Konzept eines COM-basierten Technischen Produktinformationssystems (TPIS)

Vom Fachbereich 12 Maschinenwesen der Universität GH Essen
zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte
Dissertation

von

Markus Schütten

aus Essen

Tag der mündlichen Prüfung: 15. August 2001

Vorsitzender

der Prüfungskomission: Prof. Dr.-Ing. E. Schmachtenberg

Gutachter: Prof. Dr.-Ing. H. J. Stracke

Prof. Dr.-Ing. D. Bergers

Vorwort

Diese Arbeit entstand während meiner Tätigkeiten als wissenschaftlicher Mitarbeiter am Institut für Prozess- und Datenmanagement des Fachbereichs Maschinenwesen der Universität Essen.

Herrn Univ.-Prof. Dr.-Ing. H. J. Stracke danke ich für die Anregung, Ermöglichung und Förderung dieser Arbeit und den mir stets gewährten Freiraum bei der Durchführung.

Herrn Univ.-Prof. Dr.-Ing. D. Bergers danke ich für das meiner Arbeit entgegengebrachte Interesse und die Übernahme des Koreferats.

Essen, im August 2001 Markus Schütten

1 Einleitung

Der Produktentstehungsprozess in Klein-, Mittel- und Großbetrieben des produzierenden Gewerbes wird heute durch datenverarbeitende Systeme unterstützt. Hier haben sich vor allem Lösungen in den Bereichen Kosntruktion (CAD), Fertigung (CAM), Arbeitsvorbereitung (CAP), Qualitätssicherung (CAQ), Softwareengineering (CASE) und Organisation (CAO) durchgesetzt. Alle Systeme haben für sich betrachtet bemerkenswerte Erfolge erzielt, allerdings mit dem Nachteil, dass es sich jeweils nur um sogenannte Insellösungen handelt. So summieren sich die mit diesen Programmen erstellten Daten zu einer Datenflut, die je nach Größe des Betriebs und in Abhängigkeit der Bearbeitungszeit keine wirkliche Datentransparenz mehr zulässt. Um dem bereichsweise entgegenzuwirken, wurden von führenden CAD-Herstellern und einigen unabhängigen Anbietern sogenannte PDM-Systeme (Produktdatenmanagementsysteme) entwickelt, um zumindest auf der CAD-Seite eine Verwaltung der Dokumente zu ermöglichen. Eine bereichsübergreifende Datenverwaltung (Abbildung 1-1) und eine optimale Abstimmung der Prozessabläufe verlangt jedoch die Implementierung eines unternehmensweiten Produktdatenmanagement. Man kann in diesem Zusammenhang auch von einem Technischen Informationssystem (TIS) sprechen, das als Pendant zum Kommerziellen Informationssystem auf der Grundlage von Enterprise Resource Planing (ERP) existiert [1].

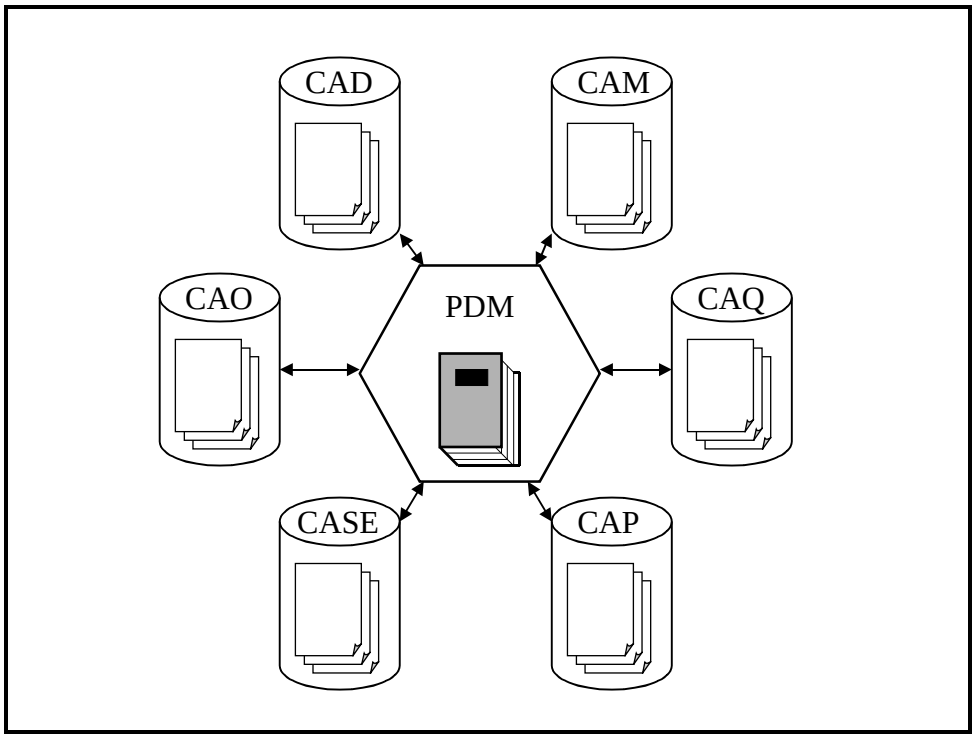


Abbildung 1-1: PDM als technisches Informationssystem

Die zur Jahrtausendwende auf dem Markt befindlichen PDM-Systeme erfüllen diese Anforderungen nur unzureichend. Die Systeme berücksichtigen leider nicht alle Bereiche der Produktion gleichermaßen. Vielmehr beschränken sich die meisten Systeme im Wesentlichen auf den Entwicklungsabschnitt der Konstruktion und unterstützen hierbei die Kopplung zu den Zeichensystemen und der damit verbundenen abteilungsspezifischen Zeichnungsverwaltung. Diese Systeme bieten nicht den von vielen Unternehmen erhofften Ersatz für die oben angesprochenen Bearbeitungsinseln, da durch ihren Einsatz lediglich nur eine lokal optimierte Datenverwaltung aufgebaut wird.

Eine ähnliche Entwicklung hat es schon einmal in den 80er- bzw. 90er-Jahren gegeben. Damals wurden von verschiedenen Anbietern entweder aus der Sicht der Fertigung oder aus der Sicht der Konstruktion eine Vielzahl von CAD-Systemen konzipiert, die alle auf verschiedenen Datenstrukturen basierten. Aus diesem Grunde konnten die jeweils guten Funktionalitäten einzelner CAD-Systeme nicht in die jeweils anderen CAD-Systeme überführt werden, deren Funktionalität in diesem speziellen Bereich nicht so gut ausgeprägt war. Erst mit der Entwicklung eines anbieterunabhängigen 3D-CAD-Kernes wurde die Qualität der CAD-Systeme insgesamt besser [2].

Hier soll in ähnlicher Weise der Versuch unternommen werden, auf der Basis geeigneter Softwaretechnologien ein PDM-Konzept für den Einsatz als TIS zu entwickeln, welches den heutigen Anforderungen genügt und somit den unternehmensweiten Zugriff auf ein virtuelles Produkt gestattet, dessen digitales Abbild das reale Produkt darstellt. Hierbei soll insbesondere gewährleistet sein, das durch die Nutzung neuer Technologien im Bereich der Schnittstellen zu den Erzeugersystemen alle Bereiche des Unternehmens berücksichtigt werden, die entweder Informationen erzeugen oder nutzen. Die Steuerung des Informationsflusses soll hier unternehmensübergreifend von sogenannten Workflows übernommen werden. Darüber hinaus sollen die Zugriffe auf die Daten der Produktionsunternehmen auch über das Internet möglich sein, was die Zusammenarbeit mit Zulieferfirmen und die Informationsversorgung der Außendienstmitarbeiter wesentlich vereinfacht.

Im folgenden soll zunächst der Stand der Technik analysiert und die Schwachpunkte der auf dem Markt befindlichen PDM-Systeme festgestellt werden, aus welchen sich die Anforderungen an ein technisches Produktinformationssystem **(TPIS)** ableiten lassen. Die Vorstellung der momentan verfügbaren Routinen für die softwaretechnische Gestaltung sollen die Möglichkeiten

für die Gestaltung des **TPIS** verdeutlichen. Im Anschluß daran wird das Konzept für das **TPIS** unter Einbeziehung eines neuen Datenbank-Kernes vorgestellt. Die softwaretechnische Umsetzung verlangt mehrere Mannjahre und ist im Rahmen einer Dissertation nicht möglich, deshalb soll anschließend zumindest exemplarisch der Einsatz des **TPIS** aufgezeigt werden.

2 Ist-Zustand des Produktdatenmanagement

Betrachtet man den Prozess der Produktentstehung, so lässt sich dieser in einen fertigungsvorgelagerten Bereich und in einen Bereich der eigentlichen Fertigung aufteilen. Aus dem Blickwinkel der Informationstechnologie lässt sich diese Aufteilung auch anders beschreiben. So ist im fertigungsvorgelagerten Bereich der Informationsfluss vorherrschend, während der Fertigung der Materialfluss (vgl. Abbildung 2-1). Die Arbeitsergebnisse der fertigungsvorgelagerten Bereiche eines Produktionsunternehmens sind heute digitale Informationen. Zwar werden auch während der Fertigung Informationen benötigt, doch diese bestehen zum größten Teil aus immer wiederkehrenden Informationen, die zur Steuerung und Ablauforganisation während der Produktion beitragen.

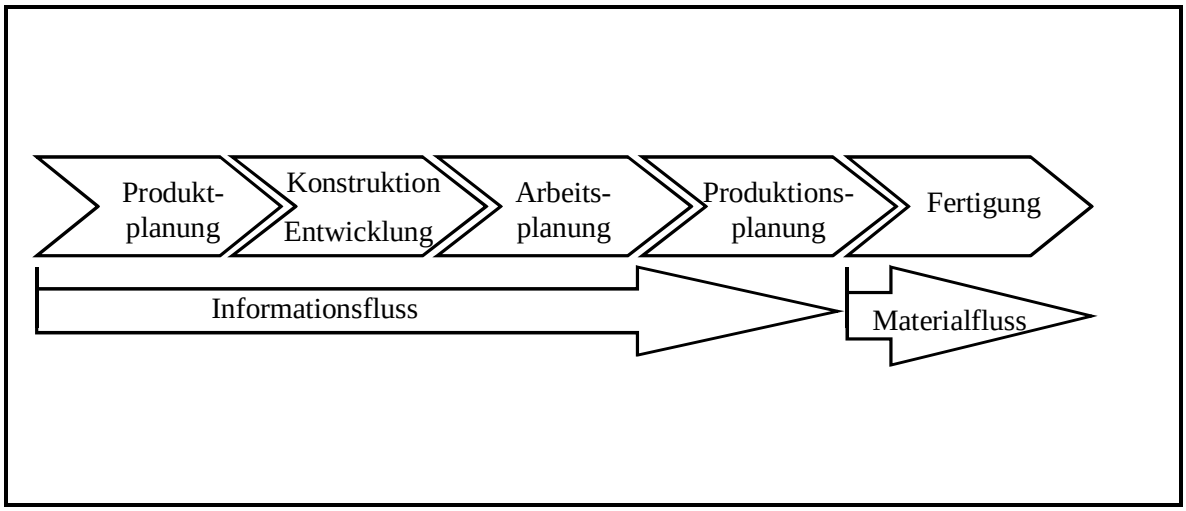


Abbildung 2-1: Aufteilung des Produktentstehungsprozesses unter den Gesichtspunkten des Informations- und Materialflusses [2].

Im Zuge des Informationsflusses werden die produktbeschreibenden Daten im fertigungsvorgelagerten Bereich entlang der vorher definierten Ketten der an der Produktion beteiligten Abteilungen weitergereicht. Die jeweils folgende Abteilung verarbeitet eingegangene Daten und nimmt Erweiterungen vor. Diese Vorgehensweise hat zur Folge, dass die Quantität der Informationen entlang des Entstehungsprozesses stetig zunimmt. Damit die Qualität der Informationen nicht im gleichen Maße abnimmt wie die Quantität zunimmt, sollte die Lösung eines jedes Prozessschrittes durch eine iterative Vorgehensweise innerhalb eines Prozessschrittes erlangt werden. So sind beispielsweise die im Bereich der Konstruktion erarbeiteten Modelle auf Machbarkeit zu überprüfen und gegebenenfalls an den Anfang des jeweiligen Schrittes zurückzuführen, um eine Überarbeitung und damit eine Verbesserung zu veranlassen (vgl.

Abbildung 2-2). Man spricht im Zusammenhang der Quantitäts- und Qualitätssteigerung auch von der Überführung eines Produktmodells aus einem diffusen in einen immer schärferen Bereich [1],[2]. Die erarbeiteten Informationen der fertigungsvorgelagerten Bereiche ergänzen sich zunehmend, bis alle Daten zur Fertigung eines Produktes zusammengetragen, geprüft und freigegeben sind. Die Gesamtheit aller digitalen Informationen zur Beschreibung eines Produktes wird als virtuelles Produkt bezeichnet.

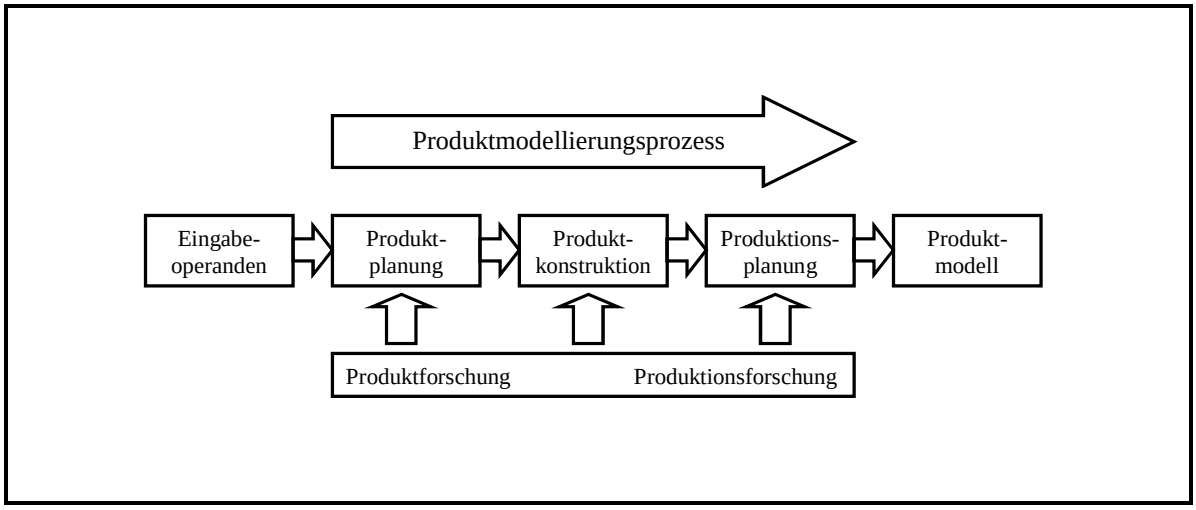


Abbildung 2-2: Virtuelle Produktmodellierung [3]

In den folgenden Kapiteln sollen die am virtuellen Produkt beteiligten Unternehmensbereiche kurz vorgestellt werden unter besonderer Berücksichtigung der dort jeweils produzierten Informationen.

Die sich anschließende Beschreibung derzeitiger Systeme zum Management dieser Produktdaten zeigt sowohl Möglichkeiten als auch Probleme auf, die sich im Umgang mit dem virtuellen Produkt ergeben. Später soll ein Konzept vorgestellt werden, das diese Schwierigkeiten überwindet und durch eine geeignete Architektur zukunftsorientierte Anforderungen erfüllt.

2.1 Produktplanung

Die Produktplanung umfasst alle Aufgaben, die marktbezogen zur Festlegung des Gestaltungsrahmens für ein herzustellendes Produkt gehören und die zur Abwicklung der Produktentwicklung organisatorisch erforderlich sind [6]. Marktanalysen und

Konkurrenzfabrikate sind Einflussfaktoren auf die Weiterentwicklung vorhandener Produkte, die weniger planerische Aufgaben umfasst als eine komplette Neuentwicklung. Beide Ziele verlangen jedoch von der Produktplanung die Festlegung eines zeitlichen Rahmens von der Produktfindung bis hin zur Markteinführung. Organisatorische Vorraussetzungen und technische Hilfsmittel sind entscheidende Faktoren für eine Verkürzung dieses Zeitrahmens. Besondere Beachtung findet dabei die Planung des Informationsflusses, die im Zuge der Produktplanung festgelegt wird und mit dem Begriff der Informationslogistik beschrieben wird [4].

Zur Bearbeitung dieser betriebswirtschaflich-planerischen Aufgaben werden Werkzeuge der Büroautomation eingesetzt. Eine besondere Rolle spielen dabei vor allem die Standardanwendungen Textverarbeitung, Präsentationserstellung und Tabellenkalkulation, die für fast alle Planungsaufgaben zu einem unverzichtbaren Hilfsmittel geworden sind und unter der Bezeichnung *Computer Aided Office (CAO)* bekannt sind. Zu nennen sind außerdem spezielle Tools zur Projektplanung, die Aufgaben der Projektüberwachung vereinfachen und sowohl den Zeitfortschritt als auch das Erreichen technischer Vorgaben überprüfen. Als Beispiel sei hier Microsoft® Project genannt.

2.2 Produktkonstruktion

Der Konstruktionsprozess umfasst die Findung von Funktionslösungen sowie die Dokumentation und Darstellung der Funktionsstruktur durch Vorschriften zur materiellen Realisierung. Das Produkt wird auf diese Art virtuell definiert, so dass zur materiellen Verwirklichung alle gestellten Anforderungen erfüllt sind. Konstruieren heißt, sich von Gewohntem zu trennen, zu analysieren und es gegebenenfalls anders zu kombinieren. Der eigentliche Konstruktionsprozess verläuft als gedankliche Vorstellung, setzt aber die Kenntnis relevanter Daten und deren ständige Verfügbarkeit voraus. Ergebnisse vorausgegangener Konstruktionsprozesse müssen für den Konstrukteur ständig verfügbar und abrufbar sein.

Einflussfaktoren auf Gestaltung und Hervorbringung neuer Konstruktionslösungen sind somit zum einen die Vorgaben der Produkteigenschaftsplanung aus dem Bereich der Produktplanung zu berücksichtigen und zum anderen die Kenntnisse aus bereits existierenden Konstruktionen mit einzubeziehen. Speziell die Kenntnis über die Produkteigenschaften setzt funktionierende Informationsflüsse aus der Produktplanung in den Bereich der Konstruktion voraus.

Auf dem Gebiet der Konstruktion haben sich Systeme zur digitalen geometrischen Beschreibung von Bauteilen bereits seit langem durchgesetzt, man spricht von der rechnerunterstützten Konstruktion bzw. dem *Computer Aided Design (CAD).* Eine relevante Entwicklung der letzten Jahre ist der Trend vom 2D-CAD hin zum 3D-CAD, das eine vollständige geometrische Beschreibung eines Bauteils erlaubt. Durch Einsatz dieser vollständigen geometrischen Beschreibung, die auch als generisches Modell bezeichnet wird, können automatisch Ansichten, Schnitte und Zeichnungen erstellt werden. Auch ist eine Variantenkonstruktion eines vorhanden Bauteils durch Vorgabe geeigneter Parameter möglich.

Auch im Hinblick auf den Informationsfluss an die der Konstruktion nachgeschalteten Bereiche hat die Existenz eines generischen Modells positive Auswirkungen. So können Festigkeitsberechnungen mit vorliegenden 3D-Modellen durchgeführt, oder auch Steuercodes für Werkzeugmaschinen direkt abgeleitet werden. Für moderne Unternehmen muss es von Interesse sein, 3D-CAD-Systeme einzuführen und die erstellten generischen Modelle allen relevanten Bereichen der Produktion zur Verfügung zu stellen [5].

2.3 Produkterprobung

Die Produkterprobung stellt den letzten Schritt im Produktentwicklungsprozesses dar. Es werden hier in der Regel Prototypen untersucht, die in Einzel- und Kleinserien gefertigt und Erkenntnisse über die Qualität der bisher erzielten Arbeitsergebnisse liefern. Eine beliebte Methode, um subjektive Kriterien bewerten zu können, ohne teure Prototypen erzeugen zu müssen, sind die sogenannten *Mock-Up-Verfahren.* Derartige Attrappen werden aber auch hergestellt, um aufgrund ihrer geometrischen Ausprägung Beurteilungen über das Verhalten bezüglich der Integrationsfähigkeit innerhalb eines komplexen Gesamtsystems zuzulassen.

Die physikalischen Mock-Up-Verfahren werden jedoch zunehmend durch *Digital-Mock-Up-Verfahren (DMU)* abgelöst. Mittels DMU werden 3D-Modelle zur Simulation der räumlichen Ausdehnung und Kollision sowie der funktionalen Zusammenhänge herangezogen. Darüber hinaus stellen Virtual Reality-Systeme Werkzeuge zur Verfügung, die einen interaktiven Zugriff auf virtuelle dreidimensionale Szenen erlauben, die im Rechner abgebildet und dem Betrachter die Beurteilung komplexer bildlicher Szenarien ermöglicht [2]. Grundlage dieses Verfahrens ist das 3D-Modell aus der Konstruktion. In Kombination mit den Vorgaben der Produkteigenschaften aus der Produktplanung können mittels DMU Aussagen über das Arbeitsergebnis der Konstruktion gemacht werden und gegebenenfalls Verbesserungsanforderungen an diese zurückgegeben werden. Der funktionierende

Informations- und Datenfluss aller bis dahin erarbeiteten Daten ist eine wichtige Vorraussetzung für ein solches Vorgehen.

Auch für rechnerische Überprüfungen, die in den Bereich der Produkterprobung fallen, sind generische 3D-Modelle aus der Konstruktion hilfreich. Mittels des Verfahrens der *Finiten Elemente Methode (FEM)* kann eine Bauteilgeometrie in finite Elemente eines vorgegebenen Typs rechnerisch zerlegt und ihr Verhalten durch Angabe bestimmter Randbedingungen mittels elementbeschreibender Differentialgleichungen mathematisch exakt bestimmt werden. Die Ergebnisse dieser Berechnungen sind entscheidend für den weiteren Informationsfluss. Muss aufgrund nicht zufriedenstellender Ergebnisse ein Bauteil überarbeitet werden, so muss die Weitergabe aller Daten gestoppt werden. Bei positiv ausfallenden Berechnungen kann ein Modell zur Fertigung freigegeben werden, die Modelle gelangen in den angeschlossenen Bereich der Produktfertigung. Die Ergebnisse der Berechnungen gehören ab diesem Zeitpunkt zum virtuellen Produkt. In der Regel sind diese visuell dargestellt und in einem neutralen Format, beispielsweise *Tagged Image File Format (TIFF)* abgespeichert.

2.4 Produktionsplanung

Im Anschluss an die Produkterprobung erfolgt die ablauforganisatorische Gestaltung des Produktionsprozesses. Als Teil der betrieblichen Gesamtplanung zielt sie auf die optimale Gestaltung und wirtschaftliche Umsetzung von Investitionsvorhaben für die Produktion [2].

Computergestützte Arbeits- und Produktionsplanung wird als *Computer Aided Planning (CAP)* oder *Computer Aided Process Planning (CAPP)* bezeichnet. Beides stellen Sammelbegriffe datenverarbeitender Systeme dar, die zur Produktionsplanung mit dem Ziel eingesetzt werden, die Erstellung von Arbeitsplänen datentechnisch zu unterstützen. Ein Arbeitsplan beschreibt detailliert alle Maßnahmen zur Überführung eines Rohteils oder Halbzeugs in ein fertiges Bauteil. Ausgangspunkt sind die aus der Konstruktion stammenden Modelle und die damit verknüpften Stücklisten (vgl. Abbildung 2-3).

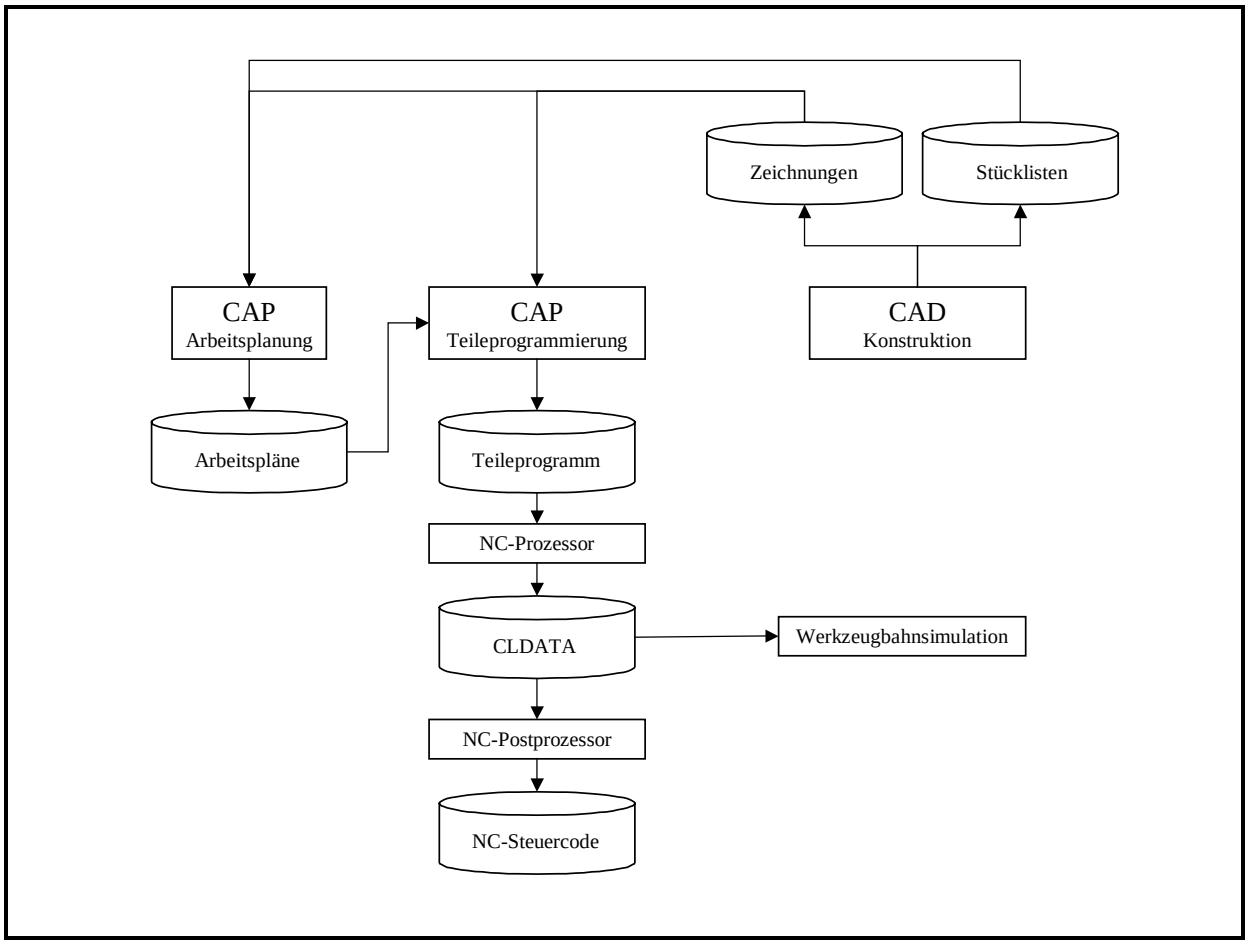


Abbildung 2-3: Erstellung von Arbeitsplänen und NC-Steuercodes aus Zeichnungen und Stücklisten [1].

Wesentlich für einen reibungslosen Ablauf der Arbeitsplanerstellung ist auch hier wieder der Informationsfluss mit den Ergebnissen aus der Konstruktion. Das virtuelle Produkt kann mit den dann vorliegenden Daten durch Arbeitsplanung und Teileprogrammierung weiter durch Arbeitspläne und NC-Steuercodes ergänzt werden.

2.5 Produktionssteuerung

Die Produktionssteuerung gewährleistet die Realisierung des Produktes unter Berücksichtigung der in einem Unternehmen auftretenden Störgrößen. Dabei zielt die Produktionssteuerung insgesamt auf die Bereitstellung von Ressourcen im richtigen Zeitraum am richtigen Ort unter Berücksichtigung aller Unternehmensbereiche, die an der Leistungserbringung für einen bestimmten Auftrag beteiligt sind [6].

Computergestützte Produktionssteuerung wird als *Computer Aided Manufacturing (CAM)* bezeichnet und hat zur Aufgabe, die Steuerung und Überwachung der Teilefertigung zu übernehmen (Abbildung 2-4). Unterteilt man die Produktion in logische Ebenen, so erhält man eine Ausführungsebene, in der Roboter, Werkzeugmaschinen und manuelle Arbeitsplätze zu finden sind. Darüber angesiedelt ist die Steuerungsebene mit Rechnern zur Steuerung der Maschinen der Ausführungsebene. Koordiniert werden die Steuerungsrechner durch einen Fertigungsleitrechner, der eine Ebene höher auf der Leitebene zu finden ist. Als oberste Schicht befindet sich die Planungsebene, mit Ihren Einflussgrößen aus der Konstruktion, Arbeitsplanung und Produktionsplanung und -steuerung. Anhand der Einflussgrößen auf der Planungsebene lässt sich auch hier wieder die Notwendigkeit eines funktionierenden Informationsflusses erkennen, der das gesamte virtuelle Produkt in den Bereich der Produktionssteuerung gelangen lässt. Nur anhand dieser Informationen kann eine Produktionssteuerung entwickelt und die Produktion später überwacht werden.

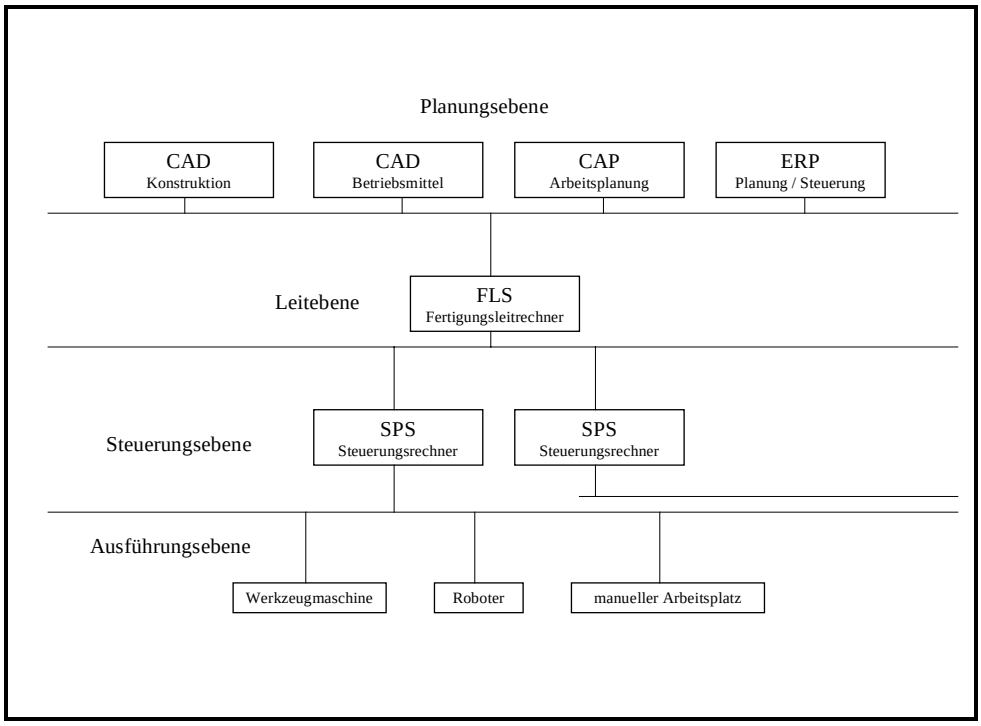


Abbildung 2-4: Unterteilung der Produktionssteuerung in logische Ebenen

2.6 Datenhaltung im Produktentstehungsprozess

Das in Kapitel 2 beschriebene virtuelle Produkt ist in der Fertigungsindustrie derzeit nicht durchgängig bzw. noch gar nicht eingeführt. Die Haltung produktbeschreibender Daten geschieht im Regelfall bereichsorientiert, d.h. es werden Datenbasen für Aufgabengebiete geschaffen, die untereinander nicht gekoppelt sind (Abbildung 2-5). Ein Informationsfluss oder gar die Weitergabe und Vervollständigung eines virtuellen Produkts ist auf diese Art nicht möglich.

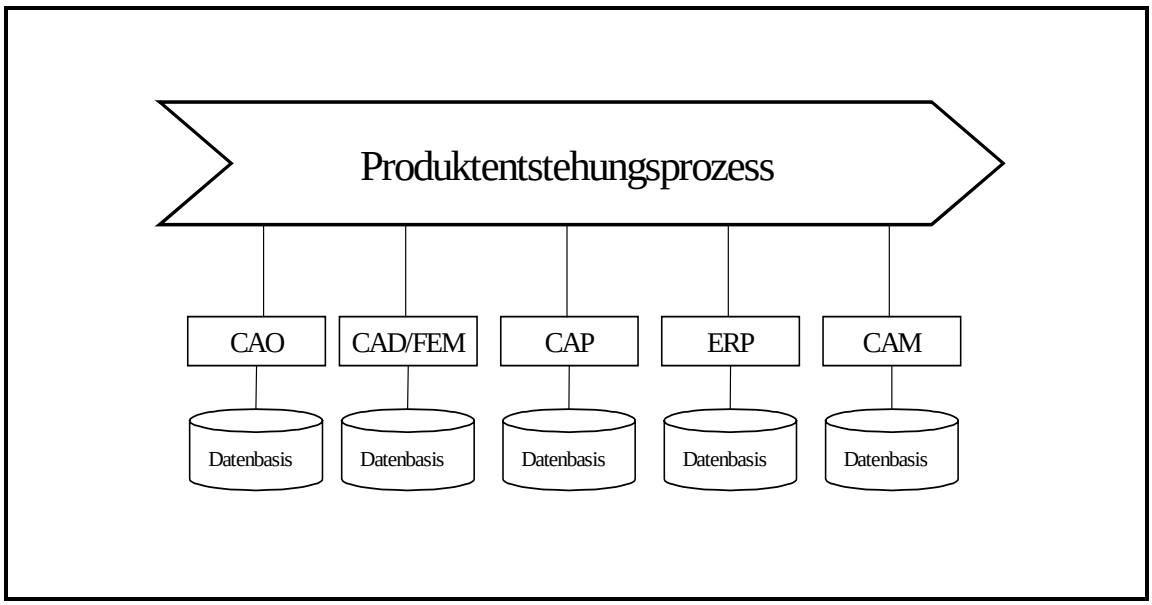


Abbildung 2-5: Systembezogene Datenhaltung [1]

Es muss berücksichtigt werden, dass alle erzeugten Daten ihren eigentlichen Nutzen verlieren, wenn nicht innerhalb kurzer Zeit auf sie zugegriffen werden kann. Suchfunktionen und Klassifizierungen erleichtern das Auffinden benötigter Daten, erfordern jedoch ihrerseits wieder beschreibende Daten. Es ist zur eigentlichen Datenbasis also eine zweite Datenbasis nötig, die zur Speicherung und Verwaltung sogenannter Metadaten eingesetzt wird.

Im Bereich des Computer Aided Office haben sich für derartige Aufgaben bereits *Dokumentenmanagementsysteme (DMS)*, auf dem Gebiet des Computer Aided Design sogenannte *Produktdatenmanagementsysteme (PDM)* durchgesetzt. Beide Systeme operieren in der Regel jedoch getrennt voneinander, die Erstellung eines virtuellen Produktes durch beide Systeme ist nur durch aufwendige Schnittstellenlösungen möglich. Die Koexistenz beider Systeme ist häufig überflüssig, da das Dokumentenmanagement eine Untermenge des Produktdatenmanagements und somit Teil eines jeden PDM-Systems ist. Aufgrund ihrer zunehmenden Bedeutung soll die Funktionsweise und Aufgaben klassischer PDM-Systeme im folgenden Kapitel erläutert werden.

2.7 Module klassischer PDM-Systeme

Die in Kapitel 2.6 angesprochenen Systeme zum Management von Produktdaten werden mit dem Ziel eingeführt, Kosten und Durchlaufzeit bei der Entstehung neuer Produkte, wenn möglich, zu verringern. Die Reduzierung von Entwicklungskosten wird erreicht, indem auf bereits vorhandene Konstruktionen zurückgegriffen wird und nur geringfügige Änderungen daran vorgenommen werden. Eine Verringerung der Durchlaufzeit wird durch schnellere

Informationssuche, Verkürzung von Kommunikationszeiten und der Unterstützung von *Concurrent Engineering* erreicht. Nachfolgend sollen die dafür notwendigen Teilsysteme derzeitiger PDM-Systeme vorgestellt werden. Dies sind

- Klassifizierung
- Dokumentenmanagement
- Änderungs- und Freigabemanagement
- Workflowmanagement
- Produktstrukturverwaltung

Klassifizierung

Mit Hilfe sogenannter Metadaten ist es möglich, Beschreibungen für Dokumente, Teile und Produkte innerhalb eines PDM-System zu erzeugen und als Stammsatz zu speichern. Somit wird eine sachbezogene Ablage der genannten Objekte möglich. Man spricht in diesem Zusammenhang von Klassifizierung. Eine Klassifizierung

- nach Dokumenten erfolgt hauptsächlich durch Einteilung der Dokumententypen. Als Beispiel seien hier 2D-Zeichnungen, 3D-Modelle, Textdokumente oder Tabellen genannt.
- nach Teilen wird durch eine Einteilung aller im Unternehmen befindlichen oder zu produzierenden Teile erreicht. Formelemente, Norm- und Kaufteile sowie standardisierte Teile und Baugruppen seien hier als Kriterien zur Klassifizierung genannt.
- nach Produkten bezieht sich auf die zugrundeliegende Produktstruktur und beinhaltet alle das jeweilige Produkt betreffenden Informationen.

Bei allen Arten der Klassifizierung gilt der Grundsatz: Je detaillierter man klassifiziert, desto mehr Aufwand steckt in der Anpassung des Systems und desto mehr Informationen muss der Benutzer beim Anlegen neuer Informationen eingeben.

Dokumentenmanagenment

Eine zentrale Aufgabe eines jeden PDM-Systems ist die Verwaltung der ständig wachsenden Menge digitaler Dokumente eines Unternehmens. Ziel ist es, allen Mitarbeitern den Zugriff auf alle im Unternehmen existierenden Dokumente in Abhängigkeit der Zugriffsberechtigung zu ermöglichen. Hierbei muß es für den Mitarbeiter unerheblich sein, wo das gesuchte Dokument physikalisch gespeichert ist und mit welcher Anwendung es erzeugt wurde. Heute unterstützen komfortable Suchmasken das schnelle Auffinden aller klassifizierter Dokumente. Darüber hinaus erleichtern die in einer Datenbank gespeicherten Stammsätze (Abbildung 2-6) die Beschreibung beim Anlegen eines neuen Dokuments.

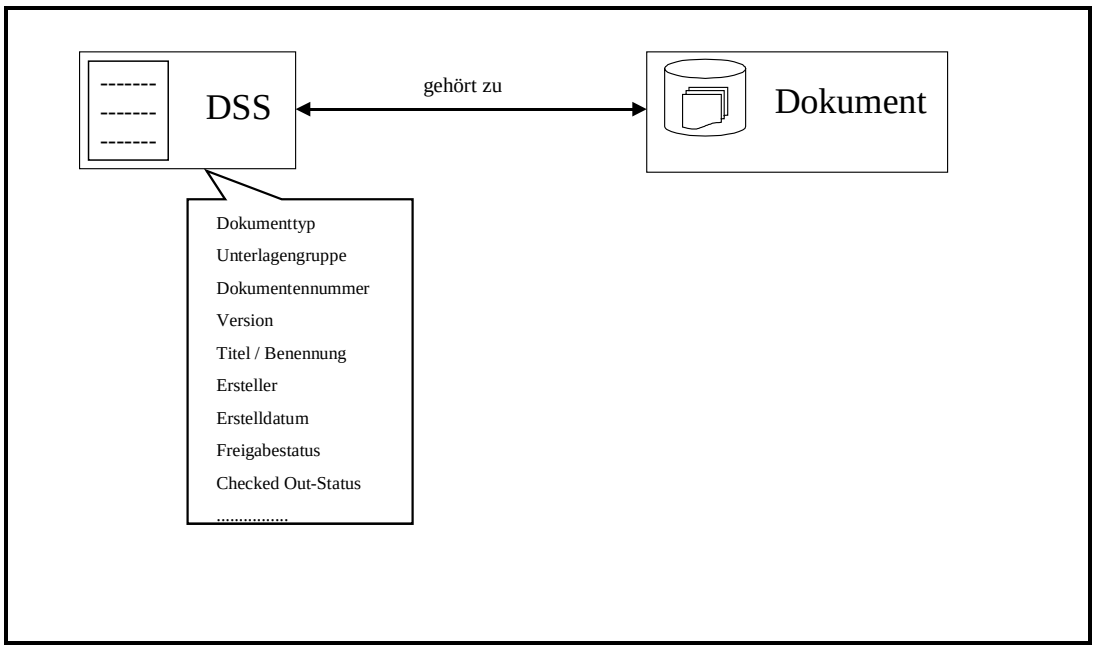


Abbildung 2-6: Ein Dokument und der dazugehörige Dokumentenstammsatz (DSS) mit Attributen

Ein weiterer Teil des Dokumentenmanagements ist das Versionsmanagement, das die Beziehung von alten zu neuen Dokumenten regelt. Dabei ist darauf zu achten, dass die eigentliche Produktstruktur aus Dokumenten besteht, die vorgegeben miteinander referenziert sind. Aufgabe des DMS ist es, Referenzierungen zu erhalten und gegebenenfalls neue Versionen einer gesamten Produktstruktur anzulegen.

