Richtungsweisender Einsatz der Komponententechnologie für CAx-Systeme

Dipl.-Ing. A. T. Janocha und Dipl.-Ing. M. Gandyra, Kaiserslautern

Zusammenfassung

Ist „Programmieren ganz ohne Code“[1] auch im CAx-Bereich möglich? Die Vielzahl heterogener CAx-Anwendungen und die wachsende Komplexität der Entwicklungsprozesse bedarf neuer Lösungen in der CAx-Technik. Ziel dieses Beitrages ist es, die richtungsweisende Rolle der Komponententechnologie im CAx-Bereich aufzuzeigen. Es werden die Grundlagen der Komponenten sowie die wichtigen Komponentenarchitekturen (ActiveX und Java Beans) vorgestellt. Die Erwartungen der Anwender und der Systemhersteller, die Potentiale und die Auswirkungen dieser Technologie auf die neuen Systeme werden analysiert. Die zur Zeit verfügbaren ersten Ansätze werden präsentiert. Die Rolle der internationalen Standards für die technische Umsetzung und für die Akzeptanz von CAx-Komponentensystemen wird aufgezeigt.

1 Einleitung

Die Vielzahl der CAx-Anwendungen in der Industrie und die mangelnde Integration dieser Anwendung untereinander verursacht Störungen und Reibungsverluste in den Entwicklungsprozessen. Deswegen sind die Anwender bestrebt, die Anforderungen an die Systemhersteller so zu formulieren, daß ein einheitliches Datenmodell und eine weitgehende Integration umgesetzt wird. Mit der rasanten Entwicklung integrierter Lösungen in anderen Bereichen wächst der Druck auf die Systemhersteller, dasselbe im CAx-Bereich zu vollziehen. Im Gegensatz zu Büro- und Business-Software existiert im CAx-Bereich noch keine Integrationsplattform für derartige Anwendungen. Die Anwender besitzen unterschiedliche Hardware- und Softwareplattformen, für die gezielt bestimmte CAx-Anwendungen entwickelt wurden. Eine Systemerweiterung bedarf dann oft einer Neuprogrammierung der gewünschten Lösung, obwohl eine ähnliche von einem anderen Hersteller auf einer anderen Plattform zur Verfügung stünde. Mittlerweile kommen aber auch die Systemhersteller zu der Überzeugung, daß es nicht zweckmäßig ist, alles selbst zu entwickeln. Eine Reduzierung der Entwicklungstiefe, die in anderen Branchen, z.B. in der Automobilindustrie, schon seit langem praktiziert wird, findet ihren Weg in die Softwarehäuser.

Zwei Entwicklungen der letzten Jahre beeinflussen die Entwicklung von CAx-Systemen:

- STEP (*St*andard for the *E*xchange of *P*roduct Model Data)
- und die verteilte objektorientierte Programmierung

Das standardisierte Datenmodell STEP ermöglicht einen (fast) verlustfreien Datenaustausch zwischen verschiedenen CAx-Systemen. Darüberhinaus stellt es aber auch eine Grundlage für die Entwicklung neuer Systeme dar.

1. VDI Nachrichten, 8. 8. 97

Auf der anderen Seite, existieren zur Zeit mehrere Software-Technologien nebeneinander. In Wechselwirkung mit STEP ergeben sich daraus verschiedene Strategien für die Entwicklung von CAx-Systemen. Bild 1 stellt einige dieser Strategien mit den entsprechenden Produkten oder Projekten dar.

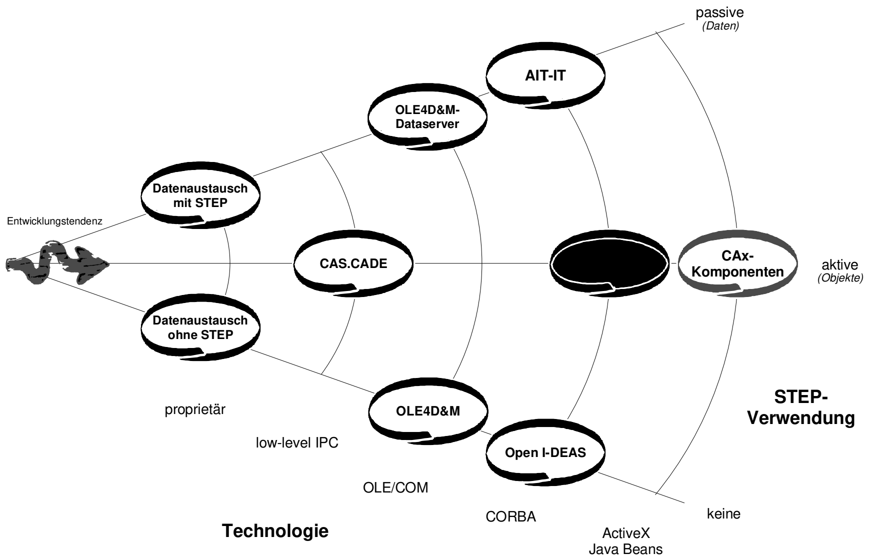

Bild 1: Verschiedene Strategiern der Entwicklung von CAx-Systemen [9]

Abhängig vom Systemhersteller und Konzept, findet eine passive oder aktive Anwendung von STEP statt. Die passive Anwendung bedeutet, daß passive STEP-Entities zwischen den Systemen ausgetauscht werden. Entweder in Form eines STEP-Physical-File (STEP-Datei) oder über die Schnittstelle SDAI (STEP Data Access Interface). Darauf lassen sich objektbasierte CAx-Systeme aufbauen.

Ein objektorientiertes System läßt sich erst bei einer aktiven Anwendung von STEP realisieren. STEP-Entities werden auf eine Hierarchie von Objektklassen abgebildet, d.h. es werden STEP-Klassen im Sinne objektorientierter Programmierung entworfen. Legt man solch einer Entwicklung statt CORBA (Common Object Request Broker Architecture) eine Komponentenarchitektur (wie ActiveX oder Java Beans) zugrunde, so entstehen statt einer STEP-Klassenhierarchie CAx-Komponenten (siehe auch [9]).

