Jürgen Ebert

MetaCASE: Generierung und Anpassung von CASE-Werkzeugen

1. Einleitung

Das Ziel von ***CASE (Computer Aided Software Engineering)*** *i*st die möglichst vollständige Unterstützung der Softwareentwicklung durch passende Werkzeuge. Praktisch setzt der Begriff CASE meist den Fokus auf Werkzeuge, die die ***frühen Phasen der Softwareentwicklung*** - die Anforderungsanalyse und den Entwurf - unterstützen.

Diese Phasen zeichnen sich dadurch aus, daß in hohem Maße auch ***graphische Dokumente*** verwendet werden, wie beispielsweise Klassen-Beziehungs-Diagramme, Zustands-Übergangs-Diagramme, Datenfluß-Diagramme oder Kontrollfluß-Diagramme. Hinzu kommen mit viele Arten von Tabellen und Übersichten.

CASE-Werkzeuge erlauben das Edieren und Analysieren dieser Dokumente. Sie unterstützen damit den Einsatz konkreter Analyse- und Entwurfsmethoden, erzeugen Code für konkrete Programmiersprachen, führen die Versionsverwaltung für die Dokumente durch und fördern so die Zusammenarbeit mehrerer Entwickler. Für nahezu alle bekannten Analyse- und Entwurfsansätze zur objekt-orientierten Systementwicklung sind heute Werkzeuge vorhanden, die diese Methoden mehr oder weniger gut unterstützen.

In diesem Beitrag werden von CASE-Werkzeugen aus einer ***konzeptuellen Sicht*** dargestellt, die sich aus der Auffassung von CASE-Umgebungen als Systeme zur Verarbeitung verschieden-sprachlicher Dokumente herleitet. (Es handelt sich also nicht um eine Marktübersicht.) Es werden die Hauptaspekte von CASE-Werkzeugen herausgearbeitet, woraus sich verschiedene Möglichkeiten der Anpassung dieser Systeme an konkrete Anforderungen ergeben. Dies führt zum Begriff der MetaCASE-Systeme.

2. CASE-Werkzeuge

Man kann die Entwicklung von Softwaresystemen aus der ***Prozeßsicht*** der dabei auszuführenden ***Aufgaben*** oder aus der ***Dokumentensicht*** der hierbei anfallenden ***Teilprodukte*** betrachten. Beide Sichten sind nicht voneinander zu trennen.

Auch CASE-Werkzeuge können danach unterschieden werden, ob sie eine eher prozeß-orientierte Sicht der Softwareentwicklung unterstützen oder eine eher dokumenten-orientierte Sicht. Gleich welche Sicht im Vordergrund steht, der Einsatz von Werkzeugen spielt bei der Erstellung großer Systeme in jedem Fall eine große Rolle, da allein die ***Anzahl*** der Aufgaben und der Dokumente die Werkzeugunterstützung zu einer Notwendigkeit macht.

Bei den meisten CASE-Werkzeugen steht eine ***dokumenten-orientierte*** Sicht im Vordergrund, d.h. die zentrale Leistung dieser CASE-Werkzeuge ist die Unterstützung der Dokumentenverwaltung im Softwareentwicklungsprozeß.

Die Hauptanteile dieser CASE-Werkzeuge sind daher ***Editoren***, die zur Verarbeitung, d.h. zur Eingabe und Fortschreibung, verschiedener Dokumente geeignet sind, sowie ***Analysatoren***, die die Konsistenz und Plausibilität der Dokumente überprüfen. Es besteht dabei ein enger Zu-

sammenhang zwischen den Werkzeugen und den von ihnen verarbeiteten Dokumentensprachen.

2.1 Beschreibung von Sprachen

CASE-Dokumente werden in unterschiedlichen ***(Dokumenten-)Sprachen*** verfaßt. Das Spektrum dieser Sprachen reicht dabei von ***textuellen Sprachen***, wie beispielsweise natürlicher Sprache - in reinen Textdokumenten - oder formalen Zeichenkettensprachen - wie beispielsweise in Programmquelltexten - bis hin zu ***visuellen Sprachen***, mit denen graphische Dokumente notiert werden. In CASE-Werkzeugen werden also Dokumente in textuellen Sprachen und Dokumente in visuellen Sprachen gemeinsam verwaltet.