Änderungs- und Freigabemanagement

Das Änderungs- und Freigabewesen unterliegt in der Regel strengen betriebspezifischen Restriktionen, um die Vorgaben für die Qualitätssicherung nach ISO 9001 zu erfüllen. Die gesamte Änderungshistorie eines Dokumentes oder einer Produktstruktur muss lückenlos nachverfolgt und unter gegebenen Umständen muss auf jeden Stand der Produktentwicklung

zurückgebaut werden können. Nur Modifikationen an bereits freigegebenen Objekten werden als Änderungen verstanden. Änderungen an generischen Objekten wie beispielsweise an 3D-Modellen haben Auswirkungen auf die gesamte Produktstruktur. Eine Konsistenz der Produktstruktur lässt sich nach einer Änderung in der Regel nicht rechnergestützt prüfen, so dass dafür Entscheidungsträger notwendig sind. Standardisierte Änderungsprozesse sind der Änderungsantrag *(ECR, Engineering Change Request)* und der Änderungsauftrag *(ECO, Engineering Change Order).* Durch einen ECR stellt ein Mitarbeiter einen Änderungsantrag an eine geschlossene Mitarbeitergruppe. Wird dem Antrag stattgegeben, entsteht daraus ein Änderungsauftrag. Beide Prozesse werden in der Regel durch das später beschriebene Workflowmodul unterstützt.

Das Freigabewesen erfordert von Entscheidungsträgern die Freigabe von Dokumenten oder ganzen Produktstrukturen. Es soll sicherstellen, dass nur fachlich einwandfreie Dokumente und insbesondere auf Machbarkeit geprüfte Zeichnungen zur Fertigung eines Produktes verwendet werden. Der Freigabemechanismus kann einen oder mehrere Freigabestati beinhalten und muss deshalb in jedem Fall auf betriebsspezifische Bedürfnisse angepasst werden. Typische Freigabestati sind

- *in Arbeit,*
- *geprüft und*
- *freigegeben.*

Workflowmanagement

Am oben beschriebenen Änderungs- und Freigabewesen sind in der Regel verschiedene Mitarbeiter oder Mitarbeitergruppen beteiligt. Jeder Mitarbeiter führt dabei einen ihm zugeteilten Arbeitsschritt aus.

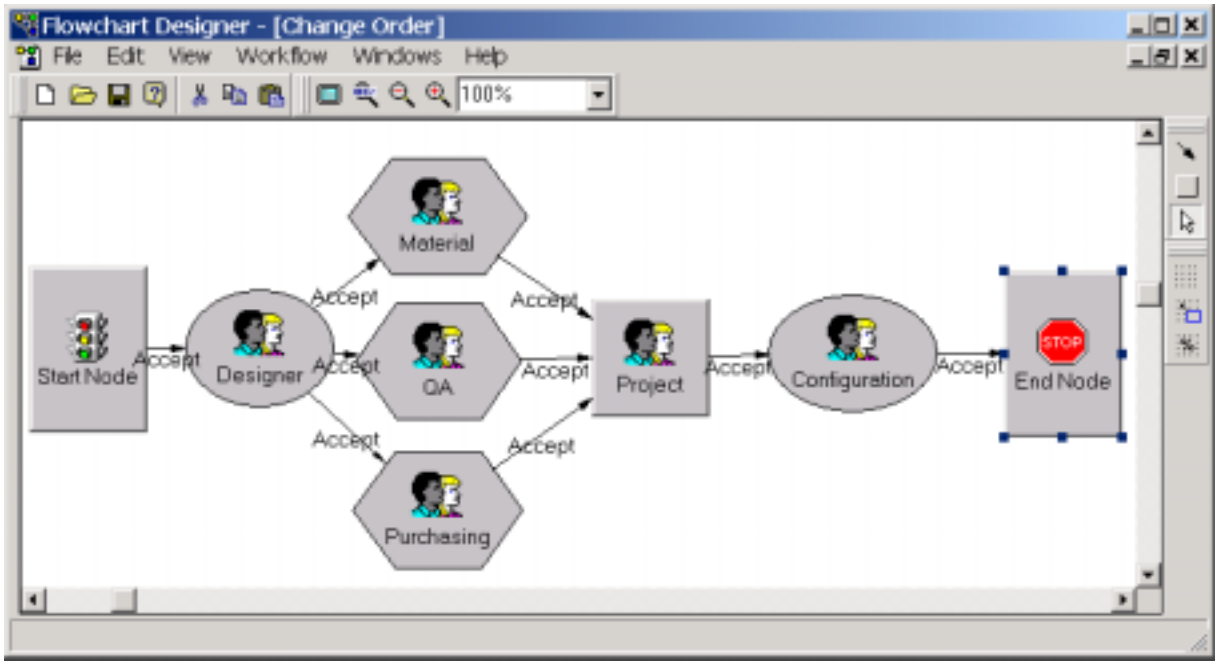


Abbildung 2-7: Graphischer Editor des Systems SmarTeam® zur Workflowmodellierung. Dargestellt ist eine Engineering Change Order (ECO).

Alle Arbeitschritte zusammen ergeben einen sogenannten *Geschäftsprozess*. Ziel des Workflowmanagements ist es, Geschäftsprozesse als Modell im Rechner darzustellen (Abbildung 2-7) und Arbeitsergebnisse gemäss dieses Prozessmodells an die jeweils nächste Mitarbeitergruppe weiterzuleiten. Die weiterzugebenden Arbeitsergebnisse können entweder nur aus Dokumenten oder aus beliebig komplexen Produktstrukturen bestehen. Man spricht in diesen Zusammenhang von sogenannten Workflowobjekten. Über das unternehmensinterne Rechnernetzwerk bekommen alle am Workflowprozess beteiligten Mitarbeiter eine Arbeitsmappe zugeteilt, in der zu bearbeitende Workflowobjekte und daran auszuführende Tätigkeiten enthalten sind.

Zunehmende Bedeutung erlangen in letzter Zeit parallele Workflows zur Unterstützung des Concurrent Engineering Ansatzes [2]. Dabei wird ereignisgesteuert ein zweiter Workflow gestartet, der zeitweise parallel zum ersten verläuft. Es ist darauf zu achten, dass bei Beendigung des ersten Workflows ein konsistentes Modell in den zweiten Workflow überführt wird. Da die Unterschiede der Produktmodelle des ersten und zweiten Workflows jedoch nicht vorausgesagt werden können, sind rechnergestützte Überführungen nicht möglich. In der Regel wird ein dritter Workflow angestoßen, der Mitarbeiter mit dieser Überführung beauftragt. Laufen mehr als zwei Workflows zeitgleich parallel, kann die Überführung beliebig komplex werden. Aus diesem Grund werden parallele Workflowmanagementsysteme kaum angeboten.

Produktstrukturverwaltung

Aufgabe der Produktstrukturverwaltung ist es, aus produktbeschreibenden Daten und Dokumenten durch Verknüpfungen ein virtuelles Produkt zu erstellen. Durch unterschiedliche Browsertechniken wird dem Benutzer die Navigation durch die Produktstruktur ermöglicht. Dabei werden Dokumenten-, Teile- und Produktstammsätze angezeigt und dem Benutzer durch einen integrierten Viewer eine Vorschau auf das selektierte Objekt ermöglicht.

Wichtige Funktionen der Produktstrukturverwaltung sind der Referenzierungs- sowie der Einbau- oder Teilenachweis. Der Referenzierungsnachweis *(where referenced)* gibt Aufschluss darüber, wo ein Dokument in der vorliegenden oder einer weiteren Produktstruktur referenziert ist. Durch einen Einbaunachweis *(where used)* wird dem Mitarbeiter ermöglicht, herauszufinden in welchen weiteren Produkten und an welcher Stelle ausgewählte Bauteile verwendet werden. Der Einbaunachweis ist besonders bei Änderungen von Teilen oder Unterbaugruppen von Bedeutung, wenn die Konsistenz aller von der Änderung betroffenen Produkte überprüft werden muss (vgl. Abbildung 2-8 und Abbildung 2-9).

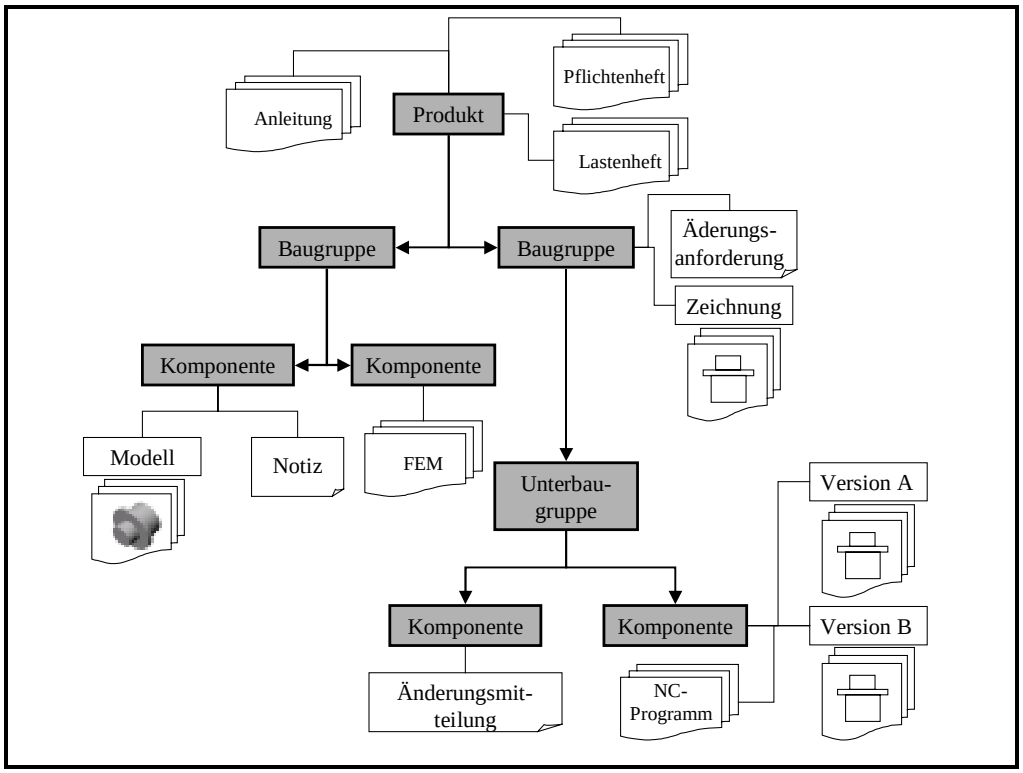


Abbildung 2-8: Beispiel einer Produktstruktur

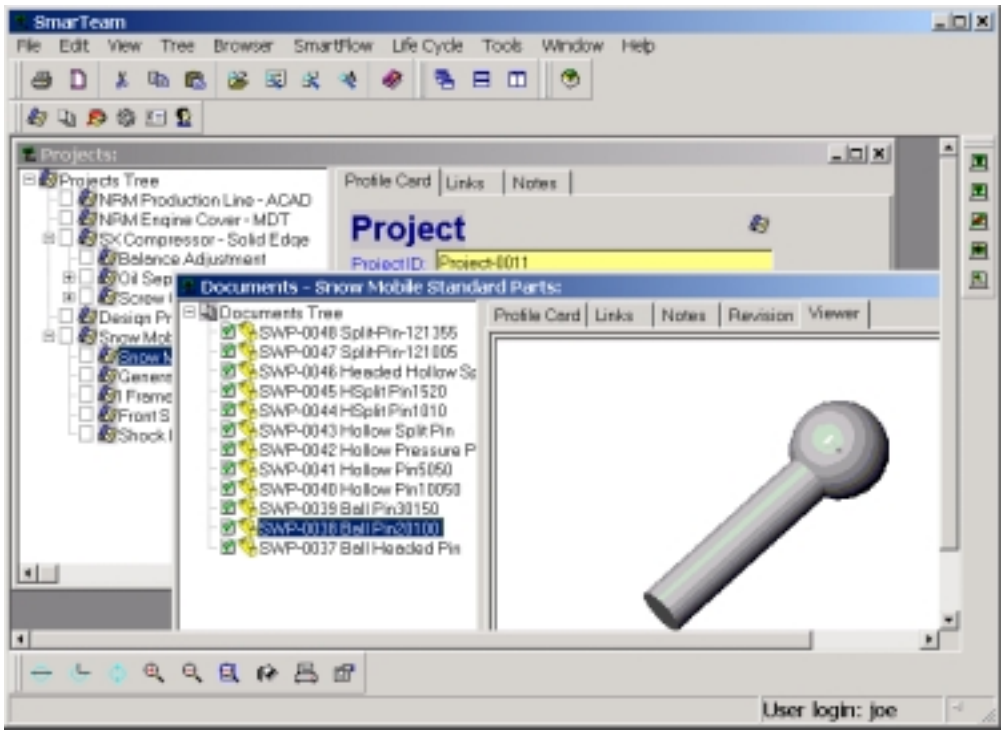


Abbildung 2-9: Darstellung einer Produktstruktur (hinten) mit integriertem Viewer.

2.8 Systemarchitektur von PDM-Systemen

PDM-Systeme wurden in der Vergangenheit hauptsächlich von CAD-Softwarehäusern angeboten. Ziel der Anbieter war es, Systeme zur Zeichnungsverwaltung auf dem Markt zu platzieren, um das eigene CAD-System zu unterstützen. Erst in letzter Zeit werden PDM-Systeme auch von Herstellern angeboten, die kein CAD-System besitzen. Diese Anbieter verfolgen die Absicht, ihr PDM-System als integriertes Informationssystem zu vermarkten. Schnittstellen zu Standardanwendungen sowie benutzerfreundliche Oberflächen zeichnen diese Systeme in erster Linie aus.

Die Architektur heutiger PDM-Systeme unterscheidet sich aus den oben genannten Gründen entsprechend ihrer Anwendungsschwerpunkte. So existiert kein einheitliches Architekturprinzip, nach dessen Vorgabe PDM-Systeme entwickelt wurden. Die Hersteller reagieren vielmehr auf Marktanforderungen um ein bestehendes System zu verbessern und zu erweitern. Dennoch lässt sich eine Referenzarchitektur beschreiben, die einem großen Teil aller PDM-Systeme gemein ist.

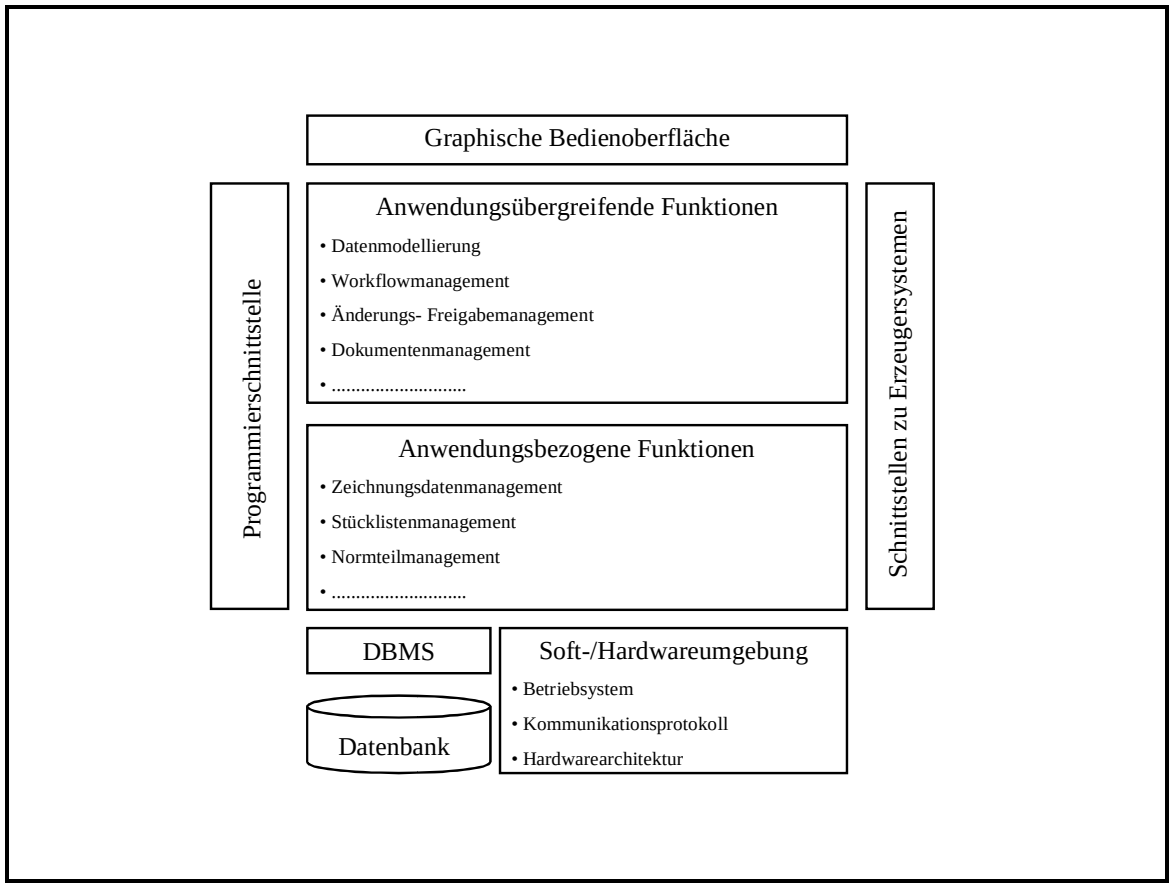


Abbildung 2-10: Referenzarchitektur heutiger PDM-Systeme

Den allgemeinen Aufbau der PDM-Systeme zeigt die Abbildung 2-10. Im einzelnen müssen die folgenden Bereiche aufeinander abgestimmt werden:

- Die *Soft-/Hardwareumgebung* besteht aus einem Betriebssystem, einem Netzwerkprotokoll zur Kommunikation mit Fremdrechnern und der Hardware selber. Parallel dazu befindet sich ein Datenbankmanagementsystem, das nicht zwingend auf dem gleichen Rechner aufgesetzt sein muss und somit eine andere Soft-/Hardwareumgebung beanspruchen kann.
- Die darüber liegenden PDM-Funktionen werden aufgeteilt in *anwendungsübergreifende und anwendungsbezogene Funktionen*. Anwendungsübergreifende Funktionen unterstützen Geschäftsprozesse über mehrere Schritte bzw. Anwendungen. Dazu gehört beispielsweise der Workflow oder das Freigabemanagement, die unabhängig vom Prozessschritt ein Objekt begleiten. Anwendungsbezogene Funktionen hingegen unterstützen Arbeitsaufgaben entsprechend ihrer Ausprägung. So bezieht sich das Normteilmangement ausschließlich auf die Arbeit mit einem CAD-System, nicht jedoch auf vorgelagerte oder nachfolgende Anwendungen.

- Die *graphische Bedienoberfläche* ermöglicht das Sichten und die Eingabe von Daten.
- Die *Schnittstellen zu Erzeugersystemen* beziehen sich sowohl auf anwendungsübergreifende als auch auf anwendungsbezogene Funktionen. Dem oben genannten Beispiel des CAD-Systems mit einer Schnittstelle zum Normteilmanagement lässt sich noch eine Kopplung zum Dokumentenmanagement hinzufügen, das für die Verwaltung aller erzeugter Zeichnungen verantwortlich ist.
- *Programmierschnittstellen* ermöglichen die Ankopplung und Integration von Anwendungsbausteinen zur funktionalen Erweiterung.

Die hier beschriebene Referenzarchitektur wird hauptsächlich in Systemlandschaften nach dem Client / Server-Prinzip eingesetzt. Es erfolgt eine Aufteilung der Module in zwei voneinander unabhängig arbeitenden Komponenten. Diese Aufteilung kann je nach Ausprägung des eingesetzten PDM-Systems stark variieren. So ist denkbar, dass alle anwendungsbezogenen und anwendungsübergreifenden Funktionen zentral auf einem PDM-Server aufgesetzt sind und lediglich die graphische Bedienoberfläche auf dem Client vorhanden ist. Häufig sind auch alle PDM-Funktionen auf dem Client vorhanden und der PDM-Server dient hauptsächlich als Datenbankserver. Wie eine Aufteilung der Softwarekomponenten stattfindet, richtet sich aber nicht nur nach der Architektur des PDM-Systems, sondern auch nach den spezifischen Anforderungen des Unternehmens. Es kann vorkommen, dass aufgrund häufiger Nutzung des Systems eine sehr hohe Netzwerkbelastung entsteht, wenn viele Benutzer die gleichen PDM-Funktionen auf dem Server aufrufen. In einem solchen Fall sollten die PDM-Funktionen auf dem Client eingebunden werden.

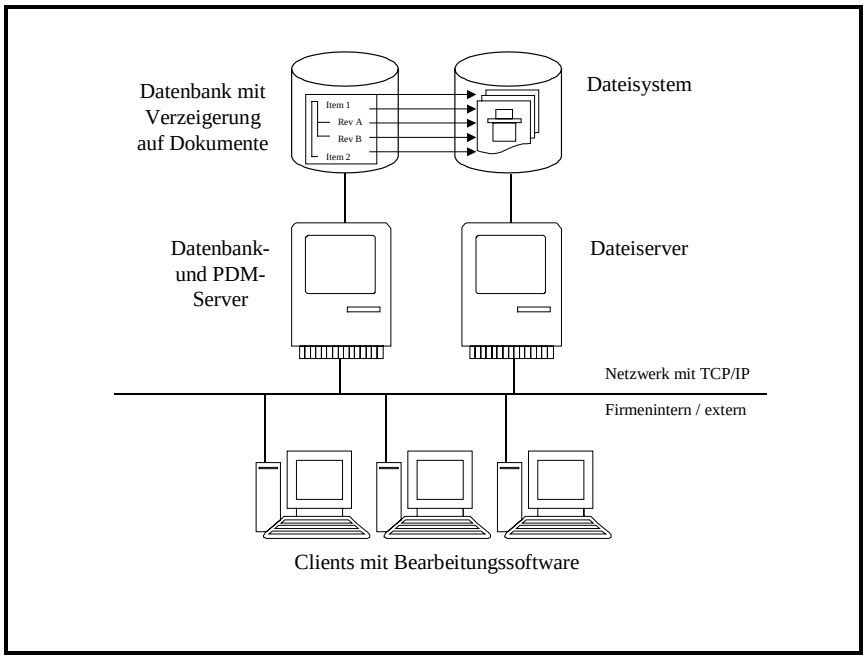


Abbildung 2-11: Client-Server-Architektur eines PDM-Systems

Abbildung 2-11 stellt schematisch den Aufbau einer Client-Server-Architektur dar, wie sie hauptsächlich in Unternehmen eingesetzt wird [1],[2]. Die Clients sind über das TCP/IP-Protokoll mit den Datenbank- und Dateiservern verbunden. Datenbankanfragen werden über TCP/IP vom Client an den Datenbankserver gestellt, um beispielsweise ein Dokument zu suchen. Soll ein gefundenes Dokument bearbeitet werden, so ist es für den Benutzer nicht notwendig zu wissen, wo sich dieses befindet. In der Datenbank sind alle notwendigen Informationen bezüglich des Dokuments gespeichert. Eine PDM-Arbeitsfunktion veranlasst das Auschecken des Dokuments, in dem das Dokument vom Dateisystem des Dateiservers auf einen bestimmten Arbeitsbereich des Clients kopiert wird. Die Dateiübertragung geschieht ebenfalls über TCP/IP, indem die Datei in kleine Pakete zerlegt und mit einem TCP-Header versehen wird.

Die oben angesprochene IT-Struktur des PDM-Systems zeigt den einfachsten, aber auch am häufigsten verwendeten Fall des PDM-Einsatzes. Eine beinahe beliebige Aufgabenverteilung bezüglich der Hardware ist hier zudem noch denkbar. So kann beispielsweise der PDM-Server zusätzlich als Dateiserver dienen oder als Client eingesetzt werden.

Durch Verwendung des TCP/IP-Protokolls wird auch der Einsatz eines PDM-Systems über Standortgrenzen hinaus ermöglicht. Im Zuge der zunehmenden Globalisierung ist das ein Aspekt, der nicht unberücksichtigt bleiben kann. Über ein Weitbereichsnetzwerk *(Wide Area Network WAN)* können PDM-Datenbasen sowohl verteilt als auch vernetzt sein. Verteilte

Datenbasen stellen in Ihrer Gesamtheit den Datenbestand einer PDM-Installation dar. Anders ausgedrückt heißt das, der Datenbestand eines Unternehmens wird nicht an einem zentralen Standort verwaltet, sondern über verschiedene Standorte verteilt. Jede Datenbasis eines Standortes entspricht bezüglich der Datenstruktur allen anderen Datenbasen. Die Daten der jeweiligen Datenbasen entsprichen im Allgemeinen denen, die an den Standorten auch erzeugt wurden. Aufgrund des einheitlichen Datenmodells ist das Konzept der verteilten Datenbasen relativ unflexibel, da eine Änderung an einem Standort erst die Zustimmung und Änderung an allen anderen Standorten erfordert. Es ist dann einzusetzen, wenn eine enge Zusammenarbeit gleichartiger Unternehmensbereiche möglich ist.

Vernetzte Datenbasen (Abbildung 2-12) bieten die Möglichkeit der Kopplung unabhängiger PDM-Installationen. Die Datenmodelle können verschieden aufgebaut sein und ermöglichen so auch die Zusammenarbeit unterschiedlicher Unternehmen. Jede PDM-Installation ist für sich eigenständig und hat auch so alle administrativen Aufgaben zu übernehmen. Sogenannte Proxy-User werden mit geringen Berechtigungen ausgestattet und sind Stellvertreter für Benutzer aus anderen PDM-Installationen. Ein Benutzer einer fremden PDM-Installation kann sich auf dem System als ein solcher Proxy-User anmelden und sich Informationen beschaffen. Die Übernahme der beschafften Daten in das Heimatsystem ist im Regelfall aufgrund unterschiedlicher Datenstrukturen jedoch nicht möglich. Die Vernetzung von Datenbasen ist zudem nur dann möglich, wenn die beteiligten Unternehmen das PDM-System eines Herstellers verwenden. Unterschiedliche PDM-Systeme lassen keine Vernetzung der Datenbasen zu.

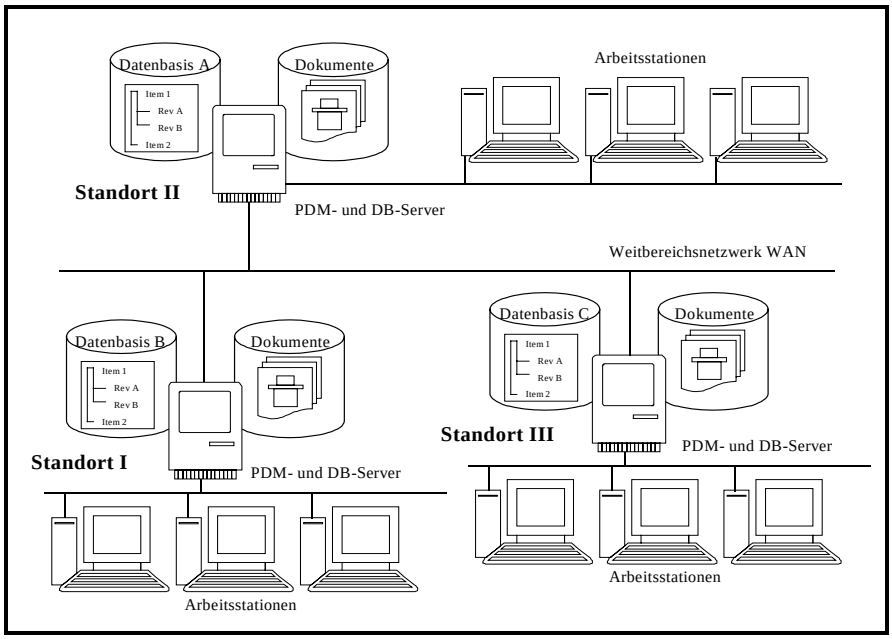


Abbildung 2-12: Vernetzte Datenbasen dreier PDM-Installationen

2.9 Problemfelder derzeitiger PDM-Systeme

PDM-Systeme verschiedener Hersteller decken mit ähnlichen Algorithmen Standardfunktionen des Produktdatenmanagements ab. Diese Standardfunktionen beziehen sich immer noch vornehmlich auf die Anbindung zu CAD-Systemen und unterstützen so nur den eigentlichen Konstruktionsprozess. Zudem kreieren die Funktionen Datenformate, die nur in einem PDM-System Verwendung finden können und so Kopplungen zu Fremdsystemen fast unmöglich machen. Aus diesem Grund sind für zukünftige PDM-System folgende Problemfelder aufzuarbeiten [2] :

- *Standardisierung*

  Jedes PDM-System arbeitet intern mit eigenen Formaten, die teilweise von Modul zu Modul differieren. Gleichartige Module verschiedener PDM-Systeme arbeiten mit unterschiedlichen Formaten. Diese Formate bestimmen die Grenzen der Integrationsfähigkeit, da die Daten der integrierten, spezialisierten Systeme in ein anderes PDM-System übertragen werden müssen. Dabei gehen immer Informationen verloren. Das Formatproblem ist besonders im Fall der Metadaten zu bearbeiten, da zur Zeit kein Austausch von Metadaten und somit keine Verbindung verschiedener PDM-Systeme möglich ist.

- *Integration*

  Um dem Anspruch der unternehmensweiten Informationsversorgung gerecht zu werden, genügen heutige Integrationen von Daten und Funktionen nicht. Integrationen beschränken sich im Regelfall auf ein CAD-System und diversen Standard-Office-Anwendungen. Kopplungen zu ERP-Systemen und Planungsmodulen, sowie Standardanbindungen an CAM-Systeme bilden mehr die Ausnahme als den Regelfall. Mit Integration ist hier aber auch die Anpassungsfähigkeit eines PDM-Systems an die unternehmensweiten Gegebenheiten gemeint. Für viele PDM-Systeme stellt es immer noch ein Problem dar, Datenmodelle frei zu konfigurieren und so auf die spezifischen Bedürfnisse eines Unternehmens anzupassen.

- *Parallelität*

  Die Unterstützung paralleler Entwicklungsprozesse innerhalb der Unternehmen und im besonderen über Unternehmensgrenzen hinaus stellt für PDM-Systeme ein unüberwindbares Hindernis dar. Immer noch ist der Informationsfluss

projektbezogener Entwicklungsteams über Unternehmensgrenzen hinweg mit hoher Datenredundanz und Synchronisationsproblemen behaftet. Mit Parallelität ist aber auch die gleichzeitige Verwendung von PDM- und ERP-Systemen gemeint, ohne dass durch aufwendige Kopplungen bzw. Schnittstellen die Arbeit des jeweiligen Sachbearbeiters negativ beeinflusst wird.

Im Folgenden soll nun ein Konzept für ein systemunabhängiges **TPIS** auf der Basis einer geeigneten Softwarearchitektur entwickelt werden, das alle diese Problemfelder innerhalb eines Unternehmens bereichsübergreifend lösen kann. Hierbei soll besondere Aufmerksamkeit dem Gedanken des virtuellen Produktes entgegengebracht werden, das standortunabhängig abgerufen und bearbeitet werden kann. Für die Kommunikation der verschiedenen beteiligten Software-Komponenten sollen nicht jeweils spezifische Schnittstellen entwickelt werden, sondern die auf dem Markt befindlichen Schnittstellen zum Einsatz kommen. Durch die Verwendung von Standardschnittstellen bleibt der Aktualisierungsaufwand der Schnittstellen für die Zukunft in einem vertretbaren Rahmen. Bevor nun auf das Konzept näher eingegangen wird, sollen zunächst die für das Konzept geeigneten Schnittstellenkonzepte kurz erläutert werden.

3 Grundlagen der Schnittstellen-Konzepte

Das Konzept des noch zu entwickelnden **TPIS** stützt sich im Wesentlichen auf das *Componenten Object Model (COM)* der Firma Microsoft®. Genutzt wird zudem die universelle Datenbankschnittstelle ADO, die ebenfalls als COM-Objekt ausgelegt ist. Aufgrund der Bedeutung für das **TPIS** soll in diesem Kapitel eine Vorstellung dieses Modells und der Datenbankschnittstelle erfolgen. Für das Verständnis der eingesetzten Programmiersprachen C++ und VisualBasic, des Internets und der benötigten DBMS sei hier auf die einschlägige Literatur verwiesen [7],[8],[9],[10].

3.1 Das Componenten Object Model (COM)

COM ist eine Spezifikation, die eine Erstellung von Softwarekomponenten ermöglicht, die sowohl programmiersprachen- als auch standortunabhängig sind. Sie schreibt vor, wie Softwarekomponenten aufgebaut sein müssen, um den o.g. Anforderungen gerecht zu werden. Diesen Spezifikationen folgend, kann COM auf fast allen Plattformen eingesetzt werden. Als bekannteste Betriebssysteme sind HP-UX®, Linux, SUN Solaris® und alle Versionen von Microsoft Windows® zu nennen .

3.1.1 Softwarekomponenten

Der oben angesprochene Begriff *Softwarekomponente* stellt per Definition [11] einen binären, wiederverwendbaren und programmiersprachenunabhängigen Softwareteil dar, der zusammen mit anderen Teilen eine gesamte Softwareapplikation ausmacht. Wichtig ist an dieser Stelle den Unterschied zwischen Softwarekomponente und Applikation zu verstehen. Eine Applikation stellt eine für sich abgeschlossene Programmeinheit dar. So ist beispielsweise Microsoft Word® als Applikation zu verstehen, solange dieses Programm zum Schreiben diverser Texte oder Briefe genutzt wird. Microsoft Word® kann aber über ein *Application Programming Interface (API)*[1] auch als Softwarekomponente genutzt werden. Rufen Fremdkomponenten zur Verfügung gestellte Dienste über diese API auf, so kann MS Word beispielsweise als Berichtgenerator für Datenbanksysteme eingesetzt werden.

[1] Application Programming Interface (API) (engl.): Programmierschnittstelle

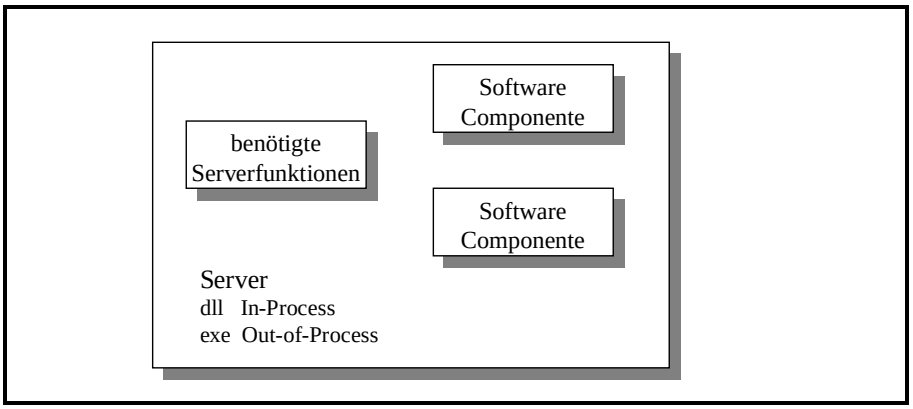


Abbildung 3-1: Zusammenlegung mehrer Softwarekomponenten in einem Server

Abbildung 3-1 zeigt, wie Softwarekomponenten im Betriebssystem Windows zusammenarbeiten. Dazu werden diese über einen Server veröffentlicht, der entweder als *dynamic link library (dll)*[2] oder als ausführbare Datei eingesetzt werden kann. Ein dll-basierter Server wird In-Process Server genannt, weil er im selben Prozessraum abläuft wie der dazugehörige Client. Ein Server, der aus einer ausführbaren Datei besteht, befindet sich in einem eigenen Prozessraum und wird deshalb Out-of-Process Server genannt.

3.1.2 Programmiersprachenunabhängigkeit

Eines der primären Ziele von COM ist die Wiederverwendbarkeit bereits erstellter Softwarekomponenten in beliebigen Programmiersprachen. Mit COM ist es möglich, die Dienste eines COM-Objekts unabhängig von der verwendeten Programmiersprache oder der Programmiersprache, in der das COM-Objekt implementiert wurde, zu nutzen. Verschiedene Programmiersprachen besitzen wiederum verschiedene Datentypen und Aufrufkonventionen für Funktionen, so dass theoretisch eine Konvertierung für den sprachunabhängigen Einsatz notwendig ist. Um dieses Problem zu lösen definiert COM ein Softwareinterface auf binärer Ebene, das Softwarekomponenten zur Kommunikation untereinander nutzen können. Hersteller von Entwicklungsumgebungen bieten Kompiler an, die Datentypen und Aufrufkonventionen der verwendeten Programmiersprache auf den durch COM definierten Standard umsetzen. Abbildung 3-2 stellt die Umsetzung durch COM-Mapping dar.

[2] dynamic link library (dll) (engl.): dynamisch zu ladende Bibliothek

Die in den Programmiersprachen C++, Java und Delphi geschriebenen Softwarekomponenten können durch COM-Mapping auf eine Geschäftsanwendung zugreifen, die in VisualBasic programmiert wurde. Dazu ist ebenfalls ein COM-Mapping des VisualBasic-Codes notwendig. Zwischen der Mappingkomponente von VisaulBasic und derer anderer Programmierspreachen ist COM als Schnittstellenstandard durch einen waagerechten Balken definiert. Dieser Balken stellt dass sogenannte Implementation Hiding durch COM dar. Die auf die in VisualBasic programmierte Geschäftsanwendung zugreifenden Softwarekomponenten machen keinerlei Annahmen über die Art der Implementierung innerhalb dieser Anwendung. Es lässt sich festhalten, daß COM keine Spezifikation der Strukturierung von Applikationen, sondern der Interoperabilität von Applikationen ist.

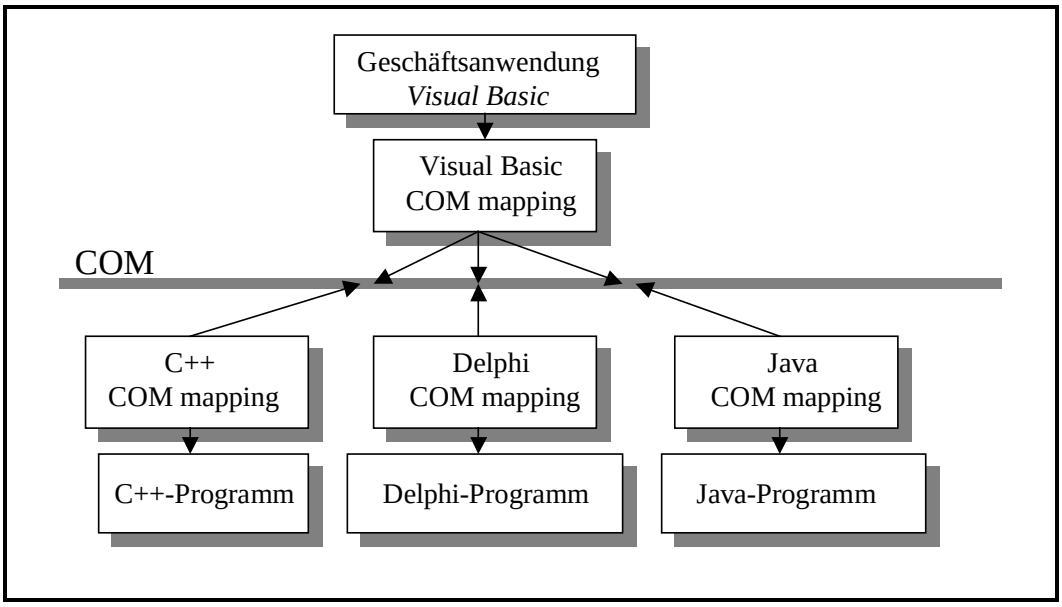


Abbildung 3-2: Programmiersprachenunabhängigkeit durch COM-Mapping

3.1.3 Komponententypen

Die in Kapitel 3.1.1 bereits angesprochenen Komponententypen unterteilen sich in prozessinterne und prozessexterne (lokale) Komponenten. Daneben existiert noch ein Remote-Komponententyp, der wie die anderen Typen im folgenden vorgestellt werden soll.