2 Komponententechnologie

2.1 Softwarekomponenten

Von der Softwareindustrie werden Komponenten vor allem als eine Möglichkeit der Produktivitätssteigerung gesehen. Je größer und umfangreicher die Komponenten, desto größer der potentielle Produktivitätszuwachs. Das Zusammensetzen von Objekten oder Applikationen zu einer komplexeren

Applikation ist effektiver als das „Neuprogrammieren“ von Funktionen. Komponenten sind auch ein Weg zur Beherrschung der wachsenden Komplexität der Anwendungen. Es ist einfacher eine aus Komponenten bestehende Software neu zu strukturieren als ein monolithisches Gebilde zu verändern.

Der Ursprung der Komponententechnologie liegt im Bereich der Wiederverwendung von Software (Software-Reuse). Die Intention des Software-Reuse ist es, den Anteil an neu programmierter Software möglichst zu reduzieren und durch vorgefertigte Bausteine zu ersetzen.

Prinzipiell können Einheiten beliebiger Granulation zur Komponente erklärt werden. Makroprogramme, Quell-Code, ausführbare Programme und Dokumentation werden gleichberechtigt als Komponenten deklariert. Dementsprechend groß ist in der Software-Entwicklung die Vielfalt an Definitionen des Begriffs „Komponente“. Eine Auswahl an Definitionen, die verschiedene Schwerpunkte setzen wird im folgenden präsentiert.

Die im Workshop WCOP ’96 [13] erarbeitete Definition stellt wohldefinierte Schnittstellen der Komponente und die Verwendung der Komponente im Prozeß der Komposition heraus:

> *A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.*

Sametinger [12] gibt eine breite Definition, die sowohl funktionierende Programme als auch Beschreibungen einschließt. Desweiteren betont er eine klar definierte Schnittstelle und die Dokumentation der Komponente:

> *Reusable software components are self-contained, clearly identifiable artefacts that describe and/or perform specific functions and have clear interfaces, appropriate documentation and a defined reuse status.*

Ciupke und Schmidt [2] konzentrieren sich auf die konzeptuelle und technische Unabhängigkeit der Komponente von dem Kontext, in dem sie entstanden ist:

> *Components are software units that are context independent both in the conceptual and the technical domain.*

Bei Microsoft [11] wird die Funktion (Service) eines binären Bausteins betont:

> *A component is a piece of compiled software, which is offering a service.*

Stal [15] hebt die Unabhängigkeit der Entwicklung von der Applikation, in der die Komponente eingesetzt wird, hervor:

> *Komponente ist ein binärer Softwarebaustein, dessen Entwicklung unabhängig von Container-Applikationen erfolgt.*

Waskiewicz [17] betont die Funktionalität der Komponente und die Integrität des Komponentensystems:

> *A component is the minimal piece of functionality in a system or subsystem that can be removed without affecting the integrity of the system or subsystem.*

2.2 Komponentensysteme

Komponentensysteme sind Software-Systeme, die aus Komponenten entstehen. Die Entwicklung von Software-Systemen aus Komponenten wird *Komposition* genannt. Die zentralen Aspekte der Entwicklung von Komponentensystemen werden hier in Anlehnung an [13] dargestellt.

Unabhängige Entwicklung und Erweiterung. Bei traditioneller Software-Entwicklung besteht eine starke Abhängigkeit der Module voneinander zur Übersetzungszeit, d.h. alle Module müssen in einer geeigneten Form (z.B. als Object-Library) vorliegen und werden dann zu einem Applikationsprogramm zusammengebunden. Bei Komponentensystemen wird diese Abhängigkeit aufgehoben. Komponenten können unabhängig vom späteren Zielsystem entwickelt werden.

Integration. Der geforderte Grad an Integration ist richtungsweisend beim Entwurf der Komponentenschnittstellen. Standards für Schnittstellen können hier die Entwickler unterstützen.

Implementierung. Bei der Implementierung ist oft ein Kompromiß zwischen Performance und Abstraktion zu finden. Insbesondere trifft dies bei der Benutzung von verteilten Objekten zu, wo oft die Netzwerke den „Flaschenhals“ darstellen.

Struktur. Die richtige Wahl zwischen einer strengen Hierarchie und Komposition ist nicht einfach zu finden. In der Hierarchie wird der Implementierungsaufwand durch Vererbung reduziert. Es ist aber nicht klar, wie die Vererbung über Komponentengrenzen funktionieren soll.

Sicherheit. Sicherheit und Flexibilität eines Software-Systems stehen oft im Konflikt. Es ist Aufgabe der Software-Entwickler eine für den Kunden optimale Kombination zu finden.

Komposition. Eine effektive Komposition muß durch graphisch-interaktive Werkzeuge unterstützt werden.

Markt für Softwarekomponenten. Die Vorteile der Komponententechnologie werden erst dann spürbar werden, wenn auf dem Markt eine ausreichende Auswahl an Komponenten zur Verfügung stehen wird. Die Komponenten müssen in geeigneten Repositorien abgelegt werden, so daß der Kunde anhand der Komponentenbeschreibung schnell feststellen kann, ob die Komponente für ein bestimmtes Zielsystem geeignet ist.

3 Komponentenarchitekturen

Zur Zeit existieren zwei bedeutende Komponentenarchitekturen, die im folgenden näher vorgestellt werden: ActiveX von Microsoft und Java Beans von Sun Microsystems. Neue Komponentensysteme, insbesondere CAx-Systeme, können auf einer dieser Technologien aufbauen.

3.1 ActiveX

Unter der Bezeichnung „ActiveX" wurde von Microsoft eine Technologie eingeführt, die eine Erweiterung der OLE/COM-Spezifikation darstellt. In diesem Abschnitt werden die Grundlagen der wichtigsten Konzepte von OLE und ActiveX nähergebracht, um den möglichen Einsatz dieser Technologien für CAx-Komponenten zu prüfen.

3.1.1 Erste Komponenten mit dem SDK

Der erste komponentenähnliche Ansatz beruhte auf der Spezifikation der „*Zusatzsteuerelemente*". Windows-Programmierern wurde damit die Möglichkeit gegeben individuelle Steuerelemente in Dialogfeldern zu erstellen und zu benutzen. Entwickelt und getestet wurden sie zusammen mit dem Dialog-Editor und dem Microsoft *Software Development Kit* (SDK) in C oder C++. Da dieses Konzept obsolet ist und wohl kaum noch ein Entwickler derartige Komponenten entwickelt, soll hier nicht näher darauf eingegangen werden [5].

