IKE Institut für Kernenergetik
und Energiesysteme

Entwicklung eines komponentenbasierten Frameworks zur Bewertung heiz- und raumlufttechnischer Anlagen

Darko Sucic

November 2002 IKE 4-156

Universität Stuttgart

IKE Institut für Kernenergetik
und Energiesysteme

Entwicklung eines komponentenbasierten Frameworks zur Bewertung heiz- und raumlufttechnischer Anlagen

Von der Fakultät der Energietechnik der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Darko Sucic

geboren in Sarajewo/Bosnien und Herzegovina

Hauptberichter:	Prof. Dr.-Ing. habil. F. Schmidt
Mitberichter:	Prof. Dr.-Ing. R. Rühle

Tag der Einreichung:	20. November 2001
Tag der mündlichen Prüfung:	14. November 2002

ISSN 0173-6892

Universität Stuttgart

November 2002 IKE 4-156

Kurzfassung

In der vorliegenden Arbeit wird ein komponentenbasiertes Framework zur energetischen Bewertung heiz- und raumlufttechnischer Anlagen entwickelt. Heiz- und raumlufttechnische Anlagen können mit der Hilfe der Methode der Bedarfsentwicklung anhand ihres Energieaufwandes während der Planung und des Betriebes bewertet werden. Die Bewertung erfolgt durch:

- die Feststellung des Energiebedarfs und der ihn zu beeinflussenden Größen mit Hilfe einer Gebäudesimulation
- die Feststellung des Aufwands, der eine Anlage zur Befriedigung des Bedarfs benötigt, mit Hilfe einer Anlagesimulation
- den kontinuierlichen Vergleich der durch die Messung gewonnenen Raum- und Anlagezustände mit simulierten Werten mit Hilfe einer gekoppelten Gebäude- und Anlagesimulation.

Dem komponentenbasierten Framework liegt eine Architektur zugrunde, welche sich durch eine klare Trennung der Komponente Gebäude- und Anlagedatenmodell von den Berechnungskomponenten auszeichnet. Das Kernstück der Arbeit ist die Entwicklung und die Implementierung der Komponente Gebäude- und Anlagedatenmodell aus Sicht der Technischen Gebäudeausrüstung. Dabei liegen die Schwerpunkte auf der Behandlung von Fragen der Gebäudetechnik und hier insbesondere auf der Einbindung von Gebäude- und Anlagesimulationen während der Planung und des Betriebs. Der Rahmen für die Entwicklung des Datenmodells ist durch das aktuelle Austauschmodell der Industrieallianz für Interoperabilität (IAI) vorgegeben. Das Datenmodell erweitert dieses Austauschmodell um die Beschreibung heiz- und raumlufttechnischer Anlagen. Nach der Vorgaben der IAI ist es die Aufgabe des Datenmodells, den Datenaustausch zwischen Komponenten zu ermöglichen. Die Aufgaben des hier entwickelten Datenmodells gehen allerdings über den Datenaustausch hinaus, da es zur einheitlichen Beschreibung von Gebäuden und deren Anlagen während des gesamten Lebenszyklus dient. Dazu muss es dynamisch erweiterbar sein und eine so große Informationsdichte besitzen, dass die Daten in verschiedenen Phasen des Gebäudelebenszyklus ohne Informationsverluste verwendet werden können.

Das Komponentenframework ermöglicht das Zusammenfügen verschiedener Berechnungskomponenten zu Anwendungen. Die Voraussetzung dafür ist, dass sie das OLE DB Interface unterstützen und für den Austausch der Gebäude- und Anlagedaten das in dieser Arbeit entwickelte Datenmodell verwenden. Die Berechnungskomponenten liefern ihre Ergebnisse in Form von Zeitreihen. Die Abstraktion dieser Zeitreihen vereinfacht die Verwaltung der Berechnungsergebnisse und ermöglicht deren direkten Vergleich mit Messwerten. Das Datenmodell und die Berechnungskomponenten werden durch die Verwendung einer Skriptsprache zu Anwendungen integriert.

Abstract

In this thesis a component-based framework for the evaluation of HVAC performance in buildings is developed. Following the method of the demand development HVAC systems are evaluated by means of their energy effort during planing and maintenance. The evaluation is achieved through:

- determining the energy requirement and energy requirement's impacting factors using a building simulation
- determining the effort, which an HVAC system requires in order to meet the demand, using an HVAC system simulation
- continuos comparison of the measured states of both building and HVAC system with simulated values using a coupled building and HVAC system simulation.

