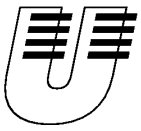

UNIVERSITÄT
KOBLENZ · LANDAU

Institut für
Wirtschaftsinformatik

Fachbereich Informatik
Universität Koblenz-Landau

LUTZ KIRCHNER
JÜRGEN JUNG

EIN BEZUGSRAHMEN ZUR EVALUIERUNG VON UML-MODELLIERUNGSWERKZEUGEN

März 2001

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 26

UNIVERSITÄT KOBLENZ · LANDAU

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

LUTZ KIRCHNER
JÜRGEN JUNG

EIN BEZUGSRAHMEN ZUR EVALUIERUNG VON UML-MODELLIERUNGSWERKZEUGEN

März 2001

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 26

Die Arbeitsberichte des Instituts für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i.d.R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The "Arbeitsberichte des Instituts für Wirtschaftsinformatik" comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Anschrift der Verfasser
Address of the authors:

Lutz Kirchner
Jürgen Jung
Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
D-56075 Koblenz

Arbeitsberichte des Instituts für Wirtschaftsinformatik
Herausgegeben von / Edited by:

Prof. Dr. Ulrich Frank
Prof. Dr. J. Felix Hampe
Prof. Dr. Gerhard Schwabe

Bezugsquelle / Source of Supply:

Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
56075 Koblenz

Tel.: 0261-287-2520
Fax: 0261-287-2521
Email: iwi@uni-koblenz.de
WWW: http://www.uni-koblenz.de/~iwi

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

Zusammenfassung

Im Rahmen dieses Berichts wird ein Bezugsrahmen entworfen, der es ermöglicht, am Markt erhältliche Software zur Modellierung mit der UML zu bewerten und die Produkte der Hersteller miteinander zu vergleichen. Der Bezugsrahmen versteht sich als erster Vorschlag für ein Vorgehen bei der Evaluation entsprechender Software-Werkzeuge in einer späteren Arbeit. In deren Verlauf wird er noch einigen Änderungen unterliegen. Ausgehend von einem Kriterienkatalog werden die für die Durchführung der Evaluation nötigen Qualitätskriterien vorgestellt und erläutert. Dabei wird grob zwischen drei Kriterienkategorien unterschieden:

- Allgemeine Kriterien für Software
- Kriterien für Modellierungswerkzeuge
- Spezielle Kriterien bzgl. der Modellierung mit der UML

Die erste Kategorie behandelt die für die Anschaffung von beliebiger Software generell gültigen Qualitätskriterien. Die zweite befaßt sich mit den spezielleren Anforderungen an ein Modellierungswerkzeug, dessen erklärte Funktionalität die Erstellung eines Modells sowie die Erzeugung von Quellcode aus diesem ist. In der letzten Kategorie wird schließlich auf die für die Modellierung in UML relevanten Aspekte eingegangen. Insgesamt ist jedoch zu bemerken, daß die aufgelisteten Punkte zur Bewertung der Software in keiner Weise disjunkt zu betrachtende Teilmengen innerhalb des Bezugsrahmens bilden, sondern daß hier eine Überlappung und gegenseitige Beeinflussung bzw. Implikation verschiedener Kriterien zu beobachten ist. Das beinhaltet die Zuordnung zu den Kategorien dahingehend, daß bspw. die Bewertung der Qualität der Bedienungsoberfläche von Software in der ersten Kategorie nicht völlig getrennt von ihrem Zweck - also der Erstellung von Modellen - gesehen werden kann, was somit ab einer gewissen Betrachtungstiefe automatisch Kriterien der zweiten Kategorie tangiert. Auch ist es nicht möglich, die Versionskontrolle eines Tools unabhängig von der Art und Weise zu betrachten, wie Teile eines Modells in das Repository eingebracht werden und wie die Konsequenzen für den Mehrbenutzerbetrieb aussehen. Wenn sich im Rahmen der Evaluation eine solche Überlappung zwischen den einzelnen Kriterien ergibt, wird bei der Bewertung der einzelnen Tools darauf hingewiesen, bei der hier vorliegenden Aufstellung der Kriterien aber davon noch weitestgehend abstrahiert.
Ein weiteres wichtiges Unterscheidungsmerkmal der im folgenden aufgeführten Kriterien ist die Art und Weise der Bewertung. Die Zweckmäßigkeit der Anordnung von Menüs bspw. läßt sich von einigen grundlegenden Aspekten abgesehen nicht ausschließlich objektiv bewerten und ist dabei abhängig von den Bedienungsgewohnheiten des Benutzers und somit stark subjektiv. Andere lassen sich durchaus objektiv gegeneinander abwägen. Auf diese Bewertungsschwerpunkte soll jeweils deutlich hingewiesen werden, um aufzuzeigen, an welcher Stelle die Präferenzen des Benutzers stark ausschlaggebend sind und somit den Bezugsrahmen in seiner Interpretation nicht zu sehr an eine kleine Personengruppe zu binden.
Die der Evaluation zugrundeliegende Software ist generell unter Windows 9x/ME und NT/2000 lauffähig. Diese Betriebssysteme wurden als Plattformen ausgewählt, da die Hersteller aller Tools, die in der Testphase Berücksichtigung finden sollen, eine unter diesen Umgebungen lauffähige Version anbieten und somit die Marktdichte in diesem Bereich am höchsten ist.

Inhaltsverzeichnis

1. Allgemeine Kriterien für Software

Ein Modellierungswerkzeug ist allgemein gesehen zunächst eine Software zur Erreichung eines definierten Ziels in einem bestimmten Einsatzbereich. Somit gelten für dedizierte Modellierungswerkzeuge auch allgemeine Kriterien für softwaretechnische Anwendungen. Diese allgemeinen Kriterien werden im Verlauf dieses Kapitels konkretisiert. Anschließend diskutiert das folgende Kapitel 2 Anforderungen an Modellierungswerkzeuge und Kapitel 3 Eigenschaften von Werkzeugen zur Modellierung mit der Unified Modeling Language (UML).

1.1. Wirtschaftliche Rahmenbedingungen

Wirtschaftliche Rahmenbedingungen betreffen die mit der Anschaffung und dem Einsatz eines Tools auftretenden Kosten. Zu den Anschaffungskosten zählen die unmittelbar durch den Kauf der Software anfallenden Kosten (s. Abschnitt 1.1.1.). Darüber hinaus sind mit den reinen Anschaffungskosten auch finanzielle Aufwendungen für notwendige Hardware verbunden, so daß die Hardware-Voraussetzungen für ein Werkzeug in die Aufwendungen mit einzubeziehen sind (s. Abschnitt 1.1.2.). Auch Preismodelle der Anbieter bzgl. differenzierter Ausstattungsvarianten werden im Rahmen der Betrachtung der Wirtschaftlichkeit berücksichtigt (Abschnitt 1.1.3.).