3.1.2 VBX-Controls

Einer der ersten echten „Komponentenkleber" die unter Microsoft Windows eingesetzt wurden ist Microsofts Visual Basic [5]. Während der Konzeption erkannten die Entwickler, daß benutzerdefinierte Steuerelemente ein großes Potential besitzen und führten mit der Weiterentwicklung die Visual-Basic-Steuerelemente (*VBX-Control* oder kurz: *VBX* – Visual Basic Extension) ein. VBX-Steuerelemente sind in allen Varianten, beispielsweise vom einfachen *Button* über komplizierte Prozeßanzeigen bis hin zu vollständigen Textverarbeitungen realisierbar.

Die Anzahl der verfügbaren VBX-Komponenten, die sich am Anfang nur auf die von Microsoft mitgelieferten beschränkte, erhöhte sich sehr schnell durch die VBX-Entwicklung von Drittanbietern. Die Attraktivität der VBX-Controls ist letztendlich zurückzuführen auf:

- die einfache Erlernbarkeit
- einen geringeren Implementierungsaufwand gegenüber anderen Entwicklungsmethoden
- die Verwendung einer einheitlichen Entwicklungsumgebung – Visual Basic

Schnell erkannte man den Nutzen dieser Steuerelemente und führte Ihre Verwendung auch unter Visual C++ ein.

Allerdings hat das VBX-Konzept auch Nachteile:

- VBX-Controls sind 16-Bit-Objekte und nicht ohne Probleme für 32-Bit-Anwendungen zu portieren
- VBX-Controls unterstützen keine benutzerdefinierten Methoden
- VBX-Elemente waren für die Verwendung unter Visual Basic ausgelegt und werden somit von anderen Sprachen schlechter unterstützt

Um diese Nachteile zu eliminieren hat Microsoft mit *OLE* (*Object Linking & Embedding*) den nächsten Schritt in Richtung 32-Bit-Komponente gemacht.

3.1.3 OLE und das Component Object Model (COM)

OLE wurde mit Windows 3.1 erstmals eingeführt und bezeichnete in der Version 1 (*OLE1*) die ursprüngliche Fähigkeit, Objekte in ein Dokument zu integrieren, entweder eingebettet (*Embedding*) oder mit der objekterzeugenden Anwendung (Serveranwendung) verknüpft (*Linking*). Der Unterschied zwischen diesen Beiden Konzepten ist im Bild 2 dargestellt:

- *Linking*. Das eingefügte Objekt wird bei der Objektintegration nicht zum Bestandteil der Anwendung die das Objekt aufnimmt (Client- bzw. Containeranwendung). Es wird lediglich ein Verweis auf das entsprechende Objekt angelegt.
- *Embedding*. Durch Kopieren wird das eingefügte Objekt bei dieser Art der Objektintegration zum festen Bestandteil der Container-Anwendung. Selbst wenn die Quelldatei gelöscht wurde, so ist das Objekt als Ganzes in der Containeranwendung integriert.

Mit OLE1 wurden erstmals Funktionalitäten zur Verfügung gestellt, mit denen man unterschiedliche Anwendungen kombinieren und Daten zwischen diesen austauschen konnte [10]. Damit gehörten die überladenen monolithischen Anwendungen der Vergangenheit an und OLE1 bildete den ersten praktischen Ansatz für das *Compound Document* Konzept.

Ein Compound Document ist ein Dokument, das aus vielerlei Objekten bestehen kann, die von unterschiedlichen Anwendungen mit unterschiedlichen Methoden erzeugt wurden. Dabei brauchen die Serveranwendungen nichts voneinander zu wissen. Sie müssen lediglich OLE-fähig sein.

Mit der Einführung von *OLE2* wurde das Compound Document Konzept erweitert und weitere Konzepte kamen hinzu. Zusammen mit OLE2 wurde ein neues Objektmodell eingeführt, das die Basis für diese neuen Konzepte und für alle zukünftigen Entwicklungen unter Windows bildet, das *Component Object Model*.

Das Component Object Model (COM) definiert einen binären Objektstandard, der die Schnittstellen aller Objekte nach außen hin einheitlich erscheinen läßt. Dabei ist es gleichgültig mit welcher Programmiersprache die Objekte entwickelt wurden. Das heißt, eine mit Visual Basic erstellte Anwendung kann über diesen Mechanismus auf die Methoden eines Objektes zugreifen, dessen Funktionalitäten beispielsweise in C++ implementiert wurden. Damit ist COM ein Standard für die Interaktion zwischen Objekten.

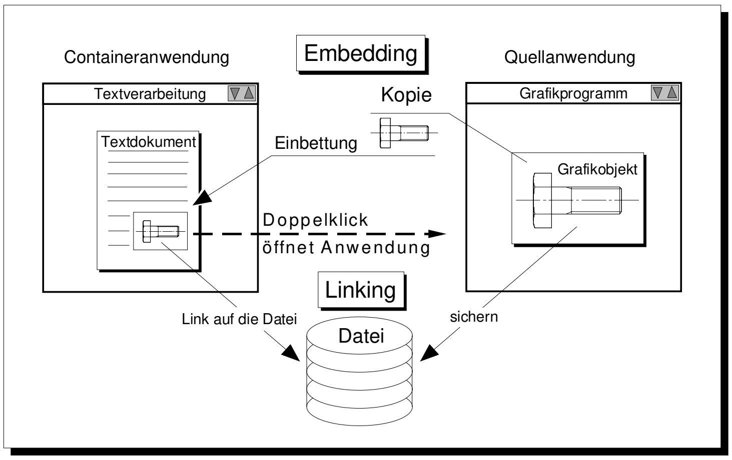

Bild 2: Unterschied zwischen Linking und Embedding

Die wichtigsten neuen Konzepte die damit einhergehen, sind im folgenden kurz beschrieben (vgl. Bild 3).