Prozessinterne Komponenten

Prozessinterne Komponenten werden zur Laufzeit dynamisch in den Prozessraum des Clients geladen. Verantwortlich dafür ist das Betriebssystem, das entsprechende Aufrufe an die Bibliothek erkennt und diese dann in den 32-Bit-Adressraum des Clients lädt, da Zugriffe auf nicht privaten Adressraum aus Gründen der Stabilität verboten sind. Ein großer Vorteil dieser prozessinternen Server liegt in der Performance, da zur Nutzung der bereitgestellten Dienste der Kontext nicht gewechselt werden muss. Nachteilig ist allerdings die Auslegung dieser Komponenten als DLL, die in dieser Form keine eigenständig ausführbaren Anwendungen darstellen können[11].

Lokale Komponenten

Lokale Komponenten werden im Gegensatz zu prozessinternen Komponenten nicht in den Adressraum des Clients geladen, sondern laufen in einem eigenständigen 32-Bit-Adressraum ab. Zugriffe auf Dienste dieser Komponenten gestalten sich deshalb auch sehr viel langsamer, da das Betriebssystem zwischen den Prozessräumen hin- und herwechseln muss. Vorteil der lokalen Komponenten ist allerdings die eigenständige Ausführbarkeit ohne aufrufenden Client. Die bereits angesprochene Textverarbeitung MS Word stellt eine solche lokale Komponente dar [11].

Remote-Komponenten

Remote-Komponenten befinden sich in Prozessräumen entfernter Rechner, die als Remote-Server über ein Netzwerk mit dem Client-Rechner verbunden sind. Im ersten Schritt konnte diese Funktionalität durch Einsatz von *DCOM*[3] *(Distributed COM)* realisiert werden, mit Einführung von *COM+*[4] wurde dieses Modell jedoch abgelöst. Applikationen lassen sich durch den Einsatz von Remote-Komponenten auf verschiedene Rechner verteilen, um damit beispielsweise eine ausgeglichene Lastverteilung zu erreichen. Dazu wird ein Proxy-Objekt im Prozessraum des Clients eingesetzt, das Schnittstellenaufrufe abfängt und diese über *Remote Procedure Call (RPC)*[5] zur realen Instanz transportiert. Im Prozessraum der realen Instanz befindet sich ein Stub-Objekt, das wiederum den Client repräsentiert. Der Client führt die

[3] DCOM Distributed COM (engl.): Komponenten Objekt Modell für den Einsatz in verteilten Anwendungen

[4] COM+ : Erweiterung des COM-Standards zur Verwendung in verteilten Anwendungen

Funktionsaufrufe auf die gleiche Art und Weise aus, wie er dies bei einer prozessinternen Komponente tut [3].

Das Proxy-/Stub-Objekt besteht aus einer dll, die wie bereits angesprochen in die jeweiligen Prozessräume geladen wird. COM bietet *Standard-Marshaling* für die Kommunikation mit prozeßexternen Komponenten, d.h. COM stellt durch den *MIDL-Compiler* ein generisches Proxy-/Stub-Objekt bereit. Zwei solcher Objekte können über Remote Procedure Calls miteinander kommunizieren und kennen die für Standardschnittstellen nötigen Marshaling-Operationen (Abbildung 3-3). Für das Marshaling von Methodenaufrufen über benutzerdefinierte Schnittstellen sind weitere von COM bereitgestellte Mechanismen nötig [10].

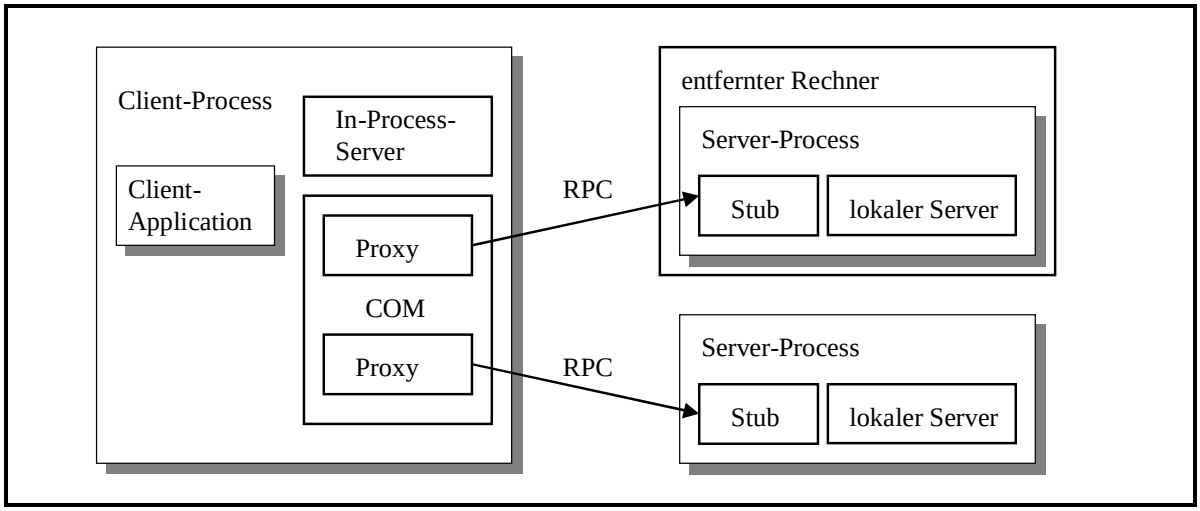


Abbildung 3-3: Komponententypen und Zugriffe auf diese durch einen Client

3.1.4 Interface Definition Language

Die *Interface Definition Language* (IDL) hat zum Ziel, sowohl Client- als auch Server-Projekte darin zu unterstützen, sich auf Kommunikationsschnittstellen zu einigen, sie zu spezifizieren und dann diese Vereinbarungen einzuhalten. Microsoft hat diese, von der OSF entwickelten, Schnittstellendefinition übernommen und um die Möglichkeit zur Definition von COM-Schnittstellen und -klassen erweitert. Das Ergebnis ist der Microsoft IDL-Compiler (MIDL).

[5] RPC Remote Procedur Call (engl.) : Funktionsaufruf über Netzwerke an entfernete Komponenten, definiert von der Open Software

Technisch gesehen ist MIDL kein Compiler, da kein Maschinencode erzeugt wird, sondern ein Übersetzer. Er übersetzt IDL-Quelltext in C/C++ und erzeugt dabei Dateien für das Proxy-/Stub-Objekt und Header-Dateien, die bei der Implementierung von Client und Komponente zu benutzen sind.

Da MIDL den Quelltext in einem einzigen, universellen Format generiert, das alle Programmiersprachen verstehen, sind Client und Komponente dadurch sprachenunabhängig implementierbar.

```
import "unknwn.idl";

[ object, uuid (00000001-0000-0000-0000-000000000001) ]
interface ISum : IUnknown
{
HRESULT Sum ([in] int x, [in] int y, [out,retval] int* retval);
};
```

Abbildung 3-4: Schnittstellendefinition in IDL

Dieser dehnbare Mechanismus funktioniert durch die Definition der in den generierten Dateien verwendeten universellen Schlüsselwörtern. Hierzu müssen spezielle Header-Dateien, welche diese Definitionen für die jeweils verwendeten Programmiersprachen enthalten, in Projekte, die mit MIDL definierte Schnittstellen benutzen, importiert werden. Um Inkonsistenzen bei der Verwendung einer Schnittstelle in verschiedenen Programmiersprachen vermeiden zu können, legt IDL die Größe aller Basisdatentypen konkret fest [12].

```
interface DECLSPEC_UUID("00000001-0000-0000-0000-
000000000001")

ISum : Public IUnknown
{
public:
  virtual HRESULT STDMETHODCALLTYPE Sum (
    /* [in] */ int x,
    /* [in] */ int y,
    /* [retval,out] */ int __RPC_FAR * retval) = 0;
};
```

Abbildung 3-5: Die übersetzte Schnittstelle als C/C++- Header-Datei

Foundation (OSF) und als Distributed Cvomputing Environment (DCE) bezeichnet.

3.1.5 Schnittstellen

Abbildung 3-4 stellt eine Schnittstellendefinition in der Beschreibungssprache IDL dar. Eine derartige Definition ist von großer Bedeutung, da die darin beschriebene Schnittstelle der einzige Weg ist, um auf eine COM-Komponente zuzugreifen. Sie ist der Vertrag zwischen der Komponente und dem Client, der sie nutzt. Der Vertrag legt nicht nur die verfügbaren Funktionen, sondern auch das Verhalten einer Komponente fest. Diese semantische Definition wird nicht im Rahmen einer spezifischen Implementierung des Objekts formuliert, es gibt also keine Möglichkeit, sie in C++-Code darzustellen, obwohl wir eine spezifische Implementierung in C++ darstellbar ist. Stattdessen bezieht sich die Definition auf das Objektverhalten, so dass Versionen des Objekts oder neue Objekte möglich sind, welche die Schnittstelle ebenfalls implementieren. Tatsächlich hat das Objekt die Freiheit, den Vertrag auf beliebige Weise zu implementieren, solange er eingehalten wird. Mit anderen Worten, der Vertrag muss außerhalb des Quellcodes dokumentiert werden. Dies ist besonders wichtig, weil Clients den Quellcode normalerweise weder erhalten noch benötigen. Der COM-Schnittstellenvertrag wird mit Auslieferung einer Komponente veröffentlicht und darf danach nicht wieder geändert werden, da sich andere Softwarekomponenten auf diesen verlassen.

Über Schnittstellen ist ein COM-Objekt in der Lage, seine Methoden beliebigen Clients anzubieten. Dabei kann ein einzelnes COM-Objekt mehrere Schnittstellen unterstützen. COM definiert eine Anzahl von Standardschnittstellen, die jedem Objekt gleich sind. Diese müssen, wie die vom Entwickler entworfenen benutzerdefinierten Schnittstellen, ebenfalls in die Komponente integriert werden. Zur Lokalisierung der Schnittstellen werden sogenannte *GUIDs*[6] eingesetzt, die sowohl den Komponenten als auch den darin implementierten Methoden zugeordnet sind. Der GUID ist ein 16-Byte-Wert, der in der Registry eines Rechners eingetragen (Abbildung 3-6) und zwischen Klassen-GUIDs (CLSID) und Schnittstellen-GUIDs (IID) unterschieden wird [13]. In Abbildung 3-4 und Abbildung 3-5 ist eine IID in der zweiten bzw. ersten Zeile zur Beschreibung der Schnittstelle *IUnknown* angegeben.

```
HKEY_CLASSES_ROOT
 CLSID
 {E312522E-A7B7-11D1-A52E-0000F8751BA7}=„PDM Class"
InprocServer32="D:\\PDMKernel\\PDM\\Debug\\PDM.dll"
```

Abbildung 3-6: Registry-Eintrag eines In-Process-Servers und dessen GUID

[6] GUID Global Unique Identifier (engl.): Globales, einzigartiges Erkennungszeichen

Die bereits angesprochene Schnittstelle *IUnknown* gehört zu den ebenfalls schon erwähnten Standardschnittstellen eines jeden COM-Objektes. Die Schnittstelle IUnknown ist von Bedeutung, da diese die Grundlage aller weiteren Standard- und Benutzerschnittstellen darstellt. Sie stellt drei Methoden zur Verfügung:

- *QueryInterface*. Bei der Erzeugung eines COM-Objektes bekommt der erzeugende Client immer einen Zeiger auf die angeforderte Schnittstelle der Server-Komponente zurück. Über diesen Schnittstellenzeiger kann der Client Methoden dieser Schnittstelle aufrufen. Für die Rückgabe des Zeigers ist die Methode QueryInterface verantwortlich, die per Definition Bestandteil jeder COM-Komponente sein muss.
- *AddRef*. Wird ein COM-Objekt im lokalen Spiecher eines Servers erzeugt, so wird, wie oben beschrieben, dem Client ein Schnittstellenzeiger zurückgegeben. Ein zweiter Client, der die Dienste des Objektes nutzen möchte, bekommt ebenfalls einen Zeiger auf die Schnittstelle zurück, es wird jedoch im Speicher des Servers kein zweites Objekt angelegt. Statt dessen existiert ein Referenzzähler, der die Anzahl der Clients wiedergibt, die Dienste eines Objektes nutzen. Die Methode *AddRef* wird für jeden Client einmal gerufen und erhöht den Referenzzähler jeweils um den Wert eins.
- *Release*. Benötigt ein Client die Dienste eines COM-Objektes nicht mehr, so ist die Rückgabe des Schnittstellenzeigers erforderlich. Dies wird durch die Methode *Release* ausgelöst, deren eigentliche Aufgabe darin besteht den Referenzzähler um den Wert eins zu erniedrigen. Ist der Wert des Referenzzählers nach Aufruf von Release auf Null gesetzt, wird das erzeugte COM-Objekt aus dem Speicher des Servers gelöscht.

Die zweite Standardschnittstelle *IClassFactory* benötigt die Existenz eines *COM-Klassenobjektes*, das typischerweise kein Verhalten besitzt, abgesehen davon, dass es neue Instanzen anderer Klassen erstellt. Eine Komponente muss mindestens ein *Klassenobjekt* besitzen, das eine Anzahl der in der Komponente enthaltenen Klassen instantiieren kann. Zuvor muss dieses natürlich selbst instantiiert werden. Ein Klassenobjekt ist ein Objekt, das die Schnittstelle *IClassFactory* unterstützt.

Es ist möglich, dass einige Klassen aufgrund besonderer Anforderungen durch ein alternatives Klassenobjekt der Komponente angesprochen werden. Beispielsweise könnte ein Entwickler eine Klasse mit einem Konstruktor erstellt haben. Die Standardschnittstelle *IClassFactory* bietet aber keine Möglichkeit zur Übergabe von Parametern bei der Instantiierung an den Konstruktor einer

Klasse. In diesem und in ähnlichen Fällen ist in COM eine benutzerdefinierte Schnittstelle von Nöten. Die Vorgehensweise bleibt dann dem Entwickler der Komponente überlassen.

IClassFactory besitzt zwei Methoden:

- *CreateInstance*. Instantiiert die gewünschte COM-Klasse und liefert dann einen Schnittstellenzeiger auf die Schnittstelle *IUnknown* des neuen Objekts.
- *LockServer*: Mit dieser Methode kann der Client erzwingen, daß eine Komponente im Speicher bleibt und somit weitere Objektinstanzen schneller erstellt werden.

3.1.6 Delegation und Aggregation

Sollte eine Komponente benötigt werden, welche zusätzlich zu Funktionen einer bestehenden COM-basierten Komponente eine Erweiterung bieten soll, lassen sich die entsprechenden Objekte von der neuen Komponente wiederverwenden. Um diesen Vorgang zu ermöglichen, sind die beiden Mechanismen *Delegation* und *Aggregation* entwickelt worden. Zur Laufzeit verwendet dabei ein Objekt andere Objekte, um deren Funktionen anbieten zu können. Eine einfache Vererbung der benötigten Klassen auf Quelltextebene ist nicht möglich, weil es sich bei COM um einen binären von der Implementierungsebene losgelösten Standard handelt. Eine COM-Klasse darf ausschließlich von abstrakten Basisklassen abgeleitet sein.

Delegation

Bei dieser Technik enthält ein Objekt ein anderes Objekt vollkommen. Der Trick ist hier, dass sich das deligierende Objekt, genannt *Container*, wie ein Client des eingebetteten Objekts verhält. Der Container gibt vor, eine oder mehrere Schnittstellen zu unterstützen. Dazu reicht er Anfragen, welche diese Schnittstellen betreffen, einfach an das innere Objekt weiter (Abbildung 3-7).

Die Funktionsweise der Delegation läßt sich wie folgt beschreiben: Über die Mehrfachvererbung implementiert die Containerklasse nicht nur die eigene Schnittstelle, sondern auch die des enthaltenen Objekts. Sobald das Container-Objekt instantiiert wird, erstellt es automatisch eine Instanz des inneren Objekts. Ein Client-Wunsch nach einer Methode resultiert nun zunächst in einem Aufruf dieser an die äußere Schnittstellenattrappe, innerhalb dieser Methode wird dann ein Funktionsaufruf der echten Methode durchgeführt, deren Ergebnis danach einfach zurückgeleitet wird.

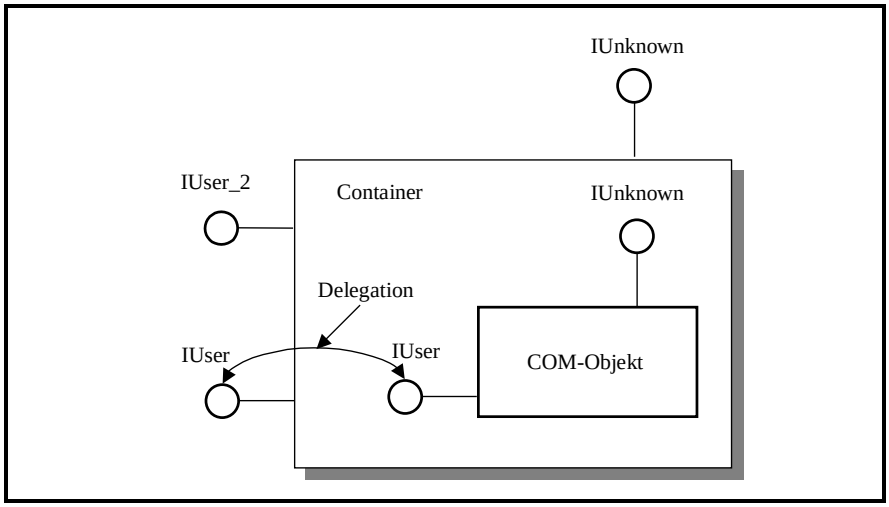


Abbildung 3-7: Delegation der Schnittstelle IUser

Aggregation

Dieser Mechanismus kann nur zwischen prozeßinternen Komponenten eingesetzt werden und meint, dass die Schnittstellen des internen Objekts tatsächlich als Schnittstellen des äußeren Aggregators offengelegt werden. Im Gegensatz zur Delegation kann der Client also direkt mit dem internen Objekt kommunizieren. Aggregation wird von COM in besonderer Weise unterstützt. In Abbildung 3-8 ist die Aggregation eines Objektes zu sehen. Auch hier wird das interne Objekt bei einer Instantiierung des Aggregators automatisch erstellt. Der Aggregator implementiert allerdings nur die Schnittstelle *IUser_2*. Wenn ein Client die Methode *IUser* benutzen möchte, nachdem er gerade vom Klassenobjekt einen Schnittstellenzeiger auf *IUnknown* des äußeren Objekts empfangen hat, liefert *QueryInterface* durch Weiterreichen an die Schnittstelle *INoAggregationUnknown* des internen Objekts den gewünschten *IUser*-Schnittstellenzeiger. Damit kann der Client, anders als bei der Delegation, jetzt direkt auf das interne Objekt zugreifen.

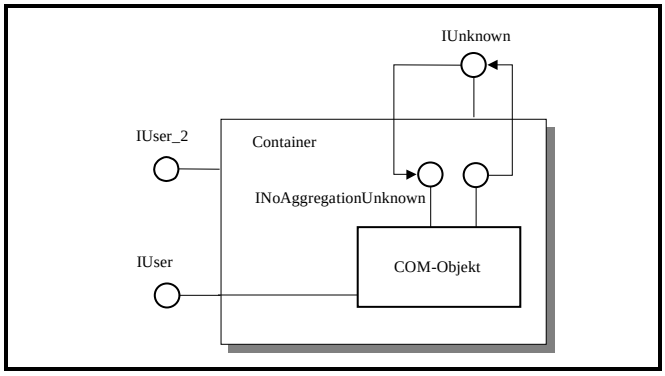


Abbildung 3-8: Aggregation einer Schnittstelle

3.2 ActiveX Data Objects ADO

Das im letzten Kapitel vorgestellte COM-Modell wurde von Microsoft ® zur Erstellung zweier Datenschnittstellen verwendet. Mit Hilfe dieser ist der Programmierer in der Lage, nicht nur auf Datenbanken, sondern auf beliebige Daten zuzugreifen. Microsoft benennt diesen Gedanken mit *Universal Data Access*[7] und legt die Technologien OLE DB und ADO[8] zugrunde. OLE DB und ADO wurden geschaffen, um die bisherigen Datenbanktreiber ODBC[9] und DAO[10] abzulösen. Beide Modelle wurden auf Grundlage des COM-Modells erzeugt und beinhalten dadurch alle Vorzüge dieses Programmiermodells [10].

OLE DB ist die Grundlage der UDA-Strategie. ADO wiederum basiert auf OLE DB und kann im Gegensatz dazu in verschiedensten Umgebungen eingesetzt werden. Dazu zählen beispielsweise Visual Basic oder diverse Skriptsprachen. Verschiedenste Einsatzmöglichkeiten sind der Grund für die Verwendung der ADO-Technologie als Datenbankschnittstelle im **TPIS**. Nachfolgend wird das zugrundeliegende Konzept näher erläutert.

3.2.1 Die ADO-Architektur

Die ADO-Architektur besteht im wesentlichen aus sieben COM-Objekten:

- Connection
- Recordset
- Command
- Error
- Field
- Property
- Parameter

[7] Universal Data Access (UDA) (engl.): universeller Datenzugriff

[8] ADO ActiveX Data Objects (engl.): COM-Bibliothek, die Objekte zum Zugriff auf Datenbanken enthält

[9] ODBC Open Database Connectivity (engl.): offene Datenbankverbindung

[10] DAO Data Access Objects (engl.): Datenzugriffsobjekte

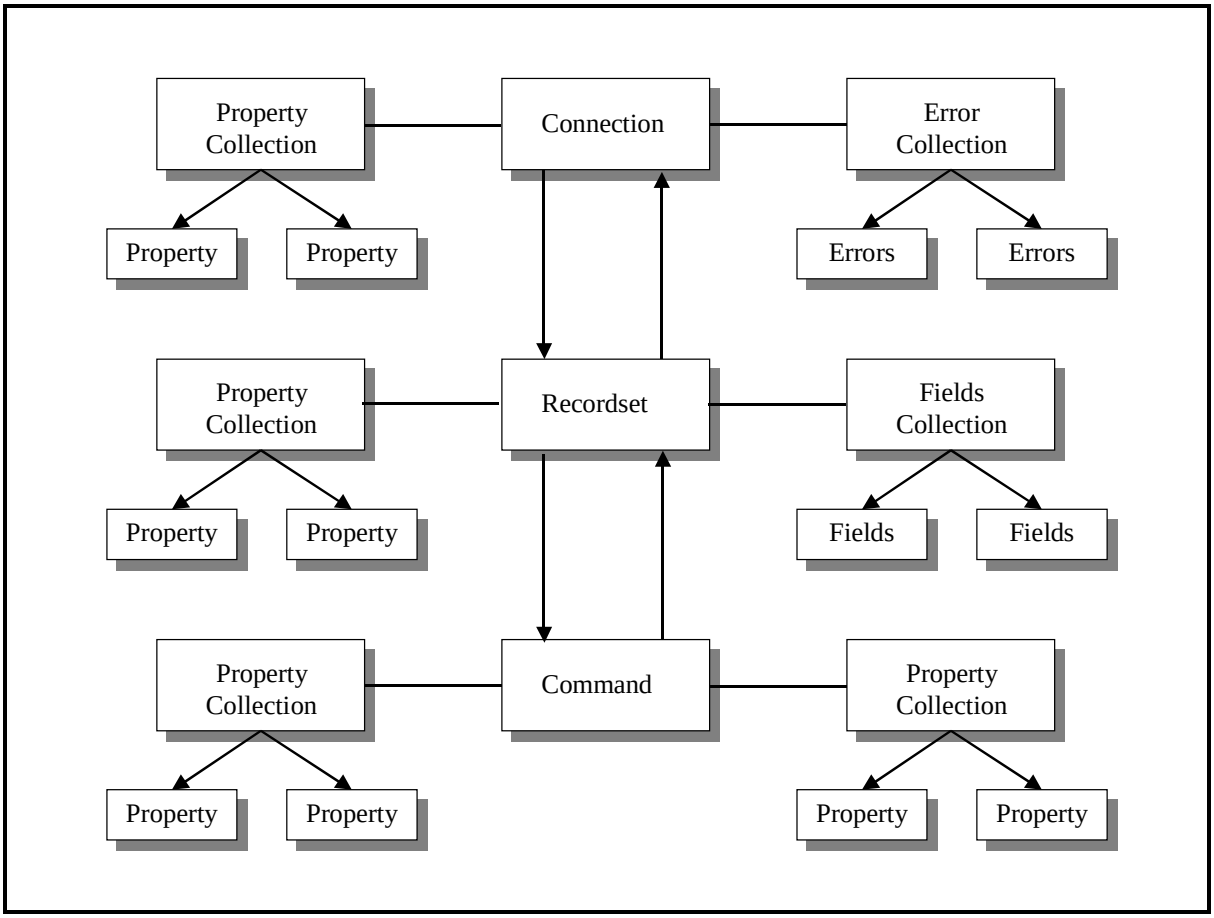


Abbildung 3-9: Das ADO-Objektmodell

Für den Ablauf einer Datenbankverbindung und –Abfrage sind jedoch nur die ersten drei Objekte von Bedeutung. Es soll deshalb eine kurze Vorstellung dieser Objekte folgen.

Das Connection-Object

Das als COM-Object ausgeführte Connection-Object repräsentiert eine aktive Datenbankverbindung. Wie in Abbildung 3-9 dargestellt, besteht es aus einer Sammlung von Eigenschaften (links) und Fehlern (rechts). Die Eigenschaften werden gesetzt um den Ort und die Art einer zu öffnenden Datenbasis anzugeben. Außerdem befinden sich hier Informationen über den angemeldeten User und das Passwort. Beim Öffnen einer Datenbank werden diese Attribute zu einem Connection-String zusammengefasst und beim Erstellen des Objektes übergeben (Abbildung 3-10).

```
' Öffnen einer Verbindung.
 strCnn = "Provider=sqloledb;" & _ "Data Source=srv;Initial Catalog=pubs;User Id=sa;Password=; „
 Set cnn1 = New ADODB.Connection
cnn1.Open strCnn
```

Abbildung 3-10: Setzen eines Connection-Strings und öffnen eines Connection-Objektes in Visual Basic

Weitere Eigenschaften, die beim Verbindungsaufbau gesetzt werden können, sind die Isolationsstufe, der Schreib- bzw. Lesemodus und der Cursorstandort, die folgende Auswirkungen haben:

- Der Cursorstandort entscheidet, welche Bibliothek geladen wird, um Navigationen innerhalb eines Datensatzes vorzunehmen. So kann die Bibliothek des Clients geladen werden um Cursorfunktionen auszuführen, was in Bezug auf die Performance deutliche Vorteile mit sich bringt. Die Performance bei serverseitiger Nutzung der Bibliothek ist zwar etwas schlechter, doch werden Änderungen durch andere Benutzer sofort übergeben. Die Nutzung der serverseitigen Bibliothek ist standardmäßig eingestellt und wird aufgrund der Aktualität auch im Datenbankkern eingesetzt.
- Die Isolationsstufe ist ein Maß dafür, wie isoliert Transaktionen nebeneinander ablaufen. Greifen mehrerer Benutzer auf eine Datenbasis zu und führen dort Transaktionen aus, so sollte eine geeignete Isolationsstufe gewählt werden, um gegenseitige Behinderungen zu vermeiden. Insgesamt sind fünf Isolationsstufen bekannt, wobei die höchste Stufe bevorzugte Schreibrechte auf die Datenbasis besitzt und Änderungen niedrigerer Isolationsstufen überschreibt. Für den Datenbankkern muss die Isolationsstufe so gewählt werden, dass beispielsweise Änderungen durch interne Mitarbeiter auf einer höheren Stufe abläuft als Änderungen externer Mitarbeiter.
- Der Schreib- und Lesemodus gibt an, ob ein Benutzer Schreib- oder Leserechte auf die Datenbasis bekommt. Dieser Modus muss im Datenbankkern ähnlich der Isolationsstufe in Abhängigkeit des Mitarbeiterstatus gesetzt werden. Insgesamt sind acht verschiedene Zugriffsrechte möglich, die Wichtigsten sind der Vollzugriff und die einzelne Schreib- oder Leseberechtigung.

Das Recordset-Object

Das Recordset-Objekt repräsentiert, wie bereits mehrfach erwähnt, einen Datensatz der als Ergebnismenge einer Abfrage gegen eine Datenbasis entsteht oder zur Datenmanipulation von Hand erstellt und durch eine Transaktion in die Datenbasis eingepflegt wird. Die zur Navigation innerhalb des Datensatzes bereits im Connection-Objekt vorgestellten Cursor können anhand vorgegebener Parameter den jeweiligen Bedürfnissen angepasst werden. Ein dynamischer Cursor erlaubt den Benutzern den sofortigen Zugriff auf Aktualisierungen durch andere Benutzer. Es wird also nur mit tatsächlich bestehenden Daten einer Datenbasis gearbeitet, was allerdings zu einem Performanceverlust führt. Ähnlich arbeitet ein Schlüsselcursor, der allerdings nur Änderungen anderer Benutzer an bereits vorhandenen Recordsets wiedergibt. Das Löschen und Hinzufügen von Recordsets durch weitere Benutzer wird durch diesen Cursortypen nicht wiedergegeben.

Ein weiterer, im Datenbankkern eingesetzter Cursortyp, ist der statische Cursor. Ein zu bearbeitender Datensatz stellt nur eine Momentaufnahme der Datenbasis dar. Änderungen durch anderer Benutzer bleiben völlig verborgen. Vor allem bei Webanwendungen kommt dieser Cursortyp häufig zum Einsatz und erlaubt zudem die Aktualisierung der Datenbank in einem Batchlauf. Dazu können beliebige Änderungen am Recordset vorgenommen werden, ohne dass diese direkt auf die Datenbasis übertragen werden. Erst zu einem bestimmten Zeitpunkt wird der gesamte Recordset durch Aufruf der Methode *UpdateBatch()* an die Datenbank geschickt und dort übernommen. Der letzte und unbedeutendste Cursortyp erlaubt eine Navigation innerhalb des Recordsets in nur eine Richtung.

Das Command-Object

Das Command-Objekt wird eingesetzt, um bei SQL-Datenbanken den SQL-Befehl aufzunehmen. Bei anderen Datenquellen kann das Command-Objekt entsprechende Datenmanipulationsobjekte darstellen. So besteht die Möglichkeit, das Command-Objekt für die Abfrage über Stored Procedures oder persistente Objekte einzusetzen. Parameterisierte Abfragen sind durch Verwendung zugehöriger Parameter-Objekte möglich, die während der Laufzeit in die Abfrage integriert werden.

3.3 XML

Die vorigen Kapitel definierten das COM-Modell und dem darauf basierenden Datenzugriff über ADO. Solange von ADO bereitgestellte Daten innerhalb einer Applikation verarbeitet werden können, eignen sich dazu die beschriebenen Recordset-Objekte. Im Zeitalter des Internets wird es jedoch von immer größerer Wichtigkeit, ausgewählte Daten einer Datenbasis über Standortgrenzen hinweg auf andere Rechner zu transportieren und dort zu visualisieren. Für derartige Funktionen wird die Sprache XML auch bei der Anwendung des **TPIS** genutzt.

Seit Jahren hat sich dazu das W3[11]-Konsortium mit diesem Thema beschäftigt und eine neue Auszeichnungssprache mit dem Namen *XML* definiert. Diese erfüllt im wesentlichen die folgenden, vom W3C aufgestellten Forderungen [14]:

- *Flexible Datenstruktur.* Die Sprache basiert nicht auf einer festen Datenstruktur, sondern ist ständig erweiterbar. Sie passt sich den wechselnden Gegebenheiten und Anforderungen bezüglich der Dateninterpretation an und lässt sich leicht in andere Formate konvertieren.

- *Plattformunabhängigkeit*. Die Daten, die mit dieser neuen Auszeichnungssprache definiert werden, sind flexibel zwischen verschiedenen Plattformen austauschbar. Dazu ist die Einhaltung eines offengelegten Standards nötig, der von jedem Unternehmen oder Institution aufgenommen und in eigene Programme implementiert werden kann. Lizenzkosten fallen beim Gebrauch dieser Sprache nicht an.

- *Textorientiert*. Der plattformübergreifende Aspekt verlangt, dass keine binäre Datenstruktur zur Speicherung verwendet wird. Deshalb präsentiert sich XML als Textformat, das Ähnlichkeiten zu *HTML* aufweist und deren Kennzeichnungen der einzelnen Daten durch Textmarken erzeugt wird. Eine Beschränkung auf die 128 Zeichen des US-ASCII-Zeichensatzes sind bei der Erstellung von XML-Dateien einzuhalten.

Dem Gedanken des W3C folgend, soll XML als Basissprache für jede Art von Daten eingesetzt werden. Aus diesen Daten können dann beliebige Arten von Dokumenttypen generiert

[11] W3-konsortium (W3C) : World WideWeb Consortium, freiwilliger Zusammenschluss von Firmen und Institutionen zur Vorantreibung des Internets.

werden. Die heutige anfallende Doppelarbeit von Erfassung und späterer Konvertierung fällt weg.

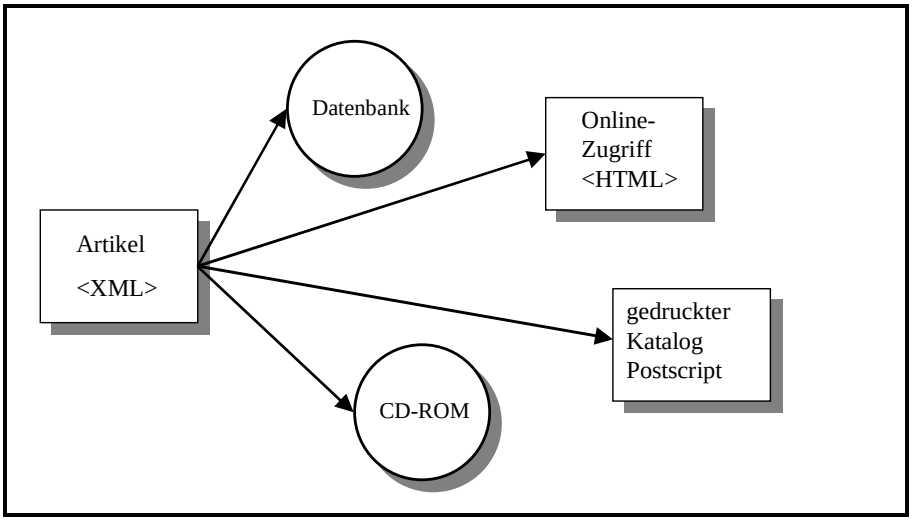


Abbildung 3-11: XML als Ausgangssprache für die Weiterverarbeitung [14]

Abbildung 3-11 stellt den Übergang einer XML-Datei, die einen Artikelstammsatz beinhaltet, in verschiedene Typen zur Weiterverarbeitung dar. Dazu wird der Text der XML-Datei entsprechend interpretiert. Im Gegensatz zu einer anderen Auszeichnungssprache wie beispielsweise HTML ist es dazu notwendig, die enthaltenen Informationen zu strukturieren. Es steht nicht im Vordergrund wie die Informationen dargestellt werden, sondern wie sie strukturiert sind.

```
<Motor>
  <Anlasser>
    <Anlasserwelle>
      <Welle.prt Beschreibung="Welle für Anlasser CrMo" Ersteller="Hubert" DocId="P1456“
      <Welle.vrml Beschreibung="Welle für Anlasser " Ersteller="Hubert" DocId="P1457"
    </Anlasserwelle>
    <Anlassergehäuse>
      <Gehäuse.prt Beschreibung="Welle für Anlasser " Ersteller="Hubert" DocId="P1457"
    </Anlassergehäuse>
  </Anlasser>
  <Lichtmaschine>
    <Welle/>
    <Gehäuse/>
  </Lichtmaschine>
</Motor>
```

Abbildung 3-12: Beispiel einer Produktstruktur im XML-Datenformat

Der in Abbildung 3-12 dargestellte Artikelstammsatz macht die Strukturierung deutlich. Die Textmarken <*Motor*> und </*Motor*> markieren den Beginn bzw. das Ende des Artikelstammsatzes. Dazwischen gliedert sich der Artikel hierarchisch auf, so dass der Motor in Anlasser und Lichtmaschine unterteilt wird. Auch diese Teilestammsätze beginnen und enden mit entsprechenden Textmarken. Die beschreibenden Attribute oder Metadaten eines Teils stehen in einer Zeile, in der auch eine Wertzuweisung erfolgt.

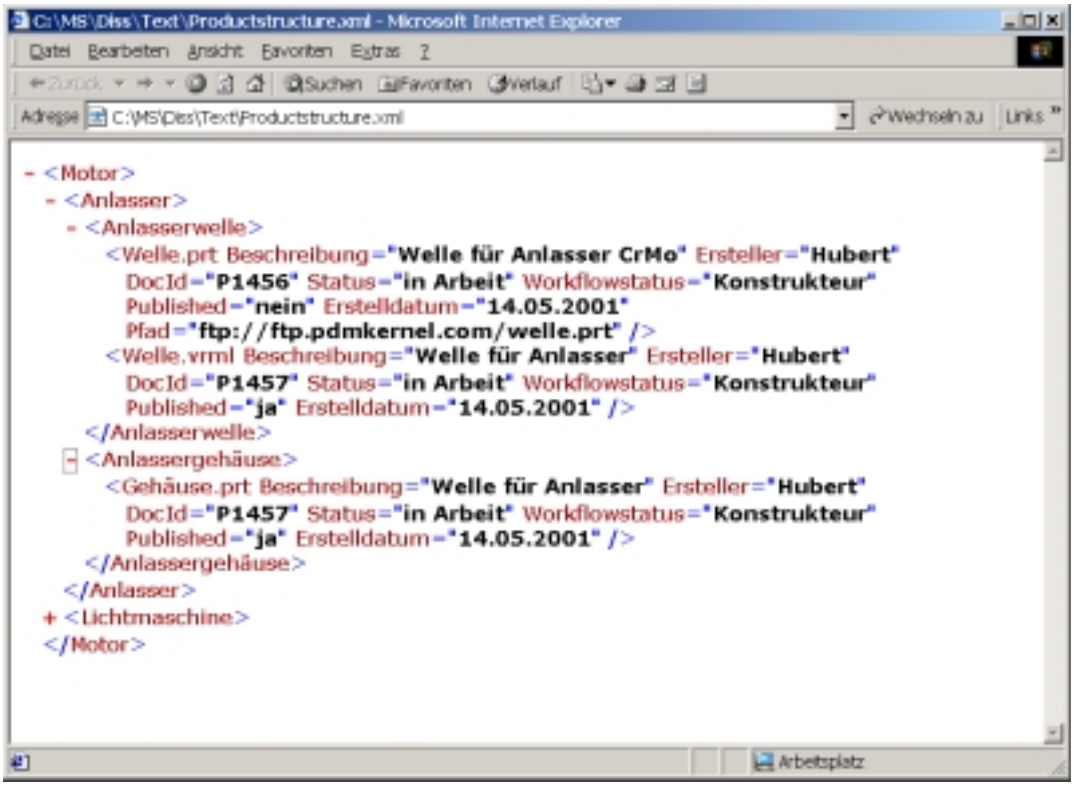


Abbildung 3-13: Beispiel für die Darstellung einer XML-Datei in einem Internetbrowser

Für Abbildung 3-13 wurde der Microsoft® Internetexplorer verwendet. Eine Visualisierung der oben beschriebenen XML-Datei ist damit möglich. Zu erkennen ist hier die strukturierte Darstellung und die Möglichkeit zur Navigation in dieser Hierarchie. Das Auf- und Zuklappen des Baumes erfolgt durch die Betätigung des „+“ bzw. „-“ –Zeichens.