In den frühen Phasen spielen gerade ***visuelle Sprachen*** eine herausragende Rolle, da sie mehr als rein textuelle Sprachen geeignet sind, von Details zu abstrahieren und auch unscharfe oder unvollständig bekannte Zusammenhänge zu skizzieren. Außerdem bilden sie eine bessere Grundlage für die Kommunikation unter den am Entwicklungsprozeß Beteiligten.

Die in der Softwaretechnik vorkommenden visuellen Sprachen sind allerdings meist sehr einfach gehalten: Häufig erlauben sie nur baumartige Übersichtsdarstellungen, oder sie bestehen aus einfachen Graphen mit verschiedenenartigen Knotenformen und verbindenden Kanten. Beispielsweise enthalten Klassendiagramme in objekt-orientierten Methoden meist nur rechteckige oder ovale Knoten, die mit verschiedenartig dekorierten Kanten verbunden werden. (Einige wenige Sprachen verwenden auch Higraphen [Harel88], d.h. Graphen in denen Knotenschachtelung verwendet wird, wie beispielsweise in Struktogrammen oder Statecharts.) Im folgenden werden alle diese Sprachen unter dem Begriff ***Diagrammsprachen*** zusammengefaßt, um sie von ausgefeilteren visuellen Darstellungen abzugrenzen.

Die Verarbeitung von Dokumenten ist in textuellen Sprachen und in Diagrammsprachen ähnlich. Die ***Beschreibung textueller Sprachen*** wird klassischerweise in die Beschreibung von Syntax, Semantik und Pragmatik aufgeteilt. Die tiefergehende Aufspaltung der syntaktischen Anteile einer Sprachbeschreibung in Lexik, kontextfreie Syntax und kontextsensitive Syntax (auch statische Semantik genannt)gibt (formal-)sprach-verarbeitenden Werkzeugen wie Compilern oder Editoren eine Grundstruktur, die sich auch in deren Aufbau widerspiegelt [AhSeUl86]

Die gleichen begrifflichen Unterscheidungen sind analog auch auf ***Beschreibung von Diagrammsprachen*** anwendbar. Hier ist die lexikalische Ebene durch die graphischen Formen gegeben. Graphische Formen sind beispielsweise Rechtecke, Kreise, Rauten etc. für knotenartige Objekte und Pfeile, Linie, etc. für kantenartige Objekte. Die kontextfreie syntaktische Ebene legt fest, welche Knoten- und Kantenverbindungen erlaubt sind, und zur kontextsensitiven Syntax gehört, wie die vorkommenden Bezeichner gewählt werden dürfen, welche Strukturanforderungen das Gesamtdiagramm erfüllen muß, wann Diagramme zueinander passen usw. Die semantischen Anteile von Diagrammen findet man bei Werkzeugen als Codeerzeugung wieder. Mit den pragmatischen Anteilen ist dann die darüber hinausgehende Unterstützung des Umgangs mit Dokumenten durch konkrete Methoden angesprochen.

2.2 Werkzeuge zu Sprachen

Eine Sichtung der Werkzeuge zur Bearbeitung sowohl textueller als auch visueller Dokumente führt zur Unterscheidung in erstellende Werkzeuge (Editoren), untersuchende Werkzeuge (Analysatoren) und verarbeitende Werkzeuge (Simulatoren und Codeerzeuger). Die ersten beiden Gruppen hängen mit der Syntax der verarbeitenden Sprache zusammen, die letzteren mit der Semantik.