- *Drag & Drop*. Beliebige Objekte können mit der Maus verschoben und in eine Containeranwendung abgelegt werden. Entsprechend dem abgelegten Objekt wird eine geeignete Aktion ausgewählt und durchgeführt, d.h. die Containeranwendung „erkennt“ das abgelegte Objekt. Beispielsweise kann man ein Serverobjekt durch „ziehen mit der Maus“ in eine Containeranwendung einbetten.
- *Automation*. Das OLE-Automation Konzept beinhaltet Schnittstellen und Konventionen zum Aufruf von Methoden unbekannter OLE2-Objekte. Dadurch können zum Beispiel beliebige automationfähige Anwendungen von anderen Anwendungen aus gesteuert werden.
- *Naming and Binding*. Objekte die über Linking in das Dokument integriert wurden, müssen benannt werden. Nur benannte Objekte, deren Datei existiert, können angebunden werden. Das Binding bietet eine Schnittstelle, um Bindungsinformation zum eingefügten Objekt zu ermitteln und eine Verbindung zu ihm herstellen zu können.
- *Uniform Data Transfer*. Konzept zur Vereinheitlichung des Datenaustausches zwischen und innerhalb von OLE-Objekten (Dieses ersetzt das DDE-Konzept – Dynamic Data

Exchange – von OLE1).

- *Structured Storage*. Dieses neue Konzept beinhaltet eine standardisierte und hierarchische Ablagestruktur zur Speicherung von Compound Documents.
- *Notification*. Konzept zum Nachrichtenaustausch zwischen OLE-Objekten.

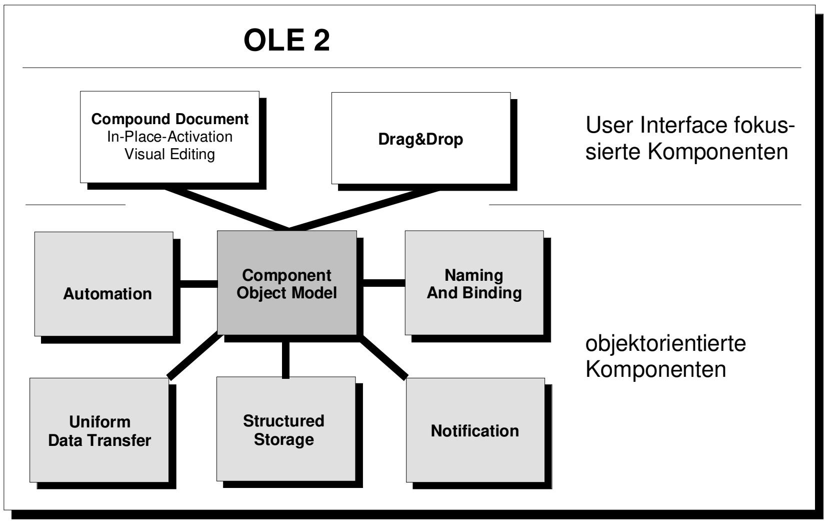

Bild 3: OLE2. Erweiterungen der Konzepte von OLE1. Die Basis aller Konzepte bildet das Component Object Model [10].

3.1.4 Die ActiveX Konzepte

Im März 1996 wurden von Microsoft neue Konzepte eingeführt. Diese entsprechen einer Erweiterung der OLE2/COM-Technologie. Alle Konzepte die bisher entwickelt wurden, sind unter der Bezeichnung ActiveX zusammengefaßt worden. So sind die Begriffe entsprechend den neuen Termini von Microsoft abzuändern [18]:

- *ActiveX-Komponente*. Ein ActiveX-Objekt, das von anderen Programmen benutzt werden kann.
- *ActiveX-Control*. Eine besondere Komponente, die wie ein Control-Fenster funktioniert. Sie Entspricht den OCX-Controls, die um Internet-Funktionalität erweitert wurden.
- *ActiveX-Client*. Jedes Programm, das irgendeine ActiveX-Komponente benutzt.
- *ActiveX-Container*. Ein Programm, das ActiveX-Document-Objects oder -Controls aufnehmen kann (bisher als Containeranwendung bezeichnet).
- *ActiveX-Server*. Ein ausführbares Programm oder eine DLL, das ein oder mehrere ActiveX-Objekte bereitstellt (bisher als Serveranwendung bezeichnet).

- *ActiveX-Document-Object*. Ein Dokument, das man in einen Container einbetten oder mit ihm verknüpfen kann (traditionelle OLE-Funktionalität).

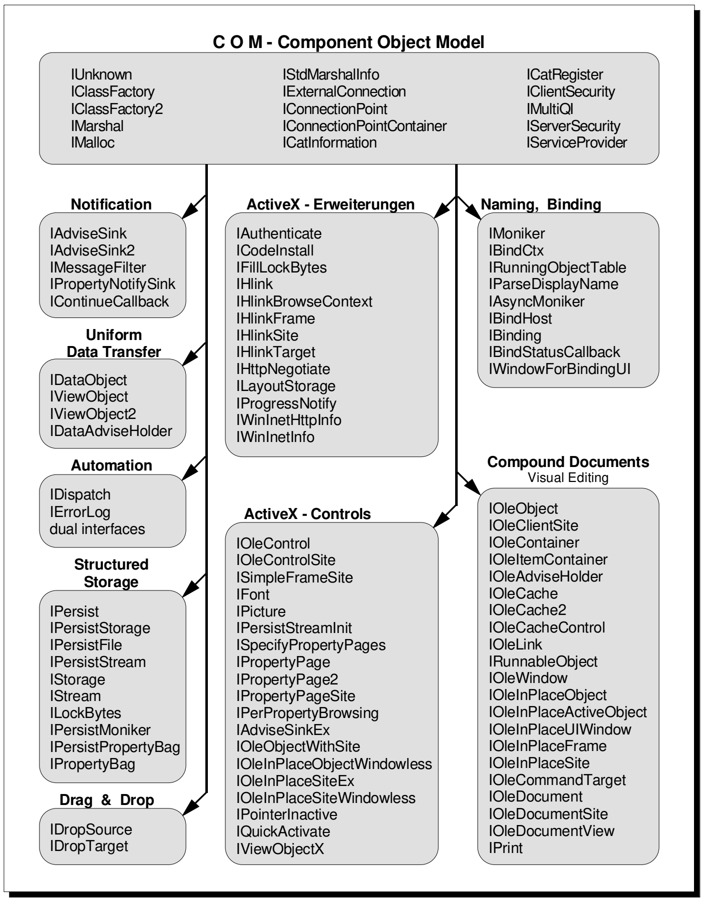

Bild 4: Was sich hinter ActiveX verbirgt: Die Erweiterung der OLE2 Spezifikation [5]

Die neuesten Teile dieser Technologie umfassen die Verwendbarkeit von ActiveX-Controls mit Web-Browsern, d.h. die Möglichkeit Daten effizient zu speichern, die über das Internet übertragen werden. Damit avanciert ActiveX zu einem Standard für die Entwicklung von Softwarekomponenten, die über die Rechnergrenzen hinweg eingesetzt werden können.