4 Konzept eines technischen Produktinformationssystem (TPIS)

In den vorausgegangenen Kapiteln wurde der Produktentstehungsprozess unter dem Aspekt des Informationsflusses auf der Basis eines virtuellen Produktes erläutert. Ferner wurde die Architektur und Arbeitsweise von PDM-Programmen angesprochen, die den Prozess der Produktentstehung vor allem im Bereich der Konstruktion unterstützen. Die abschließend diskutierten Probleme derzeitiger Systeme sollen im folgenden durch ein Konzept gelöst werden, dessen Ansatz die Systemintegration in den Vordergrund stellt.

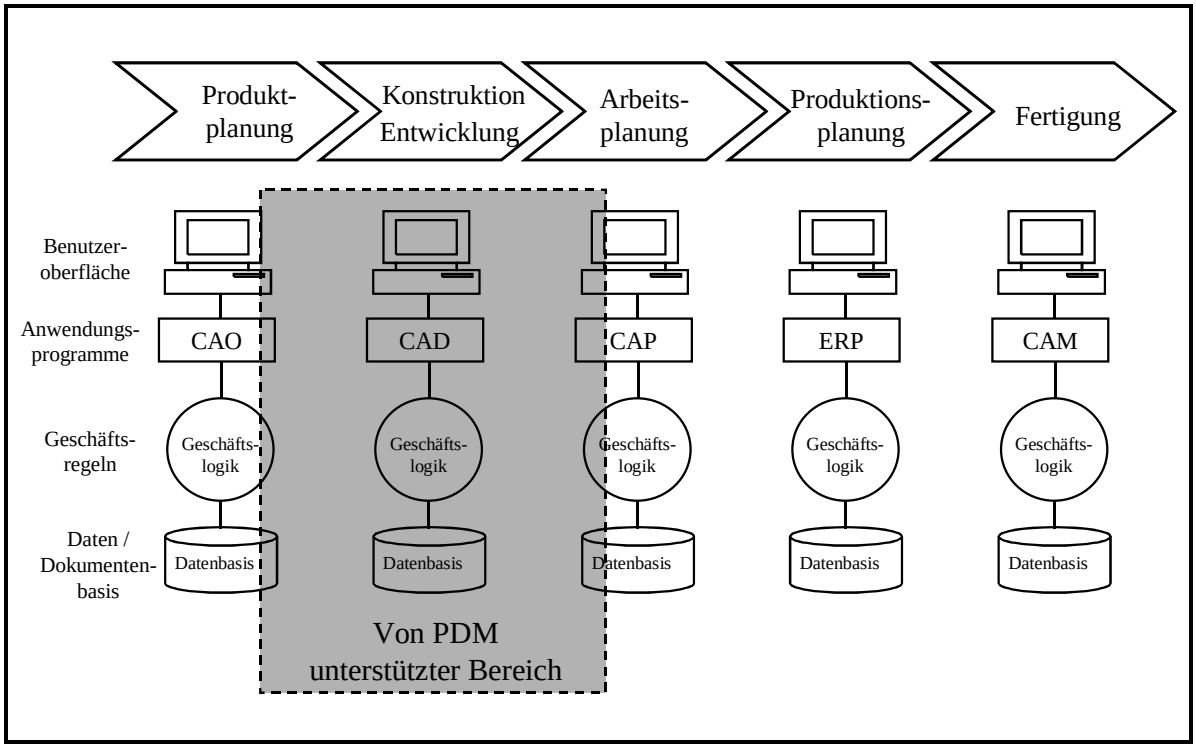


Abbildung 4-1: Einsatzbereich von PDM-Systemen bei konventioneller Datenhaltung

Abbildung 4-1 stellt den Einsatzbereich konventioneller PDM-Systeme dar. Dieser beschränkt sich bezüglich des Produktentstehungsprozesses (horizontal) auf den Bereich der Konstruktion und sich direkt anschließende bzw. vorgelagerte Bereiche. Bezüglich der Datenhaltung (vertikal) wird eine durchgängige Lösung von fast allen PDM-Herstellern angeboten. Ausgehend von einer überall vorhandenen Benutzeroberfläche findet sich auf der Anwendungsebene in der Regel eine gute Kopplung zu mindestens einem CAD-System wieder. Mit dem CAD-System erzeugte Dokumente werden durch Geschäftsregeln verwaltet. Die dafür notwendige Geschäftlogik findet sich ebenfalls im PDM-System wieder und kann dort auch angepasst werden. Gespeichert

werden die so erzeugten Daten in einer PDM-Datenbasis, die Metadaten und Dokumente beinhaltet. Dazu setzen alle PDM-Systeme auf geeignete Datenbanksysteme auf, die häufig nur für den Einsatz des PDM-Systems installiert werden.

Durch derzeitige PDM-Einsätze werden beträchtliche Erleichterungen während des Konstruktionsprozesses erreicht, doch sind die Grenzen einzelner Bearbeitungsinseln immer noch erkennbar. Bereichsübergreifender Informationsfluss in Form eines virtuellen Produktes ist nach wie vor stark eingeschränkt. Das vorrangige Ziel von Produktdatenmanagement, die Verkürzung der Durchlaufzeit eines Produktes vom Auftragseingang zur Fertigung, kann durch Mittel wie Concurrent Engineering nur teilweise erreicht werden.

Das im Rahmen dieser Arbeit entwickelte Konzept stellt hingegen weniger die vertikale Durchgängigkeit in den Vordergrund, sondern unterstützt vielmehr den Gedanken des virtuellen Produktes und damit die Durchgängigkeit in horizontaler Richtung.

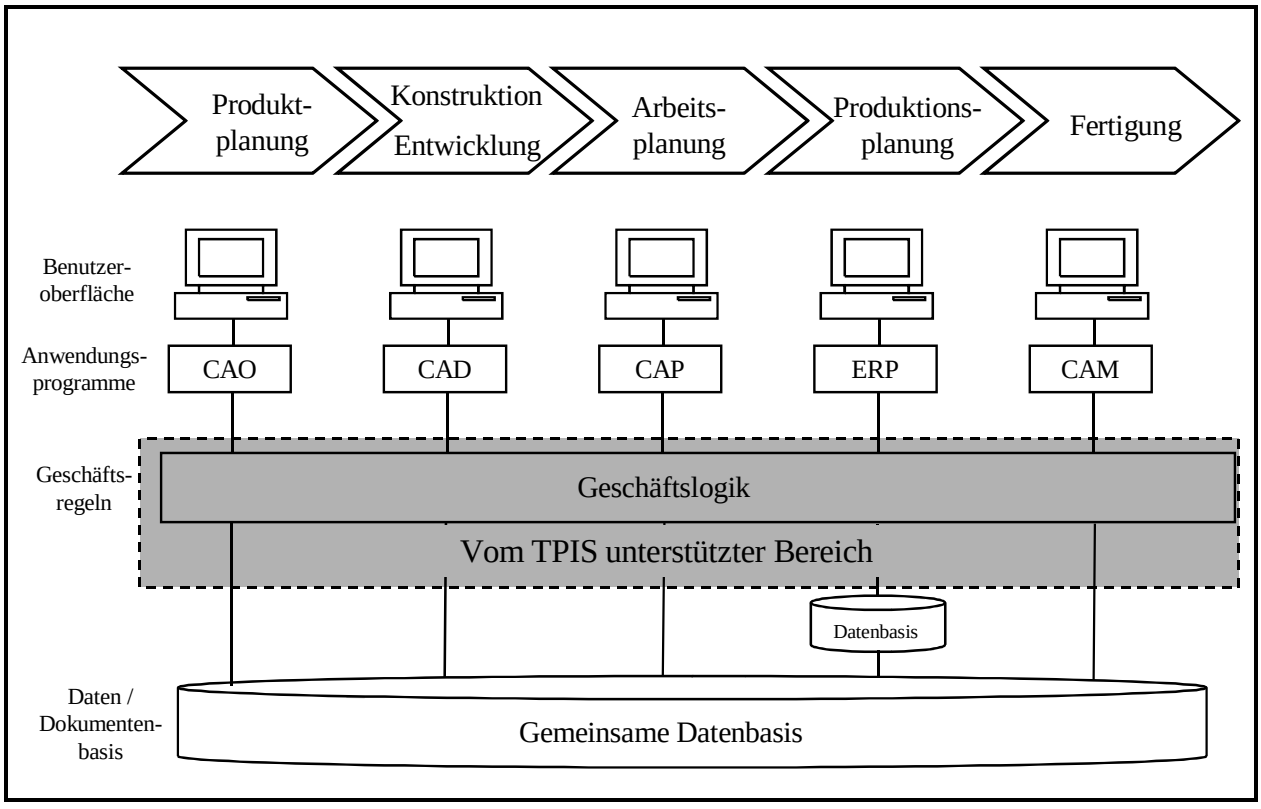


Abbildung 4-2: Unterstützung des Produktentstehungsprozesses durch ein TPIS

Es soll eine gemeinsame Datenbasis geschaffen werden, die gekoppelt mit angepassten Geschäftsregeln ein technisches Informationssystem ergibt (Abbildung 4-2). Dazu ist ein Kernbaustein notwendig, der durch eine ausgewählte Architektur Kopplungsmöglichkeiten zu

EDV-Systemen aller Bereiche bietet. Aufgabe des **TPIS** ist es dann, allen Bereichen angepasste Funktionen zu bieten, um eine sinnvolle, den Geschäftsregeln entsprechende Datenhaltung zu gewährleisten. So kann dieser als Grundlage für zukünfitge PDM-Systeme dienen.

Zunächst sind die Kriterien zu definieren, die eine horizontale Bearbeitung der Geschäftsprozesse in einem Unternehmen gewährleisten.

4.1 Kriterien an ein TPIS

Es soll zunächst ein Kriterienkatalog erstellt werden, der entsprechend aktueller Technologien einzuhalten ist, um einen wirklichen Vorteil bei der Verwendung eines allgemeingültigen **TPIS** zu erlangen. Es wird mit Anforderungen begonnen, die sich auf den allgemeinen Aufbau beziehen. Danach werden Anforderungen an die Datenbankseite, die Abdeckung typischer PDM-Funktionalitäten, der Kopplungsmöglichkeiten zu anderen Systemen und letztlich an die Verwendungsmöglichkeiten im Internet gestellt.

4.1.1 Allgemeine Anforderungen

Die allgemeinen Anforderungen an das **TPIS** sollen weniger die abzubildende Funktionalität berücksichtigen, es soll vielmehr sichergestellt werden, dass eine solche Software-Komponente den Ansprüchen gerecht wird, universell einsetzbar zu sein. Dazu gehören:

- Integrationsfähigkeit in neu zu entwickelnde Systeme
- Unabhängigkeit verwendeter Programmiersprachen
- Verteilung als Binärcode
- Modularer Aufbau
- Erweiterbarkeit

Integrationsfähigkeit in neu zu entwickelnde Systeme

Die Hersteller von PDM-Programmen entwickeln ähnlich den Herstellern von CAD-Systemen aus den 70er und 80er Jahren Systeme mit vergleichbarem Leistungsumfang. Dabei implementieren sie gleiche Funktionalitäten auf unterschiedliche Art und produzieren so untereinander inkompatible Systeme, deren Entwicklungskosten einen nicht unerheblichen Betrag ausmachen.

Das **TPIS** soll benötigte PDM-Funktionen abbilden und so Grundlage für neu zu entwickelnde Systeme sein. Die Aufgabe der Hersteller darf es nur noch sein, eine gewünschte Oberfläche und Schnittstellen zu ihrem CAx-System zu programmieren. Bei dieser Integration sollen keine Annahmen über Implementierungen innerhalb des **TPIS** gemacht werden. Dafür ist es notwendig, dass einmal gemachte Verträge über die Funktionen eingehalten werden und sich die Hersteller der PDM-Systeme auch bei Weiterentwicklung auf einmal bereitgestellte Methoden verlassen können. So kann die Verwendung eines **TPIS** dazu beitragen, Entwicklungszeit und Kosten zu senken.

Unabhängigkeit der verwendeten Programmiersprache

Soll das **TPIS** die geforderte Integrationsfähigkeit besitzen, sind Schnittstellen zur Kopplung an Erzeugersysteme offen zulegen. Um möglichst vielen Herstellern die Möglichkeit zu eröffnen geeignete Kopplungen zu implementieren, müssen die Schnittstellen in verschiedenen Programmiersprachen verwendet werden können. Als Vorraussetzung seien hier die Sprachen C++, Pascal, Visual Basic und Java genannt.

Verteilung als Binärcode

Hersteller dürfen keine Annahmen über Implementierungen innerhalb des **TPIS** machen. Das fertige Produkt wird dazu in Binärform ausgeliefert. Der programmierte Code wird nicht veröffentlicht, so können Probleme beim Compilieren und Linken eines gesamten Projektes vermieden werden.

Modularer Aufbau

Das **TPIS** muss einen modulartigen Aufbau besitzen, um entsprechend seines Einsatzes konfigurierbar zu sein. Die Module müssen Funktionsgruppen abbilden, die unabhängig voneinander einsetzbar sind.

Erweiterbarkeit

Die Erweiterbarkeit des **TPIS** sei als einer der wichtigsten Punkte der allgemeinen Anforderungen genannt. Nichts lässt die Akzeptanz eines Softwareproduktes so sehr sinken, wie

die Vorstellung mit diesem am Ende einer Entwicklung angekommen zu sein. Aus diesem Grund muss das **TPIS** nicht nur flexibel eingesetzt, sondern auch genauso flexibel erweitert werden können. Neue, vom Markt geforderte Funktionen müssen schnell integriert werden können, ohne dabei die vorhandene Funktionalität zu beeinflussen.

4.1.2 Anforderungen an die Datenbankfunktionen

Die Anforderungen an die Datenbankfunktionen lassen sich in zwei Bereiche aufteilen:

- Unabhängigkeit von einem Datenbanksystem
- Unabhängigkeit der erzeugten Daten vom Datenbanksystem

Unabhängigkeit von einem Datenbanksystem

Moderne PDM-Systeme erlauben die Verwendung mehrerer Datenbanksysteme. Bei windowsbasierten Programmen wird häufig Gebrauch von ODBC oder der BDE gemacht, die Schnittstellen zu Standard DB-Systemen bieten. Dennoch sind PDM-Programme zu finden, die unter Verwendung eigener Schnittstellen lediglich die Verwendung einer Datenbank erlauben. Damit ist eine Abhängigkeit von der Datenbank gegeben, die dazu führen kann, dass die Pflege eines PDM-Systems aus wirtschaftlichen Gründen nicht mehr vertretbar ist und die Produktion eingestellt werden muss, sobald ein Datenbankhersteller sein Produkt nicht mehr unterstützt. Für das **TPIS** stellt sich so die Forderung, unabhängig vom Datenbanksystem operieren zu können.

Unabhängigkeit der erzeugten Daten vom Datenbanksystem

Unterstützt ein PDM-System mehrere Datenbanksysteme, so kann es dennoch zu Inkompatibilitäten kommen, wenn ein bestehender Datenbestand von einem Datenbanksystem auf ein weiteres DB-System portiert werden soll. Der Grund ist in der Verwendung von Systemtabellen zu finden, aber auch im Einsatz von gespeicherten Prozeduren. Um derartige Schwierigkeiten zu vermeiden, soll der erzeugte Datenbestand unabhängig vom eingesetzten DB-System bleiben.

4.1.3 Anforderungen an typische PDM-Funktionalitäten

Die Einsatzmöglichkeiten des **TPIS** hängen in entscheidendem Maße davon ab, welche Funktionalitäten dieser bietet. Für das Konzept des **TPIS** sollen deshalb folgende typische Funktionen berücksichtigt werden:

- Administrationsfunktionen
- Managementfunktionen
- Arbeitsfunktionen

Administratorfunktionen

Aufgaben bezüglich der Administration und des Customizings haben sich seit Einführung sogenannter *out-of-the-box*[12] Systeme immer weiter überlagert. So gehört zu den eigentlichen Aufgaben eines Administrators die Verwaltung von Benutzern, die Vergabe von Benutzerrechten sowie das Archivieren und Sichern erzeugter Datenbestände. Hinzugekommen sind mittlerweile die Erzeugung und Verwaltung von Workflows, die Erstellung neuer Speicherklassen und die Programmierung kleiner Skripte, die entsprechend der Arbeitsweise eines Unternehmens immer wiederkehrende Aufgaben übernehmen.

Das **TPIS** muss alle administrativen Aufgaben ermöglichen, in wie weit diese jedoch später an den Kunden übertragen werden, hängt von der Implementierung des Herstellers ab. Dafür ist es notwendig, die entsprechenden Schnittstellen möglichst offen zu halten.

Managementfunktionen

Die Managementfunktionen eines PDM-System setzen sich in der Regel aus Dokumentenmanagement, Teilemanagement, Produktstrukturmanagement, Workflowmanagement und Projektmanagement zusammen. Diese, bereits in Kapitel 2.7 erläuterten Funktionen gelten als Mindestanforderung an die im **TPIS** implementierten Funktionen.

[12] out-of-the-box : bezeichnet den Gedanken von Softwareherstellern, auch komplexe Programme ohne Anpassungsaufwand direkt einsetzen zu können, oder den Aufwand der Anpassung stark zu verringern.

Arbeitsfunktionen

Mit dem Begriff Arbeitsfunktionen werden alle Funktionen umschrieben, die für die Bearbeitung von Dokumenten bzw. Teilen notwendig sind. Dazu gehören beispielsweise das Registrieren, Ein- und Auschecken und die Revisionierung von Objekten. Diese grundsätzlichen Arbeitsfunktionen müssen im **TPIS** abgebildet sein.

4.1.4 Kopplungsmöglichkeiten zu Fremdsystemen

Wie aus Abbildung 2-10 ersichtlich, ist die Kopplung zu Erzeugersystemen eine der wichtigsten Vorraussetzungen für den gewinnbringenden Einsatz eines PDM-Tools. Die Qualität der Kopplung hängt in entscheidendem Maße davon ab, wie gut die Programmierschnittstellen zu den beteiligten Systemen sind. Die Anforderungen an das **TPIS** bezüglich seiner Kopplungsmöglichkeiten gehen weit über die typische Schnittstellenprogrammierung hinaus (Abbildung 4-3).

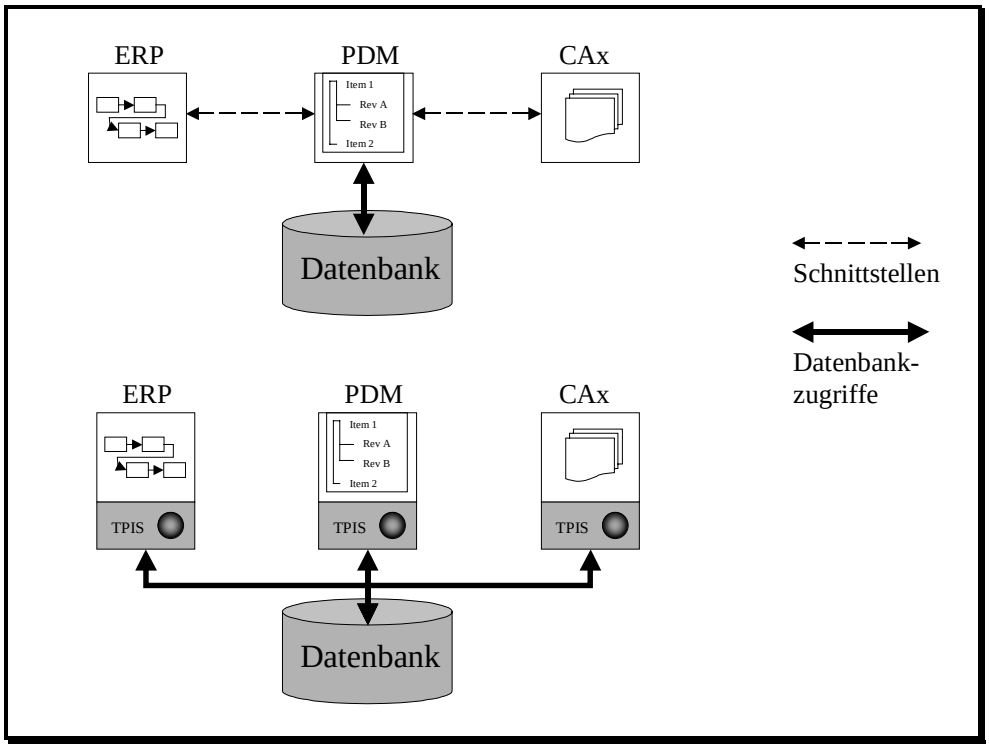


Abbildung 4-3: Verwendung konventioneller Schnittstellen (oben) und Integration über Einsatz eines gemeinsamen TPIS (unten)

Das **TPIS** als modular aufgebaute Softwarekomponente soll in mehrgestaltiger Form jedem Hersteller zur Verfügung stehen und als solche integriert werden. Als Beispiel sei hier die ERP-Kopplung genannt, die abgesehen von der CAD-Schnittstelle den höchsten Anspruch an die

Funktionalität stellt. Hersteller von ERP-Systemen sollen in die Lage versetzt werden, das **TPIS** so zu konfigurieren, dass eine optimales Zusammenwirken des ERP-Systems und des **TPIS** ermöglicht wird. Durch einen solchen Einsatz kann auf eine unternehmensweite PDM-Datenbank zugegriffen und es können relevante Daten für die Produktionsplanung und –steuerung gewonnen werden.

4.1.5 Anforderungen an den Einsatz im WAN

Der unternehmensweite Einsatz eines PDM-Tools stellt eine strukturierte Verwaltung von engineering-know-how sicher, solange sich die Mitarbeiter in einem physischen Subnetz befinden. Die fortschreitende Globalisierung verlangt für die Zukunft, dass dieses Wissen unternehmensübergreifend weltweit erreichbar zur Verfügung zu stellen. Es muss möglich sein, eine Produktdefinition zusammen mit Kunden, Zulieferern und Partnern gemeinsam zu erarbeiten und den Produktlebenszyklus für die beteiligten Unternehmen transparent zu machen. Des weiteren sei noch der Bedarf relevanter Daten für Aussendienstmitarbeitern genannt, die unabhängig ihres Standortes auf Produktdaten zurückgreifen müssen. Ein populärer Ansatz ist die Verwendung des Weitbereichsnetzwerks unter Verwendung des TCP/IP-Protokolls. Folgende Funktionalitäten müssen für den Einsatz im WAN abgedeckt sein:

- Sichten der Produktdaten
- Anlegen/Ändern der Produktdaten
- Unterstützung unternehmensübergreifender Workflows.

Sichten der Produktdaten

Die projektbezogene Kooperation mehrerer Unternehmen erfordert die Verteilung aktueller Informationen bezüglich des Projektstands und der Produktdaten. So kann es beispielsweise für einen Zulieferer notwendig sein, geometrische Daten und den Freigabestatus einer Komponente zu erfahren, für die ein Anbauteil zu konstruieren ist. Ausgewählte Daten müssen daher von einer PDM-Installation veröffentlicht und Kooperationspartnern verfügbar gemacht werden.

Anlegen/Ändern der Produktdaten

Bei projektbezogenen Kooperationen muss die Übertragung ausgewählter Daten eines Zulieferers in die PDM-Installation eines Unternehmens gewährleistet werden. Soll beispielsweise in einem Unternehmen eine DMU-Untersuchung durchgeführt werden, so kann es nötig sein von Zulieferern zur Vervollständigung einer Baugruppe die Hüllgeometrie eines Bauteils einzubinden. Dem Gedanken des virtuellen Produktes folgend, ist diese Hüllgeometrie vom Zulieferer über das WAN im PDM-System des Unternehmens einzubinden. Ändern sich im Verlauf des Entstehungsprozesses relevante Daten, so müssen auch diese in die PDM-Installation des zu beliefernden Unternehmens eingebunden werden.

Unterstützung unternehmensübergreifender Workflows

Der Produktentstehungsprozess im Rahmen eines Jointventures verlangt die Erstellung eines virtuellen Modells aus Daten aller beteiligten Firmen. Die Konsistenzprüfung dieses Modells kann aufgrund seiner beliebigen Komplexität jedoch nicht rechnergestützt ablaufen, weil dafür die Zusammenarbeit der Mitarbeiter aller beteiligter Unternehmen notwendig ist. Die zu durchlaufenden Prozesse zur Konsistenzprüfung und der damit verbundenen Freigabe der Produktstruktur erfordern einen unternehmensübergreifenden Workflow. Nur so kann ein auf den Produktenstehungsprozess beschleunigendes Concurrent Engineering über Unternehmensgrenzen hinaus verwirklicht werden.

4.2 Grundkonzept des TPIS

In Kapitel 4.1 wurden grobe Anforderungen an die Gestalt, Funktionen und Einsatzmöglichkeiten des **TPIS** gestellt. Im Verlauf dieses Kapitels soll nun ein grundsätzliches Konzept erstellt werden, das diesen Anforderungen gerecht wird.

4.2.1 Aufbau des TPIS

Die Forderung nach Vielgestaltigkeit und modularem Aufbau des **TPIS** wurde bereits mehrfach angesprochen und soll aufgrund seiner Bedeutung als erstes in das Konzept einfließen. Es ergibt sich durch Gliederung bezüglich der Funktionalität eine physische Aufteilung des

TPIS, die es möglich macht, einzelne Module entsprechend vorgegebener Beziehungen einzusetzen.

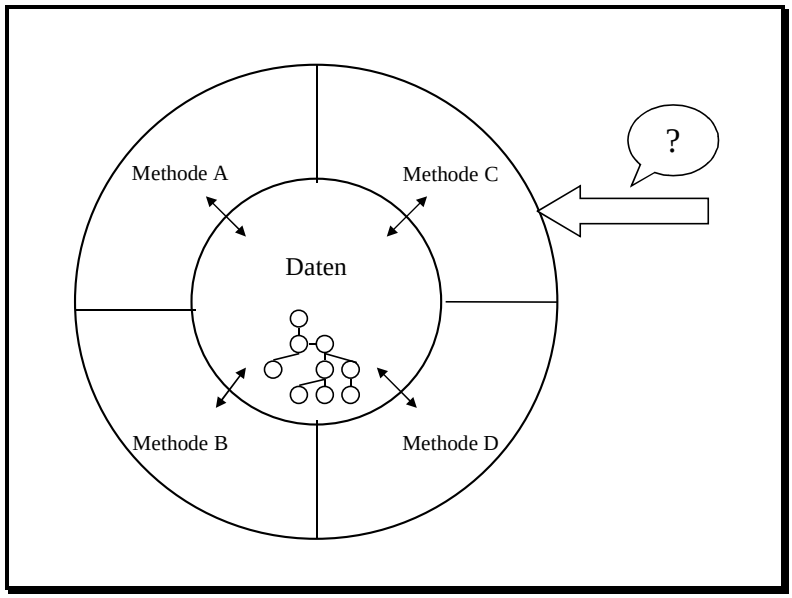


Abbildung 4-4: Grundmodell des TPIS mit Daten und Methoden zur Manipulierung der Daten

Orientiert man sich an der Grundvorstellung des **TPIS** zur Abbildung einer Datenstruktur und Bereitstellung von Methoden zur Manipulierung dieser Daten, so zerfällt dieses in zwei große Bereiche:

- Datenbankkern
- PDM-Kern

Abbildung 4-4 stellt ein Grundmodell des **TPIS** als Donut-View dar. Diese Art der Ansicht hat sich bereits bei der Darstellung von Klassen im Umfeld der objektorientierten Programmierung bewährt und erfüllt hier einen ähnlichen Zweck. Die Daten liegen im Inneren des **TPIS** und werden von umliegenden Methoden geschützt, d.h. eine Manipulation der Daten ist nur über fest definierte Methoden möglich. Eine Anfrage von Außen, dargestellt als Blockpfeil, kann nicht direkt an diese gestellt werden. Es können nur Methoden beauftragt werden eine Abfrage auszuführen und diese geben eine entsprechende Ergebnismenge zurück. Diese Anschauung soll auf das oben angesprochene **TPIS** angewendet werden. Dazu soll zunächst definiert werden, welche Aufgaben und Elemente der geschützte innere Bereich zu übernehmen hat.

Der innere Bereich des Informationssystems ist nach Abbildung 4-4 für die Bereitstellung und Speicherung der Daten verantwortlich. Die Daten eines PDM-Systems sind jedoch von derart komplexer Natur, dass üblicherweise ein Datenbanksystem dazu eingesetzt wird. Die

Speicherung der Daten findet also nicht im **TPIS**, sondern in einer ausgelagerten Datenbank statt. Diese Daten zu schützen und nur geregelte Zugriffe auf die Datenbasis zuzulassen, muss dann die Aufgabe des inneren Kerns sein, der aufgrund dessen auch als Datenbankkern bezeichnet wird.

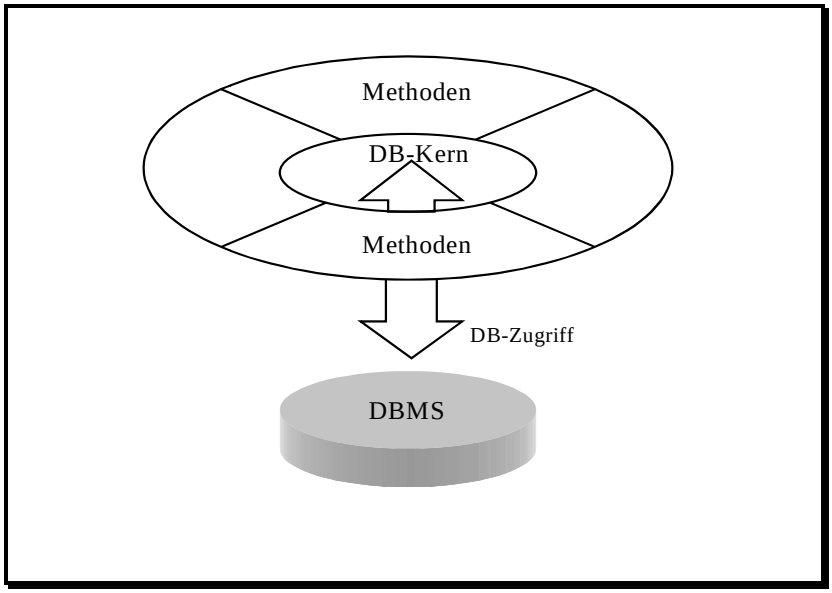


Abbildung 4-5: TPIS mit Datenbankkern und Zugriff auf ein Datenbanksystem

Jede Art der Transaktion muss zwangsweise über diesen Datenbankkern ausgeführt werden, um nur klar definierte Zustandsänderungen des Datenbestandes zuzulassen und Sicherheit gegen unbefugten Zugriff zu gewährleisten. Methoden, die diesen Kern benutzen, dürfen keine Annahmen über das Verhalten der Datenbank machen (Abbildung 4-5). Im Idealfall darf hier sogar keine Annahme über den Typ des Datenbanksystems gemacht werden.

Der Einsatz eines PDM-Systems besteht jedoch nicht nur allein aus dem Zugriff und die Gewährleistung der Sicherheit von Daten, sondern auch aus den bereits in Kapitel 4.1.3 angesprochenen PDM-Funktionen. Diese Funktionen abzubilden und entsprechend definierter Geschäftsprozesse Anfragen und Modifizierungen an der Datenstruktur zu übernehmen ist Aufgabe der äußeren Schicht. Diese ist dafür verantwortlich, eine Schnittstelle zwischen Fremdsystemen und dem Datenbankkern herzustellen.

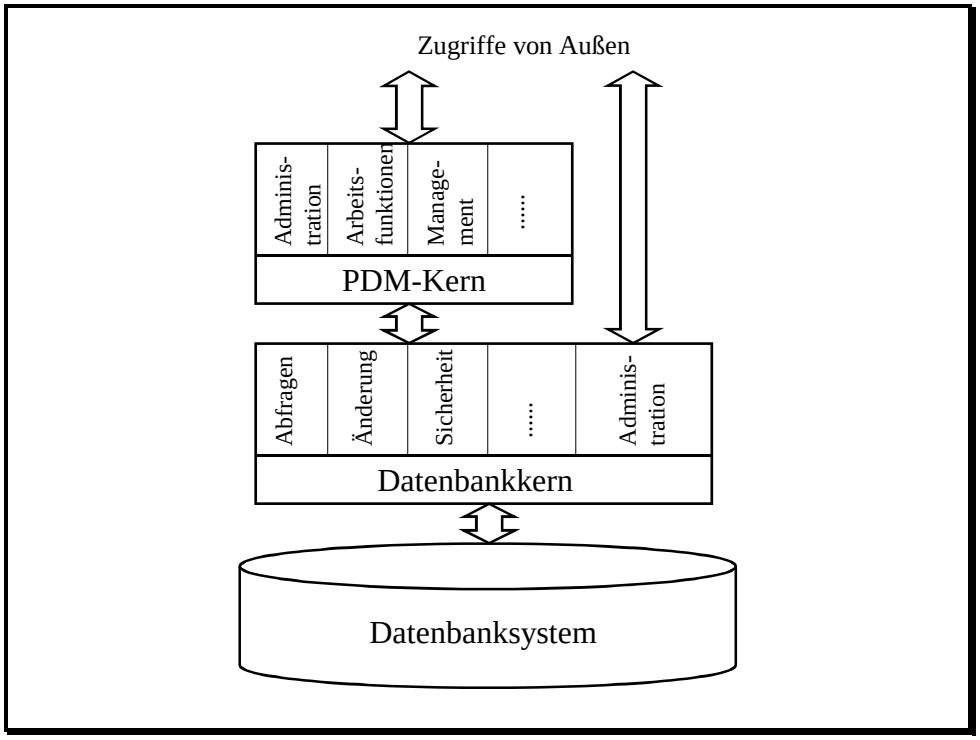


Abbildung 4-6: Zugriffe auf das Datenbanksystem von außen über PDM- und Datenbankkern

Abbildung 4-6 verdeutlicht das Zusammenspiel der beiden Schichten Datenbankkern und PDM-Kern. Ein sich auf die PDM-Daten auswirkender Geschäftsprozeß wird allgemein als Zugriff von Außen benannt. Findet ein solcher Zugriff statt, so geschieht dies nur über die obere PDM-Schicht. Hier wird der Zugriff ausgewertet und notwendige Funktionen gestartet. Alle an diesem Prozess beteiligten Datenobjekte werden zusammengestellt und Abfragen über diese an den Datenbankkern gestellt. Dieser leitet die Abfragen an das Datenbanksystem weiter und erhält eine Ergebnismenge als Antwort. Die Ergebnismenge wird an den PDM-Kern übergeben, der wiederum für die Aufbereitung der Daten und Weitergabe an ein beteiligtes System verantwortlich ist. Änderungen am Datenbestand erfordern den gleichen Ablauf. Das TPIS wird also in einen Datenbankkern, der benötigt wird um definierte Änderungen des Datenbestands zuzulassen, und in einen PDM-Kern zur Abbildung definierter PDM-Funktionen aufgeteilt.

Außer den Zugriffen über den PDM-Kern ist in Abbildung 4-6 auch ein direkter Zugriff auf den Datenbankkern dargestellt. Diese Zugriffe sollen eine Ausnahme bilden um Funktionen zur Administration der Datenbank aufrufen. Als Beispiel seien hier Manipulationen am Datenmodell genannt, um neue Speicherklassen anzulegen. Die Bereitstellung dieser Funktionen erfordert wiederum eine Unterteilung des Datenbankkerns in einen Arbeitsbereich der für Abfragen und Änderungen zuständig ist, und einen administrativen Bereich, der zur Datenmodellierung nötig ist.

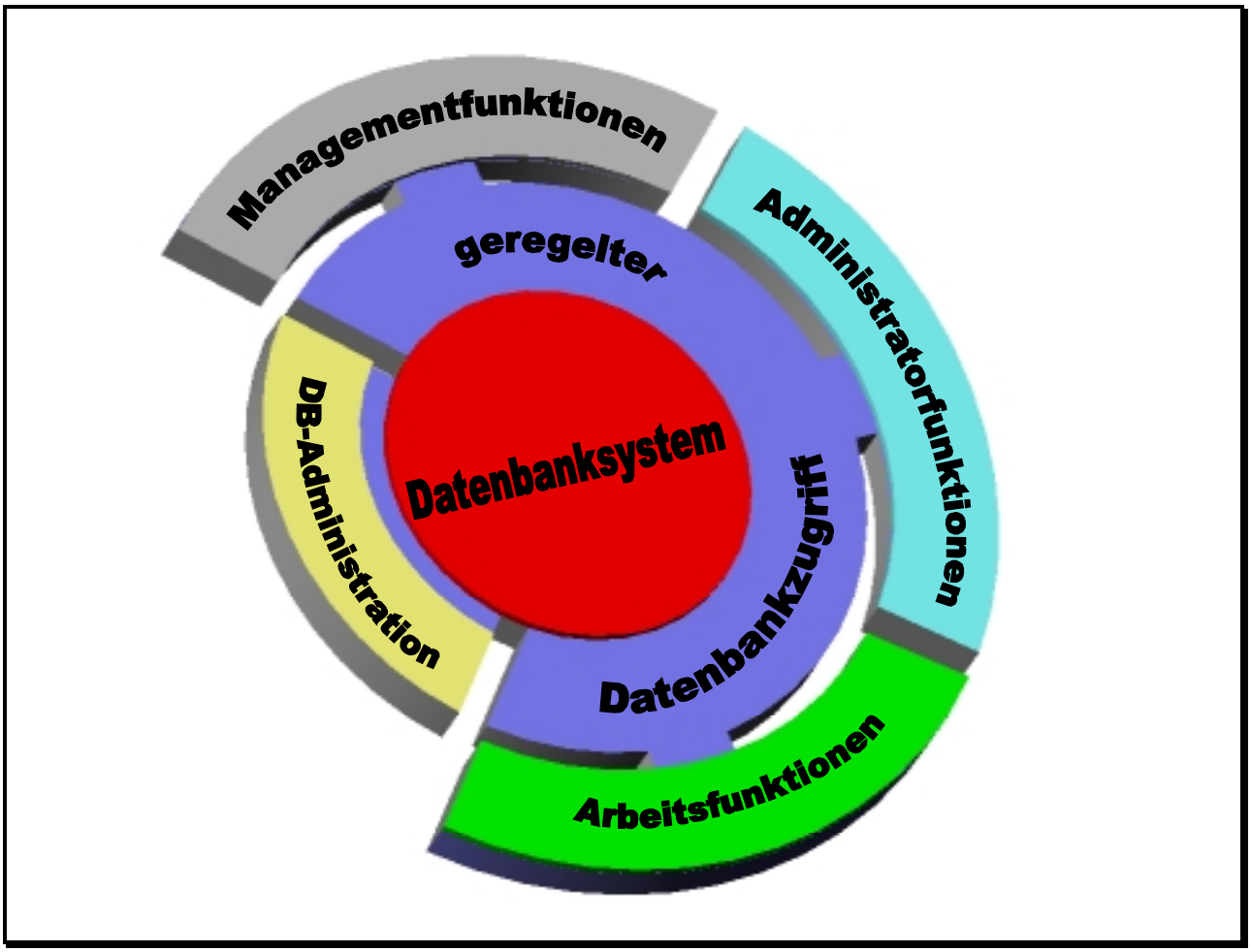


Abbildung 4-7: Das komplette TPIS mit Gliederung in Funktionseinheiten und einem gekapselten Datenbanksystem.

Eine Unterteilung des Datenbankkerns bezüglich der Administration und der Zugriffsregelung, sowie des PDM-Kerns bezüglich der Management-, Arbeits- und Administratorfunktionen ergibt für das **TPIS** die in Abbildung 4-7 modellierte Gestalt. Festzuhalten ist, dass diese Unterteilung auf funktioneller Ebene stattfindet um einen Einsatz des Kerns durch Weglassen einzelner Funktionsblöcke auf entsprechende Anforderungen anzupassen.

4.2.2 Integration des TPIS in eine 3-Tier Applikation

Bisher wurde das **TPIS** als eigenständig betrachtet, um nötige Funktionen und den modularen Aufbau darzustellen. Es wurde noch nicht diskutiert, wie eine derartige Softwarekomponente mit den beschriebenen Eigenschaften in eine Umgebung integriert werden kann, die zur Verfügung

gestellte Dienste nutzt. Zu diesem Zweck soll eine Softwarearchitektur herangezogen werden, die das oben beschriebene **TPIS** ausmacht.

