FLEXOB - Entwicklungstool für dynamische modellbasierte CAD-Systeme

Wehner, R., Steinmann, F., Hübler, R., Bauhaus-Universität Weimar

EINFÜHRUNG

Die heute vorherrschende Vorgehensweise bei der Softwareproduktion sieht eine klare Trennung zwischen Analyse des Anwendungsgebietes (OOA) und dem Entwurf der geplanten Anwendungsprogramme (OOD) vor. Dabei wird von der Annahme ausgegangen, daß das Modell nur detailliert genug sein muß, um alle potentiellen Einsatzfälle abzudecken. Es wird eine zeitliche Unveränderlichkeit der modellierten Domäne über den Nutzungszeitraum vorausgesetzt. Diese Voraussetzungen sind für viele Anwendungsgebiete jedoch nicht gegeben, wie die hohe Updatefrequenz und die damit verbundenen Kompatibilitätsprobleme zeigen. Bei einer Reihe von Anwendungsgebieten ist es darüber hinaus wünschenswert, daß der Nutzer selbst das zugrunde liegende Modell erweitern und anpassen kann. Derartige Anwendungsgebiete sind :

- VEREINIGEN UNTERSCHIEDLICHER MODELLWELTEN

  Bei Facility-Management Systemen werden Daten u.a. von CAD-, Betriebswirtschafts- und Gebäudesteuerungssystemen benötigt, die wiederum aus verschiedenen Quellen stammen können. Ziel ist es, durch Abgleich der einzelnen Sichten ein Gesamtmodell zu erzeugen. Es müssen zur Programmlaufzeit neue Klassen erzeugbar bzw. vorhandene erweiterbar sein [PFE96].

- CAD-SYSTEME FÜR STARK SUBJEKTIVE ANWENDUNGSGEBIETE

  Besonders künstlerisch-kreative Anwendungsgebiete, wie der architektonische Vorentwurf, sind subjektiv geprägt und naturgemäß mit einem hohen Maß an Vagheit und Unschärfe behaftet. Sie unterliegen Modeströmungen und privaten 'Vorlieben'. Daher sind *CADesign*-Systeme für solche Nutzer nur akzeptabel, wenn sie den Spielraum für Kreativität nicht zu sehr einengen. Entsprechende Systemlösungen sollten den Entwurfsprozeß im Sinne eines *'design assistant'* begleiten und dabei „ ... ein Mindestmaß an Systematik und Plausibilität" ([STE97]) sichern. Das vollständige Vordenken derartiger Systeme scheint unmöglich. Ein solches Programm wäre z.B. FUNPLAN ([HÜB95])

- PERSONALISIERBARE CAD-SYSTEME

  Viele Aufgaben der Gebäudeverwaltung sind für alle Gebäudetypen prinzipiell gleich. Es erscheint daher sinnvoll, solche Programme auf einem vereinheitlichten Modellkern zu implementieren und sie dann dynamisch an das jeweils konkret zu verwaltende Objekt anzupassen. (Facility Management Systemkern KOPERNIKUS[1])

- LANGZEITSPEICHERUNG VON PROJEKTDATEN

  Ein CAD-System sollte in der Lage sein neben den eigentlichen Projektdaten auch die diese Daten beschreibenden Schemata zu speichern. So wird eine (sehr viel) spätere Interpretation zumindest erleichtert.

- ERWEITERTER MODELLDATENAUSTAUSCH

  In der Praxis werden Projekte meist hochgradig arbeitsteilig entwickelt. Die Datenübertragung zwischen verschiedenen Systemen ist dabei meist nur verlustbehaftet möglich. In [KOL97] wird ein Ansatz beschrieben, wie dieser Problematik auf der Basis einer genormten dynamischen Objektschnittstelle (AKO) begegnet werden kann. Gegenstand der Normung ist dort kein Domänenmodell (wie etwa im Rahmen der IFC-Entwicklung[2]) sondern ein Metamodell, das semantikfreie Objekte des genormten Modellstrukturierungsparadigmas *'Objektorientierung'* austauscht.