1.1.1. Kosten

Die Anschaffungskosten sind das Kriterium, daß bei der geplanten Beschaffung von Software zumeist als erstes Berücksichtigung findet. Hierbei existieren verschiedene Blickwinkel, die es zu beachten gilt. Viele kleinere Firmen sowie öffentliche Einrichtungen - nicht zuletzt solche in Schule und Lehre - haben ein festes nicht zu überschreitendes Budget für derartige Neuanschaffungen, so daß diese Kosten als absolute Größe gesehen werden müssen. Alternativ kann versucht werden, eine Kosten/Nutzen-Relation aufzustellen, innerhalb der man den zu erwartenden Nutzen beim Einsatz der Software - in diesem Falle also mittel- bis langfristige Einsparungen durch die Vorteile, die eine solches Tools in der Softwareentwicklung mit sich bringt - zu ermitteln. Dieses läßt sich aber in der Regel nicht durch einfaches Vergleichen des vom Hersteller angegebenen Leistungsumfangs eines Tools mit den eigenen Anforderungen erreichen, sondern offenbart sich normalerweise erst nach einer gewissen Evaluierungsphase. Im Laufe dieser Phase können die Einsatzmöglichkeiten im Rahmen der speziellen Anforderungen ermittelt werden. Ein Bezugsrahmen wie der vorliegende bildet dabei ein nützliches Hilfsmittel, um Vorgaben für die Evaluation beizusteuern und diese zu beschleunigen.
Da die Kostenfrage somit von keinerlei allgemeingültigen Faktoren bestimmt wird, beschränkt sich die anvisierte Evaluation darauf, die Preise der verschiedenen Versionen der Programme aufzulisten und auf bestehende Rabatte für bspw. Schule und Lehre hinzuweisen.

1.1.2. Hardwarevoraussetzung

Falls nicht geplant ist, mit dem Erwerb der neuen Software auch neue Hardware zu beschaffen, ist es sinnvoll, die Hardwarevoraussetzung der verschiedenen Produkte zu überprüfen. Diese können im ungünstigen Falle weitere Kosten verursachen, wenn nach der Entscheidung für ein Tool oder sogar nach dessen Erwerb festgestellt wird, daß die zugrundeliegende Hardware der meisten Produktivsysteme nicht ausreichend ist.

1.1.3. Unterschiedliche Varianten

Die Ansprüche vieler Modellierer und Softwareentwickler sind aufgrund der Unterschiedlichkeit in Komplexität und Funktionalität des zu entwickelnden Modells bzw. Produkts sehr differenziert. Somit ist es von Vorteil, wenn ein Hersteller auf diese Tatsache mit mehreren Varianten seines Tools reagiert. Diese Varianten oder Editionen können dann z.B. mit einer Basis-Edition den Fall abdecken, daß kleinere Anwendungen von einzelnen Entwicklern mit Schwerpunkt auf der Implementierung entwickelt werden. Oder man geht von größeren Projekten aus, an denen mehrere Entwickler gleichzeitig arbeiten und die eine Mehrbenutzerfähigkeit benötigen und bietet dementsprechend eine umfangreichere Edition an, die auf diese Anforderungen zugeschnitten ist.

Der Hauptvorteil hierbei ist eine preisliche Staffelung je nach angebotenem Funktionsumfang, so daß ein Käufer nicht für eine Funktionalität bezahlen muß, die er in seinem Umfeld gar nicht benötigt.
Bei der Evaluierung eines Tools sollte auf die Unterschiede zwischen evtl. vorhandenen Editionen eingegangen werden. Besonderes Augenmerk wird dabei auf die Team- bzw. Mehrbenutzerfähigkeit eines Tools gerichtet, d.h. das Vorhandensein eines Repositories (möglichst auch im Rahmen einer Client/Server Installation) sowie die Möglichkeit der Versionierung, da dies grundlegende Eigenschaften für den Einsatz eines Tools in größeren Projekten sind.

1.2. Grad der Einschränkung der Evaluierungsversion

Üblicherweise bieten die Distributoren der Tools die Möglichkeit, per Internet eine Evaluierungsversion zu akquirieren oder sich diese auf CD zusenden zu lassen. Diese Versionen sind grundsätzlich eingeschränkt was entweder den Funktionsumfang oder die Dauer des Einsatzes betrifft. Dabei ist zu berücksichtigen, daß bei Beschränkung der Einsatzdauer der Zeitrahmen nicht zu eng gesteckt sein darf, da man gerade bei der gleichzeitigen Bewertung mehrerer Tools diesen Rahmen vor Abschluß überschreitet.
Bei Beschränkung des Funktionsumfangs muß trotzdem noch ein sinnvolles Arbeiten innerhalb der spezifischen Anforderungen an den Einsatz des Tools gewährleistet und keine wichtige Funktionalität komplett ausgeblendet sein (z.B. Abspeichern der Ergebnisse).

1.3. Ergonomie

Die Ergonomie der Benutzerschnittstelle wird unter Berücksichtigung der Aspekte Aufgabenangemessenheit, Erwartungskonformität, Individualisierbarkeit, Selbsterklärungsfähigkeit, Erlernbarkeit und Fehlerrobustheit[1] in den folgenden Punkten konkret überprüft:

1.3.1. Einhaltung des "Look and Feels" der Bedienungsoberfläche

Da Windows bei allen berücksichtigten Tools als Betriebssystem zugrunde liegt und das sogenannte "Look and Feel" - also die Gleichförmigkeit der Durchführung gleichartiger Arbeitsschritte wie Kopieren, Löschen, Einfügen von Komponenten und die Reaktion des Systems auf diese Abläufe - bei allen Windows-Versionen identisch ist, ist es von Vorteil, wenn dieses "Look and Feel" seitens der Software geboten wird. Damit wird die Einarbeitungszeit in die grundlegende Bedienung eines Programms für schon erfahrene Windows-Benutzer minimiert und der Lernprozeß kann sich schon nach kürzester Zeit auf die eigentliche Funktionalität der Software konzentrieren.
Negativbeispiele diesbezüglich sind Programme, die ursprünglich von einem anderen Betriebssystem auf die aktuelle Plattform portiert wurden und die Hauptcharakteristika der Oberfläche des Ursprungssystems behalten haben.

1.3.2. Anordnung der Bedienungselemente

Die Bedienungselemente, wie bspw. die Auswahl der zu zeichnenden Komponenten bei der Erstellung eines Diagramms oder die Anzeige der Eigenschaften der Komponenten, sollten kontextbezogen angezeigt und entsprechend übersichtlich angeordnet sein. Die Verschachtelungstiefe von Menüs und die Anzahl der durchzuführenden Einzelschritte zum Erreichen der gewünschten Funktionalität sollte abhängig von der Häufigkeit der Nutzung dieser sein.

1.3.3. Konfigurierbarkeit der Bedienungsoberfläche

Es sollte bis zu einem gewissen Grad möglich sein, Funktionssymbole und Menüs nach den eigenen Vorstellungen zu konfigurieren und so die Oberfläche seiner eigenen Arbeitsweise anzupassen statt umgekehrt.

[1] Eine nähere Erläuterung dieser Begriffe findet man in [WED+93], S. 11 ff und [Balz99], S522 ff.

1.3.4. Verhalten bei Benutzerfehlern

Fehlerhafte Benutzereingaben sollten seitens der Software abgefangen werden, ohne daß ein undefinierter Zustand eingenommen wird oder eine unerwartete Terminierung erfolgt. Ein Beispiel für einen solchen Fehlerfall ist die Zulassung von Umlauten für Elemente der Modellierungsebene, was dann später beim Compilieren der Zielsprache zu Fehlern führen kann.

1.4. Programmunterstützung

Üblicherweise kann die Qualität einer Software nicht ausschließlich anhand der angebotenen Funktionalität beurteilt werden. Vielmehr ist der Benutzer einer Software auf die Unterstützung seitens des Herstellers angewiesen. Dies beginnt mit einer benutzerfreundlichen Installation des Werkzeugs (Abschnitt 1.4.1.). Zusätzlich muß die Dokumentation des Programms und die unterstützte Sprache eine intuitive Einarbeitung ermöglichen (Abschnitte 1.4.2. und 1.4.3.). Sofern dies nicht gewährleistet werden kann – was bei komplexen Softwaresystemen nicht unüblich ist -, sollte der Support des Werkzeug-Herstellers greifen (s. Abschnitt 1.4.4.).