3.1.5 ActiveX-Schnittstellen-Mixtur

Die ActiveX-Konzepte bestehen im Grunde genommen aus einer Menge von *Interfaces* (Schnittstellen*, siehe Bild 4*). In der Programmierung entspricht einem Interface ein Datenfeld mit Zeigern auf Funktionen. Dementsprechend „sieht“ eine Clientanwendung von einem ActiveX-Objekt nichts anderes als eine Tabelle mit Funktionszeigern. Diese Tabelle wird vom ActiveX-Objekt exportiert, d.h.

die betreffenden Funktionalitäten werden nach außen hin zur Verfügung gestellt. ActiveX ist also nichts anderes als eine Spezifikation für Schnittstellen zwischen Objekten.

3.2 Java Beans

Java Beans ist eine Komponentenarchitektur, die auf Java aufbaut. Das Dokument „The Java Language: A Whitepaper“ [16] beschreibt Java folgendermaßen: „Java ist eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturunabhängige, portable, hochperformante, multithread-fähige und dynamische Sprache“. Eine ausführliche Erläuterung der oben genannten Eigenschaften von Java kann im Originaldokument oder in einem der zahlreichen Java-Handbücher (z.B. [7]) nachgeschlagen werden. Ein wichtiger Aspekt von Java ist die Portabilität, die sich aus dem Java zugrundeliegenden Konzept ergibt. Java ist eine interpretierte Sprache, allerdings wird vom Interpreter (der sog. Virtuellen Maschine, VM) nicht der Quell-Code sondern ein Byte-Code interpretiert. Entsprechend einer plattformunabhängiger Spezifikation arbeitet die Virtuelle Maschine auf allen Plattformen und Hardwarearchitekturen gleich. Insofern ist der Byte-Code, der vom Java-Compiler erzeugt wird, vollkommen portabel und der Slogan „Write once, use everywhere“ berechtigt.
Die Performance einer interpretierten Sprache ist schlechter als einer kompilierten Sprache (wie C oder C++). Im Java-Entwicklungslager sind massive Bemühungen im Gange, Beschleuniger für Java, sog. Just-in-Time-Compiler (JIT) zu entwerfen und zu verbreiten. Diese können die Virtuelle Maschine bei der Ausführung von Byte-Code unterstützen und die Performance der Programmausführung erheblich verbessern. Darüberhinaus besitzt Java ein „native-API“. Dies ermöglicht die Anbindung von Systemaufrufen oder binären Programmteilen (in C oder C++ geschrieben), allerdings auf Kosten der Portabilität.
Java Beans wurden für die Erstellung und für die Benutzung von Komponenten entworfen. Beans sind Java-Komponenten, aus denen Anwendungen, Applets (nicht selbstständige Anwendungen, die in einem Webt-Browser ablaufen) und komplexere Komponenten zusammengebaut werden können.

> *A Java Bean is a reusable software component that can be manipulated visually in a builder tool.*

Diese Definition betont die Wiederverwendbarkeit einer Komponente sowie die Fähigkeit der Komponente, dem Prozeß der Komposition unterworfen zu werden. Dazu werden graphische Werkzeuge eingesetzt, mit deren Hilfe die Modifikation von Eigenschaften sowie das Zusammenbauen der Komponenten in ein Komponentensystem erfolgt. Ein Beispiel solch eines Werkzeugs, das Java Studio von Sun Microsystems ist im Bild 5 dargestellt.
Die oben genannte Fähigkeit der Java Beans, in einem Editor angepaßt oder zusammengefügt zu werden, setzt die Existenz entsprechender Mechanismen voraus, mit Hilfe derer die Beans wichtige In-

formationen über ihre Properties und ihren Zustand dem graphischen Editor mitteilen können bzw. die gewünschten Einstellungen entgegennehmen können.

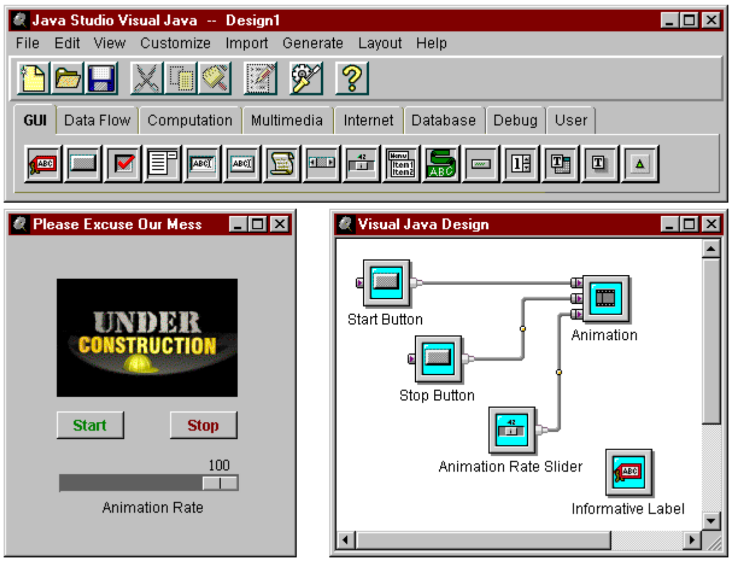

Bild 5: Java Studio – Beispiel eines graphischen Editors für Komponentensysteme [16]

Im folgenden sind die wichtigen Begriffe, Eigenschaften und Mechanismen der Java Beans-Technologie erläutert [6].

Properties sind Attribute eines Java Beans, die über einen Namen angesprochen werden.

Methoden des Java Beans sind die Java Methoden, die von der Klasse, die das Bean implementiert, für andere Komponenten sichtbar gemacht (exportiert) werden.

