeutlicht.

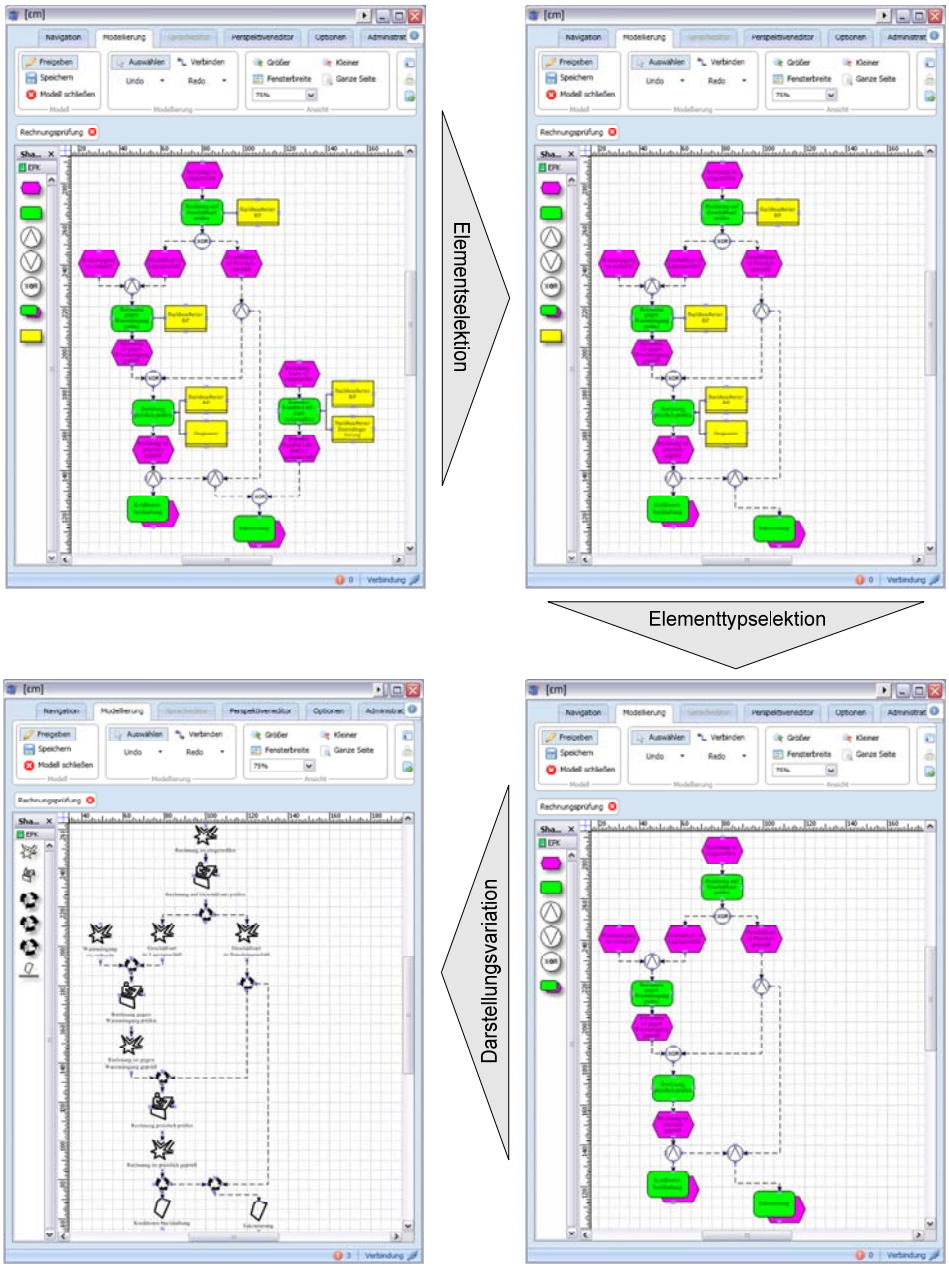

Abbildung 13: Perspektivenspezifische Modellierung in [εm]