1.4.1. Installation

Voraussetzung für den problemlosen Einsatz von Software ist eine komfortable und den jeweiligen Bedürfnissen des Benutzers angepaßte Installation. In diesem Rahmen sollten Parameter wie Installationsort, Sprache, Umfang u.a. vom Benutzer festgelegt werden können, damit die spätere Konfiguration entsprechend vereinfacht wird und keine Systemressourcen für nicht benötigte Komponenten belegt werden. Auch sollte eine Deinstallationsroutine vorhanden sein, die gerade hinsichtlich des Entfernens von Evaluierungsversionen von Interesse ist.

1.4.2. Programmdokumentation

Die Dokumentation der Tools ist grundlegend für den Nutzen, den die Software bringen soll. Bewertet werden Art und Umfang der Dokumentation sowie die Eignung für die verschiedenen Zielgruppen (Administrator, Modellierer, Implementierer). Gängige Präsentationsformen der Programmdokumentation sind die klassische Papierform, sowie HTML oder PDF-Dateien auf CD oder proprietäre Hilfesysteme. Prinzipiell ist zumindest für die Einführung in die grundlegenden Funktionen des Produkts eine Dokumentation auf Papier vorzuziehen, da diese problemlos parallel zur Arbeit am Rechner gelesen werden kann. Das setzt eine im Lieferumfang enthaltene gedruckte Version oder eine in druckbarer Form vorliegende elektronische Dokumentation voraus, wobei letzteres einen erhöhten Aufwand für den Benutzer bedeutet.
Als Ergänzung dazu sollte eine kontextsensitive Online-Hilfe angeboten werden, mit der häufiger auftretende Bedienungsprobleme gelöst werden können oder die im Bedarfsfall zumindest auf das entsprechende Kapitel der eigentlichen Dokumentation verweist.

1.4.3. Sprache

Da der Fokus der durchzuführenden Evaluation in ihrer Hauptsache auf dem deutschen Markt liegt, wird nur untersucht, ob die Software resp. die beiliegende Dokumentation in englisch oder deutsch vorliegt. Auf evtl. erhältliche Programmversionen in anderen Sprachen wird nicht weiter eingegangen. In der Regel kann man davon ausgehen, daß die meisten Tools als internationale Version in englischer Sprache verfügbar sind. Einige Hersteller bieten eine lokalisierte Variante für Deutschland an. Die Lokalisierung kann sich sowohl auf die Software als auch auf die Dokumentation beziehen, betrifft aber auch oft nur einen der beiden Punkte.
Die Dokumentation betreffend ist eine lokalisierte Fassung immer vorzuziehen, da komplexe Sachverhalte in der eigenen Muttersprache leichter aufgenommen werden können als in einer Fremdsprache. Aus den selben Gründen ist eine deutsche Version der Software durchaus von Vorteil, da sie die Einarbeitungsphase vereinfachen kann. Wenn allerdings berücksichtigt wird, daß viele Unternehmen international tätig sind und die Projekte auf englischsprachiger Basis abgewickelt werden, bringt eine ausschließlich deutsche Fassung den Nachteil mit sich, daß sie nicht von allen Projektbeteiligten benutzbar ist. Somit sollten hier im optimalen Fall beide Sprachversionen verfügbar sein.

1.4.4. Herstellerseitige Unterstützung

Ein wichtiger Aspekt bei der Auswahl einer Software, der zumeist erst einige Zeit nach Erwerb dieser als solcher erkannt wird, ist die Art und der Umfang, mit der ein Hersteller sein Produkt unterstützt. Das beinhaltet solche Supportarten wie telefonische Hotlines, die Beantwortung technischer Fragen per E-Mail, User Help Desks oder ständig aktualisierte FAQs[2] auf den Websites der Hersteller, sowie das regelmäßige Erscheinen sogenannter Bugfixes oder Updates. Bei mangelhafter Supporttätigkeit seitens der Hersteller kann beim Auftreten grundlegender Probleme im speziellen Einsatzumfeld der Software schnell der Vorteil eines günstigen Anschaffungspreis wieder verloren gehen.
Ein Überprüfen der Qualität des Supports kann durch gleichartige Anfragen an die Supportmitarbeiter der verschiedenen Herstellerfirmen ansatzweise überprüft werden, wobei aufgrund der geringen Repräsentanz der Anfragen die Ergebnisse nicht überbewertet werden dürfen.

1.5. Plattformabhängigkeit

Ein interessanter Aspekt für Unternehmen mit einer heterogenen Rechnerinfrastruktur ist die Möglichkeit, Arbeitsergebnisse zwischen den einzelnen Plattformen austauschen zu können, wenn die Software für verschiedene Betriebssysteme erhältlich ist. Das beinhaltet das komplette Repository, einzelne Diagramme und auch den generierten Code, soweit er keine per Hand ergänzten systemspezifischen Aufrufe enthält. Diese Möglichkeit stellt einen weiteren kostenreduzierenden Faktor beim Erwerb der Software dar, da die bestehende Infrastruktur optimal ausgenutzt werden kann.
Auch ist es denkbar, daß Mitarbeiter innerhalb eines Projekts an verschiedenen Plattformen arbeiten müssen. Hierbei ist ein konkurrierendes Ziel zur Vereinheitlichung des „Look and Feels“ (s. Abschnitt 1.3.) zu erkennen, da verschiedene Betriebssysteme in der Regel stark unterschiedliche Oberflächen besitzen und somit zumeist die jeweilige Version des gleichen Tools auf unterschiedlichen Systemen eine andere Bedienbarkeit aufweisen. Wenn allerdings vom Hersteller versucht wird, die erhältlichen Versionen möglichst ähnlich zu gestalten, kann es zu Widersprüchen mit der Bedienungsfunktionalität nativer Software eines Betriebssystems kommen, was die Einarbeitungszeit in das Tool tendenziell erhöht.

1.6. Stabilität

Ein immer wieder die prinzipielle Leistungsfähigkeit und Nützlichkeit von Software einschränkendes Manko ist die mangelnde Stabilität der Programme. Die eigentliche Ursache dieser Instabilität kann aufgrund des Zusammenwirkens vieler Faktoren wie Hardware, Treiber, Schnittstellen zum Betriebssystem und nicht zuletzt Programmierfehler in der eigentlichen Software nur schwer ermittelt werden.
Im Falle eines Programmabsturzes oder fehlerhaften Verhalten des Tools ist es in der Regel sehr schwer nachzuvollziehen, ob ein wirklicher Softwarefehler vorliegt oder schlicht eine schlecht konfigurierte oder mit Fehlern behaftete Betriebssysteminstallation vorliegt. Abhilfe schafft da nur eine vergleichende Installation des zu evaluierenden Tools auf einem anderen Rechner.

[2] FAQs (Frequently Asked Questions) sind Listen der am häufigsten auftretenden Probleme beim Umgang mit einer Software und entsprechende Lösungsvorschläge dazu.

2. Kriterien für Modellierungswerkzeuge

Inhalt dieses Kapitels sind besondere Anforderungen an Modellierungswerkzeuge. Diese heben sich von den allgemeinen Kriterien aus Kapitel 1 insofern ab, als der spezielle Einsatz zur Modellierung von Softwaresystemen berücksichtigt wird. Kapitel 3 konkretisiert diese Anforderungen anhand von UML-Modellierungswerkzeugen

2.1. Im- und Export

Im Fokus der Bewertung von Modellierungswerkzeugen steht zunächst die Interoperabilität mit anderen Anwendungen. Diese Interoperabilität zieht sich von der Fähigkeit des Im- und Exports der Formate fremder Modellierungswerkzeuge auf Modellebene bis zur Erzeugung von Dokumentationen und der Erzeugung von Programmcode für Software-Entwicklungswerkzeuge. Zentrale Fragestellung ist hierbei die Codeerzeugung für festgelegte Programmiersprachen und die Rekonstruktion von Modellen aus vorhandenem Programmcode.