Characteristic for the component-based framework is an architecture which separates the component *data model* form the calculation components. In the centre of this work is the development and the implementation of the component *data model* from a building service's point of view. The main focus lays in dealing with building services's questions and embedding the building and HVAC system simulations into the processes of planing and maintenance. The design of the *data model* is based on the latest version of the building information exchange model developed by the International Alliance for Interoperability (IAI). The data model extends this model with definitions of HVAC equipment and systems. In accordance with IAI guidelines the data model is used for the exchange of information among components. More important however, the data model serves for a uniform description of buildings and installations over all phases of the life-cycle. For this purpose it is extendable and has an information content large enough for an efficient information usage throughout the building's life.

The component-based framework enables assembling of different calculation components into applications. The calculation components are required to support the OLE DB interface and to exchange building and installation data using the data model. The calculation components deliver their results as time series. The abstract data type of time series relieves the management of the calculation results and allows their direct comparison with measured data. The component *data model* and the calculation components are integrated into applications that use a scripting language.

Danksagungen

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Kernenergetik und Energiesysteme an der Universität Stuttgart.

Mein besonderer Dank gilt Herrn Professor Dr.-Ing. Fritz Schmidt für die Unterstützung bei der Anfertigung der Arbeit, für die Übernahme des Hauptberichtes und für die wertvollen Ratschläge und Anregungen zur Lösung der Aufgaben. Durch seine kritische Betrachtung einerseits und positive Bestärkung andererseits hat er unverzichtbar dazu beigetragen, meine software-technischen Ideen in einen breiteren wissenschaftlichen Kontext einzubetten.

Ebenso danke ich Herrn Professor Dr.-Ing. R. Rühle für die kritische Durchsicht der Arbeit und die Übernahme des Mitberichtes.

Besonders danken möchte ich Herrn Dipl.-Ing. Raphael Haller vom Lehrstuhl für Heiz- und Raumlufttechnik der Universität Stuttgart für die stete Bereitschaft offene Fragen und Lösungsansätze zu diskutieren.

Allen Kollegen und Kolleginnen am Institut sei an dieser Stelle ebenso für die Schaffung einer angenehmen, anregenden und freundschaftlichen Atmosphäre wie für die zahlreichen – oft nahezu endlosen doch meist fruchtbaren – Diskussionen gedankt. Meinen Kollegen Dipl.-Gwl. Kurt De Marco, Dipl.-Ing. Roland Kopetzky und Dr.-Ing. Michael Weigele bin ich für den Meinungsaustausch und die ausgezeichnete Zusammenarbeit sehr dankbar. Nicht vergessen will ich in diesem Zusammenhang auch die zahlreichen Studenten, deren Diplom- und Studienarbeiten mir bei der praktischen Umsetzung meiner Ideen eine große Hilfe waren.

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Verzeichnis der Abkürzungen

AP	Application Protocol
API	Application Programming Interface
ATL	Active Template Library
ATLAS	Architecture, Methodology and Tools for computer-integrated Large Scale Engineering
BS	Building Services
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CCM	CORBA Component Model
CIDL	Component Implementation Definition Language
COM	Component Object Model
COMBINE	Computer Models for the Building Industry in Europe
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DIN	Deutsche Industrie Norm
DLL	Dynamic Link Library
DOM	Document Object Model
DSI	Dynamic Skeleton Interface
EJB	Enterprise Java Bean
EN	European Norm
EXPRESS	Modellierungssprache
EXPRESS-G	Graphische Modellierungssprache
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HVAC	Heating, Ventilation and Air Conditioning
IAI	International Alliance for Interoperability
IDM	Integrated Data Model
ISO	International Standards Organisation
IDL	Interface Definition Language
IFC	Industry Foundation Classes
IID	Interface Identifiers
IIOP	Inter-ORB Protocol
IKE	Institut für Kernenergetik und Energiesysteme der Universität Stuttgart
J2EE	Java 2 Enterprise Edition
JAF	JavaBeans Activation Framework
JDBC	Java Database Connectivity
JDK	Java Development Kit
JIT	Just In Time