ANFORDERUNGEN

Primäre Aufgabe des implementierten Modellierkerns ist die Unterstützung der Produktmodellentwicklung und -dynamisierung. Ein relevanter Weltausschnitt wird mit Mitteln der Objektorientierten Modellierung beschrieben, die Elemente der Beschreibung (*Klassen-* und *Instanzobjekte*) werden durch das System lediglich verwaltet, nicht aber maschinell interpretiert oder generiert. Die in der Künstlichen Intelligenz dominierenden interpretierenden Programmiersysteme sind vorwiegend für den Bau von Expertensystemen ausgelegt und verfügen zusätzlich zur Objektverwaltung über eine Vielzahl weiterer Komponenten wie Inferenzmaschinen, Regelinterpreter oder Debuggingtools. Zur ausschließlichen Nutzung für Objektverwaltungsaufgaben sind sie zu mächtig. Am bisher für diesen Zweck eingesetzten experimentellen Basissystem ALOS[3] haben sich insbesondere die ungenügende Laufzeiteffizienz sowie das Fehlen eines erweiterten Attribut- und Relationenkonzepts als einschränkend erwiesen. Bei der Weitergabe von ALOS-Programmen ist darüber hinaus die existentielle Abhängigkeit von AutoCAD problematisch. Im Ergebnis der mit ALOS gemachten Erfahrungen wurde mit der vorliegenden Neuentwicklung ein flexibler Objektverwaltungsmodul realisiert, der in seinem Funktionsumfang auf das eingegrenzte Anforderungsspektrum passend zugeschnitten ist und die geforderten Leistungsmerkmale bei möglichst hoher Laufzeiteffizienz erreicht.

Der hier verfolgte Ansatz der Produktmodellierung geht von der Annahme aus, das es nicht möglich ist, in einer der Nutzung vorgelagerten Phase (Produktion eines CAD-Systems) ein *allumfassendes* statisches Domänenmodell zu erstel-

[1] HochTief Software, Frankfurt/M.
[2] 'Industry Foundation Classes', Normungsbemühung der IAI für ein C++ Klassensystem
[3] AutoLISP-Objektsystem [STE93]

len, das über die gesamte Lebensdauer der zu modellierenden Objekte Bestand hat. Das gilt insbesondere für die Bauwerksmodellierung. Daraus ergibt sich die Notwendigkeit nach interaktiver, dynamischer Erweiterbarkeit und Änderbarkeit der Modelle. HEINECKE ([HEI94]) stellt fest :

"Zu einem echten Werkzeug der Konstrukteure (kann) der objektorientierte Aufbau erst dann werden, wenn diese die Möglichkeit erhalten, neue Klassen und Objekte interaktiv zu erzeugen... Wie solche Möglichkeiten realisiert werden können, ohne daß die Konstrukteure vertiefte Programmierkenntnisse benötigen, ist dabei noch weitgehend offen...Die Forderung nach interaktiven Änderungen und Ergänzungen der Klassen- und Vererbungsstruktur durch die Konstrukteure wirft neben softwareergonomischen auch softwaretechnische Fragen auf, da bei vielen Sprachen die Klassen- und Vererbungsstruktur zur Compilezeit festgelegt und nicht zur Laufzeit verändert werden kann."

Diese Feststellung hat weitgehende Konsequenzen für das Systemdesign des zugrundeliegenden Modellierwerkzeuges. In einer empirischen Studie vergleicht STEINMANN [STE95] verschiedene Systeme hinsichtlich der Erfüllung bestimmter charakteristischer Anforderungen an eine dynamische Objektverwaltung (Tab.1).

Anforderungen	**C++**	**ONTOS**	**ALOS**	**FLEXOB**
Portabel und verfügbar (auch PC!)	+	**O**	**O**	**O**
Genügend effizient	+	+	-	+
Wertvererbung	-	**O**	+	+
Laufzeitdynamische Objektstrukturen (Löschen, Erzeugen von Klassen)	-	**O**	+	+
Umklassifizieren von Instanzen/Subklassen	-	-	+	+
Unterstützung von Attributobjekten (Facetten)	-	-	-	+
Unterstützung von Relationenobjekten (Facetten)	-	-	-	+
Löschen, Erzeugen von Attributen/Relationen	-	+	+	+
Unterstützung der Kompositionshierarchie	-	**O**	+	+
Weitgehende Objekt-, Taxonomie- und Aggregationsanalyse	-	-	+	+

Tab.1 Anforderungen an Objektverwaltungssysteme

ERWEITERTES ATTRIBUT- UND RELATIONENKONZEPT

Eine weitere Erhöhung der Modelliermächtigkeit ergibt sich durch die Verallgemeinerung des Konzepts der Objektorientierung auf Attribute und Relationen. Unter einem Attribut- oder Relationenobjekt soll in FLEXOB die Aggregation von Facetten verstanden werden, denen ggf. noch prozedurale Erweiterungen zugeordnet werden können. Jedes Slotobjekt (Verallgemeinerung von Attributen und Relationen) besteht also aus mindestens einer Facette, die den eigentlichen Slotwert (*value facet*) repräsentiert. In weiteren Facetten können beliebige Annotationen zum 'value facet' angegeben werden. Diese Vorgehensweise hat gegenüber atomaren Attributen (terminale Datentypen der zugrundeliegenden Wirtssprache) und Relationen (namentliche Referenzen, Adreßreferenzen) mehrere Vorteile:

- durch (beliebige) Facettierung lassen sich domänenspezifische Zusatzinformationen ablegen (Maßeinheiten, Wertemengendeskriptoren, ...),
- die Zugriffsmethoden können Prozeduren zur Konsistenzsicherung, Plausibilitätskontrolle, Zugriffsüberwachung etc. triggern,
- Modellierung von attributiver und struktureller (relationaler) Unschärfe,
- die gesamte Semantik ist in den Attribut- und Relationenobjekten gekapselt; Klassen- und Instanzobjekte verwalten nur interpretationsneutrale Referenzen,
- Zuordnung angepaßter Dialogmasken für die alphanumerische Ein- und Ausgabe sowie individueller Visualisierungsmethoden,
- automatische Generierung inverser Relationen für antisymmetrische Relationen
- Modellierung gerichteter, ungerichteter, bewerteter, ... Relationen.
- Implementierung von Containern für nichtalphanumerische Attribute, die insbesondere das informale Wissen repräsentieren (Bilder, Klänge, Texte, ...).

Obwohl es prinzipiell möglich ist, alle vorkommenden Relationen durch Relationenobjekte zu modellieren, wurde hier ein anderer Weg beschritten. Relationen, die Bestandteil des zugrundeliegenden OO-Paradigmas sind, wie:

- *Generalisierung/Spezialisierung* (superclass_of/subclass_of),
- *Instanziierung/Klassifizierung* (instance_of/class_of),
- *Aggregation/Zerlegung* (part_of/has_parts),

deren Semantik also a priori bekannt ist, werden aus Effizienzgründen explizit vom Objektverwaltungsmodul unterstützt. Alle anderen Relationen, deren Semantik der Nutzerinterpretation unterliegt, werden durch Relationenobjekte realisiert (allgemeiner Relationentyp *Assoziation*).

DIE ZWEIPHASIGE NUTZUNGSKONZEPTION

Während von Klassen zur Laufzeit durch dynamische Spezialisierung Subklassen abgeleitet werden können, existiert diese Möglichkeit für Attribut- und Relationenklassen nicht. Hier muß die gesamte, für eine bestimmte Anwendungsdomäne geforderte Varietät mittels statischer C++-Klassen implementiert werden (Abb. 1).

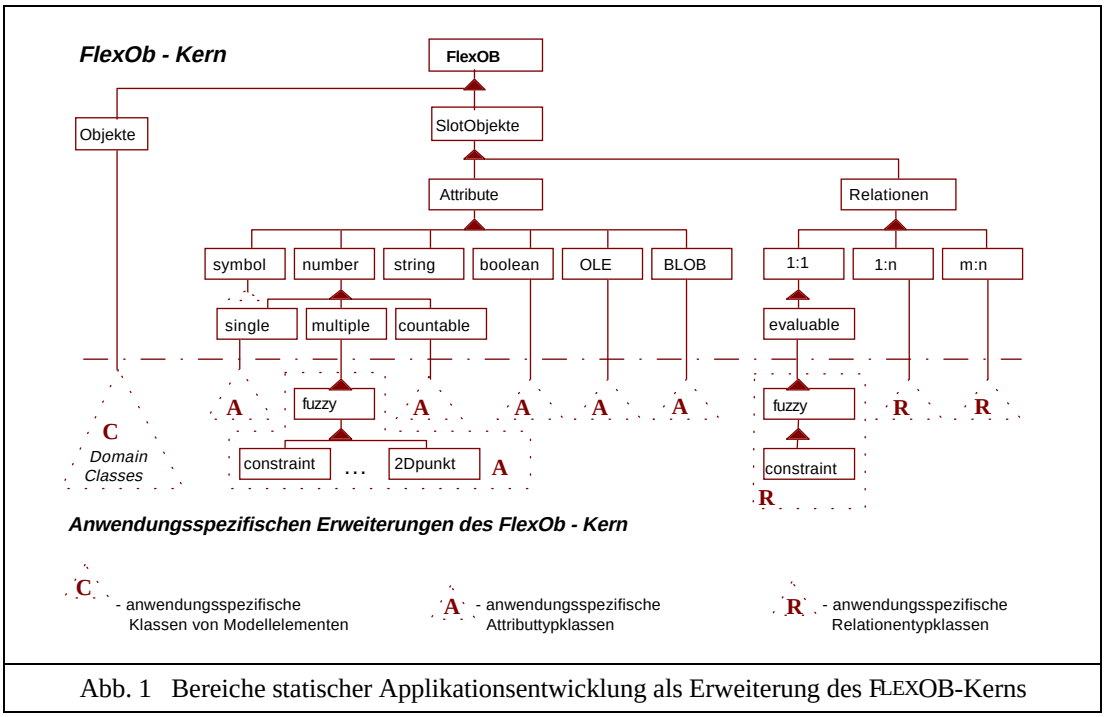


Abb. 1 Bereiche statischer Applikationsentwicklung als Erweiterung des FLEXOB-Kerns