2.1.1. Import/Export anderer Formate

Falls ein Produktwechsel im laufenden Projekt durchgeführt werden muß, da sich herausstellt, daß die Funktionalität des bisher benutzten Tools den Ansprüchen des aktuellen Projekts nicht mehr gewachsen ist oder einfach nur bereits abgeschlossene Projekte oder Musterlösungen auf das neue System übertragen werden sollen, ist eine möglichst breite Unterstützung bzgl. Import und Export von Fremdformaten eines Tools wünschenswert. Auch hinsichtlich der Kooperation bei verteilter Entwicklung ist ein problemloser Austausch von Projektkomponenten zwingend notwendig.
Der Import von Teilmodellen unter Erhaltung der Semantik, die mit gängiger Software wie bspw. Rational Rose erstellt wurden, sollte genauso möglich sein wie die Integration von Quellcode und die entsprechende Generierung des Modells (s. Abschnitt 2.1.2.). Der Export in das Format gängiger Entwicklungstools ist genauso wünschenswert.
Als universeller Austauschmechanismus bietet sich der XMI-Standard der OMG[3] an, der aufbauend auf XML DTDs[4] die Umwandlung der Modelle in ein für den Austausch - insbesondere über das Internet - geeignetes Format beschreibt. In Zukunft soll XMI von einigen Tools unterstützt werden.
Weiterhin kann es nützlich sein, Teilmodelle in andere Formate (Text oder Grafik) zu exportieren, um diese nach entsprechender Nachbearbeitung optisch aufgewertet für Präsentationen zu verwenden (s. Abschnitt 2.1.6.).

2.1.2. Round Trip Engineering

Unter Round Trip Engineering versteht man die Fähigkeit eines Tools, Modell und Quellcode in beide Richtungen konsistent zu halten[5]. Das bedeutet, daß sowohl Änderungen im Modell sofort in den Quellcode übernommen werden (Forward-Engineering) als auch umgekehrt (Reverse-Engineering). Dabei müssen u.a. auch sämtliche Kommentare, die in das Modell oder den Code zum Zwecke der Dokumentation eingefügt wurden, erhalten bleiben. Überprüft wird die Reichweite des Round Trip Engineering, d.h. ob bspw. nur Klassen- und Methodenrümpfe respektive deren Benennung Teil dieser Funktionalität sind, oder ob auch Methoden-Parameter oder sogar dynamische Aspekte berücksichtigt werden. Letzteres könnte die automatische Erstellung von Sequenz-Diagrammen anhand der Analyse (tracing) von Methodenaufrufen im Quellcode sein (Auto-Generation).
Weiterhin wird die Qualität bzgl. der Zuverlässigkeit der Konsistenzerhaltung im Rahmen des Round Trip Engineering sowie die Übereinstimmung des tatsächlichen Leistungsumfangs mit den Herstellerangaben diesbezüglich überprüft.

---

[3] XMI (XML Metadata Interchange) wird in [Brod99] kurz vorgestellt und in [OMG98] detailliert beschrieben. XML (eXtensible Markup Language) wird in Abschnitt 2.1.6.1. vorgestellt.
[4] Eine DTD (Document Type Definition) ist die Beschreibung des Aufbaus eines Dokuments.
[5] Vergleiche dazu [SeGu00], S.157 ff.

2.1.3. Codeerzeugung

Um die Qualität des vom Werkzeug generierten Quellcodes zu überprüfen, muß zuerst einmal ermittelt werden, wie weit diese Möglichkeit genutzt werden kann. D.h. ob diese sich auf die Erstellung reiner Rumpfprogramme ohne Funktionalität beschränkt oder ob die Dynamik eines Modells mittels Methodenaufrufen ansatzweise in den Quellcode mit aufgenommen werden kann (s. Abschnitt 2.1.2.). In diesem Falle geht die Funktionalität des Codes mit in die Bewertung ein, wobei diese gegenüber der gebotenen Performanz aufgrund des hier skizzierten Rahmens deutlich im Vordergrund steht.
Weiterhin muß die Zielsprache des Projekts beliebig durch einzubindende Bibliotheken erweiterbar sein, so daß es zu keinen Einschränkungen bei der Implementierung kommt.

2.1.4. Zielsprachenabhängigkeit

Bei einigen Tools ist der Sprachschatz der Modellierungssprache direkt abhängig von der gewählten Zielsprache, so daß der Anspruch der Modellierung, insbesondere der UML, zielsprachenunabhängig zu sein, hier untergraben wird. Das liegt an der beabsichtigten Implementationsnähe der Entwicklungswerkzeuge, da es nun mal im Sinne der problemlosen Umsetzung des Modells auf eine Programmiersprache wenig Sinn zu machen scheint, Konzepte wie die Mehrfachvererbung jedesmal wieder auf eine Einfachvererbung zu reduzieren, wenn die am häufigsten benutzte Zielsprache nur letztere beherrscht.
Zum Problem wird eine zu starke Zielsprachenabhängigkeit spätestens dann, wenn im laufenden Projekt die Zielsprache geändert wird. Deshalb ist zu überprüfen, wie sich das Tool in diesem Falle verhält und was das für den bisherigen Fortschritt im Projekt bedeutet.
Falls das Produkt Phasen eines Software-Lifecycles oder eines Prozesses unterstützt (s. Abschnitt 2.3.1.), muß überprüft werden, ob eine Zielsprachenabhängigkeit entgegen der den frühen Phasen inhärenten Philosophie auch dort vorhanden ist.

2.1.5. Berücksichtigung von Persistenz

Bei der Erstellung eines Modells einer zu entwickelnden Software ist es von Vorteil, wenn in der Design-Phase des Entwicklungsprozesses die Möglichkeit der Persistenz[6] bestimmter Objekte berücksichtigt wird. Weiterführend ist dann zu prüfen, inwieweit der Aspekt der späteren Nutzung von Datenbanken in die Modellierung mit einfließt, indem man bspw. die Möglichkeit hat, Datenbankschemata aus den als persistent markierten Klassen zu erzeugen. Da die Syntax der Schemata für (objekt)relationale Datenbanken bspw. bzgl. der Art und Weise, wie Primärschlüssel zu definieren sind, sehr unterschiedlich ist, kann hier noch darauf eingegangen werden, welche konkreten Datenbankprodukte diesbezüglich unterstützt werden.
Dasselbe gilt prinzipiell für objektorientierte Datenbanken, für die sich bisher noch kein einheitliches Persistenzmodell durchgesetzt hat und die damit analog zu den relationalen Datenbanken nicht generell unterstützt werden können.

2.1.6. Dokumentationsmöglichkeiten

Dokumentation wird in diesem Kontext als die Möglichkeit verstanden, das Modell oder Teile dessen während der Entwicklung entsprechend zu beschreiben und zu kommentieren und es anschließend darauf basierend in elektronischer Form oder auf Papier weiterzugeben. Das setzt zu allererst voraus, daß während der Erstellung des Modells auf allen Ebenen entsprechende Beschreibungen und Kommentare festgehalten werden können. Weiterhin muß es möglich sein, bei der Weitergabe des Modells oder Teilmodells zur Bewertung und Überprüfung durch Dritte eine Präsentationsform zu wählen, die diese Aufgabe entsprechend erleichtert und vor allem das Vorhandensein des Entwicklungstools selbst zur eigentlichen Sichtung des Modells unnötig macht. Überprüft werden also die Möglichkeiten der entwicklungsbegleitenden Dokumentation und deren Präsentation in einer zu erstellenden Dokumentation.