MIDL	Microsoft Interface Definition Language
MTA	Multi Threaded Apartment
MTS	Microsoft Transaction Server
ODBC	Open Database Connectivity
OLE	Object Linking and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
ORPC	Object Remote Procedure Call
PC	Personal Computer
REUSE	Rational Use of Energy at University of Stuttgart Building Environment
RLT	Raumlufttechnik
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAX	Simple API for XML
SHK	Sanitär- Heizung- Klimaverband Baden Württemberg
STA	Single Threaded Apartment
STEP	Standard for Exchange of Product Model Data
TCP/IP	Transmission Control Protocol/Internet Protocol
TGA	Technische Gebäudeausrüstung
TRNSYS	Transient Simulation System
UML	Unified Modeling Language
VDI	Verein Deutscher Ingenieure
VEC	Visual Energy Center
W3C	World Wide Web Consortium
VM	Virtuelle Maschine
WSchV	Wärmeschutzverordnung
WWW	World Wide Web
XM	Model Extensions
XML	Extensible Markup Language

1 Einleitung

In der Baubranche gibt es eine starke Tendenz die Kosten eines Gebäudes nicht mehr phasenabhängig (aufgeteilt in Planung, Ausführung, Inbetriebnahme und Betrieb) zu betrachten, sondern über alle Phasen des Gebäudelebenszyklus als Ganzes zu behandeln. Planungs-, Ausführungs-, Inbetriebnahme- und Betriebskosten bilden dann eine Einheit, die es zu minimieren gilt. Dies führt zu einem erheblichen Druck auf die Gebäudetechniker. Die Kosten, die durch die technische Gebäudeausrüstung und deren Betrieb entstehen, können inzwischen ein Vielfaches der Baukosten betragen [1]. Eine Aussage über die Kosten des Betriebs und hier vor allem über die dabei entstehenden Energiekosten ist daher schon im Vorfeld eines möglichen Auftrags ein entscheidendes Verhandlungsargument, dessen Richtigkeit garantiert und im realen Betrieb nachgewiesen werden muss. Dies zwingt zu einem neuen Umgang mit der Gebäudetechnik und der Ausnützung ihrer ökonomischen und zunehmend auch ihrer ökologischen Potentiale. Vor diesem Hintergrund sind die Anforderungen an Planung, Bau und Betrieb von Gebäuden und deren heiz- und raumlufttechnischen Anlagen rasant gestiegen. Die Branche hat darauf mit einem Paradigmenwechsel reagiert und die Methode der Bedarfsentwicklung als Grundlage ihres Vorgehens entwickelt [2]. Die Umsetzung dieses Paradigmas in der Praxis erwies sich als äußerst schwierig und ist nur mit Hilfe moderner Informations- und Kommunikationstechniken möglich. In der vorliegenden Arbeit wird ein Weg aufgezeigt, die neuen Methoden anwendbar zu machen.

Bereits bei der Gebäudegesamtkonzeption wird die Grundlage für den Energiebedarf gelegt, wobei zunächst Lage und Ausrichtung des Gebäudes und aus bauphysikalischer Sicht u.a. der Dämmwirkung der verwendeten Materialien sowie dem Fensteranteil der Gebäudehülle besondere Bedeutung zukommen. Der Energiebedarf wird darüber hinaus durch den beabsichtigten Nutzen bestimmt, unter dem im einfachsten Fall ein vorgegebener Verlauf der Raumlufttemperatur zu verstehen ist. Zur Einstellung des gewünschten Raumklimas sind in der Regel Anlagen erforderlich. Für einen optimalen Gebäudebetrieb ist es notwendig, die Energielieferung der Anlagen an den durch die Nutzenanforderungen vorgegebenen Bedarf anzupassen. Die Anlage soll den erwarteten Nutzen bedarfsgerecht, d.h. mit möglichst kleinem Aufwand übergeben. Mit der Methode der Bedarfsentwicklung können heiz- und raumlufttechnische Anlagen anhand ihres Energieaufwands bewertet werden. Die Überprüfung, wie eine Anlage den Bedarf befriedigt, sollte aber nicht nur während der Planung und eventuell der Inbetriebnahme erfolgen. Vielmehr ist sie insbesondere während des Betriebs kontinuierlich durchzuführen. Die bedarfsorientierte Planung ist als Vorstufe eines bedarfsorientierten Betriebs zu sehen, der sowohl bei neuen als auch bei bestehenden Gebäuden anzustreben ist.