Editoren sind Werkzeuge, die der Erstellung und Veränderung von Dokumenten dienen. Hierzu müssen sie eine Reihe von ***Operationen*** unterstützen, üblicherweise hat man etwa die folgenden Operationsgruppen:

- Hauptsteuerung (z.B. starten, verlassen, einlesen, ausschreiben, Modus definieren oder wechseln)
- Sichtenbehandlung (z.B. vergrößern, verkleinern, rollen)
- Positionierung (z.B. navigieren, suchen)
- Manipulation (z.B. eingeben, löschen, überschreiben)
- Aufbereitung (z.B. Layout erstellen)
- Pufferverwaltung (z.B. kopieren, einfügen)
- Sicherheitsmaßnahmen (z.B. rückgängig machen, automatisch abspeichern)
- Hilfemaßnahmen (z.B. kontext-abhängige Hilfe anbieten)
- Makromechanismen (z.B. Ablaufskripte erstellen)

Editoren können in unterschiedlichem Umfang mit der von ihnen verarbeiteten Sprache verknüpft sein. Hier ergibt sich ein gradueller Unterschied, der von freien Editoren ohne jeden Syntaxbezug bis zu streng syntax-orientierten Editoren reicht. Bei ***syntax-orientierten Editoren*** wird die syntaktische Struktur der unterstützten Sprache auch vom zugehörigen Editor berücksichtigt: Sie lassen nur die Eingabe (kontext-frei oder gar kontext-sensitiv) syntaktisch korrekter Programme zu.

Für die nicht vom Editor sichergestellten syntaktischen Spracheigenschaften sind dann entsprechende ***Analysatoren*** erforderlich. Editoren und Analysatoren sind insofern komplementär. Analysatoren sind Werkzeuge, die der Überprüfung von Dokumenten dienen. Solche Analysatoren sind beispielsweise Syntax-Checker, die eine vollständige syntaktische Überprüfung des Dokuments durchführen. Praktisch sind der Editoranteil und der Analysatoranteil manchmal aber schwer voneinander zu trennen.

Syntax-orientierte Editoren verringern zwar insgesamt den Eingabeaufwand, und garantieren, daß das eingegebene Dokument korrekt ist. Sie erschweren aber Veränderungen, die nicht grammatik-konform sind, und erfordern gerade bei einfachen Konstrukten, wie beispielsweise arithmetischen Ausdrücken, häufig einen höheren Aufwand als eine freie Eingabe im Klartext. Im allgemeinen sind daher Editoren gewünscht, die ***hybrid*** sind in dem Sinne, daß sie ein Umschalten zwischen verschiedenen Modi (freie Texteingabe unter Verwendung eines Analysators vs. streng syntax-geführte Eingabe) erlauben.

Editoren arbeiten i.a. auf einer eigenen Kopie des Dokuments, die innerhalb des Editors gehalten wird. Es muß daher zwischen der Darstellung von Dokumenten für den Endbenutzer (***externe Darstellung***) und deren Repräsentation innerhalb des Werkzeugs (***interne Darstellung***) unterschieden werden. Bei freien textuellen Editoren können diese Darstellungen durchaus identisch sein, da es sich um reine Zeichenketten handelt. Bei syntax-orientierten Editoren und allgemein bei Diagrammeditoren allerdings ist die interne Darstellung in einer leistungsfähigeren Datenstruktur - etwa in Form eines abstrakten Syntaxgraphen o.ä.- erforderlich. Der Editor muß dann die Fähigkeit haben, die externe Eingaben in die interne Darstellung umzuwandeln (parsing) und umgekehrt (unparsing).

Die interne Darstellung ist auch die Basis für die verarbeitenden Werkzeuge, wie Simulatoren und Codeerzeuger. ***Simulatoren*** veranschaulichen die Semantik der Dokumente durch deren (symbolische) Ausführung. ***Codeerzeuger*** erstellen Programmrahmen in verschiedenen Programmiersprachen, die dann im Zuge der Detailprogrammierung weiterverarbeitet werden.

2.3 CASE-Umgebungen

CASE-Werkzeuge unterstützen also in erster Linie das Edieren, Analysieren und Verarbeiten verschiedener Analyse- und Entwurfsdokumente, wodurch der Einsatz konkreter Entwicklungsmethoden zur Erstellung großer Systeme erst möglich wird.

Um diese Kernaufgaben herum stellen sich weitere Folgeaufgaben, die von den Werkzeugen in unterschiedlichem Umfang unterstützt werden. üblicherweise gehören dazu

- eine Projektverwaltung zur Planung, Steuerung und Überwachung von Entwicklungsprojekten
- eine Versionsverwaltung für die anfallenden Dokumente und
- eine Unterstützung der Zusammenarbeit mehrerer Entwickler.