[6] Eine Beschreibung der Arten von Objektpersistenz findet sich in [Jung00], S. 131 ff.

2.1.6.1. Elektronische Dokumentationsmöglichkeiten

Gängige Formate für die Weitergabe von Projektergebnissen sind zuallererst navigierbare Formate (Mark-Up Sprachen) wie HTML und XML[7]. Weiterhin werden auch Dokumentenformate wie PDF, RTF oder das Word-Dokumentenformat benutzt, die auch einen gewissen Grad an Navigierbarkeit bieten. Zuletzt sind Grafikformate zur Ausgabe von Diagrammen zu nennen, die einerseits vektororientiert (z.B. WMF oder SVG[8]) oder andererseits pixelorientiert sind (GIF, BMP, JPG u.a.). Prinzipiell ist bei mehreren angebotenen Formaten die Auswahl abhängig von der geplanten Verwendung der Dokumentation. Denkbar wäre die Zielsetzung, ein Modell oder großes Teilmodell in seiner Gesamtheit und mit seinen Querbezügen zu demonstrieren (mittels Mark-Up basierter Navigation), oder es soll die erstellte Dokumentation in einer anderen Software noch ergänzt und beschrieben werden (Text- oder Grafikformat).

2.1.6.2. Dokumentationsmöglichkeiten auf Papier

Wo eine elektronische Form der Weitergabe nicht gewünscht oder praktikabel ist, muß es möglich sein, entsprechend vielfältig konfigurierbare Druckroutinen zur Ausgabe auf Papier nutzen zu können. Grundlegende Funktionen wären dabei die Ausgabe der Teilmodelle entsprechend der Darstellung auf dem Bildschirm (WYSIWYG), die Wahl zwischen "fit to page" und "wall chart" und verschiedenen Vergrößerungsstufen. Auch eine Preview-Funktion zu Überprüfung des Druckergebnisses vor dem eigentlichen Druckvorgang sollte angeboten werden.

2.2. Integration vorhandener Konzepte

Wiederverwendung sollte sich in den Werkzeugen in mehreren Aspekten widerspiegeln. Ein erster Ansatzpunkt ist die Integration von Entwurfsmustern in die Modelle. Hierbei müssen auch Frameworks berücksichtigt werden. Im Hinblick auf die Wiederverwendung von Software ist die Berücksichtigung von Komponenten eine wünschenswerte Eigenschaft. Diese Wiederverwendung kann sich in der Integration vorhandener Komponenten in das Modell wiederspiegeln. Es sollte aber auch die Definition eigener Komponenten auf Basis des eigenen Objektmodells möglich sein, wobei das Werkzeug bei der Code-Erzeugung Standardschnittstellen diverser Komponentenmodelle (JavaBeans, Enterprise JavaBeans, ActiveX, etc.) berücksichtigt.

2.2.1. Musterlösungen

Das Konzept domänenabhängiger Entwicklungsprozesse (s. Abschnitt 2.3.) weiterführend ist es wünschenswert, wenn zu diesen auch schon Musterlösungen in Form von Frameworks[9] oder Entwurfsmustern[10] vorliegen würden. Weiterhin sollten eigene Muster definiert und als solche für spätere Projekte abgelegt werden können. Auch der problemlose Import von Pattern ist von Vorteil, wenn sich für bestimmte Problemdomänen schon Designlösungen etabliert haben. Damit wären u.a. von der Business Object Design Task Force (BODTF)[11] in Zukunft im Bereich von Geschäftsobjekten und deren Kommunikation untereinander geschaffene Standards problemlos in die eigene Umgebung integrierbar.

2.2.2. Unterstützung von Komponenten

Im Rahmen der Erstellung von Klassendefinitionen sollte weiterhin die Möglichkeit gegeben sein, die daraus instanzierten Objekt später problemlos mit Komponententechnologien zu kombinieren. Als solche Technologien sind CORBA-Components[12], EJB[13] und COM+[14] nennen, zu deren Nutzung

[7] XML (eXtensible Markup Language) wird in [John99] näher beschrieben.

[8] SVG (Scalable Vector Graphics) ist eine Anwendung der XML zur Beschreibung zweidimensionaler Grafiken (Spezifikation s. [W3C00]).

[9] Die Einsatzmöglichkeiten von Frameworks werden anhand eines Beispiels in [Baue00] erläutert.

[10] Das Thema Entwurfsmuster (Design-Patterns) wird in [Gamm96] ausführlich behandelt.

[11] Zum gegenwärtigen Stand der Standardisierungsbemühungen s. [BODT01].

[12] Eine Beschreibung des Prinzips von CORBA (Common Object Request Broker Architecture) und IDL (Interface Definition Language) findet sich in [OHE98].

jeweils Schnittstellendefinitionen zu erstellen sind. Das Tool sollte bei Berücksichtigung einer dieser Architekturen die Erzeugung dieser Schnittstellen durch die entsprechende Codegenerierung möglichst weitreichend unterstützen.

2.3. Entwicklungsprozeß

Die Modellierung ist nur ein Teil der mit der Entwicklung eines Anwendungssystems verbundenen Tätigkeiten. Die einzelnen Tätigkeiten sowie deren Interdependenzen werden in Form eines Softwareentwicklungsprozesses beschrieben (s. Abschnitt 2.3.1.). Daneben empfehlen etablierte Modellierungsmethoden unterstützende Pragmatiken wie bspw. Metriken (s. Abschnitt 2.3.2.).

2.3.1. Prozeß

Da die Entwicklung eines Modells gänzlich ohne Einhaltung zumindest eines rudimentären Prozesses wenig sinnvoll erscheint, stellt sich die Frage, inwiefern ein möglichst domänenunabhängiges Prozeßrahmenwerk oder mehrere häufig bezogene Domänen betreffende spezifische Prozesse im Tool als Hilfsmittel angeboten werden. Die Detaillierung des Prozesses kann von einem vereinfachten Wasserfallmodell oder Software-Lifecycle mit Analyse-, Design- und Implementierungsphase bis hin zu einem kompletten objektorientierten Prozeß wie dem Rational Unified Process (RUP)[15] reichen. Interessant ist, wie der den Prozeßphasen inhärente Fokus beim Projektfortschritt umgesetzt wird und wie die Phasenübergänge realisiert werden. Außerdem wird überprüft, ob der Prozeß, wenn er von der Software angeboten wird, bei der Entwicklung bindend ist oder optional verfolgt werden kann.

2.3.2. Integration von OO-Metriken

Mittlerweile wurden im Bereich der objektorientierten Modellierung und Softwareentwicklung eine Vielzahl von Metriken (Methoden pro Klasse, Tiefe des Vererbungsbaums, Kohäsion u.a.[16]) zur Bewertung der Komplexität und Qualität eines Modells definiert, von denen einige sehr gut geeignet sind, um eine Qualitätskontrolle schon in einer frühen Entwicklungsphase zu gewährleisten. Ein Entwicklungstool sollte die Möglichkeit bieten, diese Metriken zum Zwecke der Überprüfung eines Modells im Rahmen eines Reports zur Verfügung zu stellen. Bewertet wird sowohl die Möglichkeit der Anwendung verschiedener Metriken als auch deren Übersichtlichkeit bezüglich der Präsentation. Soweit möglich, werden auch die Laufzeiten der Auswertungsläufe qualitativ berücksichtigt, um Ausreißer zu identifizieren. Ein direkter Perfomanzvergleich ist nicht vorgesehen.