Besonderer Wert wird bei der Methode der Bedarfsentwicklung erstmals auf eine Gesamtbewertung (Übergabe, Verteilung und Erzeugung) der Anlagetechnik gelegt. Die Gesamtbewertung der Anlagetechnik bedarf daher einer interdisziplinären Betrachtung, welche in drei Stufen erfolgt:

- Feststellung des Energiebedarfs des Gebäudes ohne Anlagetechnik
- Bestimmung der Gesamtaufwandszahl zur Bewertung der Anlage
- Bewertung des Primärenergieeinsatzes mit Primärenergiefaktoren.

Damit die Anforderungen, die aus der Methode der Bedarfsentwicklung resultieren, erfüllt werden können, sind rechnergestützte Planung und rechnergestützte Betriebsführung notwendig. Die rechnergestützte Planung [3] ermöglicht die Einbettung unterschiedlicher Werkzeuge in den Planungsprozess. Dies geschieht mit herkömmlichen Technologien auf zwei Arten [4]:

- durch Entwicklung monolithischer Programme, welche meistens im Auftrag großer Baukonzerne entstehen und ein eigenes aber proprietäres Modell zur Beschreibung des Gebäudes und der Anlagetechnik festlegen. Diese Entwicklungen sind für Klein- und mittelständische Unternehmen der Baubranche nicht zugänglich.
- durch Entwicklung von Planungsplattformen, welche zwar für Klein- und mittelständische Unternehmen zugänglich sind, aber die Struktur der im Planungsprozess benötigten Daten nicht vorgeben. Hier wird lediglich eine Basis geschaffen, projektbezogene Gebäude- und Anlagedaten als Dateien unternehmensweit auszutauschen.

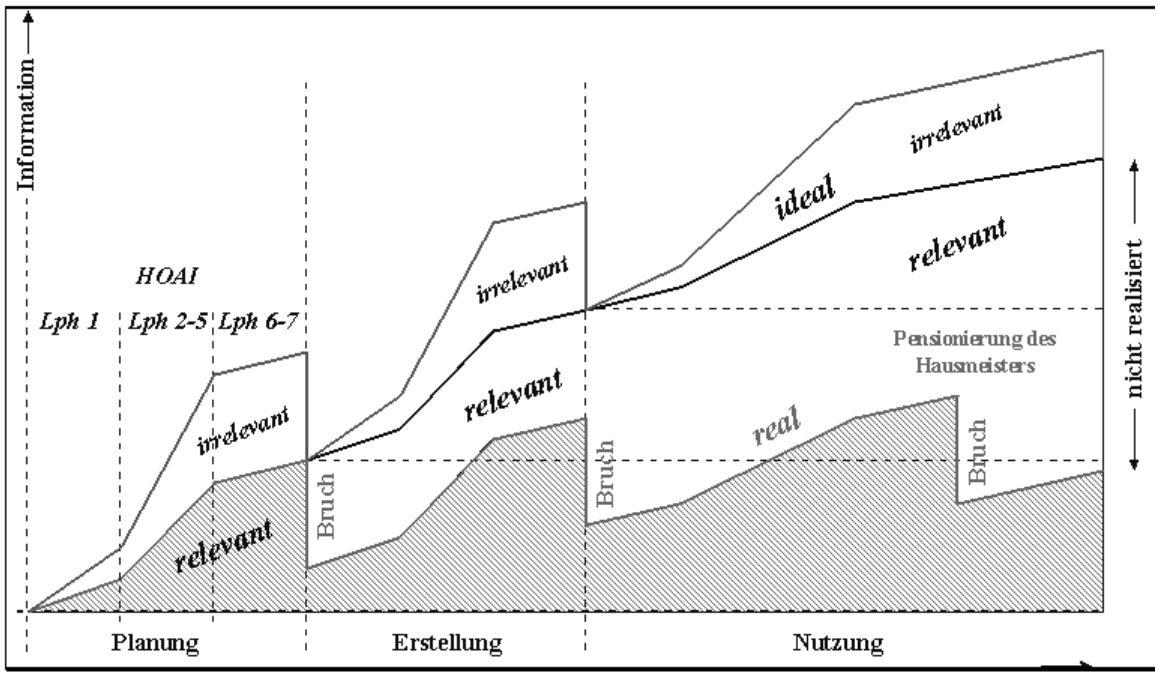


Abbildung 1: Informationsbrüche im Bauprozess[1]

[1] Quelle: [5]

Die beiden Vorgehensweisen führen wegen deren Beschränkung auf die Planung und des Fehlens eines standardisierten, offenen und für alle zugänglichen Gebäude- und Anlagemodells unvermeidlich zu Informationsbrüchen, wie in Abbildung 1 dargestellt ist. Informationsbrüche bedeuten, dass es von der Planung bis zur Inbetriebnahme des Gebäudes nicht mehr alle in der Betriebsphase benötigten Daten zur Verfügung stehen. Die Konsequenz dieser Brüche ist entweder eine teure Neuerfassung der Daten oder ein Kompromiss bei der eigentlich zu optimierenden Betriebsführung des Gebäudes.