Daraus resultiert eine semidynamischen Arbeitsweise des Modellierkerns und seine Nutzung zerfällt in zwei Phasen:

I. STATISCHE PHASE (COMPILEZEITPHASE)
Diese Phase dient der domänenspezifischen Konditionierung des Modellierers in einem der Programmlaufzeit vorgelagerten Stadium. Slotklassen, die für eine bestimmten Anwendungsdomäne benötigt werden, sind als C++-Klassen zu implementieren. Ebenso muß der Applikationsprogrammierer alle die (wurzelnahen) Klassen der Taxonomie implementieren, die:

- Methoden enthalten, deren Semantik sich von der ererbten unterscheiden,
- Methoden für Berechnungen über Objektmengen enthalten,
- Methoden für Iterationen über Aggregationsstrukturen enthalten.

Es entsteht ein C++-Klassensystem, das für die Nutzung in der nächsten Phase dynamisiert wird.

II. DYNAMISCHE PHASE (LAUFZEITPHASE)
In dieser Phase erfolgt die eigentliche Domänenmodellbildung durch Eingabe des zugehörigen deskriptiven Wissens. Das Modell kann durch dynamische Spezialisierung bereits in der Taxonomie definierter Klassen erweitert werden. Attribut- und Relationenobjekte können durch Instanziierung ihrer Klassen gebildet, mit Werten belegt und zugeordnet werden.

Projektwissen wird durch Nutzung des Domänenmodells, d.h. durch Instanziierung von Modellklassen, ggf. Änderung der ererbten Standardwerte und vor allem durch den Aufbau einer (heterarchischen) Aggregationstruktur erzeugt.

FLEXOB IMPLEMENTIERUNGSDETAILS

FlexOB wird in Form einer C++-Klassenbibliothek bereitgestellt, deren Aufbau in Abb. 2 dargestellt ist. Die angedeuteten Teilbäume bilden den vom FlexOB-Applikationsprogrammierer zu implementierenden statischen Teil des domänenspezifischen Objektmodells.
Für die Implementierung wurde dabei von folgenden Annahmen und Prämissen ausgegangen:

- die Wissensorganisation erfolgt in Taxonomien mit einfacher Vererbung
- die Erweiterbarkeit durch den FlexOB-Applikationsprogrammierer soll einfach sein
- es soll nur beschreibendes aber kein ablaufsteuerndes Wissen dynamisch erzeugbar sein
- es wird eine möglichst hohe Laufzeiteffizienz auch bei großer Anzahl zu verwaltender Objekte angestrebt
- alle zu verwaltenden Objekte finden im Hauptspeicher Platz[4]
- Objekte haben starke Objektidentität (persistente Surrogat-Objektidentifikatoren)
- es soll eine hohe Zugriffseffizienz unter der Annahme $N_{Destruktion} < N_{konstruktion} << N_{Zugriffe}$ erreicht werden.

[4] durch die dezidierte Schnittstelle zum FLEXOB-Kern laßt sich das ohne Auswirkungen für Applikationen ändern