Events sind Ereignisse, die von Komponenten verarbeitet werden (z.B. ein Mausklick).

Introspection ist ein Prozeß des Sichtbarmachen der Properties, Methoden und Events, die eine Komponente unterstützt. Die Introspection wird zur Laufzeit aber auch vor allem zum Entwurf benutzt. Ein graphischer Editor kann diese Information abfragen und darstellen.

Customization. Eine weitere Zusatzklasse ermöglicht die Anpassung der Attribute (und des Verhaltens) der Komponente für eine konkrete Anwendung.

Persistence. Um in verschiedenen Kontexten agieren zu können, müssen Beans einen Mechanismus zur Serialisierung, d.h. zum Speichen und Restaurieren der Properties und des Zustandes eines Objektes besitzen. Java Beans benutzen den portablen Mechanismus der Serialisierung von Java.

Unterscheidung zwischen Entwurf und Laufzeit. Java Beans unterscheiden und trennen Entwurf und Laufzeit voneinander. Für den Entwurf muß eine Komponente möglicherweise mehr oder andere In-formationen zur Verfügung stellen als zur Laufzeit. Beispielsweise stellt Visualisierungsinformation einer Berechnungskomponente zur Laufzeit nur Ballast dar. Für den Entwurf ist die Visualisierung der Komponente aber notwendig.

Multithreading. Java Beans benutzen den Multithreading-Mechanismus von Java. Multithreading bedeutet, daß innerhalb eines Prozesses mehrere Code-Fragmente (Threads) gleichzeitig abgearbeitet werden. Bei der Programmierung von Java Beans muß man davon ausgehen, daß die Komponente in einer Multithreading-Applikation laufen wird und muß diese darauf vorbereiten.

Sicherheit. Java Beans bieten dieselben Sicherheitsmechanismen wie Java. Diese müssen beim sorgfältigen Design der Komponenten berücksichtigt werden.

Für den unternehmensweiten Einsatz von Java Beans müssen aber einige Defizite von Java berücksichtigt werden.

Zum einen stellt Java Beans ein lokales Komponentenmodell dar, d.h. die Kommunikation zwischen den Komponenten findet primär auf einer Maschine statt. Zum Zwecke der Verteilung besitzt Java ein RMI-API (RMI, Remote Method Invocation ist eine objektorientierte Version des Remote-Procedure-Call-Mechanismus). Unterstützung durch weitere Konzepte, wie CORBA ist an dieser Stelle notwendig.

Zum anderen ist die Definition von Komponenten in Java nicht sprachunabhängig, was auch eine starke Einschränkung darstellt. Die gemeinsame Benutzung von Java und CORBA, insbesondere die Definition der Schnittstellen in der IDL (Interface Definition Language) ist die richtige Lösung.

4 CAx-Komponentensysteme

4.1 Anforderungen und Ziele

Die wichtigsten Anforderungen der Anwender an die CAx-Technik wurden in [3] und [4] zusammengestellt. Dazu gehören unter anderem:

- Integration von CAx-Systemen
- Datendurchgängigkeit
- plattformunabhängige Verfügbarkeit aller benötigten Funktionen
- Langzeitarchivierung
- optimale Unterstützung von Prozeßketten, insbesondere im Hinblick auf das Simultaneous Engineering
- Aufhebung der Zwangsbindung an einen Systemhersteller

Eine nahtlose Integration von CAx-Systemen und die Datendurchgängigkeit können durch den Einsatz von Schnittstellen wie ANICA [8] erreicht werden. Die Langzeitarchivierung ist nur aufgrund

der Benutzung eines standardisierten Datenmodells (STEP) realisierbar. Eine optimale Unterstützung von Entwicklungsprozessen sowie die Aufhebung der Zwangsbindung an einen Systemhersteller sind erst mit der Einführung von Komponentensystemen im CAx-Bereich möglich.
Dazu sind allgemeine und domänenspezifische Komponenten notwendig, die durch eine Wechselwirkung von STEP mit einer Komponentenarchitektur (ActiveX, Java Beans) entstehen (vgl. Bild 1). Aus diesen Komponenten können Anwendungen oder komplexere Komponenten erstellt werden. Eine signifikante Reduzierung der Komplexität der Prozesse wird dann eintreten können, wenn die Granulation der Komponenten in etwa den Prozeßschritten (FEM, Flächenmodellierung, NC-Bearbeitung usw.) entsprechen wird.

4.2 Definition

In [9] wird eine CAx-Komponente folgendermaßen definiert:

> *CAx-Komponente ist ein binärer, klar identifizierbarer Baustein mit einer wohldefinierten, auf STEP basierenden Schnittstelle, der die für ein Anwendungsgebiet relevante CAx-Funktionalität realisiert".*

Die Definition betont die STEP-kompatible Schnittstelle sowie die für CAx-Abläufe geeignete Granulation, d.h. den Umfang und die Größe der Komponenten.

4.3 Rolle der internationalen Standards

Für eine rasche Verbreitung von CAx-Komponenten sind einige Voraussetzungen notwendig:

- standardisiertes Datenmodell
- sprachunabhängige Definition von Komponenten
- plattformübergreifende Kommunikation

Die erste Voraussetzung ist mit der Einführung von STEP in die CAx-Systeme erfüllt. Die Erfüllung der übrigen Voraussetzungen ist komplizierter. Zur Zeit ist CORBA die einzige Architektur, die die Anforderungen der sprachunabhängigen Definition von Schnittstellen (und Komponenten) sowie einer plattformübergreifenden Kommunikation erfüllt. Das bedeutet, daß die Komponentenarchitekturen, wie ActiveX oder Java Beans als alleinige Grundlage für CAx-Komponenten nicht ausreichen. Der Entwurf der Schnittstellen für CAx-Komponenten muß sich sowohl auf STEP als auch auf CORBA stützen. Eine Möglichkeit hierfür ist die ANICA-Schnittstelle (vgl. [8]). Aus technischen Gesichtspunkten ist die Einführung von Standards, insbesondere für Schnittstellen und dessen Spezifikation, von entscheidender Bedeutung.

4.4 Auswirkungen der Komponentenarchitektur

Die Architektur eines Softwaresystems legt in erster Linie die internen Schnittstellen und Datenstrukturen des Systems fest. Bei Komponentensystemen kommen weitere Auswirkungen auf die Herstellung und Lizensierung, auf die äußere Erscheinung und auf die Arbeitsweise hinzu.