Allgemein kann festgehalten werden, dass eine nahtlose Verbindung unterschiedlicher Werkzeuge im Lebenszyklusprozess eines Gebäudes bisher nicht so gelungen ist, wie es aus der Sicht der Planung und des Betriebs energieeffizienter Gebäude wünschenswert wäre. Dies liegt zum größten Teil am fehlenden gemeinsamen Verständnis der im Bauprozess anfallenden Daten. Der erste Schritt ist es also, eine Verständigung über die gemeinsam zu nutzenden Daten zu erzielen, indem festgelegt wird, welche Daten zu erfassen sind und in welchem Detaillierungsgrad dies geschehen muss. Die Beschreibung der zu erfassenden Daten und deren Strukturierung erfolgt in Datenmodellen. Das Kernstück dieser Arbeit ist die Entwicklung eines Datenmodells zur Beschreibung von Gebäude und Anlagetechnik aus Sicht der Technischen Gebäudeausrüstung. Der Schwerpunkt wird daher auf die Behandlung von Fragen der Gebäudetechnik und hier wieder auf die Einbindung der Gebäude- und Anlagesimulationen während der Planung und des Betriebs gelegt. Das in der vorliegenden Arbeit entwickelte Datenmodell wird in die allgemeine Entwicklung der Industrieallianz für Interoperabilität (IAI) [6] eingebettet. Die IAI entwickelt aus der hier geschilderten Motivation ein Basismodell (IFC-Modell [7]) zur gemeinsamen Datennutzung im Bauwesen. Das in der vorliegenden Arbeit entwickelte Datenmodell benutzt das IFC-Modell als Leitfaden für die Gebäudebeschreibung und erweitert dieses um eine Beschreibung heiz- und raumlufttechnischer Anlagen. Es ist erweiterbar und unterstützt die semantische Interpretation der in ihm abgelegten Daten. Dadurch ermöglicht es geschlossene Anwendungen wieder in unabhängige Teile in Form von Softwarebausteinen aufzubrechen. Diese Softwarebausteine können dann zur Entwicklung neuer Anwendungen kombiniert werden.

Hinter der komponentenbasierten Softwareentwicklung steht die Idee der Zusammensetzung von Software aus bereits vorhandenen Softwarekomponenten [8]. An diesem Punkt setzt der zweite Teil der vorliegenden Arbeit an. Die Softwarekomponenten müssen in einer speziellen Art und Weise konstruiert werden, um mit anderen Komponenten in eine Anwendung zusammengefügt werden zu können. Mit anderen Worten, eine Softwarekomponente ist Teil eines Komponentenframeworks, das eine Bibliothek von Blackbox-Komponenten zur Verfügung stellt, und eine wiederverwendbare Softwarearchitektur vorgibt, in der die Komponenten geeignet integriert sind. Wurden vor einigen Jahren noch überwiegend Bausteine für graphische Benutzeroberflächen als Komponenten angeboten, so hat sich dieses Bild inzwischen grundlegend geändert. Mit den weiterentwickelten Komponententechnologien (COM, CORBA, JavaBeans) wurden die wichtigsten Voraussetzungen geschaffen, um komponentenbasierte Anwendungen für unternehmensübergreifende Lösungen zu entwickeln. In der vorliegenden Arbeit wird ein Lösungsvorschlag für ein

komponentenbasiertes Framework zur Bewertung heiz- und raumlufttechnischen Anlagen erarbeitet. Der Ansatz basiert auf einer Komponentenarchitektur, durch die eine klare Trennung der Komponente Gebäude- und Anlagedatenmodell von den Berechnungskomponenten erreicht wird. Das Gebäude- und Anlagedatenmodell vergegenständlicht eine gemeinsame fachliche Basis zur Integration und ermöglicht den Datenaustausch zwischen den Berechnungskomponenten. Es repräsentiert daher ein für alle Komponenten gemeinsames Modell.