Betrachtet man die Architektur heutiger Software, so stellt man die sehr weite Verbreitung der Client/Server-Lösungen fest. Ein Server dient dabei im Regelfall als File- oder Datenbankserver, nur selten als Applikationsserver. Sämtliche Geschäftslogik befindet sich auf dem Client, der auch als *Fat-Client* bezeichnet wird. Diese Form der Architektur wird in Zukunft immer mehr durch sogenannte 3-Schichtige-Archtikturen ersetzt. 3-Schicht-Architekturen *(engl. 3-Tier)* besitzen eine weitere Schicht zwischen Client und Server (Abbildung 4-8). Dort ist die eigentliche Geschäftslogik platziert, die Zugriffe auf einen Datenbankserver oder Fileserver ausführt. Der angekoppelte Client besitzt wie zu früheren Terminalzeiten keine Logik mehr und wird nur zur Anzeige der Daten eingesetzt. Man spricht von sogenannten *Thin-Clients.*

Ein Grund für den Einsatz dieser 3-Tier-Technologie liegt in der Wiederverwendbarkeit der einmal erstellten Geschäftslogik für mehrere Programme. Wurden früher Geschäftsprozesse in große, monolithische und schwer zu wartende Applikationen implementiert, können diese jetzt von eigentlichen Anwendungen getrennt und anderen Programmen zur Verfügung gestellt werden. Die Trennung ermöglicht zudem auch eine bessere Wartbarkeit der eingesetzten Geschäftslogik, die bei Bedarf nur an einer zentralen Stelle aktualisiert werden muss. Der Einsatz von Thin-Clients ist ein weiterer Grund für den Gebrauch der 3-Schicht-Technologie. Im Zeitalter des Internets müssen immer mehr Mitarbeiter von beliebigen Standorten auf Produktdaten zugreifen können. Dabei wird häufig auf Laptops und zukünftig immer mehr auf PDAs[13] zurückgegriffen, die allenfalls die Funktion von Datensichtgeräten übernehmen können.

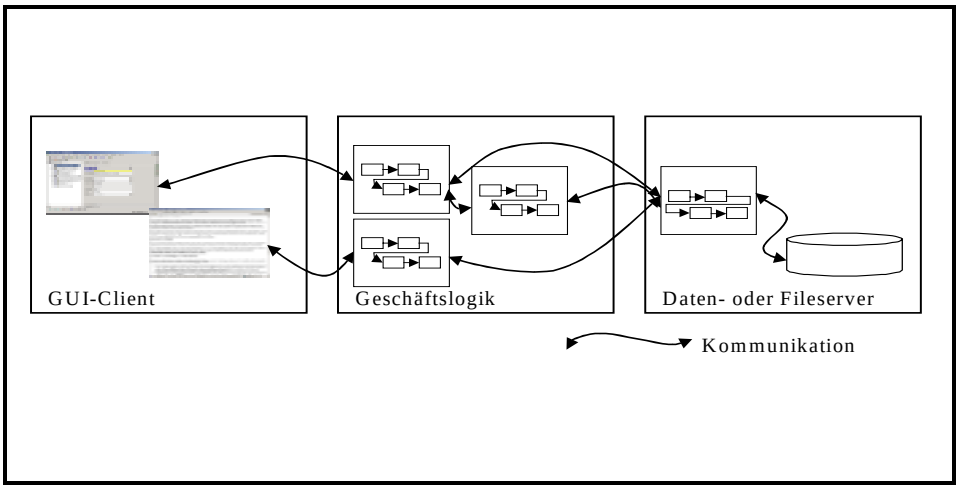


Abbildung 4-8: 3-Schicht-Architektur

[13] PDA Personal Digital Assistent (engl.) : Taschencomputer

Aus oben genannten Gründen soll das TPIS ebenfalls in eine 3-Schicht-Architektur integriert werden. Zum besseren Verständnis wird im Folgenden auf die Vorgehensweise der Integration kurz eingegangen.

Im Bereich der Geschäftslogik werden zunehmend Komponentenmodelle eingesetzt. Gängige Lösungen sind derzeit *CORBA* und *COM/COM+*. Aus diesem Grund wird für die Konzeption des hier entwickelten **TPIS** auf das Componenten Object Model (COM) von Microsoft® zurückgegriffen. Grundsätzlich ist zu sagen, dass CORBA für diese Aufgabe ebenfalls geeignet ist, doch aufgrund besserer Verfügbarkeit wird COM der Vorzug gegeben.

Die getroffene Wahl der COM-Technologie für den Bereich der Geschäftslogik als COM-Komponente soll durch eine Gegenüberstellung des Produktentwicklungsprozesses mit der Windows DNA nun noch verdeutlicht werden. Abbildung 4-9 stellt eine solche Gegenüberstellung dar. Es ist zu erkennen, dass die Windows DNA in der Präsentationsschicht unter anderem Win32-Anwendungen als auch Internetbrowser vorsieht. Im Produktentstehungsprozess ist diese Schicht im Bereich der Benutzeroberfläche und der CAx- bzw. ERP-Anwendungen zu finden. Geht man davon aus, dass die CAx- bzw. ERP-Anwendungen als WIN32-Anwendungen eingesetzt werden, so sollen die Internetbrowser durch dynamisch generierte Webseiten Informationen über Produktdaten liefern.

Die Geschäftsschicht der Windows DNA findet sich im Produktentstehungsprozess auf dem Gebiet der Geschäftslogik wieder. An dieser Stelle ist der Einsatz des **TPIS** vorgesehen, das, wie bereits erwähnt, als COM/COM+-Komponente ausgeführt wird. Die darunter liegende Datenschicht ermöglicht bei geeigneter Treiberwahl nach Vorgabe der Windows DNA den Einsatz beliebiger Datenbanken. Das in der Produktentstehung dazu äquivalente Datenbankmanagementsystem kann also beliebig gewählt werden. Voraussetzung ist die Verwendung der OLEDB- oder ADO-Datenbanktreiber. Ältere DBMS können, sofern vorhanden, immer noch über die ODBC angesprochen werden.

Die Kommunikation zwischen jeweils zwei Schichten soll nach Anforderungskatalog sowohl über das LAN als auch über das WAN möglich sein. Die von Microsoft dafür vorgesehene Technologie *Distributed COM* findet aus diesem Grund auch hier Anwendung.

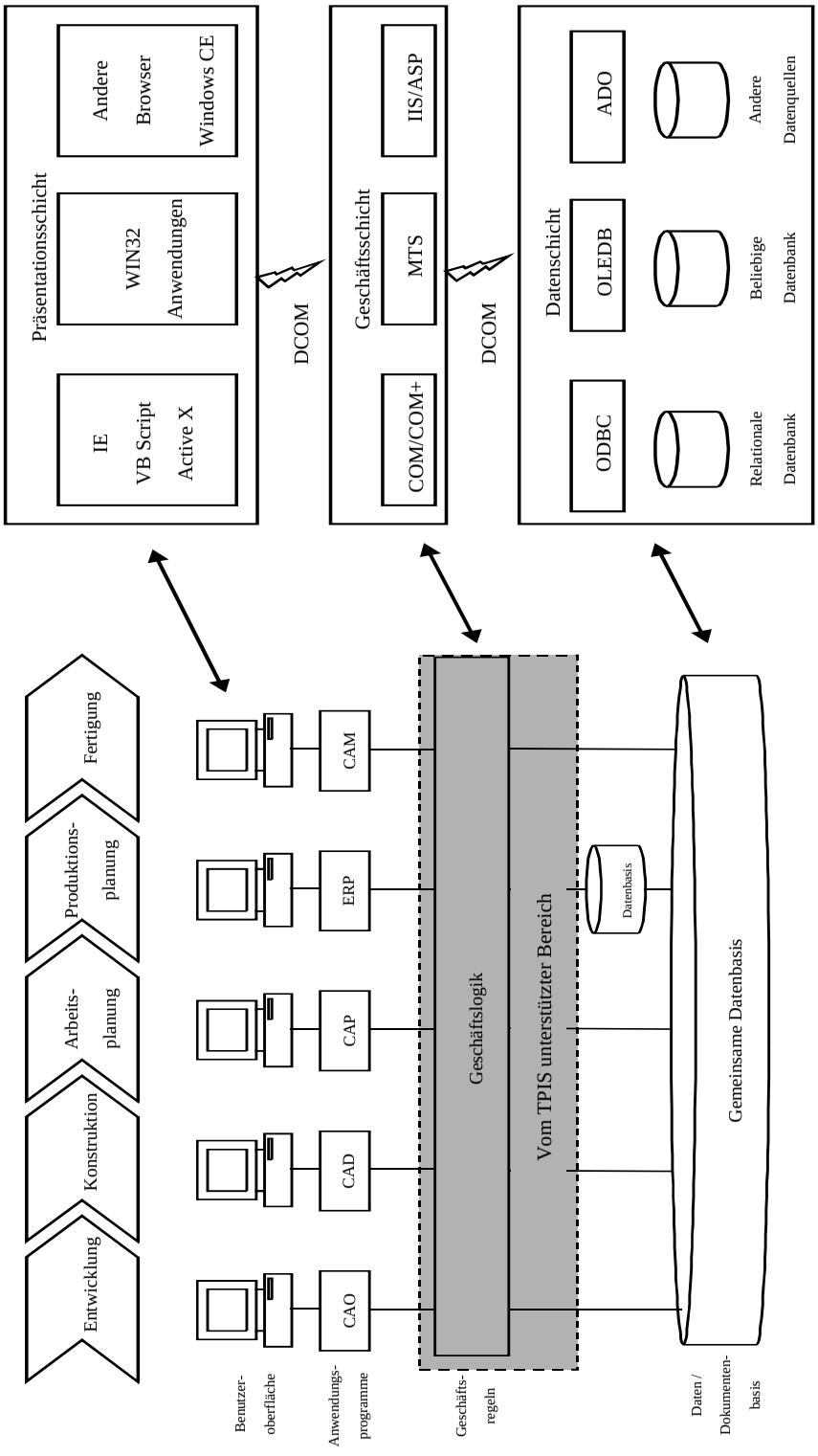


Abbildung 4-9: Gengenüberstellung der Produktentstehung (links) und der Windows DNA (rechts)

4.3 Konzeption des DB-Kerns

Der hier gewählte Ansatz für einen Datenbankkern soll im folgenden bezüglich seines grundsätzlichen Konzeptes erläutert werden. Zunächst soll ein Datenmodell hergeleitet werden, auf dem die danach beschriebenen Schnittstellenmethoden aufsetzen. Außerdem sollen Möglichkeiten zur Verbindung des Datenbankkerns mit der Datenbank erläutert werden.

4.3.1 Herleitung eines Datenmodells

Um die Welt in ihrer Komplexität begreifbar und damit nutzbar zu machen, bedient sich der Mensch der Methodik der Modellbildung. Auch auf dem Gebiet des Produktdatenmanagements ist eine derartige Modellbildung notwendig, um das in den vorhergehenden Kapiteln beschriebene virtuelle Produkt abbilden zu können. Einige PDM-Systeme bedienen sich dazu eines sehr starren Datenmodells, das hinsichtlich seiner Anpassungsfähigkeit nur geringen Spielraum zulässt. Zudem lässt sich eine Überführung der Datenbasis eines PDM-Systems auf das System eines anderen Herstellers nur sehr schwer durchführen, da man semantische Äquivalenzen zwischen den ineinander zu überführenden Modellen erkennen muss. Äquivalente Informationen sind in unterschiedlichen Modellen unterschiedlich präsent und müssen durch traditionelle Konvertierungsmethoden abgeglichen werden. Die Informationen selbst ändern sich also nicht, sie nehmen nur eine andere Gestalt an. Nur in den seltensten Fällen ist jedoch eine Überführung ohne Verluste möglich. Zentrale Aufgabe des Datenbankkerns ist es, dieses Manko zu umgehen und eine flexible Datenstruktur zu unterstützen.

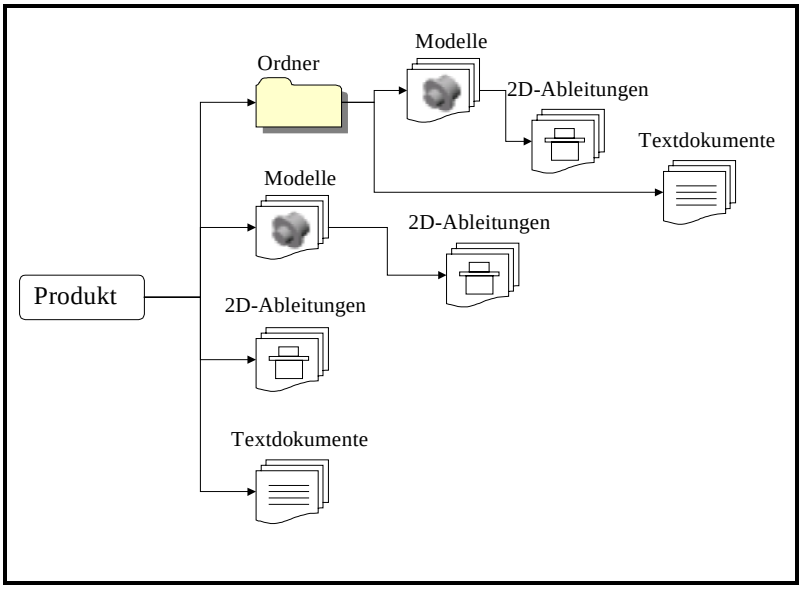


Abbildung 4-10: Beispiel einer Produktstrukur

Das in Abbildung 4-10 dargestellte Produktmodell soll als Ausgangspunkt für die Herleitung eines Datenmodells dienen. Dazu soll ein Entity-Relationship-Diagramm nach Chen erstellt werden.

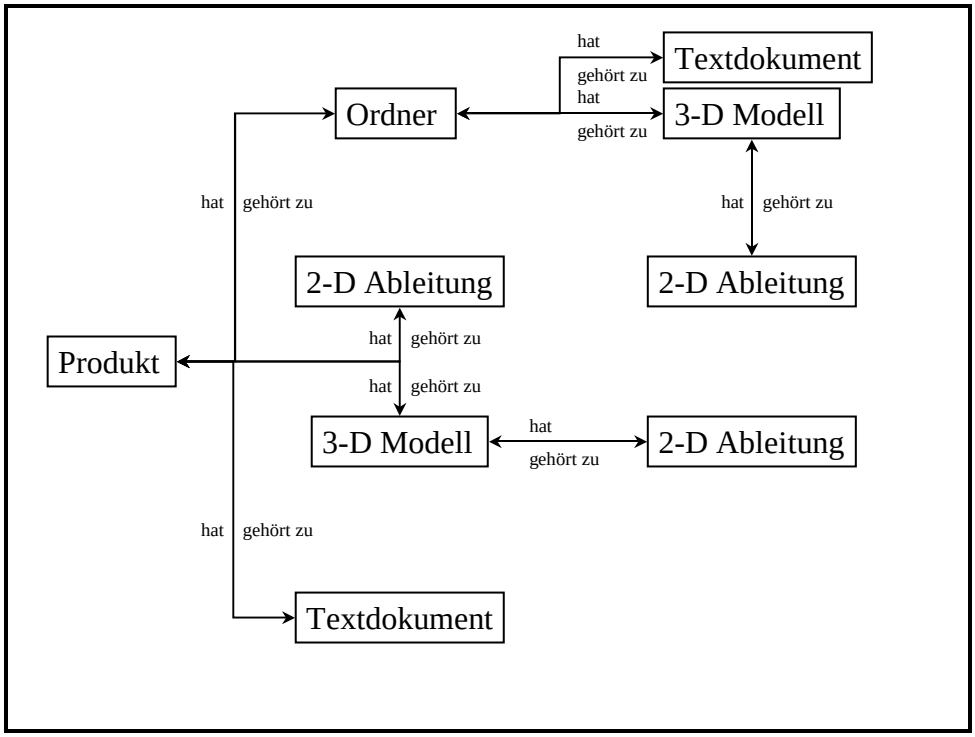


Abbildung 4-11: Entity-Relationship-Diagramm für eine Produktstruktur

In Abbildung 4-11 sind Referenzierungen der Objekte einer Produktstruktur untereinander dargestellt. Ein Produkt *hat* einen Ordner, ein Modell, eine 2D-Ableitung und ein Textdokument. Das 3D-Modell, das Textdokument, der Ordner und die 2D-Ableitung *gehören zu* einem Produkt. Die Objekte werden als Entitäten bezeichnet, ihre Beziehungen zueinander als Relationships. Aufgabe ist es nun, die Objekte und ihre Relationen zueinander in Tabellen einer relationalen Datenbank abzulegen. Von der Verwendung der Systemtabellen einiger DBMS soll hier abgesehen werden, damit eine Portierung der Datenbankinhalte ohne Verlust semantischer Beziehungen möglich ist. Ausgehend von den oben genannten Objekten und deren n:m-Beziehungen zueinander müssen Datenbanktabellen erstellt werden, die Informationen der Objekte und deren Relationen zueinander aufnehmen können.

Abbildung 4-12 zeigt die Zusammenhänge zwischen den objektrepräsentierenden Datenbanktabellen *Produkt, 3D-Modell, 2D-Ableitung*, im folgenden als Klassen bezeichnet, und einer Tabelle *Link* zur Speicherung der Beziehungen zueinander. Die Linktabelle ist demnach ein

fundamentales Werkzeug zur Verknüpfung beliebiger Objekte verschiedener Klassen zu einer Produktstruktur.

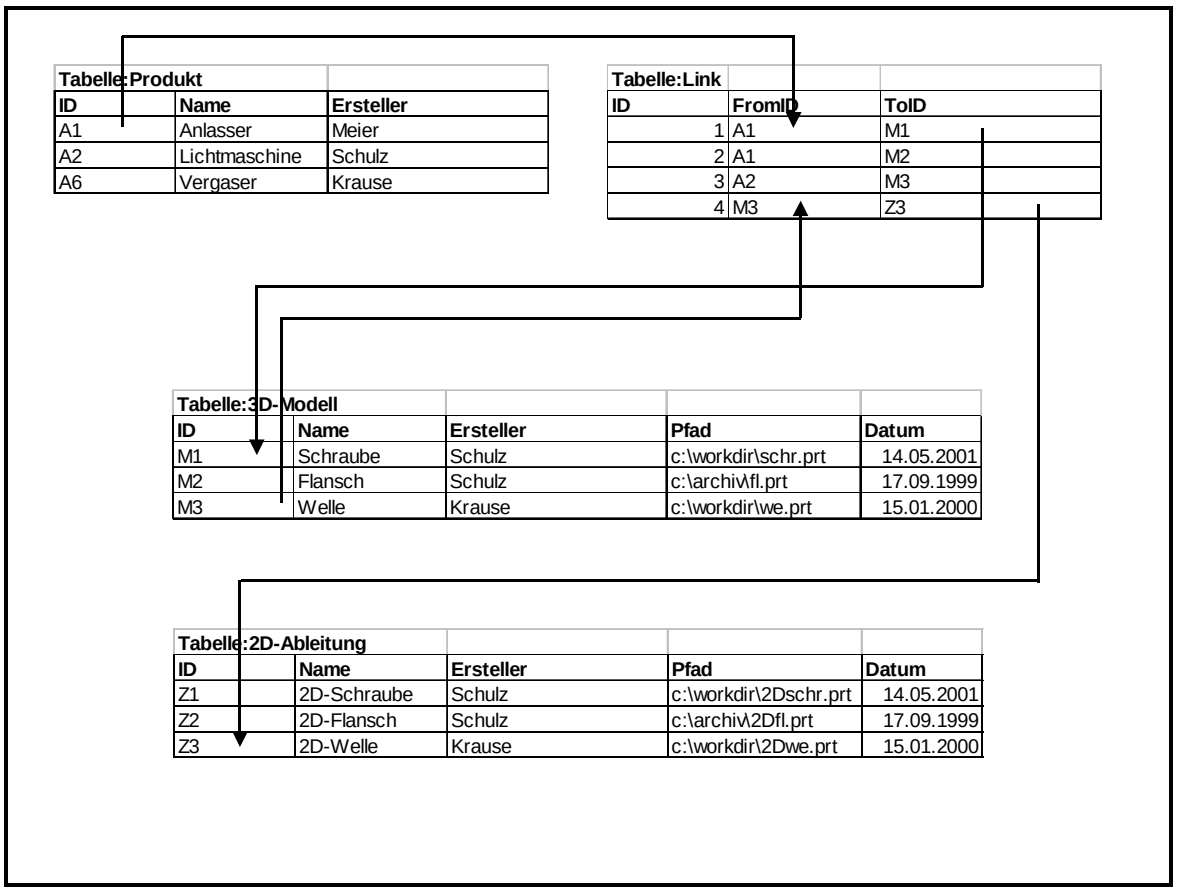


Tabelle:Produkt

ID	Name	Ersteller
A1	Anlasser	Meier
A2	Lichtmaschine	Schulz
A6	Vergaser	Krause

Tabelle:Link

ID	FromID	ToID
1	A1	M1
2	A1	M2
3	A2	M3
4	M3	Z3

Tabelle:3D-Modell

ID	Name	Ersteller	Pfad	Datum
M1	Schraube	Schulz	c:\workdir\schr.prt	14.05.2001
M2	Flansch	Schulz	c:\archiv\fl.prt	17.09.1999
M3	Welle	Krause	c:\workdir\we.prt	15.01.2000

Tabelle:2D-Ableitung

ID	Name	Ersteller	Pfad	Datum
Z1	2D-Schraube	Schulz	c:\workdir\2Dschr.prt	14.05.2001
Z2	2D-Flansch	Schulz	c:\archiv\2Dfl.prt	17.09.1999
Z3	2D-Welle	Krause	c:\workdir\2Dwe.prt	15.01.2000

Abbildung 4-12: Tabellen für Objekte, Relationen und deren Beziehungen zueinander

Ein allgemeines Datenmodell benötigt zur vollständigen Abbildung einer Produktstruktur folgende Klassen (Abbildung 4-13) :

- Eine Superclass, die als Basisklasse dient und von der alle weiteren Klassen abgeleitet werden müssen. Sie beinhaltet alle notwendigen Attribute, die zur Integration aller Kindklassen in das Datenmodell notwendig sind. Am Beispiel der oben genannten Klassen 3D-Modell, Produkt und 2D-Ableitung sind das die Attribute *ID* und *Ersteller*.
- Eine Datamodelclass, die wiedergibt welche Klassen voneinander abgeleitet wurden.
- Eine Linkclass, die wie oben bereits angedeutet, die Relationships der Entitäten zueinander wiedergibt.

- Eine Structurclass, die erlaubte Beziehungen der Klassen untereinander repräsentiert. So ist es möglich, dass eine 2D-Ableitung zu einem 3D-Modell gehört, jedoch niemals umgekehrt. Dieser Sachverhalt wird durch die Existenz der Structurclass sichergestellt.
- Eine Typclass, die Informationen bezüglich der Klassen und eines internen Nummernkreises aufnimmt. Wie später noch diskutiert wird, erfolgt die Erkennung von Objekten anhand einer ID, die in Abhängigkeit ihrer Klassenzugehörigkeit mit einem entsprechenden Präfix versehen wird. Der zur Klasse zugehörige Präfix wird in dieser Tabelle eingetragen.

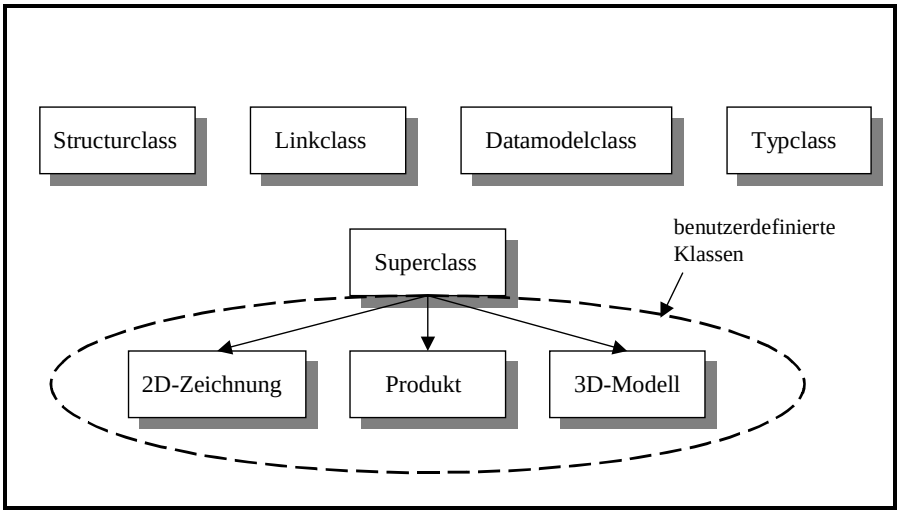


Abbildung 4-13: Basisklassen des Datenmodells (oben) mit Erweiterungen (eingekreist)

Abbildung 4-13 deutet das allgemeine Datenmodell an. Die erweiterten Klassen sind von der Basisklasse Superclass abgeleitet und besitzen zusätzlich zu den benutzerdefinierten Attributen alle Attribute dieser Klasse.

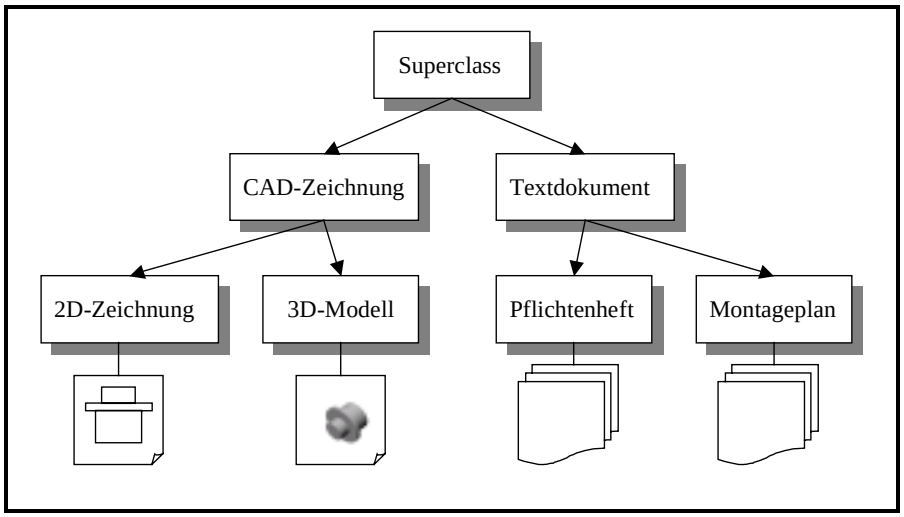


Abbildung 4-14: Datenmodell mit Klassifizierung nach Dokumenten

Die zur strukturierten Informationsverwaltung notwendige Klassifizierung ist bisher noch nicht berücksichtigt worden. Diese kann realisiert werden, indem unterhalb der Superclass weitere Klassen zur Strukturierung eingesetzt werden. Für eine Dokumentenklassifizierung werden dazu wieder Elternklassen erstellt, deren Attribute allen Kindklassen gemein sind. Zu berücksichtigen ist, dass Elternklassen niemals einen Verweis auf ein Dokument enthalten, sondern nur die jeweils untersten Kindklassen. Die in Abbildung 4-14 vorgenommene Klassifizierung beschränkt sich auf die Klassen CAD-Zeichnung und Textdokument. Die Vererbung zu den Kindklassen wird in der Tabelle Datamodelclass gespeichert. Soll beispielsweise eine Suche über alle Textdokumente stattfinden, wird die Datamodelclass nach möglichen Kindklassen befragt. Sind keine weiteren Auswahlkriterien angegeben, werden alle in den Klassen *Pflichtenheft* und *Montageplan* existierenden Entitäten als Ergebnismenge ausgegeben.

Eine Teileklassifizierung hingegen wird durch Erstellung einer Hierarchie parallel zur Dokumentenhierarchie erreicht. Auch hier sind beliebige Strukturen möglich, allerdings sollte eine Beschränkung auf das STEP ähnliche Format *Item->ItemRevision* ausreichen. Ein Item stellt dabei ein Teil bzw. eine Baugruppe dar, die dazugehörige ItemRevision alle zugehörigen Versionen dieses Bauteils.

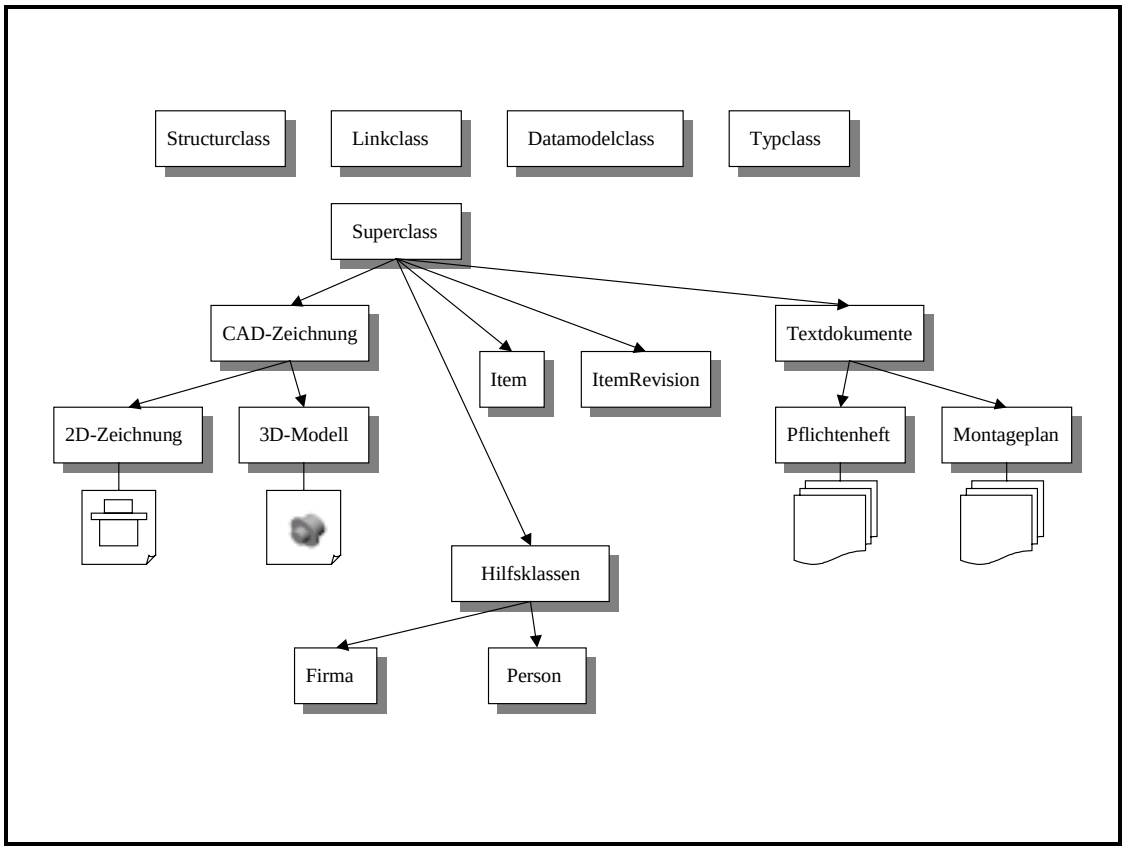


Abbildung 4-15: Ausschnitt aus einem Datenmodell mit Dokumenten- und Teileklassifizierung

Abbildung 4-15 stellt die Klassen Item und ItemRevision im Datenmodell dar. Eine Klassifizierung nach Teilemerkmalen wird durch Definition geeigneter Attribute erreicht. So entfällt eine beliebig komplexe Klassenhierarchie zur Abbildung sämtlicher Teile und Baugruppen. Im Gegensatz zu Datenmodellen moderner PDM-Systeme fehlen Klassen zur Verwaltung der Anwender. Diese Aufgabe soll beim Einsatz des **TPIS** vom Betriebssystem übernommen werden. Des weiteren werden später noch im Zuge des unternehmensübergreifenden Workflows zu implementierende Klassen diskutiert. Die im Anforderungskatalog definierte Flexibilität und Portierbarkeit ist durch das oben beschriebene Datenmodell jedoch erreicht.

4.3.2 Datenbankschnittstelle

Eine der Aufgaben des Datenbankkerns ist die Verwaltung und Bereitstellung sämtlicher in der Datenbank befindlicher Daten. Dazu ist eine geeignete Datenbankschnittstelle notwendig. Microsoft® stellt Entwicklern den Datenbankzugriff über die Datenbanktreiber ODBC und

OLEDB zur Verfügung. OLEDB entspricht bezüglich seines Aufbaus der COM-Technologie und wird durch ADO ergänzt. ADO bietet Wrapper-Klassen auf OLEDB-Komponenten und ist im Gegensatz zu diesen in verschiedenen Programmiersprachen einsetzbar. Parallel zur Konzeption ist das Testen von Datenbankfunktionen im relativ einfach zu programmierenden BASIC-Code möglich.

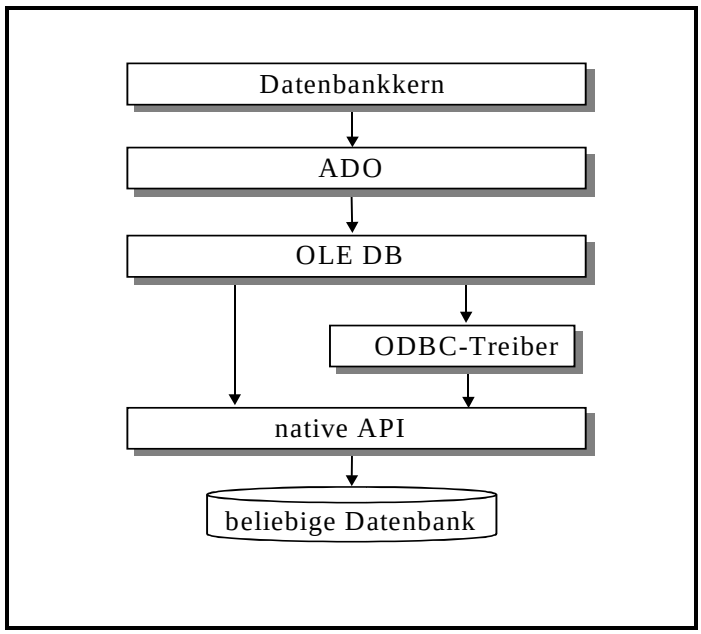


Abbildung 4-16: Datenbankzugriff über ADO

Abbildung 4-16 stellt den Datenbankzugriff über ADO und die Möglichkeit der Verwendung alter ODBC-Treiber über eine OLEDB-ODBC-Brücke dar. Das ermöglicht den Gebrauch von DBMS für die noch kein OLEDB-Provider existiert.

Ein weiterer Grund für den Einsatz von ADO ist der Gebrauch von Connection-Pooling[14] (Abbildung 4-17), das deutliche Performancegewinne beim Verbindungsaufbau zur Datenbank mit sich bringt. Möchte ein Benutzer auf Daten zugreifen, so ist nach der bisher diskutierten Architektur dieser Zugriff nur über den Datenbankkern erlaubt. Dieser stellt seinerseits wieder eine Verbindung zur Datenbank her und führt angeforderte Transaktionen aus. Bei einer Vielzahl von Benutzern entsteht auf diese Weise auch eine Vielzahl offener Datenbankverbindungen. Das bringt zum einen serverseitige Performanceverluste mit sich, zum anderen können Lizenzprobleme auftreten, wenn ein DBMS seine Lizenzen pro Verbindung abrechnet. Eine Möglichkeit dies zu umgehen ist der Auf- bzw. Abbau der Connection vor und nach einer

[14] Connection-Pooling (engl.): Verwaltung bestehender Datenbankverbindungen

Transaktion. Die daraus resultierenden Performanceprobleme sind auf den langen Verbindungsaufbau zur Datenbank zurückzuführen.

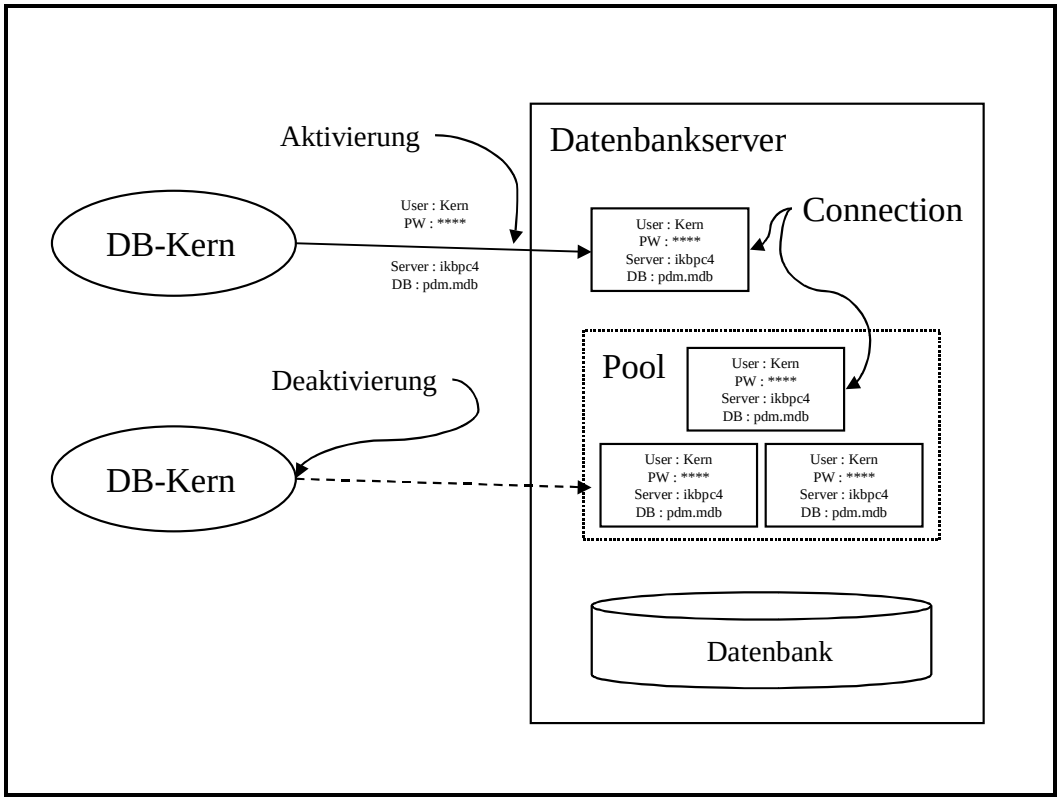


Abbildung 4-17: Verbindungsaufbau des Datenbankkerns zur Datenbank über Connection-Pooling

Das Zusammenspiel eines Resource Dispensers[15] und eines Transaktionsmanagers erlaubt jedoch die Verwendung sogenannter Connection-Pools. Darin existieren einmal geöffnete DB-Verbindungen weiter, auch wenn der entsprechende Client die Verbindung längst abgebaut hat. Beim Verbindungsaufbau eines weiteren Clients überprüft der Resource Dispenser die Connection-Strings der offenen Datenbankverbindungen und die des Clients. Diese Connection-Strings bestehen in der Hauptsache aus dem Servernamen, einem Usernamen, einem Passwort und der DB-Instanz. Sind die Connection-Strings des Clients und einer bereits geöffneten Verbindung identisch, so kann sich der Client dieser Connection bedienen ohne einen zeitintensiven Verbindungsaufbau durchlaufen zu müssen. Nach Trennung der DB-Verbindung wird die genutzte Connection wieder zurück in den Pool gegeben.

[15] Resource Dispenser (engl.) : Software zur Verteilung nicht dauerhafter Resourcen

```
...
if(SUCCEEDED(hr))
        {cnn->ConnectionString = strcnn;
        //Öffnen der Datenbankerbindung
        cnn->Open("", "", "", -1);}
HRESULT hr = rst->Open(sql, cnn.GetInterfacePtr(), adOpenDynamic, adLockOptimistic, adCmdText);
// Schliessen der Datenbankverbindung
if(cnn->State == adStateOpen)
hr = cnn->Close();
...
```

Abbildung 4-18: Verbindungsauf- und Abbau vor und nach einer Transaktion. Der Connection-String *strcnn* bleibt konstant

Der Datenbankkern nutzt diese Möglichkeit optimal, da Datenbankzugriffe nur über ihn möglich sind. So besitzen alle im Pool vorhandenen Verbindungen den gleichen Connection-String und sind somit allen Instanzen des DB-Kerns verfügbar. Die Bindung einer Connection an eine Instanz ist zudem nur von sehr kurzer Dauer, da eine DB-Verbindung nur direkt vor einer Transaktion hergestellt und danach sofort wieder abgebaut wird (Abbildung 4-18).