5. Ergebnis und weiterer Forschungsbedarf

Die Integration einer flexiblen Spezifikation grafischer Modellierungssprachen mit Funktionen des Variantenmanagements eröffnet eine Vielzahl an Anwendungsmöglichkeiten im Kontext der Informations- und Anwendungssystementwicklung. Das vorgestellte Werkzeug [εm] stellt einen entsprechenden Prototypen zur Verfügung, der eine praktische Evaluation der methodischen Konzepte der Referenzmodellierung erlaubt und darüber hinaus die Entwicklung und den Einsatz neuer Beschreibungsmittel vereinfacht.

Die potenzielle Reichweite des Werkzeugs beschränkt sich dabei jedoch auf reine *Beschreibungsmodelle* – Modelle mit unterschiedlichen Zuständen und eigenem Verhalten, wie sie bspw. im Rahmen von *Simulationen* eingesetzt werden, sind zur Zeit nicht mit [εm] abbildbar. Die Ausdrucksmächtigkeit der Sprachspezifikation ist darüber hinaus auf eine lokale Syntax beschränkt, d.h. lediglich *unmittelbare* Beziehungen zwischen Elementtypen sind definierbar. Komplexere syntaktische Restriktionen bspw. über beliebige Pfadlängen sind mit dem vorgestellten Metamodellierungsansatz nicht möglich. Ein kurzfristiges Forschungsziel ist daher, eine entsprechende Constraint-Sprache in das Werkzeug zu integrieren.

Das Konzept des Variantenmanagements durch Projektion hat zur Folge, dass innerhalb perspektivisch angepassten Modellen syntaktische und semantische Inkonsistenzen auftreten können. Diese Inkonsistenzen sind bisher durch den Modellierer im Rahmen der Perspektivendefinition zu behandeln und können ggf. durch Hinzufügen perspektivenspezifischer Elemente wieder hergestellt werden. Eine weitestgehend automatisierte Unterstützung insbesondere der syntaktischen Modellkonsolidierung stellt daher ein aktuelles Forschungsgebiet dar mit hohem Einsatzpotenzial dar. Aktuelle Forschungsarbeiten fokussieren darüber hinaus auf die Unterstützung der fachkonzeptuellen Modellierung durch automatisiert durchgesetzte Bezeichnungskonventionen – hierbei kann über die Syntax der Modellierungssprachen hinaus die Syntax der natürlich-sprachlichen Ausdrücke analysiert und ggf. transformiert werden.

Nicht zuletzt ist das entwickelte Modellierungswerkzeug im Rahmen des gewählten Design-Science-Forschungsansatzes einer umfassenden Evaluation zu unterziehen, indem es sowohl in der Lehre als auch in Praxisprojekten eingesetzt wird. Neben der Benutzerfreundlichkeit, die die Ergonomie und die Verständlichkeit einschließt, ist insbesondere zu messen, in wie weit die Modellierungseffizienz durch die Möglichkeit der individuellen Sprachspezifikation und des Sprach- und Modellvariantenmanagements gesteigert werden kann. Hierzu bieten sich bspw. Vergleichsexperimente mit herkömmlichen Modellierungswerkzeugen an.

Literaturverzeichnis

[Ba00] Balzert, H.: Lehrbuch der Softwaretechnik, Softwareentwicklung, Heidelberg u. a., 2000.

[BDK04] Becker, J.; Delfmann, P.; Knackstedt, R.: Konstruktion von Referenzmodellierungssprachen. Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle. Wirtschaftsinformatik, 46 (4) 2004, S. 251-264.

[BDK06] Becker, J.; Delfmann, P.; Knackstedt, R.: Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In Proceedings of the Reference Modeling Conference (RefMod), Passau, 2006.

[Br96] Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools Information and Software Technology, 38 (4) 1996, S. 275-280.