2.4. Beispiele und Tutorials

Zusätzlich zur Programmdokumentation sollte ein Tool eine gewisse Anzahl von vorgefertigten und gut dokumentierten Beispielen unterschiedlicher Komplexität bieten, da anhand dieser Beispiele in der Regel ein Abschätzen der Möglichkeiten der Software leichter möglich ist, als durch die ausschließliche Lektüre der Dokumentation.
Weiterhin wäre es wünschenswert, wenn exemplarisch im Rahmen eines Tutorials schrittweise Beispielapplikationen modelliert werden können, um die Bedienung praxisnah erlernen zu können.

2.5. Navigation und Übersichtlichkeit

Da innerhalb eines Modells die Diagramme der Teilmodelle semantisch nicht isoliert stehen sondern eindeutige Zusammenhänge existieren, ist es wichtig im Rahmen der Entwicklung die Möglichkeit geboten zu bekommen, diese Zusammenhänge deutlich und navigierbar zu machen. Durch die Definition von Links, also Verknüpfungen, zwischen in direkter Beziehung stehenden Elementen verschiedener Diagramme wird der Überblick über diese semantischen Zusammenhänge vereinfacht.

---

[13] Eine kurze Einführung in das Prinzip von EJB (Enterprise Java Beans) findet sich in [Word00], S. 169 ff.
[14] Das COM+ (Component Object Model+) vom Microsoft findet sich vergleichend mit CORBA-Components und EJB beschrieben in [Stal00].
[15] Der Rational Unified Process wird in [Kruc99] einführend beschrieben.
[16] Eine kurze Beschreibung dazu findet sich in [Booc94], S. 349 ff.

Es muß überprüft werden, wie weit die Verknüpfung der einzelnen Diagramme geht und wie einfach die definierten Links im Modell nachzuvollziehen sind. Auch die Navigation über einen Klassenbrowser, mit dessen Unterstützung man von einer zentralen Stelle aus jedes beliebige Objekt eines Modells erreichen kann, ist in diesem Kontext von Nutzen und sollte angeboten werden.
Weiterhin ist es bei der Erstellung von Diagrammen am Bildschirm wichtig, den Detaillierungsgrad der Darstellung der jeweiligen Zielsetzung des Modellierers anzupassen. Das beinhaltet bspw. das Ein- und Ausblenden von Attributen und Methoden in Klassendiagrammen oder das Anzeigen von Verfeinerungen jeglicher Art. Somit muß die Möglichkeit gegeben sein, verschiedene Sichten (Views) auf das Modell zu definieren.
Weitere der Übersichtlichkeit eines Diagramms zuträglichen Funktionen sind ein gut funktionierendes Autolayout sowie eine Zoomfunktion, die es ermöglicht, ein Diagramm in beliebigen Vergrößerungsstufen zu betrachten.

2.6. Konsistenzerhaltung des Modells

Da wie oben schon erwähnt die Teilmodelle innerhalb eines semantischen Kontexts existieren, muß die Konsistenz dahingehend im Gesamtmodell bis zu einem gewissen Grade vom Tool überprüft und sichergestellt werden können. Diese Funktionalität kann sich von der einfachen Überprüfung der Namensräume und Typen bis hin zur Abgleichung kompletter Diagramme mittels der in Beziehung stehenden Elemente bewegen.
Auch sollten bei der Erstellung von bspw. Klassen, Attributen und Methoden und der nachträglichen Änderung der Zuordnung dieser zueinander innerhalb eines Diagramms alle anderen betroffenen Diagramme zumindest angezeigt werden, wenn eine automatische Korrektur nicht möglich oder vorgesehen ist. Der Grad der automatischen Überprüfung sollte konfigurierbar sein, so daß der Entwickler wählen kann, inwieweit und zu welchem Zeitpunkt Änderungen des Modells abgeglichen werden.

2.7. Verwaltung des Modells

Die persistente Verwaltung eines Modells sollte ein Werkzeug für den Benutzer transparent durchführen. Allgemein kann hierzu ein Repository eingesetzt werden (s. Abschnitt 2.7.1.), wobei nicht nur der Mehrbenutzerzugriff auf das Modell Mechanismen zur Versionsverwaltung impliziert (s. Abschnitt 2.7.2.).

2.7.1. Repository

Ein wichtiger Aspekt bei der Bewertung von Modellierungssoftware ist das Vorhandensein eines Repositories[17], in dem alle projektrelevanten Komponenten wie Modell und Quellcode verwaltet werden. Ein Repository hat den Vorteil, daß alle das Projekt betreffenden Daten zentral gehalten werden und somit eine Zugriffskontrolle für mehrere Benutzer leicht realisierbar ist. Zusätzlich kann es die Rolle eines semantischen Referenzsystems bei der Integration mit externer Software übernehmen und diese entsprechend vereinfachen.
Wenn das Tool eine eigene Versionskontrollfunktionalität mitbringt, kann das Repository alternativ zu externen Server-Lösungen auch zum Verwalten verschiedener Versionen des Projekts bzw. einzelner Komponenten genutzt werden (s. Abschnitt 2.7.2.).
Falls ein Entwicklungstool kein zugrundeliegendes Repository benutzt, wird überprüft, ob und inwieweit vom Hersteller die oben genannten Funktionalitäten auch ohne dieses umgesetzt wurden.

2.7.2. Versionierung

Prinzipiell sollte ein Modellierungswerkzeug vor allem hinsichtlich der Mehrbenutzerfähigkeit die Möglichkeit der Versionierung[18] bieten. Dabei können entweder eigene Strategien Verwendung finden (s. Abschnitt 2.7.1.) oder schon bestehende Lösungen wie SCCS unter Windows oder CVS-LAN verwendet werden, wobei ein Server die Verwaltung der verschiedenen Versionen übernimmt. Die Funktionalität der Versionierung wird sowohl hinsichtlich der Granularität (betreffend verschiedene

[17] Vergleiche dazu [Fran98], S. 222 ff.
[18] Grundlegendes zur Versionierung findet sich in [Eber95] im Abschnitt „Konfigurationskontrolle“.

Versionen von Einzelkomponenten bis hin zu kompletten Modellen) als auch der Zusammenführung von Teilmodellen unterschiedlicher Benutzer bzw. Versionen überprüft.

2.8. Erweiterbarkeit

Bei entsprechend offener Architektur des Tools sollte sich die Möglichkeit ergeben, für gewisse Tätigkeiten der Entwicklungsphase externe Editoren oder andere Hilfsmittel zu nutzen. Dabei ist die Art und Weise der Integration ausschlaggebend für den Grad der Erweiterung der Funktionalität des Tools. Die parallele Nutzung eines externen Editors setzt zumindest eine statische Integration voraus, um die zu editierende Komponente austauschen zu können. Funktionale Integration liegt bei der Nutzung von externen Versionskontrollmechanismen vor, da dort vorhandene Funktionen wie "check in" und "check out" im Rahmen des Projektes genutzt werden. Weitere Arten der Integration sind denkbar und welche ein Entwicklungstool schlußendlich bietet, soll innerhalb der durchzuführenden Evaluation aufgezeigt und bewertet werden[19].

[19] Vergleiche dazu [Fran98], S. 181 ff.