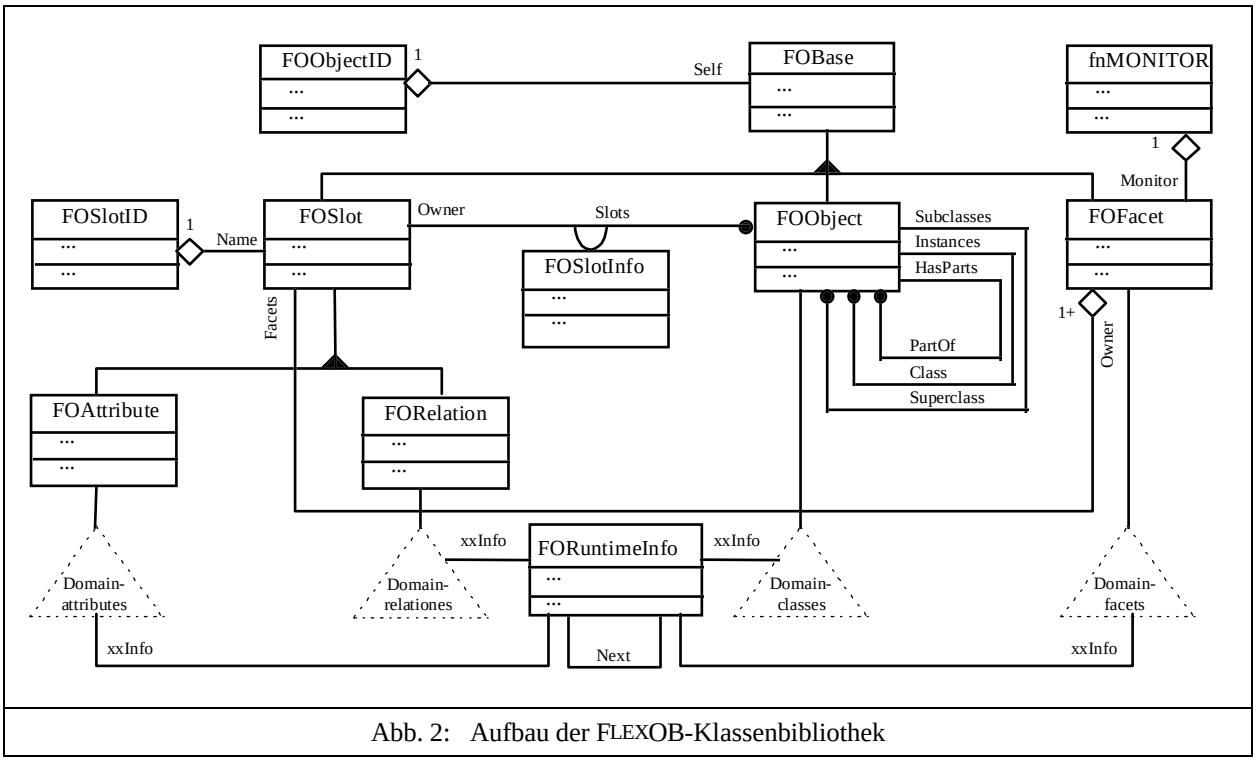


Abb. 2: Aufbau der FLEXOB-Klassenbibliothek

Der FlexOB-Kernel besteht aus einer Menge orthogonaler Elementarfunktionen zur Bereitstellung der Grundfunktionalität zur Objektverwaltung. Komplexe Funktionen lassen sich auf Folgen solcher Elementarfunktionen abbilden. Diese Basisfunktionalität unterteilt sich in die Kategorien:

- Konstruktionsfunktionen (Erzeugen von Klassen-, Instanz-, Attribut- und Relationenobjekten)
- Destruktionsfunktionen (Entfernen von Klassen-, Instanz-, Attribut- und Relationenobjekten)
- Zugriffsfunktionen (Zuweisen oder Rückgewinnen von Werten)
- Informationsfunktionen (Abrufen von Informationen zum Objekt).

Dazu gehören die Bereitstellung eines effizienten Mechanismus für die strukturelle- und Werteerbschaft, die Identifikation von Slots und das Herstellen der Laufzeitbindung, eine konsistente Namensvergabe und -verwaltung, das Herstellen von Objektpersistenz und Objektidentität sowie die Erzeugung des initialen Objektbaumes aus dem statischen C++-Domänenklassensystem (Prototyperzeugung).

KLASSEN- UND INSTANZOBJEKTE

Klassen- und Instanzobjekte werden von der Klasse FOObject repräsentiert, von der alle Applikationsklassen abzuleiten sind. Sie stellt Funktionen bereit, die wesentliche dynamische Eigenschaften von FLEXOB bestimmen. Der Nutzer kann zur Programmlaufzeit umfangreiche Informationen zur Struktur- und Aggregationsanalyse ermiteln, wie:

- Super- und Subklassen einer Klasse
- Instanzen einer Klasse, sowie die Klasse einer Instanz
- Attribute und Relationen (Slots) einer Klasse oder Instanz
- Analyse der Aggregationsstruktur einer Instanz bzw. seines Komplexes.

Weiterhin stellt FOObject alle Funktionen zur Slotverwaltung bereit. Dadurch wird der Nutzer in die Lage versetzt, zur Laufzeit Slots zu seinen Klassen- und Instanzobjekten hinzuzufügen, die in den statischen Domänenklassen (Zeitpunkt der Applikationsentwicklung) noch nicht bekannt waren (dynamische Spezialisierung).

ATTRIBUT- UND RELATIONENOBJEKTE

Attribut- und Relationenobjekte werden durch die Klasse FOSlot repräsentiert. Dabei stellen Facetten die eigentlichen Container für Werte im Sinne der Wirtssprache dar. Sie werden über ihre Namen referiert. Daher liegt die Hauptaufgabe dieser Klassen in der Definition eines einheitlichen, vom konkreten Facetten-Typ unabhängigen, Interfaces und der Bereitstellung von Basisfunktionalität für:

- FACETTENVERWALTUNG: Die Facetten eines Slotobjektes werden während dessen Erstellung erzeugt und in eine Facettentabelle eingetragen. Für $N_{Facets}<10$ ist die Verwaltung in Arrays hinreichend effizient.
- FACETTENZUGRIFF: Als einheitliches Interface zum Eintragen von Werten in und zum Holen von Werten aus Facetten dienen die Methoden GetFacet() und SetFacet(). Durch geordnetes Erzeugen in Abhängigkeit von der erwarteten Zugriffsfrequenz, läßt sich die mittlere Zugriffszeit minimieren.