Die Annäherung an das Ziel der Entwicklung des Frameworks zur Bewertung heiz- und raumlufttechnischer Anlagen erfolgt in mehreren Schritten. Zunächst gibt Kapitel 2 eine Übersicht über die Methode der Bedarfsentwicklung (Abschnitt 2.1) sowie beispielhaft den Einsatz von Werkzeugen in der rechnergestützten Planung. Kapitel 3 behandelt die wichtigsten Komponententechnologien und beschreibt die Grundlagen der komponentenbasierten Softwareentwicklung. In Kapitel 4 wird das im Rahmen dieser Arbeit entwickelte, komponentenbasierte Framework zur Bewertung heiz- und raumlufttechnischer Anlagen vorgestellt. Der Schwerpunkt liegt dabei auf der Komponente Gebäude- und Anlagedatenmodell, die in Abschnitt 4.2 dargestellt und für die Diskussion durch die Fachwelt im Internet unter http://www.ike.uni-stuttgart.de/~www_wn/projects/datenmodell/index.html veröffentlicht wird. Diese Seite ist passwortgeschützt. Die praktische Anwendbarkeit und Erprobung des entwickelten Frameworks wird in Kapitel 5 anhand von Beispielen gezeigt. In Kapitel 6 werden die gewonnenen Erkenntnisse zusammengefasst.

2 Bewertung heiz- und raumlufttechnischer Anlagen

2.1 Bedarfsorientierte Planung und Betrieb heiz- und raumlufttechnischer Anlagen

Energieeffizienz und nachhaltiger Klimaschutz werden neben dem Selbstverständnis und den Selbstverpflichtungen der umweltgestaltenden Industrie und der Wissenschaft durch energierelevante Gesetze und Verordnungen forciert. So wurden beispielsweise in den letzten 20 Jahren die Vorgaben für die Wärmedämmung, insbesondere für Wohngebäude, deutlich erhöht. Es ist davon auszugehen, dass die als 3. Novellierung der Wärmeschutzverordnung anzusehende und im Frühjahr 2002 zu erwartende Energiesparverordnung (ESPA) [9], das Niveau von sog. Niedrigenergiehäusern erreichen wird.

Ein Beispiel aus der Heiztechnik [2] verdeutlicht, dass die Anforderungen an heiz- und raumlufttechnische Anlagen mit zunehmender Dämmung etwa nicht niedriger, sondern – im Gegenteil – höher werden. Durch die erhöhten Vorgaben an die Wärmedämmung werden die Wärmeverluste durch Transmission erheblich verringert. Mit den erhöhten Standards für die Wärmedämmung von Rohrleitungen werden auch die Wärmeverluste im Bereich der Wärmeverteilung deutlich gesenkt. Weiterhin ist es möglich, die Wärmeerzeuger so zu optimieren, dass die Nutzungsgrade bei Teillast- und Volllastbetrieb nahezu gleich hoch sind. Im Gegensatz zum sinkenden Energiebedarf nimmt jedoch – durch die bessere Wärmedämmung – der Anteil der inneren und solaren Wärmelasten an den gesamten Heizlasten eines Gebäudes deutlich zu. Die dabei auftretende passiv nutzbare Sonnenenergie und die anfallenden inneren Wärmelasten können nur dann genutzt werden, wenn das installierte Heizsystem in der Lage ist, dem Heizlastprofil im Raum zu folgen. Ansonsten führt ein klassisch ausgelegtes Heizsystem dem Raum mehr Energie zu, als benötigt wird. Untersuchungen von Bauer [2] zeigen, dass bei sehr gut gedämmten Gebäuden die größten Einsparpotentiale bei der Beheizung der Räume darin liegen, dass sie möglichst eng an den Energiebedarf angepasst erfolgt.

Vor dem Hintergrund, dass die Nutzung den größten Einfluss auf den Energiebedarf hat, werden am Lehrstuhl für Heiz- und Raumlufttechnik der Universität Stuttgart neue bedarfsorientierte Planungs- und Betriebsmethoden entwickelt. Die Grundideen der bedarfsorientierten Planung und des bedarfsorientierten Betriebs entstammen einerseits der Überlegung, ob und in welcher Weise eine Anlage geeignet ist, die vom Nutzer gestellten Anforderungen zu erfüllen. Zum anderen soll die Anlage den Bedarf möglichst kostengünstig befriedigen. Mit der Bedarfsentwicklungsmethode steht eine Methode zur Verfügung, welche sowohl eine Planungsmethodik zur Ermittlung eines geeigneten Anlagesystems als auch eine Bewertungsmethodik alternativer Entwürfe und Betriebsstrategien hinsichtlich verschiedener Kriterien anbietet.