Mittelfristig wird es keine CAx-Systeme im herkömmlichen Sinne mehr geben. Sie werden auch nicht mehr nur „programmiert" sondern überwiegend „komponiert", d.h. aus Komponenten zusammengestellt. Und zwar nicht nur von den Systemanbietern sondern auch von den Anwendern, je nach Bedarf. Die Lizensierung von Software wird sich grundsätzlich ändern. Der Anwender wird nicht mehr für die Software und die folgenden Updates, sondern für die Benutzung der Software bezahlen.

Obwohl zur Zeit die verfügbaren Implementierungen die Standard GUI-Elemente zur Verfügung stellen, wird sich die äußere Erscheinung der CAx-Systeme verändern. Vor allem werden die tiefen, benutzerunfreundlichen Menüs entfallen. Die Funktionen werden immer mit den bearbeiteten Objekten verknüpft sein, so daß sie intuitiv zu finden sein werden.

Auch die Komplexität der Systeme kann dadurch reduziert werden. Beispielsweise wird in den herkömmlichen Systemen eine aufwendige Technik des Ausblendens der Menüeinträge benutzt, wenn eine Funktion nicht für ein bestimmtes Objekt (Gruppe von Objekten) relevant ist. Dies kann in den Komponentensystemen entfallen, dar der betreffende Menüeintrag gar nicht erst erscheint.

Für die Arbeitsweise mit CAx-Systemen bedeutet die Komponentenarchitektur eine konsequente Durchsetzung der Objektorientierung auf der Benutzeroberfläche der CAx-Systeme. Objekte von angemessener Granulation werden eingefügt, entfernt, strukturiert und modifiziert. Die Eigenschaften der Objekte werden jederzeit abrufbar und in Property-Editoren modifizierbar sein. Die Funktionen zum Modifizieren der Objekte werden dort zu finden sein, wo sie hingehören – zu den Objekten selbst und nicht in die vom Systemhersteller abhängigen Menüs. Da das Objekt den Property-Editor in das Komponentensystem „einbringt", wird die Erscheinung der Objekte in jedem Komponentensystem gleich bleiben. Die Bedienung der Systeme wird intuitiver werden und von den Komponentenherstellern unabhängig sein. Der Anwender wird sich dadurch besser auf seine Aufgabe konzentrieren können und an Produktivität gewinnen.

4.5 Vorteile für die Anwender und Systemhersteller

Von der Einführung der Komponentensysteme im CAx-Bereich können sowohl die Anwender als auch die Systemhersteller profitieren.

Für die Anwender bedeuten CAx-Komponentensysteme vor allem eine bessere Prozeßunterstützung. Zu jedem Prozeßschritt kann die passende, optimale (best-of-class) Komponente gewählt werden. Die herkömmliche Vorgehensweise mit mehreren verschiedenen Komplettsystemen, von denen jeweils nur ein kleiner Teil benutzt wird, verliert nach und nach an Bedeutung. Weitere Vorteile für die Anwender:

- *Flexibilität*. Die Komponentensysteme können nach Bedarf neu konfiguriert werden.
- *Stabilität*. Bei Systemerweiterungen oder Updates werden nur kleine Teile des Komponentensystems ausgetauscht. Die Auswirkungen der Änderungen bleiben lokal.
- *Bessere Qualität der Programme*. Spezialmodule mit spezifischem Know-how müssen bei der traditionellen Anbindung an die CAx-Systeme (early binding, zur Übersetzungszeit) angepaßt werden. Diese potentielle Fehlerquelle entfällt bei Komponentensystemen.
- *Schlanke Systeme*. Die Systeme beinhalten nur die Komponenten, die für die jeweilige Aufgabe tatsächlich gebraucht werden.
- *Die Interoperabilität der CAx-Systeme*. Alle Komponenten bauen auf einer standardisierten, STEP- und CORBA-kompatiblen Schnittstelle auf und sind damit uneingeschränkt interoperabel.
- *Integration in unternehmensweite Systeme*. CAx-Komponenten können einfacher in EDM (Electronic Data Management) und andere unternehmensweite Systeme integriert werden.

Für die Systemhersteller stellen Komponenten einen Weg zur Rationalisierung der Software-Entwicklung dar. Im einzelnen bedeutet das für die Systemhersteller:

- *Reduzierung der Entwicklungstiefe* durch Bezug von Komponenten von „Großanbietern" und Wettbewerbern
- Erhöhung der Produktivität
- *Verkürzung der Entwicklungszeit* (time-to-market)
- Konzentration auf eigene Stärken
- *Verbesserung der Qualität* (bessere Stabilität, Erweiterbarkeit)

5 Erste Implementierungen von CAx-Komponenten

Da sowohl Microsoft als auch Sun überwiegend im Bereich Betriebssysteme und Büroautomation tätig sind und keine technischen Systeme anbieten, sind die CAx-Systemhersteller selbst gezwungen in „Komponenten umzudenken". Die Systemhersteller konzentrieren sich zur Zeit auf die ActiveX-Technologie, die ausgereifter ist als Java Beans. Es existieren bereits erste Implementierungen, mit denen gezeigt werden kann, daß es durchaus möglich ist im CAx-Bereich effiziente ActiveX-Komponenten zu entwickeln.

Die Firma Spatial Technology hat als eine der ersten eine ActiveX-CAx-Komponente basierend auf ihrem ACIS-Modellierkern entwickelt. Diese Komponente namens „3D Building Blox" (Bild 6) dient der Entwicklung von 3D-Solid-Modelling Anwendungen und bietet Unterstützung für die Erzeugung von Solid Models, für Ansichten und deren Manipulationen und für die Modellanalyse.