- METAINFORMATIONEN: Für jedes Slotobjekt können zur Laufzeit direkt Anzahl und Namen seiner Facetten und mittelbar deren Typ ermittelt werden.
- WERTEERBSCHAFT: Ergänzend zur strukturellen Erbschaft werden hier die konkreten Wertbelegungen von Eigenschaften vererbt. Dazu wird ein laufzeit- und speichereffizienter Mechanismus bereitgestellt.

Der überwiegende Teil dieser Funktionen kann in allen abgeleiteten Slotklassen ohne Änderung verwendet werden. Damit ist eine leichte Erweiterbarkeit des Attribut- und Relationensystems durch den FlexOB-Applikationsprogrammierer gewährleistet.
Zur Erfüllung spezifischer Aufgaben ist es möglich, an jede Facette Monitorfunktionen zu binden, die durch Zugriffe auf diese Facette getriggert werden. So kann eine Konsistenzsicherung oder Zugriffsüberwachung implementiert werden. Ebenso ist es möglich bei lesendem Zugriff eine Monitorfunktion zu triggern, die statt eines statischen Facettenwertes das Resultat einer Berechnungsvorschrift zurückgibt.

Es besteht prinzipiell die Möglichkeit, wie bei Klassenobjekten auch, neue Slotklassen durch dynamische Facettierung zu erzeugen. Im Interesse einer einfachen Handhabbarkeit wird hiervon jedoch kein Gebrauch gemacht.

ERBSCHAFT

Durch die Verwendung von Attribut- und Relationenobjekten kann ein Erbschaftsalgorithmus implementiert werden, der die Vorteile des schnellen Zugriffs der statischen Erbschaft (eager inheritance) und der rationellen Datenhaltung der dynamischen Erbschaft (lazy inheritance) vereinigt. Jedes Klassen- bzw. Instanzobjekt verfügt über eine lokale Slot-tabelle[5], die aus Paaren (Slotidentifikator, Slotreferenz) bestehen. Beim Erzeugen eines neuen Objektes wird diese Tabelle von der Superklasse bzw. Klasse kopiert. Damit erbt das neue Objekt alle Eigenschaften seines Superobjektes. Es werden also nicht die Slotobjekte selbst kopiert (wie bei statischer Erbschaft), sondern nur Referenzen auf sie. Beim Zugriff auf einen Slot wird über den Slotidentifikator aus der Slottabelle die Adresse des im Scope des zugreifenden Objektes liegenden Slotobjektes ermittelt (Laufzeitbindung). Selbst wenn die Definitionsstelle der ererbten Slots in der Objekthierarchie weit oben liegt, erfolgen Zugriffe auf diese immer einstufig. Suchprozesse in der Hierarchie aufwärts, wie sie bei klassischer dynamischer Erbschaft (etwa in ALOS) auftreten, entfallen hier völlig (Abb. 3).
Bei schreibendem Zugriff wird zunächst festgestellt, ob der entsprechende Slot für das den Schreibvorgang initiierende Objekt lokal definiert ist. Ist das der Fall, kann der neue Wert direkt geschrieben werden, ohne das ein Propagationsaufwand entsteht. Im anderen Fall wird der ererbte Slot dupliziert und die Wertänderung auf der Kopie ausgeführt. Im gesamten Teilbaum unter dem schreibenden Objekt muß jetzt die Referenz auf das Original durch eine Referenz auf die Kopie ersetzt werden. Es fällt hier ein Propagationsaufwand an, der sich jedoch auf die Aktualisierung der Slotreferenzen beschränkt.
Dieses Vorgehen ist insbesondere unter der realistischen Annahme $N_{lesen\_Slot} >> N_{schreiben\_Slot}$ effizient. Beim Neuanlegen eines Slots muß in allen Klassen- bzw. Instanzobjekten des Teilbaums, dessen Wurzel das erzeugende Objekt ist, ein neues Paar (Slotidentifikator, Slotreferenz) in dessen Slottabelle eingetragen werden. Auch hier werden nur Referenzen, nicht aber ganze Slotobjekte kopiert.