4.3.3 Architektur des Datenbankkerns

Der Datenbankkern greift wie diskutiert über vertraglich festgelegte Schnittstellen der ADO-Komponente auf DBMS zu. Diese Datenbankzugriffe werden im Zuge einer Auftragserteilung durch Drittkomponenten ausgeführt, die ihrerseits die Dienste des Datenbankkerns in Anspruch nehmen. In der Hauptsache soll der über dem DB-Kern liegende PDM-Kern diese Dienste in Anspruch nehmen, aufgrund der Forderung nach modularem Aufbau und Wiederverwendbarkeit sollen aber auch beliebige andere Softwarekomponenten diese nutzen können. Um Dienste in dieser Weise bereitstellen zu können, müssen sie ebenfalls über vertraglich festgelegte Schnittstellen erreichbar sein. Aus diesem Grund ist der Datenbankkern als COM+-Komponente aufgebaut. Der Zugriff auf seine Dienste ist dadurch geregelt und diese sind über Rechnergrenzen hinaus verfügbar.

Aus Sicht der Komponentenprogrammierung besteht eine COM-Komponente aus einer Black-Box und nach aussen geführten Interfaces. Dabei wird das Standardinterface IUnknown

um benutzerdefinierte Interfaces erweitert. Diese Interfaces sollen nach logischen Gesichtspunkten gegliederte Methoden in sich aufnehmen. Im Fall des DB-Kerns können zwei zusätzliche Interfaces definiert werden, so dass sich folgende Schnittstellen ergeben (Abbildung 4-19):

- IUnknown
- IAdmin
- IWork.

Das Interface IAdmin stellt Methoden zur Administration der Datenbank zur Verfügung. Diese Methoden werden benötigt um beispielsweise das Datenbankschema individuellen Bedürfnissen anzupassen. Zwar ist eine Anpassung auch direkt im Datenbanksystem möglich, doch durch Gebrauch dieses Interfaces soll die Konsistenz des Datenmodells sichergestellt werden. Zugriff auf diese Schnittstelle ist nur dem Administrator zu gewähren. Der Einsatz eines entsprechenden Schutzmechanismus wird später diskutiert.

Methoden zur Datenmanipulation und zur Datenabfrage sind in der Schnittstelle IWork zusammengefasst. Diese Schnittstelle soll im Regelfall nur vom PDM-Kern oder anderen Softwarekomponenten angesprochen werden. Die in diesem Interface bereitgestellten Dienste ermöglichen das Arbeiten mit dem Datenbanksystem auf objektorientierter Basis. Der PDM-Kern macht in seinen Aufrufen an den DB-Kern immer Gebrauch von Objektbezeichnungen. Da der Einsatz eines objektorientierten DBMS zwar möglich aber derzeit noch sehr unwahrscheinlich ist, muss der DB-Kern aus den Objektbezeichnungen die Objekte und die dazugehörigen Tabellen ermitteln und daran Manipulationen vornehmen. Sollen beispielsweise die Metadaten eines Dokumentes ermittelt werden, so erfolgt vom PDM-Kern der Aufruf *DBKern.GetObjectAttributes (OID)* mit der Angabe der Objektnummer *OID*. Anhand dieser ermittelt der DB-Kern die zugehörigen Tabellen und gibt einen Recordset mit allen Metadaten des Dokumentes zurück.

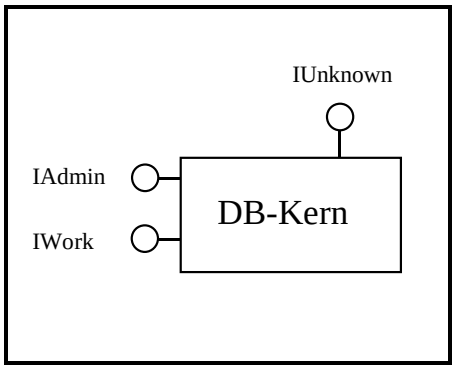


Abbildung 4-19: Darstellung des Datenbankkerns und seiner Schnittstellen

Eine Möglichkeit zur Ermittlung dieser Metadaten ist die Verwendung sogenannter *Stored Procedures*[16]. Fast alle relationalen DBMS unterstützen Stored Procedures, die als kleine Programme eigenständig auf einem Datenbankserver laufen, um dort Datenmanipulationen oder Abfragen vorzunehmen. Damit ließe sich theoretisch ein großer Teil des DB-Kerns auf die Datenbank selber portieren. Der Vorteil liegt in der Performance der Stored Procedures, da diese vom Datenbanksystem vorkompiliert und optimiert werden. Zwischenergebnisse von Stored Procedures können von weiteren Stored Procedures verwendet werden, so dass diese nicht erst zum Client gesendet und dort ausgewertet werden müssen. Das führt zu einer Verringerung der Netzwerkbelastung.

Der DB-Kern sieht von der Verwendung derartiger Stored Procedures jedoch ab. Ein Grund dafür ist die Optimierung der Stored Procedures auf Client/Server-Architekturen. Die hier verwendete 3-Schicht-Architektur bietet zwischen Client und Server noch eine weitere logische Schicht, die viel enger mit dem Server arbeiten kann als der Client des klassischen Client/Server-Prinzips. Beispielsweise müssen auch hier die Zwischenergebnisse einer Abfrage nicht zum Client, sondern nur bis zum dazwischenliegenden Businesslayer transportiert werden, der sich unter Umständen sogar auf dem selben Rechner befindet. Auch dadurch wird eine deutliche Verringerung der Netzwerkbelastung erreicht. Ein weiterer Punkt der gegen die Verwendung von Stored Procedures spricht, sind deren nicht standardisierte Programmiersprachen. So können Stored Procedures eines Microsoft SQL-Servers nicht auf einen Oracle-Server übertragen werden, ohne überarbeitet werden zu müssen. Der Datenbankkern soll jedoch beliebige Datenbanksysteme unterstützen und die Datenbasis eines DBMS soll problemlos auf ein weiteres portierbar sein. Diesem Anspruch an Flexibilität kann man durch Nutzung von Stored Procedures nicht gerecht werden.

Alle datenmanipulierenden Methoden verwenden aus o.g. Gründen keine Stored Procedures sondern greifen ausschließlich per SQL auf die Datenbank zu. Die notwendigen SQL-Kommandos können dabei je nach Zugriff beliebig komplex werden. Um nicht innerhalb jeder Methode einen SQL-String erzeugen zu müssen, existiert deshalb eine für den internen Gebrauch zu verwendende SQL-Klasse, die Methoden zur Generierung von SQL-Strings enthält.

[16] Stored Procedures (engl.) : gespeicherte Prozeduren

```
BSTR SqlClass::InnerJoin(int anzTab, BSTR Tab1, BSTR Col1, BSTR Tab2, BSTR Col2 ,BSTR Tab3, BSTR Col3 ,BSTR Tab4,
                         BSTR Col4, BSTR queryField, BSTR queryTable)
{
        _bstr_t Befehl(_bstr_t("SELECT ") + _bstr_t(queryField) + _bstr_t(".") + _bstr_t(queryTable);
        Befehl += ("FROM");
        for (i=1;i<=anzTab;++i)
        {
                switch (i)
                case 1:
                        BSTR += _bstr_t(Tab1) + _bstr("INNER JOIN") + _bstr("[") + _bstr(Tab2) + _bstr("]")
                                + _bstr_t("ON") + _bstr(Tab1) + _bstr(".") + _bstr(Col1)
                                + _bstr_t("=")  + _bstr(Tab2) + _bstr(".") + _bstr(Col2);
                        break;
                case 2:  .....
                case 3:  .....
                case 4:  .....
        }
        return Befehl.copy();
}
```

Abbildung 4-20: Ausschnitt aus einer Methode zur Generierung einer SQL-Abfrage

Abbildung 4-20 zeigt einen Ausschnitt einer solchen Methode. Der Rückgabewert *Befehl* beinhaltet das SQL-Kommando und ist vom COM-spezifischen Datentyp *BSTR* und deshalb diesen der Rückgabewert sofort Verwendung in einem ADO-Command-Objekt finden.

Außer der SQL-Klasse existiert eine weitere interne Klasse, die Toolklasse (Abbildung 4-21), die allgemeine Methoden zur Verwendung in den drei bisher aufgeführten Klassen bereitstellt. Aufgaben dieser Methoden sind beispielsweise die Generierung eindeutiger *Object Identifier* (OID) bei der Erzeugung neuer Objekte oder die Wandlung COM-spezifischer Datentypen in Typen zur Verwendung in SQL-Befehlen.

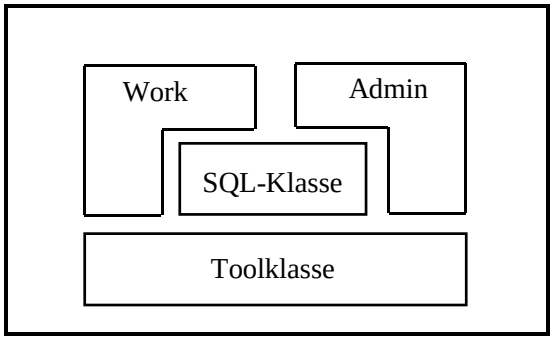


Abbildung 4-21: Die Klassen des Datenbankkerns

Bevor die Schnittstellen IAdmin und IWork diskutiert werden, soll zunächst auf die Probleme hingewiesen werden, die bei der Realisierung dieser Interfaces auftraten.

Umfangreiche Navigationsmöglichkeiten und Funktionen zur hierarchischen Strukturierung machen ADO-Recordsets zu einem idealen Rückgabetyp für Abfragen. So ist der Rückgabewert einiger Methoden der o.g. Schnittstellen vom Typ ADO-Recordset. Um Recordsets als Ein- bzw. Ausgabeparameter über COM-Schnittstellen zu nutzen, muss die ADO-Type-Library in die IDL-Datei zur Schnittstellendefinition importiert werden. Das dazu notwendige *importlib* Statement wird mit der zugehörigen dll in die Bibliotheksdeklaration des Datenbankkerns aufgenommen. Trotz dieses Imports ist die Verwendung von Recordsets als Ein- bzw. Ausgabeparameter nicht möglich. Es muss zusätzlich eine fundamentale Änderung an der von Visual C++ erzeugten Struktur der IDL-Datei erfolgen. Die in der automatisch erzeugten Schnittstellendefinition vorhandenen Interfacedeklarationen IWork und IAdmin müssen zusätzlich in die Bibliotheksdeklaration verschoben werden. Nur so können diese vom Import der ADO-Type-Library profitieren.

Recordsets als Rückgabeparameter bringen jedoch noch eine weitere Problematik mit sich. Vor der Verwendung eines Recordset muss dieses durch den Befehl CreateInstance als COM-Object instanziiert werden. Der zugehörige Instanzenzähler erhöht sich dabei um den Wert eins. Direkt nach einer Abfrage muss zur optimalen Verwendung des Connection-Poolings (Kap. 4.3.2) die Datenbankverbindung abgebaut und die Verbindungsreferenz des Recordsets auf *NULL* gesetzt werden. Wird die Methode verlassen, so erniedrigt sich der Instanzenzähler des Recordsets wiederum um eins auf den Wert null. Dadurch wird das Recordset zerstört und steht für den Gebrauch im Client nicht mehr zur Verfügung. Aus diesem Grund muss bei Verwendung eines Recordsets als Rückgabetyp der Instanzenzähler von Hand um eins erhöht werden (Abbildung 4-22). Der Client muss dafür Sorge tragen, dass der Zähler nach Auswertung des Recordsets auf null gesetzt wird, um somit eine Freigabe des allozierten Speicherbereichs zu ermöglichen.

```
if(SUCCEEDED(hr)){
        rst->AddRef();                  //erhöhen des Instanzzählers
        *rs = rst;                      //Eingangsparameter setzen
        rst->PutRefActiveConnection(NULL); // Datenbankverbindung auf NULL setzen
}
        return hr;
```

Abbildung 4-22: Erhöhen des Instanzenzählers des Recordset-Objektes

4.3.4 Die Schnittstelle IAdmin

Wie bereits erwähnt, dient die Schnittstelle IAdmin der administrativen Arbeit am Datenbankkern. Diese Arbeiten beziehen sich in der Hauptsache auf die Anpassung des Datenmodells. Dabei ist zu berücksichtigen, dass der Administrator keinerlei Annahmen über Tabellen im Datenmodell machen darf, sondern Erweiterungen und Anpassungen nur über Klassendefinitionen vorzusehen hat. Nachfolgend sollen einige der entwickelten Methoden zum besseren Verständnis der Schnittstelle vorgestellt werden.

- *CreateClass*. Um direkt unterhalb der Superclass neue Klassen zu erstellen, wird diese Methode verwendet. Als Übergabeparmeter bekommt sie nur den Namen der neu anzulegenden Klasse. Dieser Klassenname wird zur Erstellung einer DB-Tabelle verwendet. In der Tabelle Datamodellclass wird die Vererbung von der Superclass eingetragen.

- *CreateSubClass*. Diese Methode dient der Erzeugung von Kindklassen, die alle Eigenschaften, d.h. Attribute, ihrer Elternklasse erben. Die Parameterliste besteht aus dem Namen der neu anzulegenden Klasse und dem Namen der Elternklasse. Die Methode durchsucht die Tabelle der Elternklasse nach ihren Attributen und legt eine neue Tabelle mit diesen Attributen und dem Namen der anzulegenden Klasse an. Auch hier erfolgt danach ein Eintrag in der Datamodelclass, um diese Vererbung zu dokumentieren.

```
BSTR sql = Sqlsetr::Tab_erstellen(SubTyp);
        HRESULT hr = cnn->Execute(sql, pvtEmpty2, adCmdText);
        if(SUCCEEDED(hr))
        {
                hr = rst->Open("SELECT * FROM " + _bstr_t(Typ), cnn.GetInterfacePtr(),
                adOpenDynamic, adLockOptimistic, adCmdText);
                if(SUCCEEDED(hr))
                        for(int loop = 0; loop < rst->Fields->Count; loop++)
                        {
                          _bstr_t Attribut = rst->Fields->Item[_variant_t(loop)]->Name;
                          _bstr_t typ = BSTR(rst->Fields->Item[_variant_t(loop)]->Type);
                         ret = Farid.TypFind(typ, &sqlTyp);
                        _bstr_t command = Sqlsetr::ColAdd(SubTyp, Attribut, sqlTyp);
                        hr = cnn->Execute(command, pvtEmpty2, adCmdText);
                        }
        }
```

Abbildung 4-23: Methode zur Generierung von Kindklassen. Im oberen Teil wird die Tabelle angelegt, im unteren Teil werden die Attribute der Elternklasse eingefügt.

- *ClassAddAttribute*. Diese Methode ermöglicht die Definition der Eigenschaften, die außer dem Namen der Klasse und des Attributs den Typ des Attributs verlangt.

- *CreateStructure*. Die bereits angesprochene Klasse Structurclass repräsentiert erlaubte Beziehungen zwischen den einzelnen Klassen des Datenmodells. Um eine neue Beziehung anzulegen muss diese Methode gerufen und eine Parameterliste aus den in Beziehung zu setzenden Klassen übergeben werden.
- *GetSuperclass*. Bei der Programmierung einer Benutzeroberfläche zur Administration wird es notwendig sein, die Elternklasse einer Kindklasse zu bestimmen. Die Verwendung dieser Methode mit Übergabe der Kindklasse erlaubt durch Abfrage der Tabelle Datamodelclass die Bestimmung der zugehörigen Elternklasse.
- *GetSubclass*. Diese Methode erlaubt die Bestimmung der Kindklasse bei Übergabe einer Elternklasse. Der Ablauf ist der Methode GetSuperclass sehr ähnlich.

Zusätzlich zu diesen Methoden existieren noch weitere Methoden, die nicht im Detail diskutiert werden sollen. So wurden beispielsweise noch Methoden zum Löschen von Klassen, Attributen und Beziehungen programmiert. Zum Verständnis der hier beschriebenen Schnittstelle sollen diese Methoden ausreichen.

4.3.5 Die Schnittstelle IWork

Die hier entwickelten Methoden dienen der Datenabfrage und Datenmanipulation. Ähnlich der Schnittstelle IAdmin soll auch hier nur ein Ausschnitt dieses Interfaces diskutiert werden.

- *GetActiveDatabaseConnection*. Der als COM+ ausgelegte Datenbankkern kann von beliebigen Standorten instanziiert werden. Zur Ermittlung der aktuell vom PDM-Kern genutzten Instanz, dem Server und der Datenbank kann diese Methode verwendet werden, die entsprechende Informationen als Basicstring zurückgibt.
- *GetObjAttributes*. Diese Methode ist eine der grundlegenden Arbeitsmethoden. Werden im PDM-Kern die Metadaten eines Objektes benötigt, so ruft dieser die hier beschriebene Methode durch Übergabe einer ObjectID (OID) auf. Innerhalb der Methode wird zunächst der Typ anhand der OID durch Abfrage der Klasse TypClass ermittelt. Danach werden die Metadaten des Objekts bestimmt und ein Datensatz vom Typ ADO-Recordset zurückgegeben.
- *CreateNewObject*. Zum Anlegen eines neuen Objektes muss diese Methode gerufen werden. Als Parameter wird der Objekttyp benötigt, um im internen Nummerngenerator der Toolklasse eine neue OID zu generieren.

- *WhereReferenced*. Gibt einen hierarchischen Recordset zurück. Dieser beschreibt, zu welchen anderen Objekten ein angegebenes verknüpft ist. Soll beispielsweise festgestellt werden in welchen Baugruppen ein bestimmtes Modell verwendet wird, so ist diese Methode mit der OID des Modells aufzurufen.
- *WhatReferenced*. Zur Darstellung einer Produktstruktur im PDM-Kern ist die Information über alle zu einem Objekt referenzierten Objekte notwendig. Der Aufruf dieser Methode mit Übergabe der OID eines Objektes gibt einen Recordset mit OIDs aller dazu referenzierten Objekten zurück.

Alle Arbeitsmethoden, deren Rückgabewert vom Typ Recordset ist, erfordern zudem noch Angaben über den Lockingmechanismus der zurückgegebenen Recordsets. Die Lockingmechanismen entsprechen denen von ADO und ermöglichen ein vollständiges und teilweises Sperren der Datensätze sowie die Stapelaktualisierung von getrennten Recordsets. Welche dieser Lockingmechanismen verwendet wird, hängt vom Einsatz der Methoden ab und ist aus diesem Grund vom Client vorzugeben.

4.3.6 Datensicherheit und Zugriffsregelung im Datenbankkern

Die bisher diskutierten Schnittstellen und deren Methoden unterliegen noch keinen Sicherheitsmechanismen. So ist es allen Usern möglich, über geeignete Clients auf die Interfaces des Datenbankkerns zuzugreifen und Veränderungen am Datenmodell oder den Produktdaten vorzunehmen, ohne über besondere Privilegien zu verfügen. Datensicherheit und Zugriffregelungen sind jedoch bedeutende Aufgaben des Datenbankkerns, der per Definition die Manipulation des Produktdatenbestandes regeln soll.

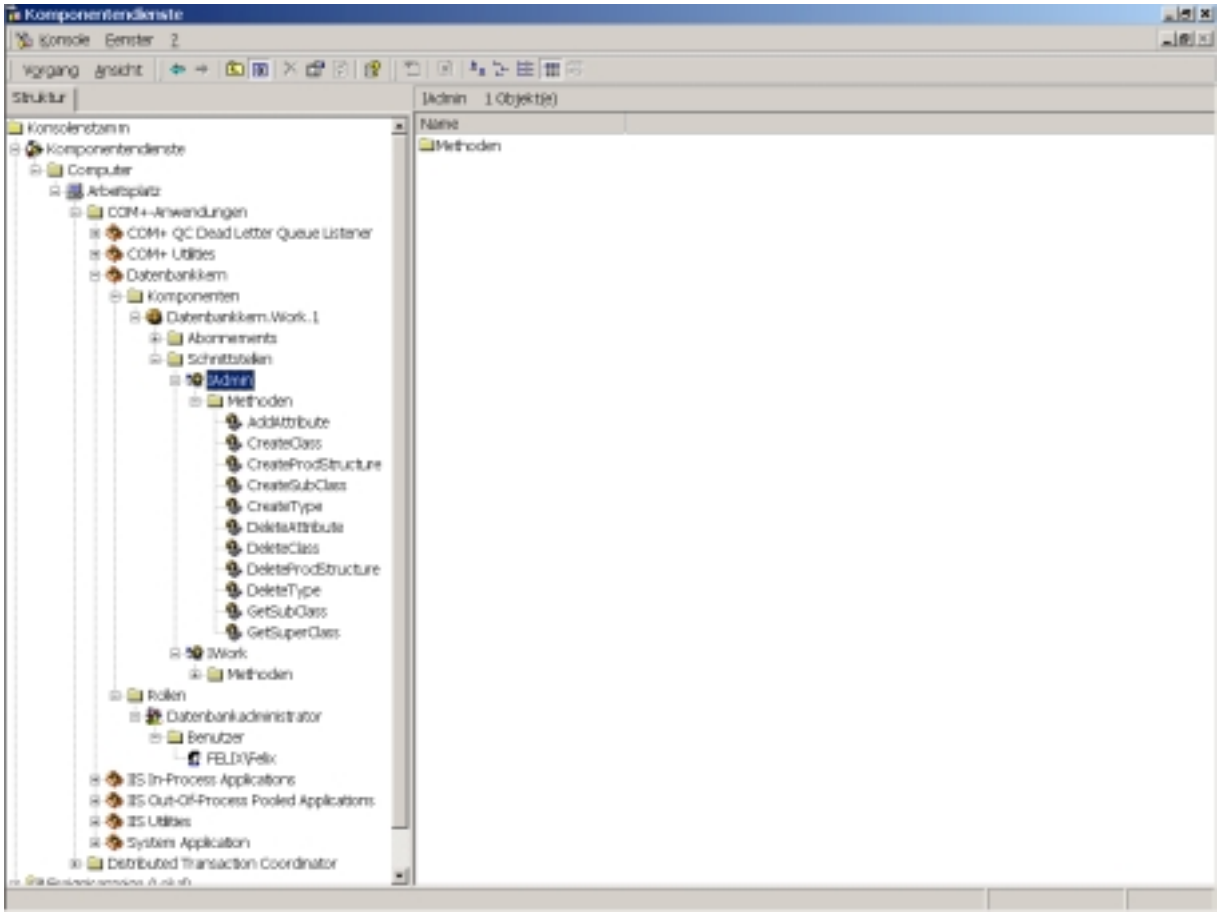


Abbildung 4-24: Der Datenbankkern als Eintrag in den COM+-Komponentendiensten. Dargestellt ist die Schnittstelle IAdmin und deren Methoden.

Die Auslegung des Datenbankkerns als COM+-Komponente ermöglicht die Verwendung der COM+-Zugriffsregelungen auf Schnittstellen und Methoden des Kerns. Der als *Fine-Grained Security*[17] bezeichnete Sicherheitsmechanismus führt eine Zugriffsüberprüfung auf Prozess- und Komponentenebene durch und besteht in der Hauptsache aus zwei Teilen:

[17] Fine-Grained Security (engl.): feinstrukturierte Sicherheit

- ein deklarativer Teil, der durch Konfiguration auf Betriebssystemebene Zugriffsberechtigungen auf Schnittstellen und Methoden vergibt und
- ein programmierbarer Teil, der durch Programmierung innerhalb der implementierten Methoden Berechtigungen des angemeldeten Users überprüft.

Der im Datenbankkern eingesetzte Sicherheitsmechanismus soll ausschließlich aus deklarativer Konfiguration bestehen. Programmierbare Sicherheit auf Methodenebene wird später im PDM-Kern eingesetzt und dort diskutiert.

Die deklarative Zugriffsregelung verlangt die Definition von Usergruppen. Diese Gruppen bestehen aus Mitarbeitern, die aufgrund ähnlicher Aufgaben innerhalb des Unternehmens gleiche Zugriffsberechtigungen auf die Produktdaten bekommen. Im Fall des Datenbankkerns soll jedoch nur ein direkter Zugriff auf die Schnittstelle IAdmin gewährt werden. Die Benutzergruppe besteht dann ausschließlich aus den Datenbankadministratoren. Die Schnittstelle IWork soll nur von anderen Softwarekomponenten angesprochen werden, so dass hier kein realer Benutzer Zugriffsberechtigungen erhalten darf.

Alle Administratoren müssen als Benutzer im Betriebssystem vorhanden sein und können dann zu einer Rolle *Datenbankadministrator* zusammengefasst werden. Die erforderliche Konfiguration der Zugriffsberechtigung auf die Schnittstelle IAdmin erfolgt in den Komponentendiensten (Abbildung 4-24). Hier wird definiert, dass die Benutzer in der Rolle Datenbankadministrator auf diese Schnittstelle zugreifen dürfen. Dazu werden gemäss Abbildung 4-25 die Eigenschaften der Schnittstelle IAdmin konfiguriert.

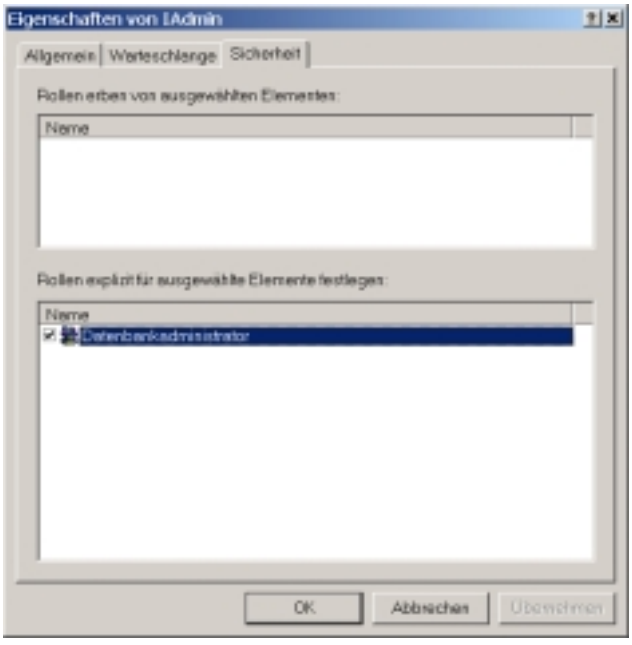
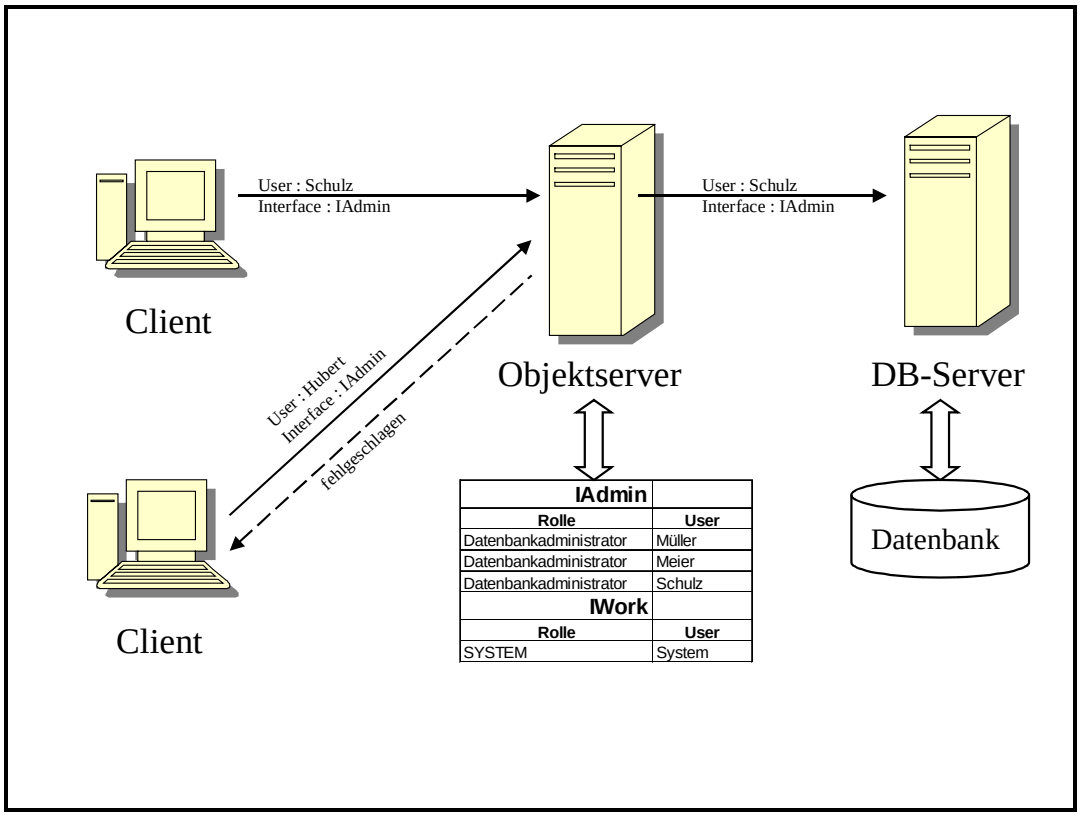


Abbildung 4-25: Eigenschaften der Schnittstelle IAdmin

Das Interface IWork lässt nach entsprechender Konfiguration nur Zugriffe der Rolle *System* zu. Mitglied der Rolle System ist nur das Benutzerkonto *System*, das den PDM-Kern dazu autorisiert, Zugriffe auf die genannte Schnittstelle auszuführen.

Der Grund für den Verzicht einer eigenen Benutzerverwaltung liegt in der besseren Skalierbarkeit und Konsistenz der Zugriffsrechte. Einmal im Betriebsystem vorhandene Benutzer können sofort auf entsprechende Dienste des **TPIS** zugreifen und brauchen zusätzlich an keiner anderen Stelle gepflegt werden. Außerdem entfällt die gesamte Implementierung einer Benutzerverwaltung auf dem Datenbankserver. Die Benutzerauthentifizierung erfolgt nach Auslegung des DB-Kerns als 3-Tier-Applikation auf der Geschäftsebene. Ein Verbindungsaufbau zum DB-Server entfällt für fehlgeschlagene Anmeldungen ganz, was wiederum zu einer Entlastung des Datenbankservers und des Netzwerks führt (Abbildung 4-26).



IAdmin	
Rolle	**User**
Datenbankadministrator	Müller
Datenbankadministrator	Meier
Datenbankadministrator	Schulz
IWork	
Rolle	**User**
SYSTEM	System

Abbildung 4-26: Zugriffsüberprüfung des Objektservers auf Geschäftsebene

4.4 Konzeption des PDM-Kerns

Der im vorigen Kapitel beschriebene Datenbankkern dient als Grundlage für den Datenzugriff des PDM-Kerns auf die Datenbasis. In diesem Kapitel soll der Aufbau des PDM-Kerns erfolgen. Dabei soll die Ausführung des Kerns als COM+-Komponente in den Vordergrund gestellt werden, um Einsatzmöglichkeiten des **TPIS** über Unternehmensgrenzen hinweg aufzuzeigen.

Der PDM-Kern besteht wie der Datenbankkern aus einer logischen Zusammenfassung seiner Methoden zu Funktionseinheiten. Das wiederum führt zu folgenden, im Kern zu implementierenden Schnittstellen:

- die Schnittstelle IManagament stellt Managementfunktionen zur Verfügung,
- Methoden zur Administration sind im Interface IAdmin zusammengefasst,
- IWork stellt Arbeitsfunktionen zur Verfügung und
- IXML ermöglicht einen Datenaustausch über das standardisierte XML-Format.

Diese Schnittstellen sollen im folgenden vorgestellt werden.

4.4.1 Die Schnittstelle IManagement

Managamentfunktionen derzeitiger PDM-Systeme umfassen hauptsächlich die Aufgabengebiete Workflowmanagement und Freigabemanagement. Zwar sind Freigabe- und Workflowmanagement zwei getrennt voneinander agierende Dienste, doch ergibt deren Verschmelzung zu einem Gesamtsystem erst eine Unterstützung des Produktentwicklungsprozesses. Das Freigabemanagement steuert den Status aller am Entstehungsprozess beteiligten Objekte, das Workflowmanagement hingegen steuert den Informationsfluss, der durch diese Objekte repräsentiert wird. Eine Forderung an den PDM-Kern ist es, Workflowobjekte unternehmensübergreifend zu verwalten, was derzeit über Installationsgrenzen hinweg noch nicht möglich ist. Wie oben bereits angedeutet, erfordert ein solcher Informationsfluss jedoch entsprechend angepasste Freigabeschemata. Aus diesem Grund soll die Funktionsweise eines Freigabeschema zunächst erläutert werden.

Freigabemanagement im PDM-Kern

Das im PDM-Kern zu implementierende Freigabeschema muss im im wesentlichen drei Anforderungen erfüllen:

- Unterstützung von Rollen,
- freie Skalierbarkeit des Schemas und
- Berücksichtigung aller Instanzen der im Datenmodell definierten Klassen als Workflowobjekte.

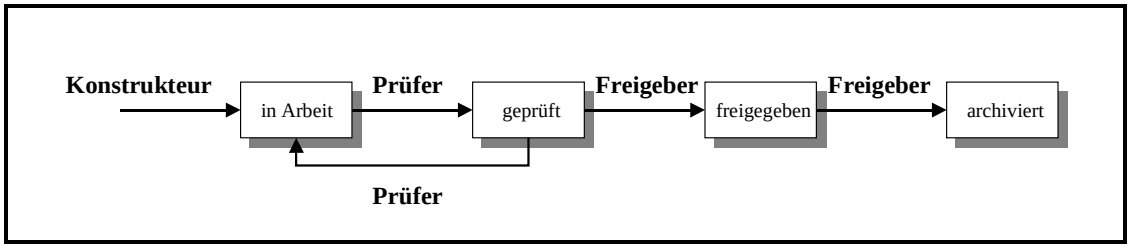


Abbildung 4-27: Typisches Freigabeschema mit Stati und Rollen

Die Unterstützung im ersten Punkt angesprochener Rollen passt sich nahtlos an die bereits im Datenbankkern eingeführten Rollen zur Verwendung deklarativer Sicherheit an. Abbildung 4-27 stellt ein Freigabeschema mit verschiedenen Freigabestati und notwendiger Rollen zur Statusüberführung dar. Die Statusüberführung eines beliebigen Objektes von *in Arbeit* nach *geprüft* ist nur Mitarbeitern erlaubt, die Mitglieder einer Benutzergruppe in der Rolle *Prüfer* sind. Der Statuswechsel wird ausgeführt, indem eine Methode der Schnittstelle IManagement gerufen wird, die als Übergabeparameter die OID des zu überführenden Objektes und den gewünschten neuen Status erfordert. Der aktuelle Status eines Objektes wird ermittelt, indem der PDM-Kern den Datenbankkern auffordert, alle Metadaten des entsprechenden Objektes in einem Recordset zurückzugeben.

ObjectTyp	StatusFrom	StatusTo	Role
3D-Model	in Arbeit	geprüft	Prüfer
3D-Model	geprüft	freigegeben	Freigeber
3D-Model	freigegeben	Archiv	Freigeber
3D-Model	geprüft	in Arbeit	Prüfer

Abbildung 4-28: Datenbanktabelle zur Konfiguration des Freigabeschemas

Daraus wird der Freigabestatus ermittelt und der PDM-Kern prüft in einer weiteren Datenbanktabelle den gewünschten Statusübergang ab. Das bisher besprochene Datenmodell ist

daher um die in Abbildung 4-28 dargestellte Tabelle zu erweitern. In dieser Tabelle sind Informationen darüber enthalten, welche Rollen für den definierten Statuswechsel angegebener Objekttypen erforderlich sind. Ist der gewünschte Statusübergang eines Objektes per Definition erlaubt, so findet eine Überprüfung der Rollen statt. Dazu wird die Rolle des angemeldeten Users mit der in der Datenbank definierten Rolle zum Statuswechsel verglichen. Da der PDM-Kern als COM+-Komponente ausgelegt ist, kann sich der Kern der Fine-Grained Security der Komponentendienste bedienen. Im Gegensatz zur deklarativen Sicherheit im Datenbankkern findet diese Überprüfung jedoch programmtechnisch statt, da der Aufruf der Methode zum Statuswechsel grundsätzlich jedem Benutzer gewährt bleiben muss. Erst innerhalb der Methode kann ein Ausschluss stattfinden.

```
STDMETHODIMP CManagement::ChangeReleaseStatus(BSTR newStatus, long OID)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState())
        HRESULT hRes, hRetval;
        ISecurityCallContext *pAufruf;
        Bool isInRole;
        //alter Status
        BSTR oldStatus = GetCurrentStatus(OID);
        //Rolle zum Statuswechsel
        BSTR roleAccess = GetNeededRole(oldStatus, newStatus, OID);
        hREs = CoGetCallContext(IID_SecurityCallContext,(void **) &pAufruf);
        if SUCCEEDED(hRes)){
                // Abfrage ob User in Role
                pAufruf->IsCallerInRole(roleAccess,&isInRole)
                        if (!isInRole){
                                hRetval=E_CALLERNOTALLOWED
                                return hRetval;

                        }
        }
        // Ändern des Status
        ChangeObjStatus(OID,newStatus);
        return S_OK;
}
```

Abbildung 4-29: Methode zur programmtechnischen Überprüfung des Statuswechsels

Abbildung 4-29 stellt ein Listing zur Überprüfung des Statuswechsels dar. In der ersten fett dargestellten Zeile wird ein Zeiger auf das COM-Interface ISecurityCallContext zurückgegeben, der in der zweiten markierten Zeile zur Rollenüberprüfung benötigt wird. Ist der diese Methode aufrufende User nicht in der notwendigen Rolle, wird ein entsprechender Returnvalue zurückgegeben und der Statuswechsel nicht durchgeführt.

Workflowmanagement im PDM-Kern

Das oben diskutierte Freigabemanagement soll, wie bereits angesprochen, sinnvoll mit einem Workflowmanagement gekoppelt werden, um Status und Ort von Informationsobjekten definiert wiederzugeben. Das nun herzuleitende Workflowmanagement soll dabei derart ausgelegt sein, dass es unternehmensübergreifend eingesetzt werden kann. Um das Aufgabengebiet eines unternehmensübergreifenden Workflows besser zu definieren, soll zunächst die projektbezogene Zusammenarbeit zweier Unternehmen und deren Informationsaustausch untereinander erläutert werden. Dazu sind zwei Szenarien vorstellbar:

- Ein Zulieferer bekommt technische Zeichnungen eines herzustellenden Bauteils oder
- ein Zulieferer bekommt Vorgaben für die Herstellung eines Bauteils und entwickelt dieses selber. Während der Entwicklung und vor der Fertigung muss der Auftraggeber das Ergebnis der Entwicklung und Konstruktion positiv bestätigen.

Im ersten Fall ist der Informationsfluss relativ einfach und überschaubar. Nach Produktionsfreigabe eines Bauteils erstellt ein Sachbearbeiter technische Zeichnungen eines Modells und versendet diese zum Zulieferer. Dieser fertigt eine Kleinserie und lässt diese vom Auftraggeber begutachten. Danach erfolgt die eigentliche Serienproduktion.

Der zweite Fall erfordert jedoch sehr viel mehr Informationsaustausch und an dieser Stelle ist der Einsatz eines unternehmensübergreifenden Workflows durchaus praktikabel. Ein Unternehmen entscheidet, die Entwicklung und Konstruktion eines Bauteils einer Baugruppe von einem Zuliefererunternehmen durchführen zu lassen.