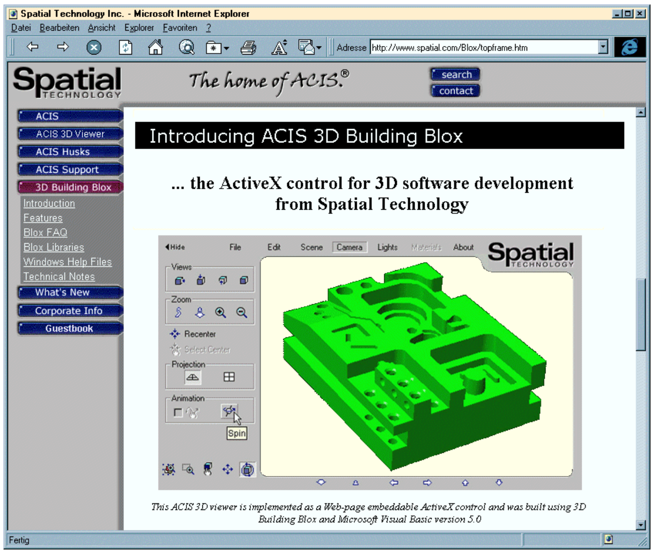

Bild 6: Erster Ansatz für ActiveX CAx-Komponenten von Spatial Technology [14]

Es wurden bereits einige Anwendungen damit erstellt, unter anderem:

- ein 3D Viewer, zum sichten von ACIS-konformen Modellen im SAT-Format (kann bei Spatial Technology [14] heruntergeladen werden)
- eine Anwendung zur Visualisierung von 3D-Daten
- ein Kran-Simulationspaket
- Ein Chemie-Lern-System

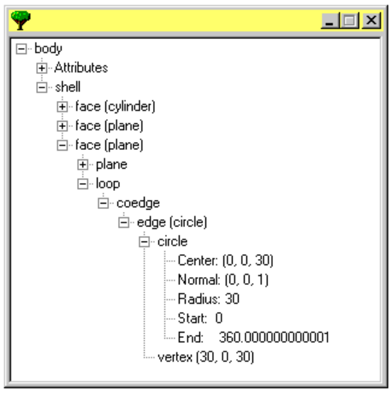

Bild 7: Der Entity-Browser von Building-Blox [14]

Wie im Bild 6 zu sehen ist, kann ein ActiveX-Control, das unter Verwendung von 3D-Building-Blox erzeugt wurde, in ein HTML-Dokument eingebettet werden und von jedem Web-Browser aus bedient werden.

Zur Modellanalyse läßt sich mit Building-Blox ein Entity-Browser einsetzen, mit dem man sich durch die Modellstruktur bewegen und alle Entities abfragen kann (siehe Bild 7).

Einen ähnlichen Ansatz wie Spatial Technology verfolgt die Firma Matra Datavision mit ihren Entwicklungstoolkit CAS.CADE unter Windows NT.

6 Ausblick

Die nächste Generation von CAx-Systemen wird in Form von Komponentensystemen entwickelt werden. Die zur Zeit angebotenen, ersten CAx-Komponenten bauen auf der ausgereiften und verbreiteten ActiveX-Technologie auf. Potentielle Entwicklungen mit Java Beans befinden sich noch im Experimentalstadium.

Komponentensysteme können aber nur dann eine ernstzunehmende Alternative zu herkömmlichen Entwicklungen darstellen, wenn sich ein Markt für CAx-Komponenten etabliert hat und eine ausreichende Auswahl an plattformübergreifend einsetzbaren Komponenten zur Verfügung steht. Darunter sind im einzelnen zu verstehen:

- allgemeine Komponenten für Visualisierung, Datei-Auswahl, Drucken usw.
- domänenspezifische CAx-Komponenten für Konstruktion, Flächenmodellierung, FEM, NC, Simulation usw.

Darüberhinaus sind Repositorien und Komponentenkataloge sowie flexible Kompositionswerkzeuge und Editoren notwendig.

In einem funktionierendem Komponentenmarkt werden die Kosten für Komponenten sinken. Damit können die Anwender und die Systemhersteller die Investition rechtfertigen, die mit der Einführung von Komponentensystemen verbunden ist.

7 Schrifttum

[1] Brockschmidt K.: *Inside OLE 2,* Microsoft Press, Redmond, 1994.

[2] Ciupke O., Schmidt R.: *Components as Context-Independent Units of Software,* in: Mühlhäuser M. (Ed.): Special Issues in Object-Orineted Programming. Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP ’96, Linz, July 1996.

[3] Dankwort C. W.: *CAx Systems Architecture of the Future*, in: Roller D., Brunet P. (Eds.): CAD Systems Development. Tools and Methods, Springer, 1997, pp. 20–31.

[4] Dankwort C. W., Kellner P., Leu D., Müller G., Renz W., Weißberger G.: *CAD in der Automobilindustrie - Aktuelle Bedürfnisse und mittelfristige Ziele*, CAD-CAM REPORT Nr.1, 1997, S. 46–53.

[5] Denning A.: *ActiveX-Steuerelemente*. Microsoft Press, Redmond, 1997.

[6] Englander R.: *Developing Java Beans*. O'Reilly & Associates, 1997.

[7] Flanagan D.: *JAVA in a Nutshell*. O'Reilly & Associates, Inc., 1996.

[8] Janocha A. T.: *CAx-Systemintegration auf Basis von CORBA und STEP*, Produktdaten Journal, Nr.1, 1996, S. 45–48.

[9] Janocha A. T., Dankwort C. W.: *Moderne CAx-Systemarchitekturen: Von proprietären Systemen zu „Komponentware"*, CAD-CAM REPORT Nr. 9, 1997.

[10] Meyer H.-M., Obermayr K.: *Objekte integrieren mit OLE 2*. Springer-Verlag, 1994.

[11] Microsoft, *The Component Object Model: Technical Overview*.

[12] Sametinger J.: *Software Engineering with Reusable Components*. Springer-Verlag, Berlin Heidelberg, 1997.

[13] Szyperski C., Pfister C.: *Component-Orinted Programming: WCOP'96 Workshop Report*, in: Mühlhäuser M. (Ed.): Special Issues in Object-Orineted Programming. Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP '96, Linz, July 1996.

[14] *http://www.spatial.com.*

[15] Stal M.: *Componentware – von der Komponente zur Applikation.* OBJEKTspektrum 3/97, S. 86–89.

[16] *http://www.sun.com.*

[17] Waskiewicz F.: *An Object-Oriented Framework for Manufacturing Applications.* OMG BOM-SIG, Ottawa, 1995.

[18] Williams A.: *ActiveX Web Controls entwickeln.* International Thomson, 1997.