3. Spezielle Kriterien bzgl. der Modellierung mit der UML

3.1. UML 1.3 Konformität

Die Konformität der in dem Entwicklungswerkzeug zu erstellenden Diagramme zu der in der UML Version 1.3 festgelegten Syntax und Semantik ist Gegenstand dieses Teils der Bewertung. Dazu muß im vorhinein festgestellt werden, daß ein Betrachtung dahingehend nicht das Ziel hat, eine einhundertprozentige Übereinstimmung in beiden Punkten zu überprüfen.
Die Syntax betreffend wird es als von Vorteil angesehen, wenn die in der Referenz der UML vorgegebene nicht einfach verändert wird, es sei denn es geschieht im Rahmen einer Erweiterung (s. Abschnitt 3.2.). In der Regel ist aber davon auszugehen, daß nicht der gesamte Sprachschatz der UML realisiert wird, da dieser teilweise einer hohen semantischen Redundanz unterliegt, d.h. verschiedene Darstellungsmöglichkeiten für den selben Sachverhalt zur Verfügung stehen. In welchem sinnvollen Rahmen sich die Unterstützung der vielen verschiedenen grafischen Notationsmöglichkeiten der UML bewegen kann und sollte, wird versucht, im Verlauf der Evaluation zu klären.
Bei der Semantik ist eine Überprüfung der Konformität dahingehend recht kompliziert, daß diese in der Referenz der UML selbst nicht immer eindeutig geregelt ist (z.B. bei Semantik der Aggregation[20]). In diesem Rahmen kann nur konkret auf eventuelle Widersprüche mit der Referenz hingewiesen und eigene Interpretationsansätze der Hersteller aufgezeigt werden.
Im folgenden sind die im bisherigen Verlauf der Evaluierung als zur Überprüfung relevant erscheinenden Teilaspekte der UML aufgelistet. Da nicht jedes Werkzeug alle Diagrammarten der UML unterstützt, können selbstverständlich nicht alle unten aufgeführten Punkte bei allen Programmen untersucht werden. In diesem Falle geht der Bewertung eine Auflistung der unterstützten Diagrammarten voraus.
Die Liste ist Gegenstand von weiteren Ergänzungen, soweit sie sich bei der Evaluierung weiterer Tools ergeben, und erhebt somit keinen Anspruch auf Endgültigkeit.

Klassendiagramme

Folgende Modellelemente, die innerhalb eines Klassendiagramms relevant sind, werden daraufhin überprüft, ob sie zur Modellierung angeboten werden und der Syntax und Semantik des UML-Standards entsprechen:

- abstrakte Klassen
- Komponentenklassen
- individuelle Klassen (Singleton)
- Assoziationsklassen
- Metaklassen
- Implementierungsklassen
- Klassenkategorien
- Interfaces

Weitere Fragestellungen:

- Ist eine alternative Darstellung der Klassen (bspw. Interfaces in der Lollipop-Schreibweise statt als Kästchen) möglich?
- Ist es möglich, Qualifier grafisch und logisch darzustellen? Wird in diesem Falle automatisch ein Attribut erzeugt?
- Wie werden Aggregation und Komposition semantisch umgesetzt?

[20] Eine Diskussion der Semantikbeschreibung der Aggregation in der UML-Spezifikation findet sich in [HeBa99].

- Inwiefern können Integritätsbedingungen in der OCL[21] formuliert werden und in welchem Umfang werden diese ausgewertet? Werden die Constraints in den erzeugten Code eingefügt?
- Können Tagged Values und Notizen im Diagramm erzeugt werden?
- Ist Mehrfachvererbung möglich?
- Wie ist die Vererbung von Assoziationen (u.a. bzgl. Redefinition) und Rollen implementiert?
- Ist die Linienführung von Assoziationen wählbar (rechtwinklig oder direkt)?
- Kann die Leserichtung einer Assoziationen mit Hilfe eines Pfeils verdeutlicht werden?
- Wie werden Dependencies interpretiert?

Interaktionsdiagramme

Hier ist zuerst zu klären, ob in dem Tools sowohl Sequenz- als auch Kollaborationsdiagramme angeboten werden.

Weitere Fragestellungen:

- Können Zeitinformationen in Sequenzdiagrammen (Dauer von Methoden und Dauer der Aufrufe) modelliert werden?
- Werden alle Pfeilarten, die die Art des Nachrichtenaustauschs in Sequenzdiagrammen verdeutlichen sollen, angeboten?
- Ist es möglich, Mulitobjekte in Kollaborationsdiagrammen zu modellieren?

Aktivitätsdiagramme

- Kann ein Diagramm in sogenannte Swimlanes aufgeteilt werden?

Zustandsdiagramme

- Können Ereignisse und Bedingungen entsprechend der Notation vollständig angebracht werden?
- Können sowohl Aktivitäten als auch Aktionen modelliert werden?

Stereotypen

- Sind vorgefertigte Stereotypen vorhanden und sind beliebige weitere vom Benutzer zu ergänzen?
- Wie wird ihre Semantik definiert?
- Können einige Stereotypen wie z.B. Interfaces und Aktoren grafisch gesondert dargestellt werden?

Verfeinerungen

- Ist es generell möglich, das Konzept der Verfeinerung (bspw. von Use Cases oder Zuständen) anzuwenden?

Constraints

- Wird die grafische Repräsentation gewisser Constraints (z.B. {xor} für Assoziationen) unterstützt?

[21] Die Object Constraint Language (OCL) ist in [IBM97] spezifiziert.

3.2. Erweiterungen

Unter Erweiterung wird auf der einen Seite das Anbieten von nicht UML-konformen Notationsvarianten innerhalb von UML-Diagrammen verstanden. Dies kann eine rein syntaktische Erweiterung zur Erhöhung der Übersichtlichkeit oder eine semantische Ergänzung sein, mit der die Aussagekraft des Diagramms verstärkt wird.
Auf der anderen Seite kann ein Modell um nicht in der UML vorgesehen Diagrammarten erweitert werden. In diesem Fall muß überprüft werden, ob und wie sich diese in das Gesamtmodell integrieren lassen. Erweiterungen der Notation können unter Einhaltung des Sprachumfangs der UML durchgeführt werden. Hierbei ist insbesondere die Definition von Profilen von Bedeutung. Profile erlauben die Adaption der Modellierungssprache UML an domänenspezifische Randebedingungen und Anforderungen[22].

3.2.1. Ergänzungen der UML-Diagrammarten

Einige Werkzeuge erweitern die Darstellung von UML-Diagrammen durch eigene Notationselemente. Beispiele hierfür sind:

- Schleifen in Sequenzdiagrammen
- Superklasse im Klassenheader einer Subklasse alternativ zum Aufzeigen der Vererbungshierarchie mittels Pfeilen

3.2.2. Erweiterungen mit anderen Diagrammarten

Als weiteres Kriterium wird die Bereitstellung von Diagrammarten jenseits der UML untersucht:

- Entity Relationship Modell (ERM)
- Prozeßmodell

[22] Profile werden im UML Reference Manual (s. [RJB99]) auf S. 104 kurz erwähnt und nicht weiter erläutert.

Literatur

[Balz96] Balzert, H. (1996): "Lehrbuch der Softwaretechnik." Heidelberg; Berlin; Oxford: Spektrum Akademischer Verlag.

[Baue00] Bauer, G. (2000): "Bausteinbasierte Software: Eine Einführung in moderne Konzepte des Software-Engineering." 1. Auflage. Vieweg.

[Booc94] Booch, G. (1994): "Objektorientierte Analyse und Design: Mit praktischen Anwendungsbeispielen." 1. Auflage. Bonn; Paris; Reading, Mass. [u.a.]: Addison-Wesley.