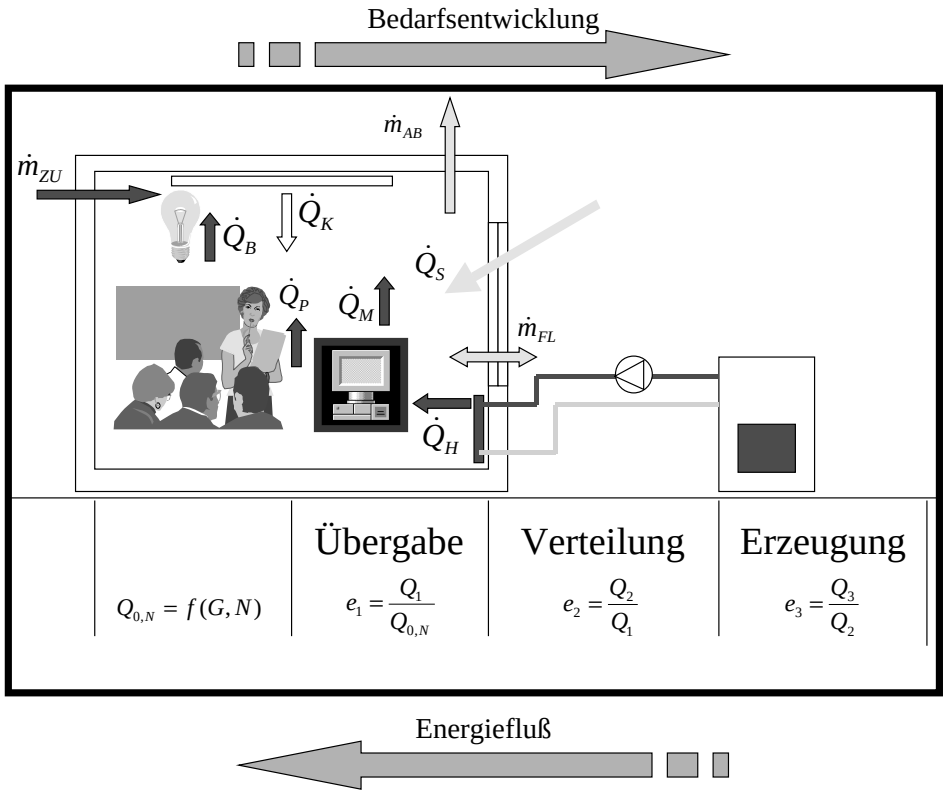


Abbildung 2: Methode der Bedarfsentwicklung

Die Methode der Bedarfsentwicklung lässt sich wie folgt beschreiben. Aufgabe des Anlagesystems ist es, den sich zeitlich verändernden Energiebedarf des Raumes so zu decken, dass die Nutzenanforderungen befriedigt werden. Die Deckung des Bedarfs durch ein bestimmtes System wird als Nutzenübergabe (vgl. Abbildung 2) bezeichnet. Zur Deckung des Bedarfs ist aufgrund von Regelabweichungen, thermischer Trägheit usw. ein Aufwand nötig. Aufwand bedeutet, dass die von einzelnen Komponenten oder dem System dem Raum zugeführten Energiemengen größer sind, als eigentlich benötigt. Der Bedarf der Nutzenübergabe wird von der Verteilung gedeckt. Dabei entsteht wiederum ein Aufwand; der Aufwand der Verteilung. Der Aufwand der Erzeugung ist schließlich diejenige Stoff- und Energiemenge, welche benötigt wird, um den Bedarf der Verteilung zu decken. In Abbildung 2 ist die Bedarfsentwicklung in beheizten Gebäuden dargestellt. Abbildung zeigt, dass die Bedarfsentwicklung genau entgegengesetzt zum Energiefluss gerichtet ist. Wird der zusätzliche Aufwand in den einzelnen Bedarfsbereichen auf die eigentlich erforderlichen Bedarfswerte bezogen, ergeben sich die Aufwandszahlen e_i, die als energetische Hilfsgrößen zum Bewerten der Anlagen in den Bereichen Nutzenübergabe, Verteilung und Erzeugung herangezogen werden können. Die Aufwandszahlen sind zum einen von den Anlageeigenschaften abhängig, zum anderen von der Betriebsführung.