Insgesamt erfordert der praktische Einsatz daher ganze ***CASE-Umgebungen***, die sich insgesamt auffassen lassen als integrierende Zusammenfassungen von unterschiedlichen Editor- und Analysekomponenten, evtl. ergänzt um weitere Teilwerkzeuge zur Methodenunterstützung.

Bei der ***Integration*** verschiedener Tools in einer Umgebung unterscheidet man die Integration

- an der Benutzeroberfläche,
- in der internen Repräsentation und
- in den Abläufen

In echten Umgebungen sind alle drei Aspekte der Integration berücksichtigt: Die Teilwerkzeuge haben das gleiche Aussehen an der Oberfläche und analoges Verhalten in all ihren Teilen. Sie arbeiten auf einem gemeinsamen Datenspeicher (Repository), in dem alle Dokumente nach dem gleichen Datenmodell abgelegt sind, und ein Wechsel zwischen den Teilwerkzeugen ist nahtlos möglich.

3. Meta-Werkzeuge

Gerade im Bereich der objekt-orientierten Analyse- und Entwurfsmethoden gibt es eine kaum noch überschaubare Menge von Büchern mit zwar verwandten, aber im Detail unterschiedlichen Ansätzen. Die Entscheidung für eine konkrete Analyse- oder Entwurfsmethode erfordert dann i.a. die Auswahl eines zugehörigen Werkzeugs und damit eine ***Festlegung*** in mehrfacher Hinsicht. So sind beispielsweise hierdurch das Aussehen der Symbole, die Syntax der Modellierungssprachen, die Analysebedingungen und auch die Codeerzeugungsstrategie bereits vollständig festgelegt.

Diese Festlegungen schränken den Einsatz von Werkzeugen stark ein. So wird die Weiterbearbeitung vorhandener Entwurfsdokumente, die Wiederverwendung von Dokumenten aus anderen Projekten und der Wechsel von Entwicklungsstrategien und Methoden erheblich erschwert.

Meta-Werkzeuge können diese Probleme teilweise lösen: Meta-Werkzeuge sind Werkzeuge zur Erzeugung von ***Werkzeug-Instanzen***, die den spezifischen Anforderungen der Benutzer in einem konkreten Umfeld entsprechen. Unter Erzeugung wird dabei die zumindest halbautomatische Entwicklung verstanden. Ein MetaCASE-Werkzeug ermöglicht es also, angepaßte CASE-Werkzeuge zu erstellen.

3.1 Klassische Meta-Werkzeuge

In der Entwicklung von Systemen für textuelle Sprachen haben Meta-Werkzeuge eine lange Tradition.

Compiler-Generatoren - wie beispielsweise das in Unix-Systemen vorhanden Scanner-Parser-Generator-Paar `lex` und `yacc` [AhSeUl86] - erlauben es, aus der Beschreibung von Sprachen mit Hilfe regulärer Ausdrücke und kontext-freien Grammatiken in Backus-Naur-Form (BNF) Werkzeuge zu generieren, die genau die beschriebene Sprache zu verarbeiten (beispielsweise zu übersetzen) geeignet sind.

Was für Compiler möglich ist, ist grundsätzlich auch für Editoren möglich. So gibt es auch ***Editor-Generatoren***. Beispielsweise erlaubt der Synthesizer Generator [RepTei84] erlaubt die Generierung von syntax-orientierten Editoren aus (abstrakten) attributierten Grammatiken, die

durch spezielle Regeln für die Eingabe der lexikalischen Einheiten und durch Annotationen ergänzt werden, die das Layout der edierten Texte festlegen. Diesen Entwicklungen liegt ein sehr allgemeines Konzept attributierter Grammatiken zugrunde.