OBJEKTIDENTITÄT

Objektidentität soll die eindeutige Unterscheidbarkeit eines Objektes von allen anderen Objekten sicherstellen und steht daher insbesondere in engem Zusammenhang mit der Problematik der Persistenz. Nach [HUG91] können an die Implementierung der Objektidentität verschiedene Anforderungen wie Unabhängigkeit vom Ablageort (location independence), Unabhängigkeit vom Wert (value independence) und Unabhängigkeit von der Struktur (structure independence) gestellt werden. Bestimmte Funktionen, wie die Trennung von Modell und Projekt oder die Vereinigung mehrerer Projekte, lassen sich überhaupt erst sinnvoll über das Konzept einer starken Objektidentität realisieren. In FlexOB werden zur Herstellung globaler Eindeutigkeit 10 Byte lange maschinengenerierte Surrogat-Objektidentifikatoren verwendet, die alle obigen Anforderungen gleichermaßen gut erfüllen. Jedem Klassen-, Instanz-, Attribut-, Relationen- und Facettenobjekt wird zum Zeitpunkt seiner Generierung ein solcher Identifikator zugewiesen, den es über seine gesamte Lebenszeit behält. Durch ausschließliche Verwendung nur lesender Zugriffe wird die Invarianz gegenüber Fremdeinwirkungen sichergestellt. Aus Laufzeitgründen werden diese Objektidentifikatoren jedoch nur zur Identifikation persistenter Objekte benutzt. Für transiente Objekte wird wie bei den meisten Systemen Objektidentität durch Adressierung hergestellt.

[5] Zur Sicherung der Laufzeiteffizienz auch bei hoher Zugriffsfrequenz wurde für die Implementierung die Brent'sche Variante des offenen Hashing gewählt. Solche Tabellen garantieren auch bei hohem Füllgrad im Mittel 2,5 Suchoperationen im Falle der erfolgreichen Suche.

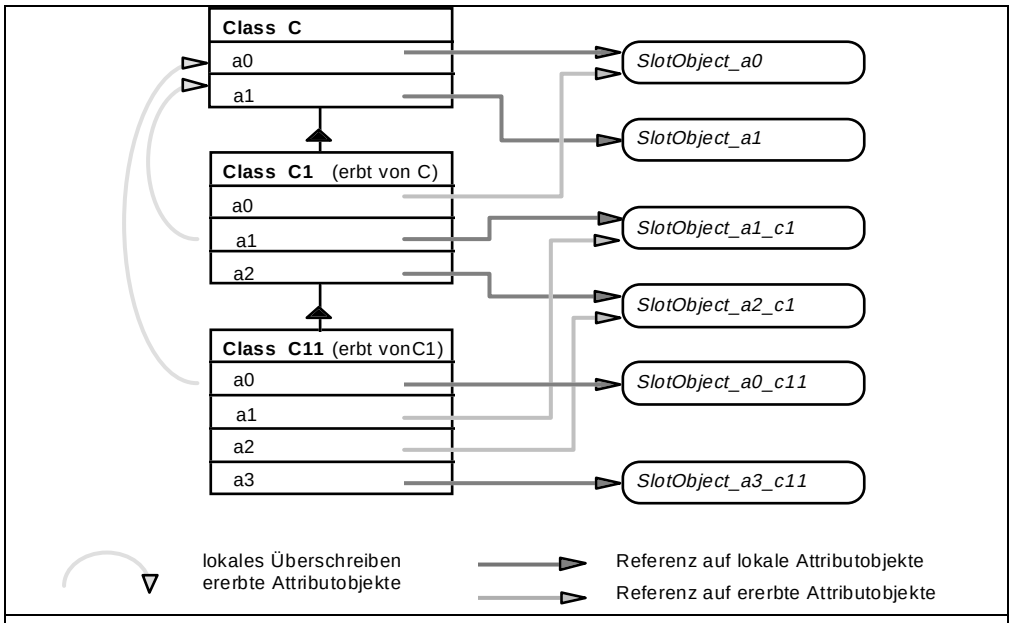


Abb. 3 Erbschaft durch Propagation von Slotobjektreferenzen

FLEXOB NUTZUNGSMODI

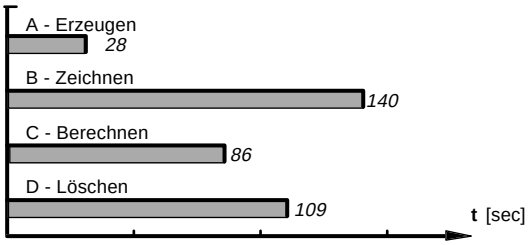
FlexOB ist durch Applikationsprogrammierer in zwei Varianten nutzbar als:

- C++ KLASSENBIBLIOTHEK: Der Applikationsprogrammierer bindet die FlexOB-Klassenbibliothek in seinen Applikationscode ein. Gegenüber der Nutzung anderer Klassenbibliotheken ergeben sich hier keine Besonderheiten. Diese Variante kommt in Frage, wenn auch die Implementierung des Anwendungsprogrammes in der Sprache C++ erfolgt.
- OBJEKTSERVER: Diese Variante wurde insbesondere für die Nutzung in anderen Programmiersprachen geschaffen. Der FlexOB-Objektverwaltungsmodul wird hier in der Form einer Windows-DLL verwendet. Dazu wird in einer Schnittstellenschicht ein Interface definiert, auf dem die Applikationsprogramme aufsetzen können. Dabei hat es sich als günstig erwiesen, ein klassisches C-Funktioneninterface zu implementieren, da die meisten Sprachen keine C++-Klassen importieren können. Die Hauptaufgabe dieser Interfacefunktionen liegt dann in der Umsetzung von Funktionsaufrufen in die C++-Methodenaufrufe (wrapperfunctions) der entsprechenden FlexOB-Objekte. Diese Schnittstelle wurde erfolgreich zur Kopplung von FlexOB mit einem in der Programmiersprache Delphi 2.0 implementierten graphischen Viewer (FLEXEDIT, [LAN96]) sowie zur Überwindung der „Sprachbarriere" zwischen C++-Systeme verschiedener Hersteller (Borland C++ 5.0, Microsoft C++ 4.0) eingesetzt.

BEWERTUNG

Der mit Hilfe von FLEXOB implementierte generische Objekt- und Schemaeditor FLEXEDIT weist dessen Funktionsfähigkeit nach. In einem Testrahmen einer dynamischen Modellwelt für geometrische Objekte konnte auch für größere Objektmengen ein effizientes Laufzeitverhalten nachgewiesen werden (siehe Abb. 4b).
Es bleibt zu bemerken, daß ein zusätzlicher Aufwand für Programmierung und Einarbeitung entsteht. Dies betrifft vor allem die Phase **I** der Anpassung des statischen FlexOB-Kerns zur Applikationsentwicklung. Dies mag sich für Applikationen die klein oder statisch programmierbar sind nicht lohnen. Immer dann, wenn Anwendungsprogramme häufigen Änderungen unterliegen oder kundenspezifisch adaptiert werden müssen, führt diese Technologie letztlich zu einer Reduktion des Aufwandes.

Abb. 4a: Generischer Schema- und Objekteditor FLEXEDIT



A - Aus nebenstehender dynamisch attributierter Klassenhierarchie werden 250000 Instanzen nach Zufallsprinzip erzeugt und mit Werten belegt.
B - Alle Instanzen werden entlang einer Relation gezeichnet
C - Entlang der Relation werden ca. 50 % der Instanzen selektiert und deren Fläche berechnet
D - Alle Instanzen werden entfernt

Testumgebung: Pentium100, 32 MBRam, Windows NT 3.51

Abb. 4b: Laufzeittest für eine dynamische Modellwelt aus 2d-Flächen

LITERATUR

[HEI94] Heinecke A.M., Fleßner H.C.: „Auswirkungen des Objektorientierten Modellierens auf die Arbeit des Konstrukteurs und die Gestaltung der Benutzungsoberfläche“, IKM Abstracts: HAB Weimar 1994

[HÜB94] Hübler, R., Kolbe, P., Steinmann, F.: „Wissensbasierte Computerunterstützung früher Phasen des architektonischen Entwurfs“, HAB Weimar: Wissenschaftliche Zeitschrift 4 Heft 1994, S75-82

[HÜB95] Hübler, R. , Kolbe, P. , Steinmann, F. : „Wissensbasierte Computerstützung des früher Phasen des architektonischen Entwurfs, Teil I - Konzeption und Realisierung des Systems PrePlan“, in „Computer und Architektur - Computereinsatz in frühen Entwurfsphasen“, Wissenschaftliche Zeitschrift der HAB Weimar, Heft 4/94, Weimar, 1994

[HUG91] Hughes, John G.: „Objektorientierte Datenbanken“, München, Wien: Hanser; London: Prentice Hall 1992

[KOL97] Kolbe, P., Ranglack, D., Steinmann, F.: „Eine Schnittstelle für dynamische Objektstrukturen für Entwurfsaufgaben“, in diesem Band, IKM ‘97, Weimar, 1997

[LAN96] Langenhan, C.: ‘Untersuchung zur Konsistenzsicherung zwischen Modellobjekten und Implementationsobjekten am Beispiel eines Editors für Funktionsplangraphen, Diplomarbeit Bauhaus-Universität Weimar, Weimar, 1996

[PFE96] Pfennigschmidt, S., Kolbe, P., Pahl, J.P.: „Integration von Datenmodellen - Eine Alternative zum Produktdatenaustausch“, Fortschrittsberichte VDI-Reihe 4 Nr. 135., Düsseldorf, VDI-Verlag 1996

[STE93] Steinmann, F.: „Experimentelles AutoLisp-Objektsystem ALOS - vorläufiges Referenzhandbuch“, HAB Weimar 1993

[STE95] Steinmann, F.: „Flexibles Objektsystem FLEXOB“, DFG-Verteidigung, Bonn, März 1995

[STE97] Steinmann, F., Hübler, R. : „Vorgehensmodelle als Basis der Gestaltung durchgängiger CAD-Systeme“, in diesem Band, IKM ‘97, Weimar, 1997