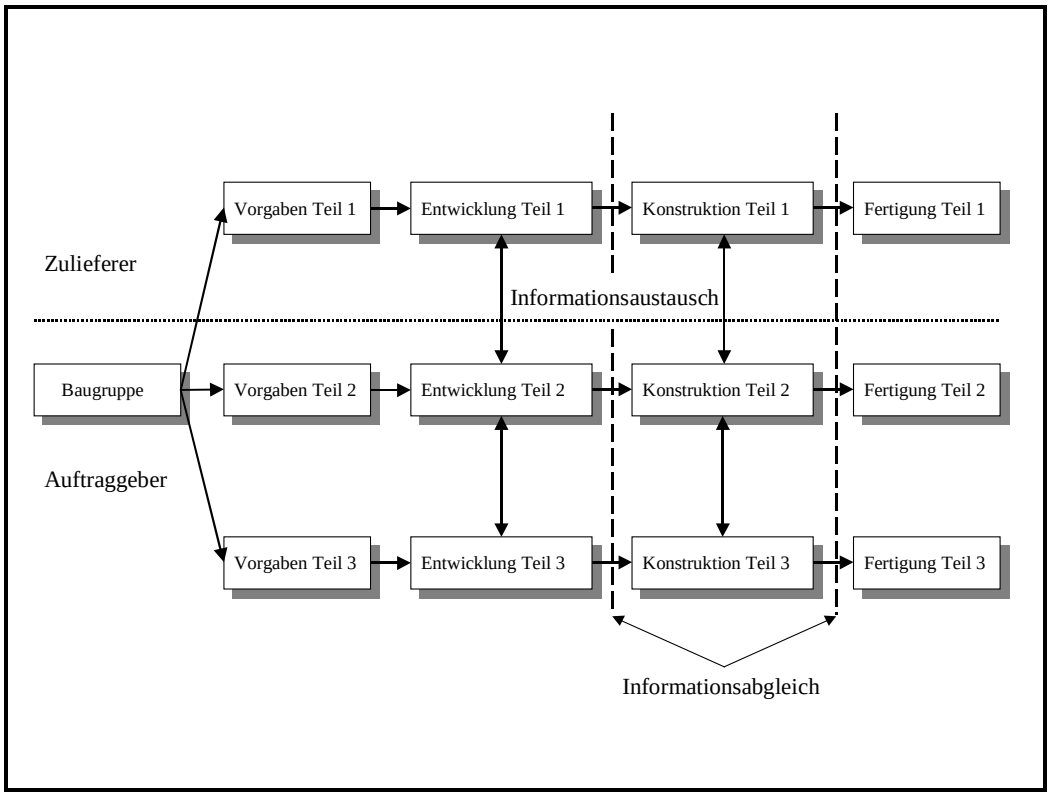


Abbildung 4-30: Unternehmensübergreifender Informationsfluss

Nach Abbildung 4-30 werden Vorgaben für die Fertigung der Baugruppe auf die entsprechenden Abteilungen und Zulieferer verteilt. Dem Gedanken des Concurrent Engineering folgend, sollen die einzelnen Teile möglichst parallel entwickelt und konstruiert werden, um den Produktentstehungsprozess möglichst kurz zu halten. Der Einsatz herkömmlicher PDM-Systeme erlaubt jedoch nur die Datenverwaltung und die Einbeziehung von Entwicklungsabteilung bezüglich einer lokalen Installation. Für den Informationsaustausch bedeutet das, Informationen müssen zu einem bestimmten Zeitpunkt aufbereitet und zwischen Zulieferer und Auftraggeber von Hand ausgetauscht werden. In der Regel geschieht der Informationsaustausch zu einem sehr späten Zeitpunkt, an dem sich eine der kooperierenden Parteien über das Ergebnis der Entwicklungs- und Konstruktionsarbeit sicher ist und dieses freigibt. Dem Prinzip des Concurrent Engineering ist auf diese Art und Weise jedoch nicht genüge getan. Um diese Art der Zusammenarbeit zu ermöglichen, müssen nach Abbildung 4-30 zwei Informationsflüsse zwischen den Unternehmen möglich sein:

- Informationsaustausch und
- Informationsabgleich.

Der Informationsaustausch findet während der Entwicklungs- und Konstruktionsphase statt. Dabei ist es möglich, auf ausgewählte und veröffentlichte Daten des jeweils anderen

Unternehmens zuzugreifen um so Erkenntnisse für die eigene Entwicklungsarbeit zu erlangen. Der Informationsaustausch findet zu keinem definierten Zeitpunkt statt, sondern unterstützt die Mitarbeiter des jeweiligen Kooperationspartners während der Entwicklung. Der Informationsabgleich hingegen findet zu einem Zeitpunkt statt, wenn eine Entwicklungs- oder Konstruktionsphase abgeschlossen ist. Der Zulieferer oder Auftraggeber muss über diesen Schritt informiert werden, und alle notwendigen bzw. feststehenden Informationen möglichst schnell erhalten. Setzt der Zulieferer den Auftraggeber über eine Freigabe in Kenntnis, so muss der Auftraggeber das erhaltene Ergebnis prüfen und seinerseits freigeben. Dann erfolgt die Fertigungsfreigabe an den Zulieferer und eine Mitteilung an die eigene Konstruktionsabteilung.

Erst durch das Zusammenspiel von Informationsaustausch und Informationsabgleich lässt sich der Gedanke des Concurrent Engineering auch unternehmensübergreifend in die Praxis umsetzen. Für den Workflow und dessen Anpassungen ist jedoch nur der Informationsabgleich von Bedeutung. Seine Umsetzung im PDM-Kern soll deshalb zunächst diskutiert werden, bevor Lösungen zur Unterstützung des Informationaustausches im PDM-Kern im Bereich der Schnittstelle IWork aufgezeigt werden.

Der in Abbildung 4-30 dargestellte Informationsabgleich lehnt sich an ein zuvor eingetretenes Ereignis. Diese Ereignisse werden in der Regel ausgelöst, wenn die im Workflow definierten Aufgaben an den dazugehörigen Workflowobjekten ausgeführt wurden. Diese Aufgaben sind oftmals an den zuvor besprochenen Freigabemechanismus gekoppelt. Wird beispielsweise ein Objekt freigegeben, so nimmt das Objekt den nächstdefinierten Workflowstatus an und wechselt damit unter Umständen auch seinen aktuellen Besitzer. Das verlangt aber nach der Auslösung eines Ereignisses bei der Freigabe, um damit Prozesse anzustossen, die für die Weiterverarbeitung des Objektes innerhalb des Workflows zuständig sind. Häufig genutzte Mittel für die Weiterverarbeitung sind Mailprogramme zur Benachrichtigung der Mitarbeiter über den Eingang zu bearbeitender Workflowobjekte.

Derartige Ereignisse auszulösen muss Aufgabe des PDM-Kerns sein, der damit weitere Softwarekomponenten zur Ereignisbehandlung auffordert. Der als COM+-Komponente ausgelegte PDM-Kern bedient sich dazu sogenannten COM+-Events.

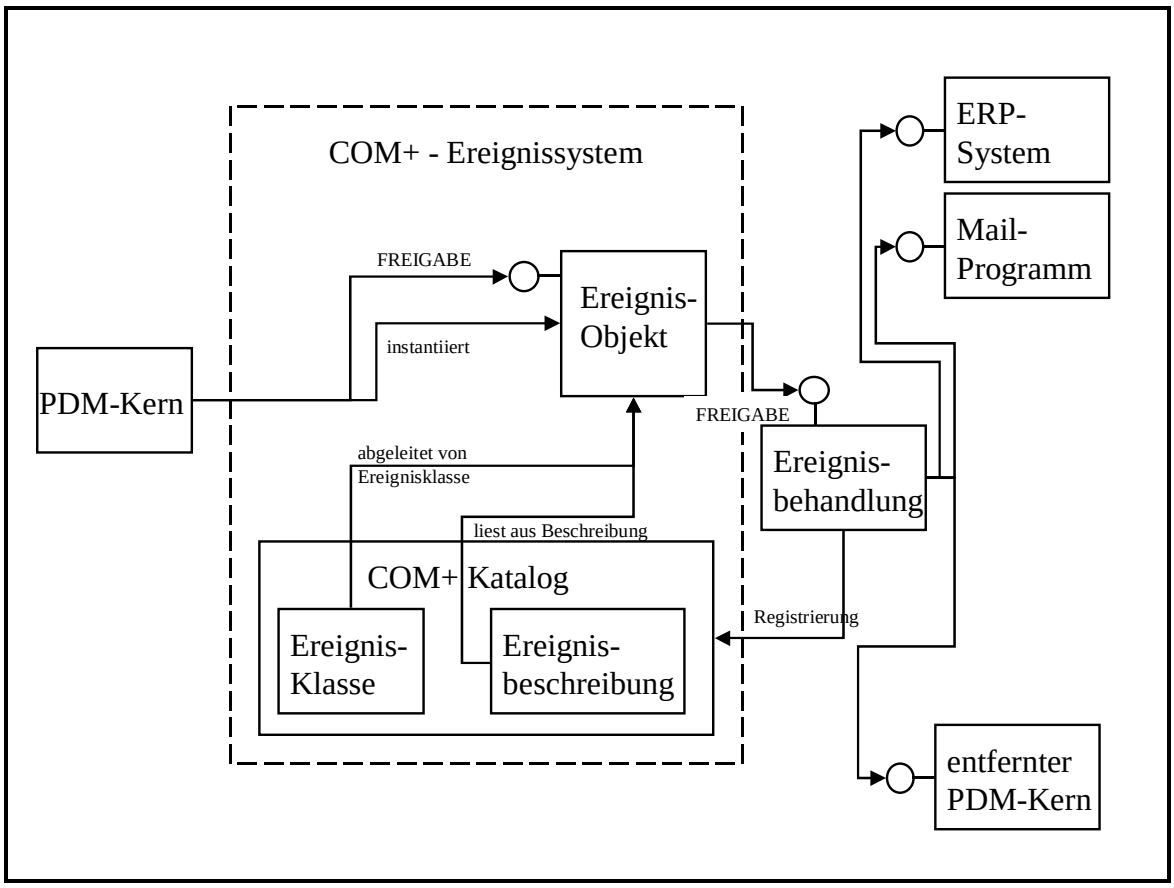


Abbildung 4-31: Zusammenspiel des PDM-Kerns mit dem COM+-Eventsystems

Abbildung 4-31 deutet das Zusammenspiel des PDM-Kerns mit dem COM+-Eventsystem an. Der Hauptbestandteil des COM+ -Ereignissystems ist der COM+- Ereigniskatalog. Ein Teil des PDM-Kerns ist als Ereignisklasse registriert. Diese Ereignisklasse besteht aus leeren Schnittstellenmethoden, die nur dazu dienen, zur Auslösung eines Ereignisses gerufen zu werden. Im Falle der Freigabe existiert beispielsweise eine Methode *Release* in der Ereignisklasse. Wird im PDM-Kern nun eine Freigabe ausgelöst, so soll die auf der rechten Seite dargestellte Klasse zur Ereignisbehandlung darauf reagieren. Dazu instantiiert der PDM-Kern ein Ereignisobjekt von der Ereignisklasse aus dem COM+ Ereigniskatalog. In dieser ist, wie bereits angesprochen, eine Methode Release vorhanden, die der Kern aufruft. Im COM+- Ereigniskatalog ist außer der Ereignisklasse noch die Klasse zur Ereignisbehandlung registriert. Durch diese Registrierung findet eine Zuordnung der gerufenen Schnittstellenmethoden der Ereignisklasse zu den dazu auszuführenden Methoden der Klasse zur Ereignisbehandlung statt. Wird also die Methode Release aufgerufen, so sorgt das Ereignissystem für einen Aufruf der entsprechenden Methode in der Ereignisbehandlungsklasse. Diese Ereignisbehandlungsklasse ist frei konfigurierbar, d.h. beim Eintreten eines Ereignisses können von dieser frei definierbare Dienste in Anspruch genommen werden. Das kann, wie in Abbildung 4-31 dargestellt, der Aufruf eines Mailprogramms zur Verfassung einer Mail oder die Nutzung beliebiger Dienste eines ERP-Systems sein. Bei der Freigabe eines Teils kann somit beispielsweise der zugehörige Teilestammsatz automatisch in das ERP-System geschrieben werden.

Die Registrierung der COM+ -Ereignisklassen und die Verknüpfung der Methoden darauf reagierender Softwarekomponenten ist durch die Verwendung der Komponentendienstkonfiguration durchzuführen. Wichtig ist an dieser Stelle jedoch festzuhalten, dass bei der Ausführung des PDM-Kerns keine Rücksicht auf später zu reagierende Ereignisse genommen werden muss. Alle Ereignisse können später in einer abstrakten Ereignisklasse implementiert werden. Ein weiterer wichtiger Grund für die Verwendung der COM+ -Ereignisse ist die Kopplungsmöglichkeit an beliebige Softwarekomponenten.

Bisher noch nicht diskutiert wurde die Ereignisbehandlung über Unternehmensgrenzen hinaus, was zur Umsetzung eines unternehmensübergreifenden Workflows jedoch notwendig ist. Abbildung 4-31 stellt zu diesem Zweck auch die Möglichkeit des Aufrufs eines entfernten PDM-Kerns dar. Dieser PDM-Kern, der seinerseits Teil einer unternehmensfremden PDM-Installation ist, kann von der Klasse zur Ereignisbehandlung über das WAN instantiiert werden. Damit können Dienste des entfernten PDM-Kerns genutzt werden. Diese Dienste sind, wie bereits im Konzept des Datenbankkerns diskutiert, durch die Verwendung der Fine-Grained Security vor unbefugten Zugriffen geschützt. Ein sogenannter Proxy-User, wird dazu in der entfernten PDM-

Installation eingerichtet und repräsentiert stellvertretend alle User oder Systeme, die Dienste dieses PDM-Kerns nutzen wollen. Beziehen sich die genutzten Dienste wiederum auf ein Workflowereignis, so wird im entfernten PDM-Kern ebenfalls ein Ereignis ausgelöst, das seinerseits über entsprechende Ereignisbehandlungsklassen darauf reagiert und ggf. vordefinierte Softwarekomponenten mit der Bearbeitung diverser Aufgaben beauftragt.

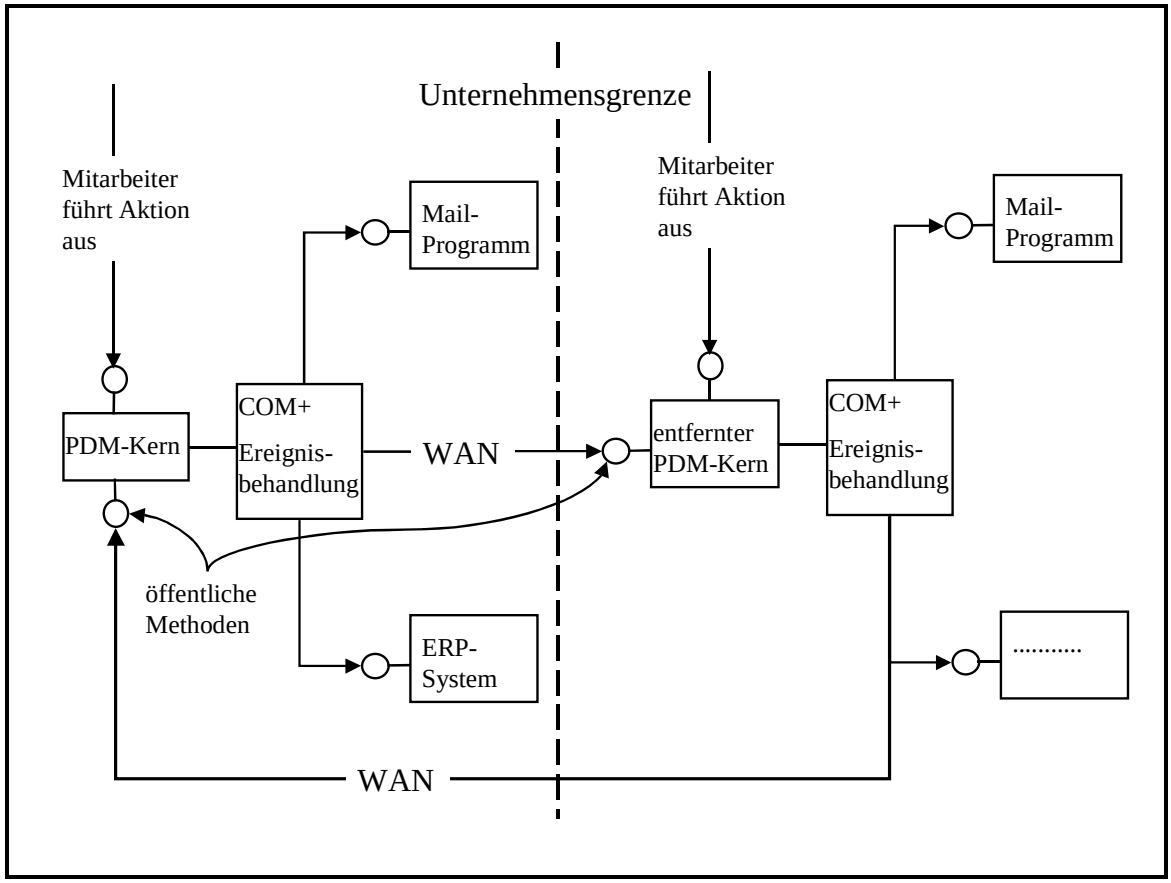


Abbildung 4-32: Schematischer Ablauf zu Ereignisbehandlung über Unternehemensgrenzen hinaus

Abbildung 4-32 deutet einen Ablauf zu einer derartigen Ereignisbehandlung an. Zu beachten ist hier die Tatsache, dass auch der entfernte PDM-Kern in der Lage ist, durch entsprechende Konfiguration der Ereignisbehandlung öffentliche Methoden des unternehmenseigenen PDM-Kerns durch Instantiierung über das WAN zu nutzen.

Die Instantiierung des entfernten PDM-Kerns und die geforderten Ereignisbehandlungen erfolgen nach Abbildung 4-32 durch synchrone Methodenaufrufe. Derartige Methodenaufrufe erfordern jedoch zu jeder Zeit eine Netzwerkverbindung zwischen der aufrufenden Ereignisklasse und dem entfernten PDM-Kern. Existiert eine solche Verbindung nicht, ist ein

synchroner Methodenaufruf nicht möglich und die zugehörige Ereignisbehandlung schlägt fehl. Der Workflow kann also durch Netzwerkprobleme in seinem Ablauf stark eingeschränkt werden oder überhaupt nicht zu Stande kommen. Aus diesem Grund sollen für die Ereignisbehandlung *Queued Componentes*[18] zur asynchronen Verarbeitung der Methodenaufrufe genutzt werden. Dazu wird Gebrauch vom sogenannten *Message Queuing*[19] gemacht. Notwendig ist ein Message-Recorder, der clientseitig Ereignisse der Ereignisklasse aufnimmt und sich dabei ähnlich einer Proxy-Komponente des entfernten PDM-Kerns verhält. Serverseitig nimmt ein Message-Player bei vorhandener Netzwerkverbindung die aufgezeichneten Ereignisse und damit verbundenen Methodenaufrufe entgegen und führt diese am PDM-Kern aus.

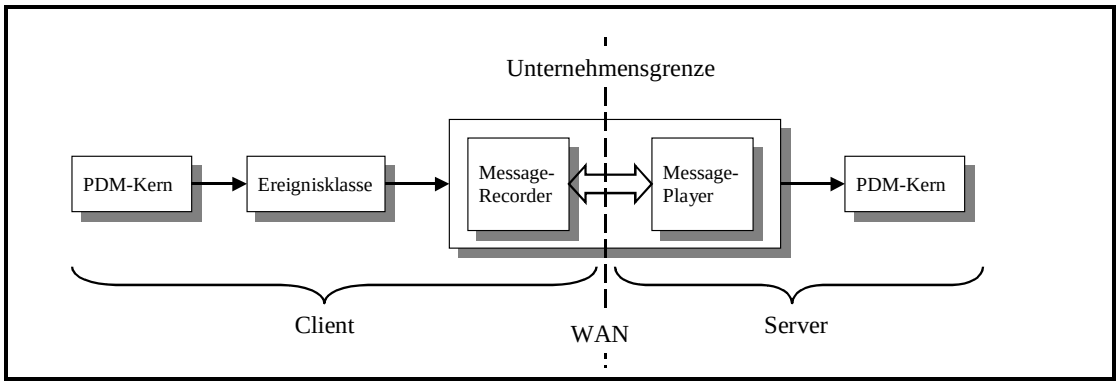


Abbildung 4-33: Message Queuing zwischen entfernten PDM-Kernen

Abbildung 4-33 stellt den Einsatz eines Message-Recorders und eines Message-Players dar. Diese werden über die Unternehmensgrenze hinaus durch das WAN miteinander verbunden. Das Unternehmen, in dem ein Ereignis ausgelöst wird, ist in diesem Zusammenhang als Client bezeichnet, da es Dienste vom entfernten, als Server bezeichneten, PDM-Kern anfordert.

Die optimale Abstimmung der Ereignisbehandlungen beider Unternehmen aufeinander ist Voraussetzung für einen gewinnbringenden Einsatz dieser Technologie und damit der Nutzung des Concurrent-Engineering über Unternehmensgrenzen hinaus. Je feinstrukturierter die Ereignisbehandlung aufgebaut ist, desto früher gelangt der Kooperationspartner an relevante Informationen zur Entwicklung der eigenen Komponenten, was wiederum zu einer Verkürzung der Gesamtentwicklungszeit führt. Abbildung 4-34 stellt exemplarisch einen Workflowprozess

[18] Queued Components (engl.): Gestapelte Komponenten, hier zur Stapelung eingetretener Ereignisobjekte und Abarbeitung dieser zu einem späteren Zeitpunkt.

[19] Message Queuing (engl.): Nachrichtenstapelung, Microsofts® Technologie zur asynchronen Nachrichtenbehandlung

ECO dar, der durch Anpassung der Ereignisbehandlungen abzubilden ist und so den Realisierung des Concurrent-Engineering über Unternehmensgrenzen hinweg ermöglicht.

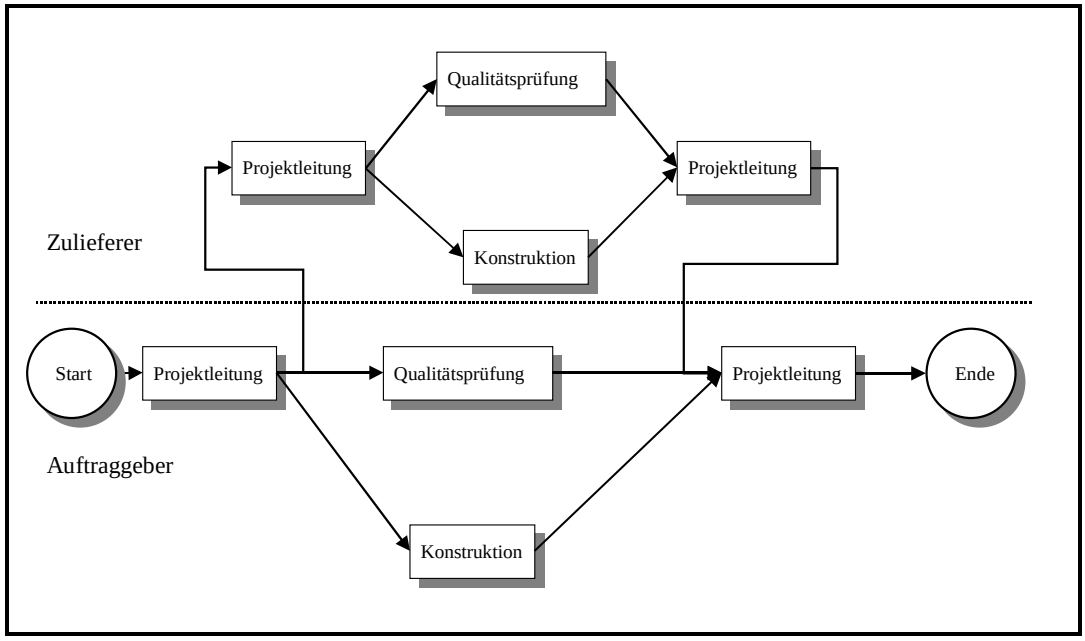


Abbildung 4-34: Beispiel für einen unternehmensübergreifenden Workflow ECO

Die Strukturierung der Ereignisbehandlung erfolgt in aller Regel durch Definition eines Workflow-Schemas. Entsprechend dieses Schemas werden Vorgänge von der in Abbildung 4-31 dargestellten Klasse zur Ereignisbehandlung ausgeführt. Diese Vorgänge und deren Verknüpfung miteinander sind in Datenbanktabellen abgelegt.

WorkflowID	VonKnoten	NachKnoten	Aufgabe	Vorgangsknoten	Status
1	StartKnoten	Designer	pruefen	1	1
1	Designer	Materialprüfung	pruefen	1	2
1	Designer	Qualitätsüberwachung	pruefen	1	2
1	Designer	Kostenüberwachung	pruefen	1	2
1	Materialprüfung	Projektleiter	pruefen	3	3
1	Qualitätsüberwachung	Projektleiter	pruefen	3	3
1	Kostenüberwachung	Projektleiter	pruefen	3	3
1	Projektleiter	EndKnoten		1	4

Abbildung 4-35: Beispiel einer Workflowdefinition

Abbildung 4-35 stellt eine Datenbanktabelle zur Abbildung eines ECO-Prozesses dar. Jedem Prozess wird eine WorkflowID zugeordnet. Diese WorkflowID wird den Workflowobjekten zugewiesen, wenn sich diese entlang eines definierten Workflows bewegen. In den Attributen *VonKnoten* und *NachKnoten* werden Rollen definiert, die beim Übergang in den jeweils nächsten Workflowstatus berücksichtigt werden müssen. Das Attribut *Vorgangsknoten* beinhaltet die Angabe über die Anzahl der vorhergehenden Knoten, damit am Beispiel des ECO-Prozesses der Projektleiter darüber informiert werden kann, wie viel positive Überprüfungen eingegangen sein müssen, damit dieser seine Aufgabe erledigen kann. Der jeweilige Workflowstatus wird durch das Attribut *Status* repräsentiert und in den Attributen des Workflowobjektes eingetragen (Abbildung 4-36).

DocID	WorkflowID	WorkflowStatus	Dateiname	Ersteller	ErstellDatum	.....
1	1	2	griff.sldprt	3	21.08.1999	
2	2	4	anlasser.prt	5	03.04.1998	
3	2	5	lichtmaschine.prt	1	19.05.2001	

Abbildung 4-36: Datenbanktabelle mit Dokumenten und deren WorkflowID mit Workflowstatus

4.4.2 Die Schnittstelle IAdmin

Die Konfiguration der oben beschriebenen Managementfunktionen muss Aufgabe der Schnittstelle IAdmin sein. Dazu gehört die Definition eines Freigabeschemas mit dazugehörigen Rollen- und Rechteverteilungen. Da die Rollenverteilung durch die Fine-Grained Security und deren Nutzung bei der Überprüfung der Statusübergange schon vorgegeben ist, besteht die eigentliche Aufgabe der Freigabekonfiguration in der Berarbeitung der Datenbanktabelle aus Abbildung 4-28. Ebenso muss der besprochene Workflow definiert und konfiguriert werden. Dafür müssen die bereits vorgestellten Datenbanktabellen bearbeitet werden. Eine administrative Bentutzerverwaltung, die bei typischen PDM-Systemen immer vorhanden ist, entfällt beim Einsatz des PDM-Kerns aufgrund der schon beschriebenen Benutzerverwaltung durch das Betriebssystem. Auf eine Diskussion der notwendigen Methoden soll an dieser Stelle verzichtet werden.

4.4.3 Die Schnittstelle IWork

Die Schnittstelle IWork beinhaltet, wie oben bereits angedeutet, Arbeitsfunktionen des PDM-Kerns. Dazu zählen das Aus- und Einchecken von Dokumenten, die damit unter Umständen verbundene Revisionierung und der in Kapitel 4.4.1 angesprochene Informationsaustausch. Zur Vervollständigung des bereits beschriebenen Konzeptes zur Umsetzung des Concurrent Engineering über Unternehmensgrenzen hinweg, soll dieser Informationsaustausch zuerst beschrieben werden.

Unternehmensübergreifender Informationsaustausch

Die parallele Entwicklung verschiedener Bauteile eines Produktes durch mehrere Unternehmen erfordert die in Abbildung 4-30 dargestellten Informationsflüsse. Der Informationsaustausch erfolgt bedarfsweise zu nicht definierten Zeitpunkten. Art und Umfang der zwischen den Kooperationspartnern auszutauschenden Informationen müssen von diesen vorher festgelegt werden. Keines der beteiligten Unternehmen wird alle produzierten Informationen für die am jeweiligen Projekt mitwirkenden Partner zugänglich machen. Es ist also eine genaue Definition der freizugebenden Informationen notwendig, die in aller Regel aus Textdokumenten, Tabellen, oder Zeichnungen bestehen und somit dokumentgebunden sind. Für die Freigabe von Text- oder Tabellendokumenten lässt sich relativ einfach eine Klassifizierung erarbeiten, nach der die Informationsfreigabe erfolgen kann. Bei technischen Zeichnungen oder 3D-Modellen gestaltet sich ein derartiges Vorgehen sehr viel schwieriger. Modelle sind während der Produktentwicklung die Hauptinformationsträger, in ihnen steckt das gesamte erarbeitete know-how eines Unternehmens.

In der Praxis haben sich für derartige Anwendungen sogenannte *light-weight-3D-Modelle*[20] als praktikable Lösung erwiesen. Ein gängiges Format für die Beschreibung solcher Modelle ist die *VRML*[21]. Mit Hilfe dieser können wichtige Informationen über die Abmessung und Gewicht eines Bauteils gewonnen werden. Bei Bedarf kann ein VRML-Modell zu DMU-Zwecken eingesetzt werden. Als positiv kann auch der Einsatz VRML-basierender Files im Web bewertet werden.

[20] light-weight-3D-Model (engl.), sinngemäß : vereinfachtes 3D-Modell

[21] VRML Virtual Reality Markup Language (engl.): Sprache zur Beschreibung virtueller Realität

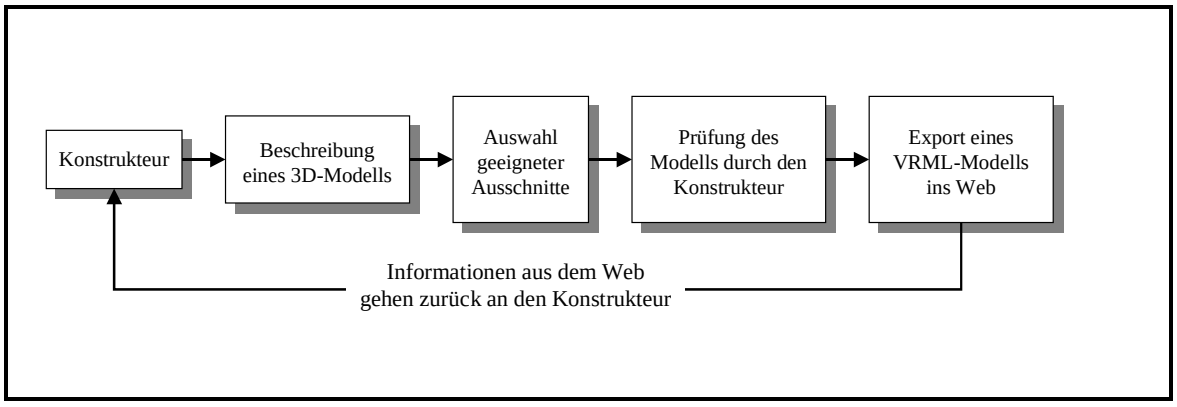


Abbildung 4-37: Informationsaustausch durch VRML-Modelle über das Web [8]

Fast alle gängigen Browser unterstützen ein Plug-In um derartige Files darzustellen und beispielsweise Abmessungen an den dargestellten Modellen vorzunehmen. Nach [8] lässt sich der Informationsaustausch zwischen Kooperationspartnern wie in Abbildung 4-37 dargestellt durchführen. Der Konstrukteur erzeugt aus einem 3D-Modell ausgewählte Ansichten in einem VRML-File. Zur Veröffentlichung des Modells wird die Datei auf einen Webserver gelegt. Dieser Schritt ist als Arbeitsfunktion *Publish* im PDM-Kern abgelegt. Nachdem der Konstrukteur eine VRML-Datei seines Modells erzeugt hat, ruft er diese Methode auf. Der PDM-Kern legt unter der entsprechenden Teilestruktur einen weiteren Dokumentenstammsatz parallel zum internen 3D-Modell an und versieht dieses mit einem Datenbankeintrag *published*. Gleichzeitig wird der VRML-File auf einen vorgesehenen Bereich des Webservers kopiert. In der Datenbank wird innerhalb des Dokumentenstammsatzes ein darauf verweisender Link eingefügt.

Als Pendant zu der beschriebenen Methode Publish, stellt der PDM-Kern eine Methode *GetPublished* zur Verfügung (Abbildung 4-38). Diese Methode ist durch die Fine-Grained Security für die Benutzung durch Kooperationspartner freigegeben. Um Informationen bezüglich eines Bauteils zu erlangen, instantiiert der Kooperationspartner den PDM-Kern über das WAN und ruft diese Methode unter Angabe des relevanten Bauteils auf. Innerhalb dieser Methode stellt der PDM-Kern alle veröffentlichten Metadaten der Dokumente dieses Bauteils zusammen und liefert diese als XML-Datei zurück. Darin enthalten sind auch die entsprechenden Links zu den eigentlichen Dokumenten, auf die dann über das Web zugegriffen werden kann.

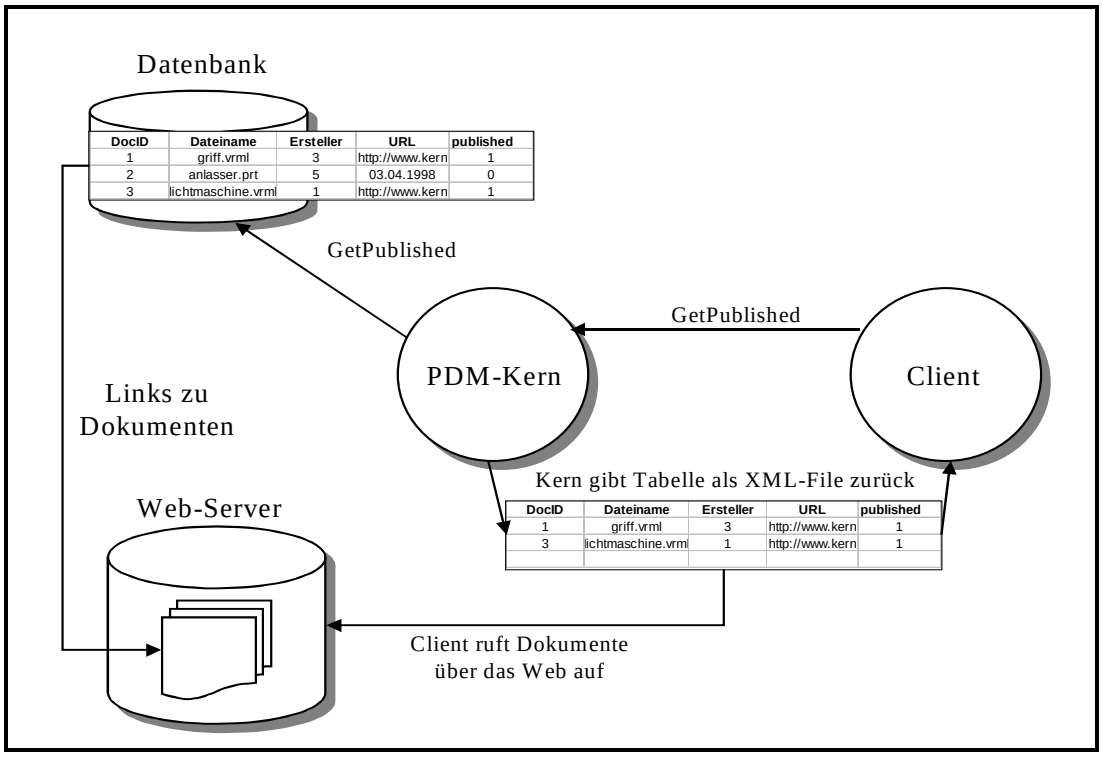


Abbildung 4-38: Zugriff auf veröffentlichte Dokumente über das Web

Die diskutierten Methoden Publish und GetPublished stellen Erweiterungen im Bereich der Arbeitsfunktionen auf dem Gebiet des Produktdatenmanagements dar. Nachfolgend soll exemplarisch eine typische Arbeitsfunktion vorgestellt werden, die demonstrieren soll, welche Technologien zu einer Umsetzung eingesetzt werden und welche Verbesserungen dadurch im Vergleich zu derzeitigen PDM-Systemen erzielt werden.

Ein- und Auschecken zu bearbeitender Dokumente

Eine von Mitarbeitern täglich mehrfach eingesetzte Arbeitsfunktion ist die des Ein- bzw. Auscheckens eines Dokuments. Bei diesem Vorgang wählt ein Bearbeiter das zu bearbeitende Dokument anhand von Metadaten aus der PDM-Datenbank aus und stößt einen Aufruf zur Bearbeitung an. Das im Filesystem befindliche Dokument wird zur Bearbeitung auf den lokalen Rechner des Bearbeiters kopiert und der dazugehörige Dokumentenstammsatz in der Datenbank allen anderen Mitarbeitern nur noch zum lesenden Zugriff zugänglich gemacht. Nach Bearbeitung des Dokuments wird dieses wieder zurück in das Filesystem kopiert und der Datensatz entsprechend den Änderungen aktualisiert. Dieses Szenario beherbergt trotz seines einfachen Ablaufs einige Stellen, an denen eine Inkonsistenz der Datenbasis verursacht werden kann, insbesondere bei Berücksichtigung eines Daten- und Dokumentenzugriffs über das WAN.

Der Vorgang des Auscheckens, im folgenden auch als Transaktion bezeichnet, verlangt die sequentielle Abarbeitung dreier Schritte:

- lesen des Dokuments aus dem Filesystem und Übertragung über das LAN/WAN,
- sperren des Datensatzes und
- schreiben des Dokuments auf den lokalen Rechner.

Läuft dieser Vorgang nicht reibungslos ab, d.h. fällt entweder einer der beteiligten Rechner oder das Netz aus, so führt das zu Inkonsistenz des Datenbestandes. Fällt beispielsweise das Netzwerk nach der Sperrung des Datensatzes aus, so kann das dazugehörige Dokument nicht zum Arbeitsrechner übertragen werden. Eine Bearbeitung und späteres Rücksichern, sowie die Entsperrung des Datensatzes ist in diesem Fall nicht mehr möglich. Derzeitige PDM-Systeme bieten an dieser Stelle keine ausreichende Sicherheit gegen derargtige Ausfälle, so dass häufig externe Programme zur Sysnchronisation des tatsächlichen Datenbestandes mit dem in der Datenbank abgebildeten Datenbestand eingesetzt werden müssen.

Aus diesem Grund sollen die o.g. Transaktionsschritte durch Resource Manager durchgeführt werden. Resource Manager sind Softwarekomponenten, deren Aufgabe in der Durchführung programmierter Aufgaben besteht. Nach erfolgreicher oder gescheiterter Durchführung der übertragenen Aufgaben wird ein entsprechendes Ergebnis zurückgegeben.

COM+ bietet für die Erstellung beschriebener Resource Manager einen *Compensating Resource Manager (CRM)*[22] Service. Dieser unterstützt die Erstellung eigener Resource Manager durch Implementierung geforderter Aufgaben in COM+ -Komponenten. Der PDM-Kern, als COM+ -Komponente ausgelegt, wird also entsprechend seiner auszuführenden Aufgaben zerlegt und durch den CRM in seiner Tätigkeit als Resource Manager unterstützt. Die Konfiguration lässt sich durch die in Abbildung 4-39 dargestellte Dialogbox erreichen.

[22] Compensating Resource Manager (engl.): überwachender, ausgleichender Resourcenverwalter, von Microsoft® zur Verfügung gestellter Dienst zur Erstellung eigener Resource Manager

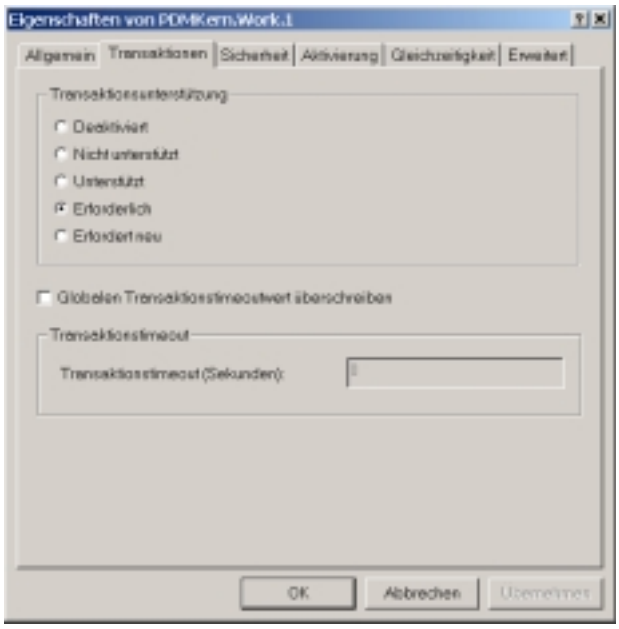


Abbildung 4-39: Konfiguration des PDM-Kerns zur Verwendung von Transaktionssicherheit

Ein *Transaction Coordinator*[23] überwacht die Ergebnisse der von den Resource Managern ausgeführten Tätigkeiten, unter Verwendung eines 2-phasigen Übertragungsprotokolls. Die in diesem Protokoll auszuführenden Schritte sind wie folgt definiert:

- Der Transaction Coordinator beauftragt den ersten Resourcen Manager das geforderte Dokument an den Clientrechner zu übertragen.
- Der Transaction Coordinator beauftragt den zweiten Resourcen Manager den entsprechenden Datensatz zu sperren.
- Der Transaction Coordinator beauftragt den dritten Resourcen Manager das Dokument auf dem Clientrechner zu überprüfen.
- Sind alle Schritte erfolgreich ausgeführt, so sendet der Transaction Coordinator das Signal *commit* an alle Resource Manager. Der Vorgang ist damit abgeschlossen, die Resource Manager stehen für andere Tätigkeiten bereit.
- Gibt mindestens einer der Resource Manager ein negatives Ergebnis zurück, so sendet der Transaction Coordinator den Befehl *Rollback*. Mit diesem Befehl werden alle Resource Manager dazu aufgefordert, die zuvor ausgeführte Tätigkeit rückgängig zu machen.