[BODT01] Business Objects Design Task Force Homepage (2001), www.omg.org/bodtf

[Brod99] Brodsky, S. (1999): "XMI Opens Application Interchange." IBM.

[Burk97] Burkhardt, R. (1997): "UML - Unified Modeling Language: Objektorientierte Modellierung für die Praxis." 1. Auflage. Bonn: Addison-Wesley.

[Doss98] Doss, M. (1998): "Evaluierung eines CASE-Tools: Reverse Engineering und Software-Entwicklung mit Designer/2000." Studienarbeit, Universität Koblenz-Landau.

[Eber95] Ebert, J. (1995): "Softwareengineering II: Vorlesung SS 1995." Mitschrift der gleichnamigen Vorlesung.

[FoSc97] Fowler, M.; Scott, K. (1997): "UML Distilled: Applying the Standard Object Modeling Language." 3. Auflage. Addison-Wesley.

[Fran98] Frank, U. (1998): "Betriebliche Informationssysteme: Anwendungen, Architekturen und Integrationstechnologien: Foliensatz zur Vorlesung.".

[Gamm96] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1996): „Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software“ 1. Auflage. Bonn et al.: Addison-Wesley.

[Graf98] Graf, J. (1998): "Analyse und Bewertung der Modellierungskonzepte zur Struktur- und Verhaltensmodellierung der Unified Modeling Language (UML)." Diplomarbeit, Universität - Gesamthochschule Siegen.

[HeBa99] Henderson-Sellers, B.; Barbier, F. (1999): "Black and White Diamonds." University of Technology, Sydney.

[IBM97] o.V. (1997): “Object Constraint Language Specification: Version 1.1, 1 September 1997.” IBM

[John99] John, V. (1999): „XML – Weltsprache für das Internet“ In: OBJEKTSpektrum, Nr. 5 September/Oktober 1999

[Jung00] Jung, J. (2000): "ODBMS in der objektorientierten Softwareentwicklung: Konkretisiert am Beispiel GemStone." Diplomarbeit, Universität Koblenz-Landau.

[Kruc99] Kruchten, P. (1999): „Der „Rational Unified Process““ In: OBJEKTSpektrum, Nr. 1 Januar/Februar 1999

[ObD01] o.V. (2001): "Choosing a UML Modeling Tool." Objects by Design.

[OHE98] Orfali, R.; Harkey, D.; Edwards, J. (1998): "Instant CORBA: Führung durch die CORBA-Welt." 1. Auflage. Addison-Wesley.

[OMG98] o.V. (1998): "XML Metadata Interchange: Proposed to the OMG OA&DTF RFP 3: Stream Based Model Interchange Format (SMIF)." Object Management Group (OMG).

[OMG99] o.V. (1999): "OMG Unified Modeling Language Specification: Version 1.3." Object Management Group (OMG).

[RJB99] Runbaugh, J.; Jacobson, I.; Booch, G.: (1999): “The Unified Modeling Language Reference Manual.” Reading (MA) et al.: Addison-Wesley

[SeGu00] Seemann, J.; von Gudenberg, J. W. (2000): "Software-Entwurf mit UML: Objektorientierte Modellierung mit Beispielen in Java." 1. Auflage. Springer.

[Stal00] Stal, M. (2000): „Reich der Mitte – Die Komponententechnologien COM+, EJB und „CORBA Components““ In: OBJEKTSpektrum, Nr. 3 Mai/Juni 2000

[W3C00] o.V. (2000): "Scalable Vector Graphics (SVG) 1.0 Specification: W3C Candiate Recommendation 02 November 2000." W3C.

[WED+93] Winter, A.; Ebert, J.; Dumslaff, U.; Mertesacker, M. (1993): "Ein Vorgehensmodell zur Software-Evaluation." Universität Koblenz-Landau.

[Word00] Worden, D. J. (2000): "Big Blue Java: Complete Guide to Programming Java Applications with IBM Tools." 1. Auflage. Wiley.

Bisherige Arbeitsberichte

Hampe, J. F.; Lehmann, S.: Konzeption eines erweiterten, integrativen Telekommunikationsdienstes. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 1**, Koblenz 1996

Frank, U.; Halter, S.: Enhancing Object-Oriented Software Development with Delegation. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 2**, Koblenz 1997

Frank, U.: Towards a Standardization of Object-Oriented Modelling Languages? Arbeitsbericht des Instituts für Wirtschaftsinformatik, **Nr. 3**, Koblenz 1997

Frank, U.: Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modelling. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 4**, Koblenz 1997

Prasse, M.; Rittgen,P.: Bemerkungen zu Peter Wegners Ausführungen über Interaktion und Berechenbarkeit, Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 5**, Koblenz 1997

Frank, U.; Prasse, M.: Ein Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen - veranschaulicht am Beispiel vom OML und UML. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 6**, Koblenz 1997

Klein, S.; Zickhardt, J.: Auktionen auf dem World Wide Web: Bezugsrahmen, Fallbeispiele und annotierte Linksammlung. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 7**, Koblenz 1997

Prasse, M.; Rittgen, P.: Why Church's Thesis still holds - Some Notes on Peter Wegner's Tracts on Interaction and Computability. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 8**, Koblenz 1997

Frank, U.: The MEMO Meta-Metamodel, Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 9**, Koblenz 1998

Frank, U.: The Memo Object Modelling Language (MEMO-OML), Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 10**, Koblenz 1998

Frank, U.: Applying the MEMO-OML: Guidelines and Examples. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 11**, Koblenz 1998

Glabbeek, R.J. van; Rittgen, P.: Scheduling Algebra. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 12**, Koblenz 1998

Klein, S.; Güler, S.; Tempelhoff, S.: Verteilte Entscheidungen im Rahmen eines Unternehmensplanspiels mit Videokonferenzunterstützung, Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 13**, Koblenz 1997

Frank, U.: Reflections on the Core of the Information Systems Discipline. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 14**, Koblenz 1998

Frank, U.: Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 15**, Koblenz 1998

Frank, U.: An Object-Oriented Architecture for Knowledge Management Systems. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 16**, Koblenz

Rittgen, P.: Vom Prozessmodell zum elektronischen Geschäftsprozess. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 17**, Koblenz 1999

Frank, U.: Memo: Visual Languages for Enterprise Modelling. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 18**, Koblenz 1999

Rittgen, P.: Modified EPCs and their Formal Semantics. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 19**, Koblenz 1999

Prasse, M., Rittgen, P.: Success Factors and Future Challenges for the Development of Object Orientation. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 20**, Koblenz 2000

Schönert, S.: Virtuelle Projektteams - Ein Ansatz zur Unterstützung der Kommunikationsprozesse im Rahmen standortverteilter Projektarbeit. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 21**, Koblenz 2000

Frank, U.: Vergleichende Betrachtung von Standardisierungsvorhaben zur Realisierung von Infrastrukturen für das E-Business. . Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 22**, Koblenz 2000

Jung, J.; Hampe, J.F.: Konzeption einer Architektur für ein Flottenmanagementsystem. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 23**, Koblenz 2001

Jung, J.: Konzepte objektorientierter Datenbanken – Konkretisiert am Beispiel GemStone. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 24**, Koblenz 2001

Frank, U.: Organising the Corporation: Research Perspectives, Concepts and Diagrams. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 25**, Koblenz 2001

Kirchner, L.; Jung, J.: Ein Bezugsrahmen zur Evaluierung von UML-Modellierungswerkzeugen. Arbeitsberichte des Instituts für Wirtschaftsinformatik, **Nr. 26**, Koblenz 2001