3.2 Klassen von Meta-Werkzeugen

Diese klassischen Meta-Werkzeuge sind ***Generatoren***, d.h. sie erzeugen aus der Werkzeugbeschreibung eine Datenstruktur, welche dann in ein Rahmenprogramm integriert wird. Dieses Rahmenprogramm interpretiert die Datenstruktur und stellt so das konkrete erzeugte Werkzeug dar. Das angepaßte Werkzeug besteht also aus einem festen (instanzunabhängigen) und einem variablen (instanzabhängigen) Teil.

Wird diese Zweiteilung genutzt, um auch in der Werkzeuginstanz noch Änderungen am variablen Teil durchzuführen, so handelt es sich um ein ***reflexives Werkzeug***. Reflexiv heißt hier, daß es seine eigene Beschreibung edieren kann. Gewisse reflexive Eigenschaften in dem Sinne, daß (kleinere) Teile des Aussehens und Verhaltens in speziellen Interaktionsboxen gesetzt werden können, haben heute die meisten Werkzeuge. Diese Tatsache zeigt insbesondere, daß die Unterscheidung in Werkzeuge und Meta-Werkzeuge letztendlich fließend ist.

Die Unterscheidung zwischen Generatoren und reflexiven Werkzeugen wird durch den ***Wirksamkeitszeitpunkt*** für die einzelnen Änderungen festgelegt. Bei reflexiven Werkzeugen wird eine Änderung sofort wirksam - mit der Gefahr, daß inkonsistente Zwischenzustände entstehen können - während bei Generatoren die Änderung erst bei der Durchführung des Generierungsschritts wirksam werden, was allerdings zur Folge hat, daß auch für kleine Anpassungen evtl. ein höherer Aufwand zu treiben ist.

Eine Zwischenstellung nehmen ***generische Werkzeuge*** ein. Ein Werkzeug ist generisch, wenn es die instanzabhängige Information zum Zeitpunkt des Startes als Parameter oder Parameterdatei mitbekommt, um dann das Verhalten zu zeigen, das durch die Parametrisierung spezifiziert ist.

In Meta-Werkzeugen wird stets eine allgemeine Grundstruktur für die Instanzen vorgegeben, von der dann einzelne Teile vom Anwender konkret anpaßbar sind. Man kann sich also eine konkrete CASE-Umgebung mit Hilfe eines MetaCASE-Werkzeugs selbst konfigurieren (customizability). Damit unterscheiden sich MetaCASE-Werkzeuge von ***CASE-Toolkits*** oder ***CASE-Frameworks***, die zwar dem Entwickler von CASE-Umgebungen eine mächtige Entwicklungsgrundlage zur Verfügung stellen, aber letztendlich doch die Zusammenstellung einer ganzen Umgebung durch Programmierung - wenn auch auf hohem Niveau - erfordern.

Eine erste grobe Unterscheidung ist also möglich in

- Generatoren,
- reflexive Werkzeuge,
- generische Werkzeuge und
- Werkzeugbausätze.

3.3 Instanzunabhängige Eigenschaften

Grundsätzlich sind der Anpaßbarkeit von Werkzeugen keine Grenzen gesetzt. Allerdings werden üblicherweise alle Aspekte der ***Benutzeroberfläche***, d.h. der Stil und das "Look-and-Feel" des Werkzeugs eher von der Plattform bestimmt, auf der es eingesetzt wird, so daß dies nicht zur Einteilung von Meta-Werkzeugen herangezogen wird.

Desgleichen ist das Datenmodell d.h. die Art der Datenstrukturen im Repository einschließlich des Repository-Verwaltungssystems (des zugrundeliegenden Datenbanksystems) eine Eigenschaft des Werkzeugs, die nicht weiter verändert werden kann, von der aber die Leistungsfähigkeit des Werkzeugs zentral abhängt.

Die Zusammenbindung in einer gemeinsamen Umgebung unter Einbeziehung

- der Projektverwaltung
- der Versionsverwaltung für die Dokumente und
- der Unterstützung der Zusammenarbeit mehrerer Entwickler

ist ebenfalls üblicherweise mit dem Tool selbst festgelegt. Gleiches gilt für die Offenheit der Umgebung, d.h. für die Schnittstellen für eigene Erweiterungen durch die Anwender.