[Da04] Dalal, N.P., et al.: Toward an integrated framework for modeling enterprise processes. Communications of the ACM, 47 (3) 2004, S. 83-87.

[DB95] Davenport, T.H.; Beers, M.: Managing information about processes. Journal of Management Information Systems, 12 (1) 1995, S. 57-80.

[De06] Delfmann, P.: Adaptive Referenzmodellierung. Methodische Konzepte zur Konstruktion und Anwendung wiederverwendungsorientierter Informationsmodelle. Logos, Berlin, 2006.

[DK07] Delfmann, P.; Knackstedt, R.: Konfiguration von Informationsmodellen - Untersuchungen zu Bedarf und Werkzeugunterstützung. In Proceedings of the 8. Internationale Tagung Wirtschaftsinformatik, Vol. 2 (Oberweis, A.; Weinhardt, C.; Gimpel, H.; Koschmider, A.; Pankratius, V.; Schnitzler, B., Eds.), Karlsruhe, 2007, S. 127-144.

[Ec91] Eco, U.: Semiotik: Entwurf einer Theorie der Zeichen. Fink, München, 1991.

[FS93] Ferstl, O.K.; Sinz, E.J.: Der Modellierungsansatz des Semantischen Objektmodells (SOM). Bamberger Beiträge zur Wirtschaftsinformatik, 1993.

[Fo02] Fowler, M.: Patterns of Enterprise Application Architecture, Reading, 2002.

[Fr99] Frank, U.: MEMO - Visual Languages for Enterprise Modelling. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Koblenz, 1999.

[Ga95] Gamma, E., et al.: Design Patterns. Elements of Reusable Object-Oriented Software, Reading, 1995.

[He04] Hevner, G., et al.: Design Science in Information Systems Research. MIS Quarterly, 28 (1) 2004, S. 75-105.

[Ka88] Karimi, J.: Requirements and Information Engineering Methods. Journal of Management Information Systems, 4 (4) 1988, S. 5-24.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". Veröffentlichungen des Instituts für Wirtschaftsinformatik (Scheer, A.-W., Ed.), Saarbrücken, 1992.

[KK84] Kottemann, J.E.; Kosynski, B.R.: Information Systems Planning and Development: Strategic Postures and Methodologies. Journal of Management Information Systems, 1 (2) 1984, S. 45-63.

[Ro07] La Rosa, M., et al.: Linking Domain Models an Process Models for Reference Model Configuration. In Proceedings of the 10th International Workshop on Reference Modeling, Vol. 4928 (ter Hofstede, A.H.M.; Benatallah, B., Eds.), Brisbane, 2007, S. 417-430.

[Mo88] Morris, C.W.: Grundlagen der Zeichentheorie. Fischer-Taschenbuch-Verl., Frankfurt am Main, 1988.

[Ob03] Object Management Group: MDA Guide Version 1.0.1, 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf.

[Ob04] Object Management Group: UML 2.0 Superstructure Specification, 2004. http://www.omg.org/cgi-bin/doc?formal/05-07-04

[Ob06] Object Management Group: BPMN 1.1, 2008. http://www.bpmn.org/Documents/BPMN%201-1%20Specification.pdf.

[Ro97] Rolland, C.: A Primer for Method Engineering. In Proceedings of the INFORSID, Tolouse, 1997.

[RA07] Rosemann, M.; van der Aalst, W.M.P.: A Configurable Reference Modelling Language. Information Systems, 23 (1) 2007, S. 1-23.

[Sc00] Scheer, A.-W.: ARIS – Business Process Modelling. 3 Edition, Berlin u.a., 2000.

[SS83] Schlageter, G.; Stucky, W.: Datenbanksysteme - Konzepte und Modelle. 2 Edition, Stuttgart, 1983.

[SGD03] Soffer, P.; Golany, B.; Dori, D.: ERP modeling: a comprehensive approach. Information Systems, 28 (9) 2003, S. 673-690.

[Su02] Sun Microsystems: Java™ Metadata Interface(JMI) Specification, 2002. http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html.