[23] Transaction Coordinator (engl.): Abwicklungskoordinator

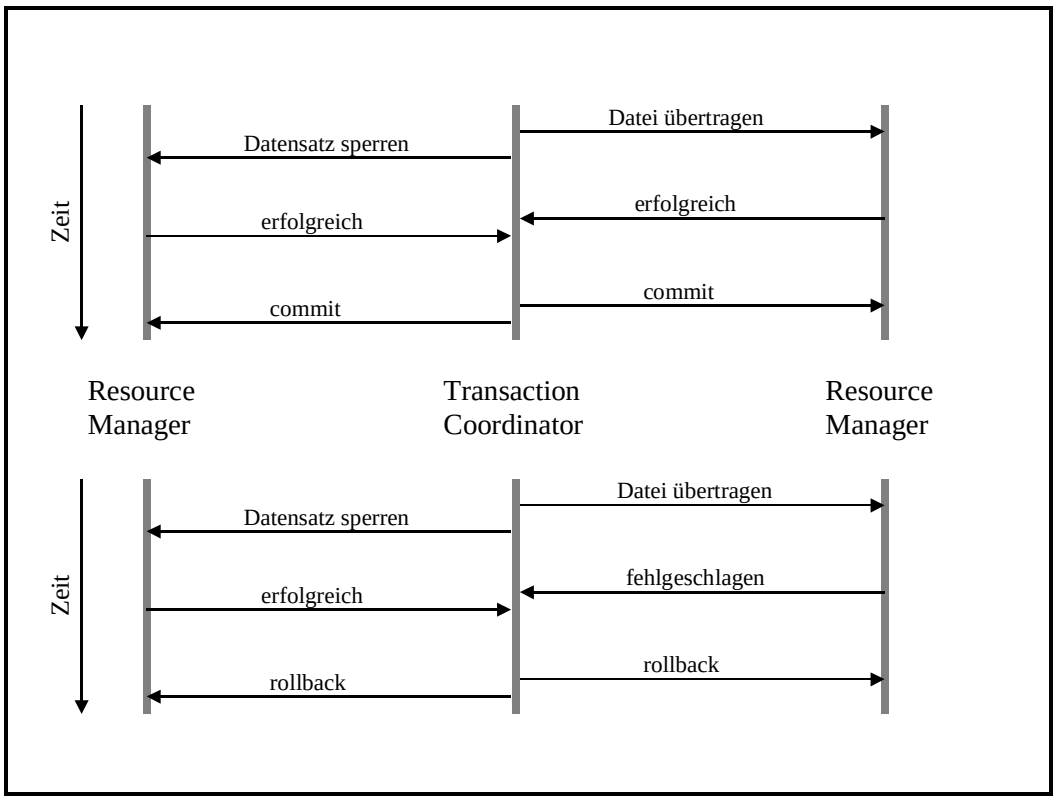


Abbildung 4-40: 2-phasiges Übertragungsprotokoll. Erfolgreiche (oben) und fehlgeschlagene Durchführung (unten).

Zur besseren Darstellung des Ablaufs sind in Abbildung 4-40 nur zwei Resourcen Manager berücksichtigt. Der auf der linken Seite dargestellte Manager führt die Transaktionsschritte auf der Datenbankseite aus, der rechtsseitig dargestellte Manager ist für die Dateiübertragung zuständig. Der in der Mitte befindliche Transaktionskoordinator sendet entsprechende Befehle und wertet die Ergebnisse der Arbeitschritte aus. Der Vorgang des Eincheckens läuft vergleichbar ab.

Einsatz eines geeigneten Fileservers

Die beschriebene Arbeitsfunktion des Ein- bzw. Auscheckens von Dokumenten greift auf ein Filesystem zurück, in dem alle zur PDM-Installation gehörenden Dateien abgelegt sind. Dieses Filesystem stellt einen Bereich dar, der gegen den manuellen Zugriff geschützt sein muss. In derzeitigen PDM-Systemen werden sogenannte File- oder Vaultserver eingesetzt. Nicht jeder dieser Server ist jedoch für den Einsatz im WAN geeignet. Der PDM-Kern soll aber auch für Mitarbeiter entfernter Standorte alle Dienste zur Verfügung stellen. So muss im Falle des Ein-

und Auscheckens eines Dokuments eine Datei über das WAN zum Client transportiert werden. Als Lösung für diese Aufgabe ist im Konzept des PDM-Kerns die Verwendung eines ftp[24]-Servers vorgesehen. Dem ftp-Server wird vom Administrator ein Plattenbereich zugeordnet, indem sich alle Dateien der PDM-Installation befinden. Schreib- und Leserechte auf diese Dateien besitzt nur der PDM-Kern, der auch als Benutzer des ftp-Servers eingetragen ist. Damit kann unbefugter Zugriff verhindert werden. Beim Auschecken eines Dokuments meldet sich der vom Resource Manager verwaltete Teil des PDM-Kerns am ftp-Server an und fordert diesen zur Übertragung der dazugehörigen Datei unter Angabe des Zielrechners auf.

```
HINTERNET hInternetSession;      // Internetverbindung
 hFTPSession;                    //FTP-Verbindung

//oeffnen der Internetverbindung
hInternetSession = InternetOpen(„PDM-Kern“, INTERNET_OPEN_TYPE_PROXY,“PDM“,NULL,0)

//Verbindung zum FTP-Server
hFTPSession = ::
InternetConnect(hInternetSession,ftp://ftp.pdmkernel.com,“PDMUser“,“PDMPassword“,INTERNET_SERVICE_FTP,
0,0);

//uebertragen der Datei
bResult = ::FtpGetFile( hFTPSession,“ftp://ftp.pdmkernel.com/pdm/lichtmaschine.prt“,“\\ikbpc4\pdmwork“, FALSE,
FILE_ATTRIBUTE_NORMAL, _TRANSFER_TYPE_BINARY,  0);

//schliessen der Verbindung
InternetCloseHandle(hFileConnection);
InternetCloseHandle(hFTPSession);
```

Abbildung 4-41: Beispiel für eine Dateiübertragung via InternetConnection

Abbildung 4-41 stellt eine Dateiübertragung über eine Internetverbindung dar. Zuerst wird eine Internetverbindung vom MFC-Typ *HINTERNET* hergestellt. Danach meldet sich der PDM-Kern über diese Verbindung am ftp-Server unter Angabe des Benutzernamens und des Passworts an. Die Methode *FtpGetFile* kopiert die geforderte Datei vom Fileserver auf einen Arbeitsbereich des Clients. Dort kann sie bearbeitet werden und später über eine äquivalente Methode zurück auf den ftp-Server kopiert werden.

[24] ftp = File Transport Protocol (engl.): Protokoll zur Dateiübertragung

4.4.4 Die Schnittstelle IXML

Der bisher diskutierte PDM-Kern besitzt Methoden zur Administration, Managementfunktionen, sowie Arbeitsfunktionen. Noch nicht angesprochen wurden Möglichkeiten zum Datenaustausch zwischen dem PDM-Kern und anderen Applikationen *(application-to-application)*, sowie zwischen dem PDM-Kern und dem Benutzer *(application-to-human)*. Es kann nicht die Aufgabe des PDM-Kerns sein, eine Benutzeroberfläche zur Verfügung zu stellen (vgl. Kap. 4.2.2).

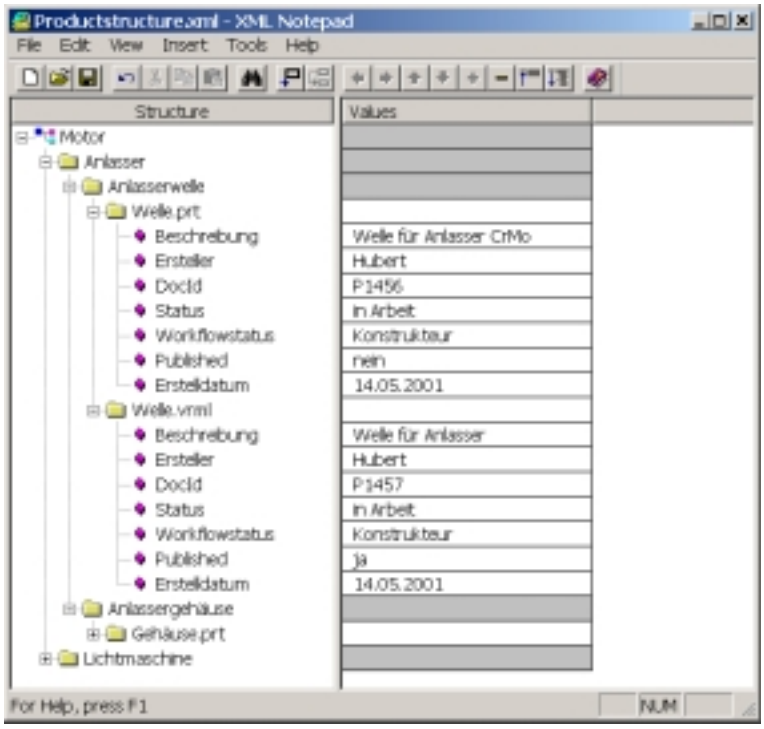


Abbildung 4-42: Editor für XML-Dateien.

Die Schnittstelle IXML soll Applikationen auf der Präsentationsebene Methoden bieten, die zur Gewinnung darzustellender Informationen genutzt werden können. Grundsätzlich besteht die Möglichkeit eines Datenaustausches zwischen Anwendungen auf Basis von ADO-Recordsets. Die Verwendung derartiger Datensätze wurde im Datenbankkern diskutiert und eignet sich generell auch für die Verwendung als Rückgabetyp des PDM-Kerns zu Benutzeroberflächen. In Bezug auf den webbasierten Datenaustausch soll jedoch auf den Einsatz derartiger Recordsets verzichtet werden, da aufwendige Individualprogrammierung notwendig ist, um den Inhalt der Recordsets darzustellen.

Als zukünftige Standardlösung für den Austausch ausgewählter Datensätze über das WAN/LAN wird sich die eXtensible Markup Language (XML) durchsetzen [15]. XML-Dateien sind nicht nur in der Lage strukturelle Gliederungen wiederzugeben, sondern auch semantische Verknüpfungen abzubilden. Durch derartige Verknüpfungen ist es möglich, mit Hilfe beliebiger Editoren einen Datensatz zu visualisieren. Abbildung 4-42 stellt einen Editor zur Bearbeitung von XML-Dateien dar. Zu erkennen ist hier vor allem die hierarchische Gliederung der Informationen, die nur auf Angaben innerhalb des XML-Dokuments basiert. XML definiert den Inhalt des Datensatzes, nicht die Art der Präsentation. Wie diese also dargestellt werden, hängt nicht von der XML-Datei, sondern vom verwendeten Browser ab. Zudem ist es möglich Informationen aus XML-Dateien zu extrahieren und in verschiedenste Präsentationsformen zu konvertieren. So ist die Konvertierung einer XML-Datei in eine HTML-Datei ein durchaus denkbarer Weg zur Datenpräsentation. Microsoft® stellt im InternetExplorer derartige Konvertierungen und Präsentationen zur Verfügung.

Durch den Datenaustausch auf XML-Basis vergrößert sich der mögliche Einsatzbereich des PDM-Kerns bedeutend. Mitarbeiter kooperierender Unternehmen können unter Verwendung eines Standardeditors alle Informationen des jeweils anderen Unternehmens sichtbar machen, ohne über eine angepasste Benutzeroberfläche zu verfügen. Der Editor passt sich entsprechend der semantischen Beziehungen selbstständig an und ermöglicht das Navigieren und Suchen innerhalb der präsentierten Informationen.

Die im PDM- und Datenbankkern verwendete Datenbankschnittstelle ADO ermöglicht die Ein- und Ausgabe von Recordsets auf Basis von XML-Dateien. Abbildung 4-43 stellt die Speicherung eines Recordsets in eine XML-Datei unter Verwendung der Methode *Save* dar. Entsprechend lassen sich ein Recordsets öffnen und in damit Aktualisierungen der Datenbank vornehmen. Dadurch können Manipulationen am Datenbestand durch Veränderung des XML-Files vorgenommen werden.

```
HRESULT hr = rst->Open(sql, cnn.GetInterfacePtr(), adOpenDynamic, adLockOptimistic, adCmdText);
        if(SUCCEEDED(hr))
                rst->Save("c:\work\anlasser.xml",adPersistentXML);
        return hr;
```

Abbildung 4-43: Speichern eines Recordsets in eine XML-Datei

5 Einsatz des TPIS

Die programmiertechnische Umsetzung des Konzeptes in seiner Gesamtheit wird ca. 3 Mannjahre erfordern. Eine genauere Abschätzung lässt sich erst nach Erstellung eines Pflichtenheftes machen. Stellvertretend für das gesamte **TPIS** ist die Realisierung des DB-Kerns erfolgt, dessen implementierte Funktionen positive Rückschlüsse auf die Realisierung des PDM-Kerns zulässt. Zur Beurteilung des Nutzens des **TPIS** soll der Einsatz exemplarisch vorgestellt werden.

5.1.1 Das TPIS im COM-Softwarebus

Die komponentenbasierte Entwicklung der letzten Jahren hat zu immer mehr Softwaremodulen geführt, die im Bereich des Betriebssystems Microsoft Windows® über eine standardisierte API angesprochen und so als Applicationserver für verschiedenste Anwendungen Dienste bereitstellen können. Darunter fallen beispielsweise die CAD-Systeme AutoCAD® und SolidWorks®, Projektplanungsprogramme wie Microsoft Project®, Standardofficeanwendungen wie MicrosoftOffice® und im Bereich des ERP existiert ein von SAP® entwickeltes Interface für das System R/3®. Die zur Verfügung gestellten COM-Interfaces oben genannter Anwendungen werden jedoch bislang nur unzureichend genutzt. Gründe dafür sind die noch recht junge COM-Technologie, aber auch der nicht ganz unproblematische Einsatz herkömmlicher COM-Komponenten über Rechner- und Netzwerkgrenzen hinweg. Bislang war dafür ein Transactionserver notwendig, der parallel zur COM-Runtime als MTS-Runtime existiert und im wesentlichen aus den Dateien mtxex.dll und mtx.exe besteht.

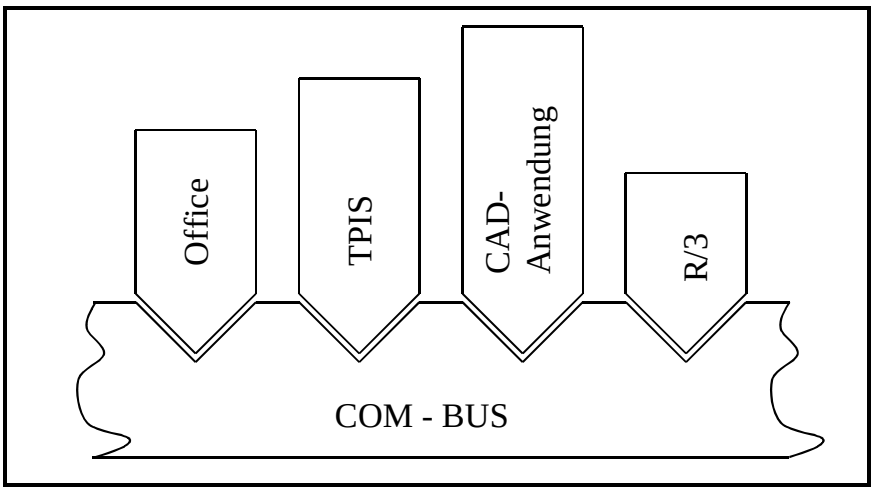


Abbildung 5-1: Das TPIS im COM-Softwarebus

Erst die Einführung der COM+-Technologie macht die Verwendung von Softwarekomponenten über das Netz attraktiv, da aus Entwicklersicht auf den Transactionserver keine Rücksicht mehr genommern werden muss. Auch die neu zur Verfügung gestellten Dienste bezüglich der Netzwerksicherheit sorgen für eine flexible Skalierbarkeit derartiger Softwarekomponenten.

Aus diesen Gründen ist das **TPIS**, wie bereits mehrfach erwähnt, als COM+-Komponente ausgelegt, kann aber über den sogenannten COM-Bus mit bereits vorhandenen Anwendungen kommunizieren (Abbildung 5-1). Als Pendant zum Hardwarenetz verläuft der COM-Bus als Softwarebus durch das unternehmensweite Netzwerk und steht damit allen daran angeschlossenen Rechnern zur Verfügung. Dadurch stehen entsprechend seiner Rollenzuteilung jedem Mitarbeiter alle für ihn notwendigen Dienste, unabhängig von der Abteilung in der er tätig ist, zur Verfügung (Abbildung 5-2).

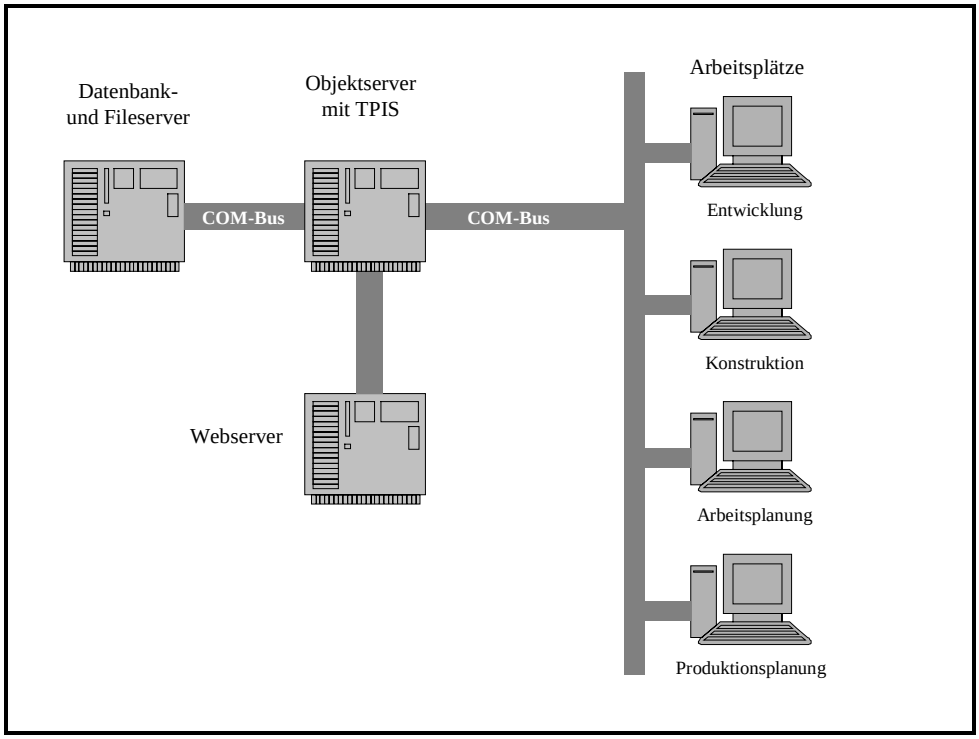


Abbildung 5-2: Der COM-Softwarebus im unternehmensweiten Netz

Das wiederum unterstützt den Gedanken des durchgängigen, virtuellen Produktes. Dieses wird auf dem Datenbankserver durch das Datenmodell abgebildet und durch das **TPIS** erweitert bzw. modifiziert.

Eine PDM-Installation jeglicher Art entfällt auf allen Client-Rechnern, wenn für die Datenvisualisierung beispielsweise ein Standardbrowser genutzt wird. Damit können relevante Informationen auch von Vertriebsleuten über das WAN abgerufen und bearbeitet werden, ohne das dafür ein leistungsstarker Rechner zur Verfügung steht. Vor dem Hintergrund immer größerer Bedeutung sogenannter PDAs erweitert das die Akzeptanz der Benutzer.

5.1.2 Schnittstellen zu CAD-Systemen

Die entfernte Nutzung aller Dienste des **TPIS** ermöglicht nicht nur Vertriebsleuten den Datenzugriff, sondern lässt auch Konstrukteure über das WAN mit der entfernten PDM-Installation kommunizieren.

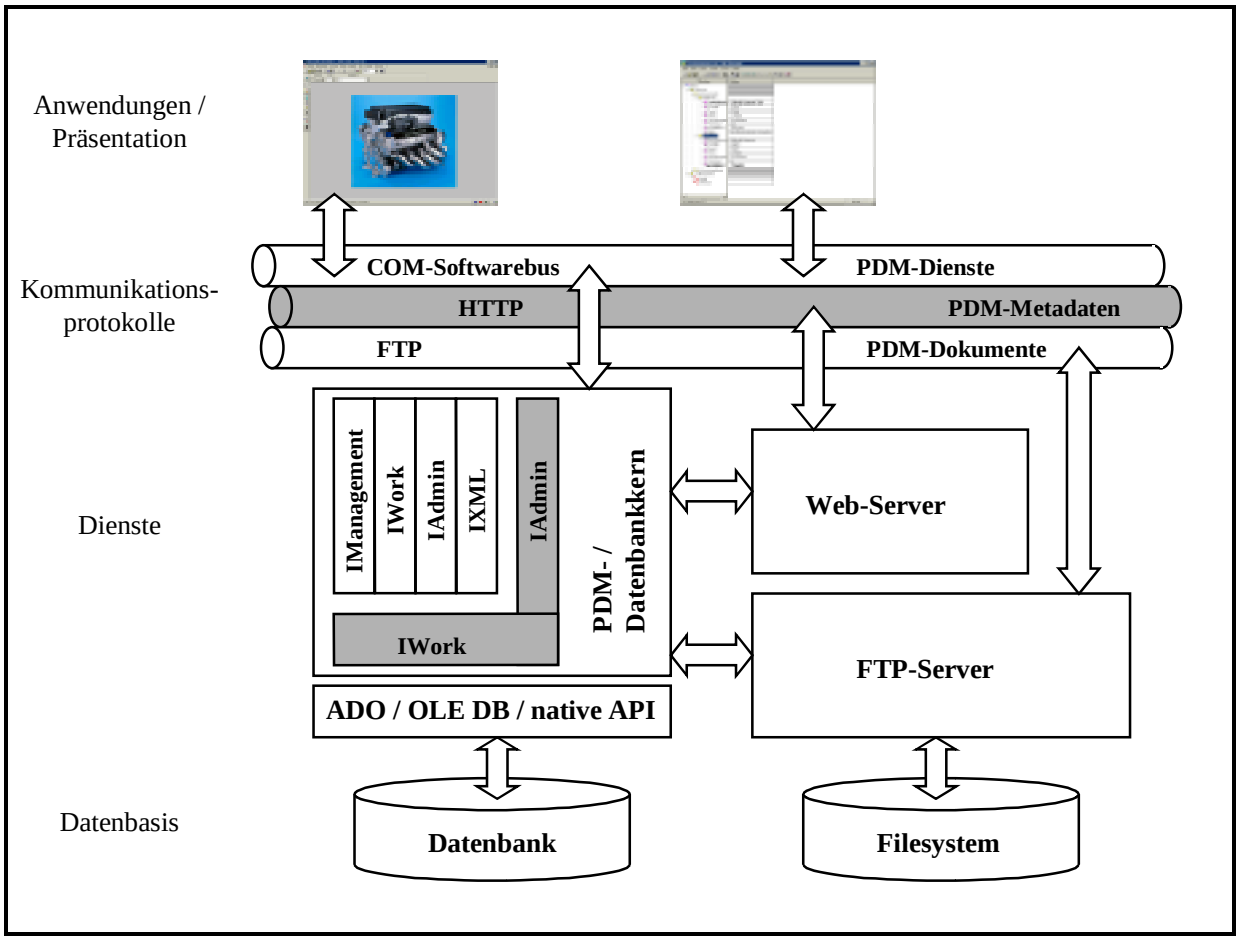


Abbildung 5-3: Komplette Umgebung des TPIS

Durch Dienstleister angefertigte Auftragsarbeit muss nicht mehr von Hand in die eigene PDM-Installation eingepflegt werden, sondern es stehen dem Auftragnehmer alle notwendigen Dienste über Standortgrenzen hinweg zur Verfügung. Auch unternehmenseigene Konstrukteure können standortunabhängig arbeiten. Das dazu notwendige CAD-System muss auf dem Arbeitsrechner installiert und mit einer PDM-Schnittstelle gekoppelt sein. Die CAD-PDM-Schnittstelle kann über eine bestehende Internetverbindung genutzt werden. Diese Schnittstelle besteht in der Regel aus einer Erweiterung der CAD-Benutzeroberfläche und einer in den Prozessraum des CAD-Systems zu ladenden dynamischen Bibliothek, die ihrerseits die API des PDM-Systems nutzt, um Arbeitsfunktionen anzustoßen. Möchte ein Benutzer beispielsweise eine CAD-Zeichnung bearbeiten, so nutzt er über die Erweiterung der Benutzeroberfläche Funktionen der geladenen DLL, die wiederum durch API-Aufrufe das PDM-System dazu veranlasst das entsprechende Dokument auszuchecken und auf den lokalen Arbeitsrechner zu kopieren. Bislang erwies sich der Einsatz der API-Aufrufe über Netzwerkgrenzen als problematisch, die Nutzung des COM-Bus macht derartige Aufrufe jedoch möglich. Abbildung 5-3 stellt schematisch die Umgebung des **TPIS** dar. Fremdanwendungen greifen über den COM-Bus via RPC (Abbildung 5-4) auf Dienste des **TPIS** zu und können diese standortübergreifend verwenden.

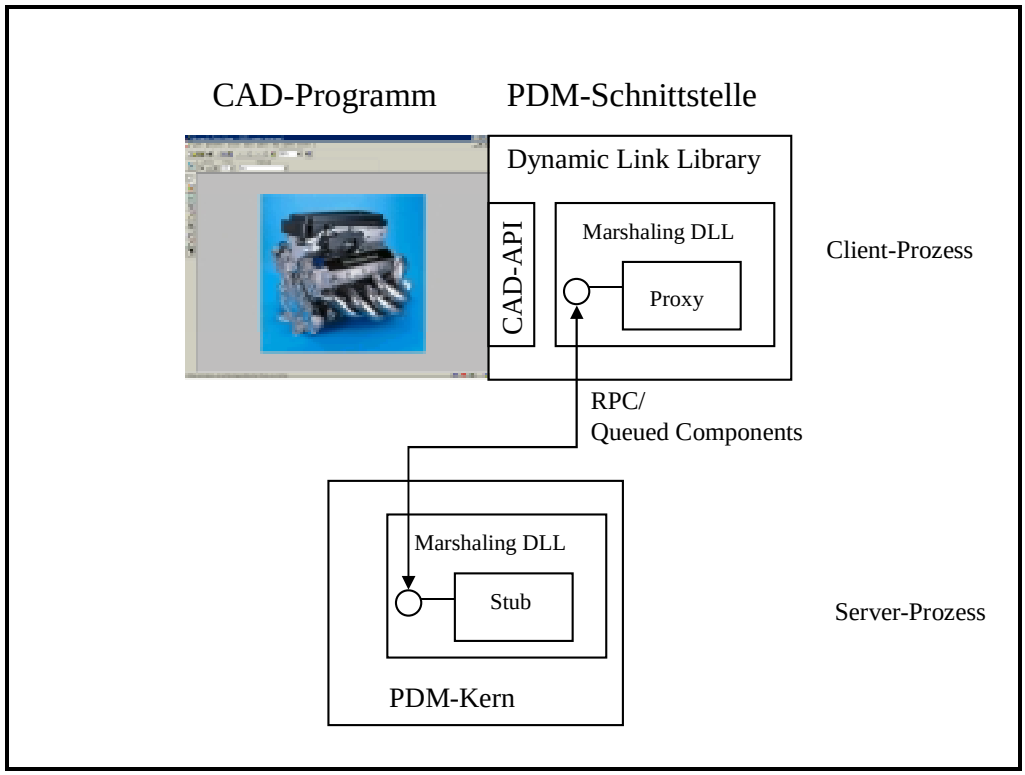


Abbildung 5-4: Verwendung eines entfernten TPIS in einer CAD-Schnittstelle durch Marshaling des PDM-Kerns

Metadaten werden über das http zum Client transportiert und dort in einem Standardbrowser sichtbar gemacht. Ausgewählte Dokumente können bei ausreichender Berechtigung des Users ein- und ausgecheckt werden. Dazu steht ein ftp-Server zur Verfügung, der über das ftp die entsprechenden Dateien auf den Arbeitsplatzrechner kopiert.

5.1.3 Schnittstelle zu ERP-Systemen

Der oben skizzierte Weg zur Kopplung eines CAD-Programms an das **TPIS** stellt nur eine mögliche Anbindung eines Erzeugersystems an den die PDM-Installation dar. Eine immer mehr an Bedeutung gewinnende Datenschnittstelle ergibt sich bei der Koexistenz eines PDM- und ERP-Systems. Datenredundanz, in der Regel unerwünscht, wird zu definierten Zeitpunkten bewusst hergeleitet, indem Informationen des technischen Informationssystems ins kommerzielle Informationssystem transferiert werden.

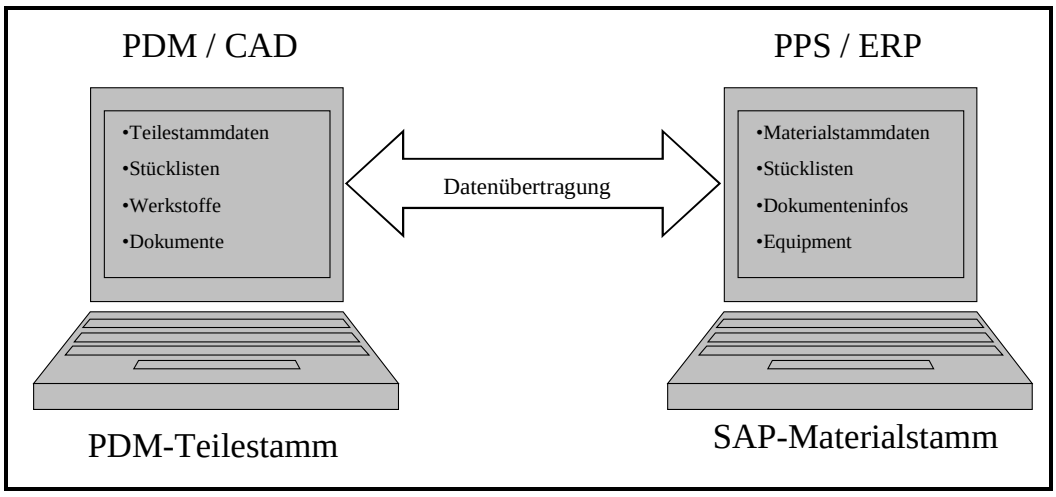


Abbildung 5-5: Datenübertragung zwischen PDM/CAD und ERP/PPS

Bei fast allen Kopplungen zwischen PDM und ERP werden Teilestammdaten bzw. Materialstammdaten zwischen den Systemen ausgetauscht (Abbildung 5-5). Dabei ist es unerheblich, in welchem der beiden Systeme die Daten generiert wurden, wichtig ist nur, dass ab einem vorher definierten Entwicklungszustand die Informationen in beiden Systemen vorliegen. Die Art und Weise der Datengenerierung hängt stark von den einzelnen Prozessschritten eines Unternehmens ab. Für die nachfolgend exemplarische Darstellung eines Datenaustausches wird davon ausgegangen, dass ein Materialstammsatz im ERP-System generiert und später ins PDM übertragen wird. Dazu soll das häufig verwendete System SAP R/3® herangezogen werden.

SAP® hat mit dem SAP COM+ Component Connector (COM+ CC), der mit dem Remote Function Call SDK ausgeliefert wird, eine Technologie entwickelt, um universelle Schnittstellen auf der Plattform von Microsoft Windows-Betriebsystemen zu unterstützen. Damit ist das gesamte System SAP R/3® über den COM-Bus erreichbar und kann so dem **TPIS** als Anwendungsserver zur Verfügung stehen. Der COM+ CC stellt BAPIs[25] und Funktionsmodule als gewöhnliche COM+-Objekte dar. Er bildet ABAP[26] Datentypen auf COM+-Datentypen ab, beispielsweise ABAP-Tabellen auf ADO-Recordsets. Der COM+ CC soll genutzt werden, um einen Materialstammsatz, repräsentiert durch ein Recordset, aus dem R/3-System auszulesen.

```
Sub ReadEntryFromTableMaterial()
.....
.....
Set SAPObj = CreateObject(„SAP.Functions“)
SAPObj.Connection.Destination=„R/3“
SAPObj.Connection.User=„Wiwi“
SAPObj.Connection.Client=„007“
SAPObj.Connection.Language=„DE“
SAPObj.Connection.Logon
.....
Set SAPObjFunc= SAPObj.Add(„RFC_MATERIAL_GET“)
SAPObjFun.exports(„I_ID“) = „DOK3256“
Set SAPRecord = SAPObjFunc.tables(1)
.....
.....
```

Abbildung 5-6: BASIC-Code zum Auslesen eines Materialstammsatzes über den COM+ CC

In Abbildung 5-6 sind die wichtigsten Auszüge aus dem dazu notwendigen Code dargestellt. Zuerst wird ein SAP-Object angelegt, dem in den folgenden Zeilen Benutzer- und Standortinformationen zugewiesen wird. Nach erfolgreicher Anmeldung wird ein ABAP-Baustein mit dem Namen *RFC_MATERIAL_GET* aufgerufen. Dieser Baustein nimmt eine I_ID als Parameter entgegen, um damit eine SQL-Anweisung auszuführen. Das Ergebnis der SQL-Abfrage wird ABAP-intern in eine Tabelle geschrieben, die wiederum mit dem letzten Befehl des BASIC-Codes in ein ADO-Recordset übertragen wird. Ab dieser Stelle stehen die so gewonnen Informationen dem **TPIS** zur Verfügung, der daraus beispielsweise einen Datenbankeintrag generieren kann.

[25] BAPI Business API (engl.): Schnittstelle zur Bereitstellung im R/3-System vorhandener Geschäftsprozesse

[26] ABAP : Programmiersprache des R/3-Systems

6 Zusammenfassung und Ausblick

Das in dieser Arbeit erstellte Konzept und die teilweise Realisierung eines **TPIS** hat gezeigt, das der Einsatz des COM+ - Modells wesentlich zur Erreichung der gestellten Forderungen beiträgt. So gibt beispielsweise der Einsatz von ADO die Möglichkeit, dass mehr Datenbanksysteme für den Einsatz in der PDM-Installation geeignet sind als bei jedem anderen PDM-System. Der COM-Softwarebus verbindet alle aktuellen Erzeugersysteme miteinander, so dass eine durchgängige Unterstützung des virtuellen Produktes über Abteilungsgrenzen hinweg möglich ist. Um dem Gedanken des Concurrent Engineering zu folgen, wurde auf die Implementierung eines unternehmensübergreifenden Workflows Wert gelegt. Auch hier konnte COM+ durch entsprechende Event-Handler Technologien zur Verfügung stellen, die eine Realisierung sehr vereinfachen. Aufgrund seiner universellen Einsetzbarkeit ist COM+ jedoch auch für die Aufgaben klassischer PDM-Funktionen geeignet. Lediglich die eingeschränkte Plattformunabhängigkeit gestaltet einen Einsatz in heterogenen Netzwerken zur Zeit etwas problematisch.

Durch seinen modulartigen Aufbau sind für das **TPIS** Erweiterungen in jeder Hinsicht möglich. Seine leicht zu erweiternde Datenstruktur und die logische Trennung zwischen Datenbank- und PDM-Kern ermöglichen die Verwendung auch als Komponente zukünftiger Datenverarbeitungssysteme. Dazu tragen auch die verwendeten Standards wie XML und http bei, die eine Kommunikation mit zukünftigen Systemen sehr vereinfachen. Für die komplette Realisierung sollte der von Microsoft neu vorgestellte Standard *Microsoft® .NET* berücksichtigt werden, der die Internetintegration sehr vereinfachen kann.

Die vorliegende Arbeit leistet einen Beitrag zur Diskussion über den Einsatz und die Verwendung technischer Informationssysteme im Produktentstehungsprozess. Dabei wird deutlich, dass auch bei jeder Form der Automatisierung ein solches System nur ein unterstützendes Hilfsmittel sein kann. Das virtuelle oder reale Produkt wird nach wie vor vom Menschen geschaffen, doch die gegenseitige Ergänzung führt zu einer Verbesserung des Prozesses.

7 Anhang

7.1 Literatur

[1] Schöttner, J.: Produktdatenmanagement in der Fertigungsindustrie, Hanser Verlag München, 1999

[2] Spur, G.; Krause, F.: Das virtuelle Produkt, Hanser Verlag München, 1997

[3] Spur, G.: Fabrikbetrieb. Carl Hanser Verlag, München, Wien 1994

[4] Schick, F.: Kommunikationsmethoden im Unternehmen – Dokumentenmanagement, Archivierung und Workflow. In: CADWORLD 06-1999.

[5] Golla, K.-M.; Heinz, S.; Iselborn, B.; Wüseke, P.: CAD-Geometrie alleine reicht auf Dauer nicht. In: Konstruktion 11/12-2000

[6] Wiendahl, H.P.: Betriebsorganisation für Ingenieure. 4. Auflage, Carl Hanser Verlag, München, Wien 1996

[7] Stroustrup, B.: Die C++ Programmiersprache, 2. Auflage. Verlag Addison-Wesley, 1998

[8] Meyers, S.: Effektiv C++ programmieren, 3.Auflage. Verlag Addison-Wesley, 1998

[9] Josuttis, N.: Objektorientiertes Programmieren in C++. Verlag Addison-Wesley, 1994

[10] Sarret, W.: Visual C++ 6 Database Programming Tutorial, WROX-Press, 1998

[11]Gordon, A.: The COM and COM+ Programming Primer, 2000

[12] Eicker, S.; Gerhards, Th.: COM/DCOM Microsofts Standard für Componentware, Seminararbeit Universität Essen, Witschaftsinformatik, Februar 1999

[13] Microsoft Developer Network

[14] Pott, O.: XML Praxis und Referenz, Verlag Markt und Technik, 2. und erweiterte Auflage, 2000

[15] Rezayat, M.: The Enterprise-Web portal for life-cycle support. In: Computer-Aided Design 32, September 1999

[16] Eddon, G.; Eddon, H.: Inside Distributed COM, Microsoft Press, 1998

[17] Vajna, S.: Die neue Richtlinie VDI 2219. Praxiserprobte Hinweise zu Einführungsstrategien und Wirtschaftlichkeit von EDM/PDM-Systemen. In: Konstruktion, März 2000.

[18] Rezayat, M.: Knowledge-Based product development using XML and KCs. In: Computer-Aided Design 32, September 1999

[19] Lobeck, F.: Konzept für ein objektorientiertes, bereichsübergreifendes Dokumenteninformations- und –verwaltungssystem, Shaker Verlag, 1999

[20] Mechlinski, Th.: MELange- ein Programmsystem zur Verwaltung von Konstruktions- und Berechnungsdaten, VDI Verlag Düsseldorf, 1998

[21] Kunzmann, U.; Löbig, S.; Benn, W.; Dube, H.: Überlegungen zu ISO 10303 (Standard for the Exchange of Product Model Data – STEP) Datenaustauschformat oder Modellierungsbasis. Verlag unbekannt , 1997

[22] Zechmann, B.: Adventures in Space. In: Automobil Industrie 05-2000.

[23] Steck, R.: Grundlagen Enterprise Resource Planing - Ressourcen richtig nutzen. In: CADWORLD 06-1999.

[24] Shephered, G.; King, B.: Inside ATL. Microsoft Press Deutschland, 1999

[25] Smart Solutions Ltd.: SmartTeam API Programmers Guide. Smart Solutions Ltd., 2000

Als Zuarbeit zu dieser Dissertation wurden vom Verfasser die folgenden Studien- und Diplomarbeiten vergeben und betreut:

Chilah, A.: Konzeption und Implementierung eines objektorientierten Freigabeschemas für PDM-Systeme.

Wang, H.: Konzeption eines objektrelationalen Datenmodells zur Abbildung typischer PDM-Funktionen.

Chilah, A.: Konzeption und Implementierung eines COM-basierten Datenbankkerns zur Unterstützung typischer Geschäftsprozesse in PDM-Systemen.