3.4 Instanzabhängige Eigenschaften

Entscheidend für die Anpaßbarkeit und damit für die MetaCASE-Eigenschaft ist eher, für welche textuellen oder visuellen Dokumentensprachen Editoren und Analysatoren zur Verfügung stehen. Die ***Sprachbeschreibung*** betrifft

- die Lexik, z.B. die graphische Formen,
- die kontextfreie Syntax, z.B. Knoten- und Kantenarten und Inzidenzen, und
- die kontextsensitive Syntax, z.B. Struktureinschränkungen für Diagramme,

jeweils bezogen auf die Editoren und die Analysemöglichkeiten.

Hinzu kommt die reine ***Funktionalität*** der Editoren, d.h.

- die Menüstruktur und
- die Operationen im einzelnen.

Die Integration der verschiedenen Editoren in einem gemeinsamen ***Repository*** erfordert ferner eine Festlegung auch

- der Beziehung zwischen der internen Repräsentation und der externen Repräsentation (parsing/unparsing) incl. Layoutberechnungen und
- der dokumenten-übergreifenden Konsistenzbedingungen, d.h. der Anforderungen an die Zusammenhänge der verschiedenen Dokumente zueinander.

Die semantischen Aspekte werden berücksichtigt durch die Definition von

- der Verfahren der Codeerzeugung aus der vorhandenen internen Repräsentation heraus,
- bzw. der Beschreibung von Simulationsschritten auf der internen Repräsentation.

Die Integration der Ablaufaspekte der einzelnen Teilwerkzeuge erlaubt dann eine Angabe der

- Methodenunterstützung

4. Zusammenfassung

Die Darstellung in dieser Arbeit sollte zeigen, wie durch Variierung verschiedener Aspekte von CASE-Umgebungen aus MetaCASE-Umgebungen konkrete Instanzen gewonnen werden können. Aus der Auffassung von CASE-Umgebungen als (formal-)sprach-verarbeitende Systeme wurden die verschiedenen Aspekte herausgearbeitet, die geeignet sind, in unterschiedlichen Instanzen variiert zu werden.

Hierdurch ergibt sich eine Taxonomie, die es erlaubt, konkrete MetaCASE-Ansätze zu klassifizieren, und so zu einer Beurteilung der MetaCASE-Fähigkeiten konkreter Werkzeuge -wie beispielsweise ToolBuilder [Alde91] und KOGGE [EbSüUh97] als Generatoren, MetaEdit+ [RoGuSm92] als generisches Werkzeug oder ToolFrame [Däbe95] als Werkzeugbausatz - herangezogen werden kann.

Literatur

[AhSeUl86] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison Wesley, Reading, MA, 1986.

[Alde91] Albert Alderson. *Meta-CASE Technology*. in: A. Endres, H. Weber (Eds.): Software Development Environments and CASE Technology. Springer, Berlin, LNCS 509, 1991, 81-91.

[Däbe95] Dirk Däberitz . *ToolFrane – an Open Extensible CASE-Tool Environment. User Manual*. Univertät Siegen., Fachbereich Elektrotechnik und Informatik. Interner Bericht.

[EbSüUh97] Jürgen Ebert, Roger Süttenbach, Ingar Uhe. *Meta-CASE in Practice: a Case for KOGGE*. in: Proc. CAiSE'97, Springer, Berlin, 1997
[Harel 88] D. Harel. *On visual formalisms*. Communications of the ACM, 31:514-530, 1988.
[RepTei84] T. Reps, Tim Teiltelbaum. *The Synthesizer Generator*. SIGPLAN Notices 19(5):42-48, 1984.
[RoGuSm92] Matti Rossi, Mats Gustafsson, Kari Smolander, Lars-Ake Johansson, Kalle Lyytinen. *Metamodeling Editor as a Front End for a CASE Shell*. Proc. CAiSE'92, Springer, LNCS 593, 1992, 546-567. s.a. http://www.jsp.fi/metacase

Autor:

Prof. Dr. Ebert, Jürgen
Universität Koblenz-Landau
Rheinau 1
56076 Koblenz
Tel.: (0261)9119-412, Fax: (0261)9119-499
email: ebert@informatik.uni-koblenz.de