Von den ermittelten Bedarfswerten streng zu unterscheiden sind Verbrauchswerte, welche sich bei einem realen Gebäudebetrieb tatsächlich einstellen. Die Verbrauchswerte sind abhängig von den realen Wetterdaten, der realen Nutzung und den aus dem Betrieb der Anlage resultierenden Aufwandszahlen. Trotzdem kann der Betrieb heiz- und raumlufttechnischer Anlagen mit Hilfe der Methode der Bedarfsentwicklung untersucht werden. Dabei ist das oberste Ziel die Gewährleistung eines störungsfreien Betriebs. Für den Bereich der Nutzenübergabe ist dieses Ziel erreicht, wenn zu keinen Nutzungszeiten Beeinträchtigungen in der Behaglichkeit auftreten. Störungsfrei ist nicht nur im Sinne des Betriebs der technischen Anlage zu verstehen. Vielmehr ist nur dann ein störungsfreier Betrieb erreicht, wenn der Nutzer zufrieden ist. Beispielsweise kann ein Lüftungssystem im Bereich der Verteilung und Erzeugung technisch ohne Störungen laufen, aber im Raum sind die Behaglichkeitsanforderungen nicht erreicht, weil die Luftgeschwindigkeit zu hoch ist. Auf der anderen Seite kann die Temperatur im Raum noch den Sollwert haben, obwohl an einer Umwälzpumpe eine Störung vorliegt. Wird die Störung rechtzeitig entdeckt und behoben, sind keine Einschränkungen im Bereich der Übergabe zu erwarten und damit ist der bedarfsgerechte Betrieb sichergestellt. Der störungsfreie Betrieb kann überprüft werden, indem die durch Messung ermittelten Raum- und Anlagezustände mit simulierten Werten verglichen werden. Untersuchungen von Madjidi [10] zeigen, dass sowohl das Erkennen von Betriebsfehlern als auch die Betriebsführung von heiz- und raumlufttechnischen Anlagen durch eine betriebsbegleitende Simulation des Anlageverhaltens verbessert werden können. Voraussetzung solcher Simulationen sind Modelle zur Beschreibung von Gebäude und Anlagetechnik. Diese Beschreibungsmodelle müssen in der Lage sein, alle Informationen über eine Anlage (Parameter, Zustandsgrößen, usw.) so bereitzustellen, dass sie von den Verhaltensmodellen der Simulationsprogramme adäquat verwendet werden können.

2.2 Datenmodelle zur Beschreibung von Gebäuden und Anlagen

In den letzten 20 Jahren wurden verstärkt digitale Modelle zur Beschreibung von Gebäude und Anlagetechnik entwickelt. Alle Ansätze bedienen sich einer Kombination von Transformation zwischen anwendungsspezifischen Teilproduktmodellen (Aspektmodellen) und dem Bemühen um ein allumfassendes, einheitliches Produktmodell. Die wichtigsten Ansätze – in Bezug auf diese Arbeit – werden im folgenden dargestellt.

2.2.1 Standard for the Exchange of Product Model Data STEP

Der von der ISO herausgegebene Standard for the Exchange of Product Model Data (STEP)[2] definiert Softwareschnittstellen für den herstellerunabhängigen Austausch von Produktdaten. STEP ist ursprünglich für den Datenaustausch zwischen CAD-Systemen ausgelegt. Inzwischen hat sich STEP jedoch zu einer Technologie entwickelt, auf der sich nicht nur Austausch der Produktdaten, sondern auch Speicherung, Archivierung und die Verarbeitung der Produktdaten in interaktiven Systemen vollziehen soll. Die

[2] Einen ausführlichen Überblick über STEP bieten [11], [12], [13].

hierarchische Anordnung der STEP-Modelle geschieht in numerierten Serien, welche in zwei Ebenen aufgeteilt sind:

- Kernmodelle (Integrated Resource Models)
- Anwendungsmodelle (Application Interpreted Models).

Kernmodelle bilden dabei die Grundbausteine für die Anwendungsmodelle und gewährleisten durch ihre integrale Funktion einen hohen Grad an Interoperabilität. Auf den Kernmodellen aufbauend wurden eine Vielzahl von standardisierten und integrierten Anwendungsmodellen (AP) für unterschiedliche Ingenieurdisziplinen erarbeitet. Das Bauwesen und damit auch Heiz- und Raumlufttechnik ist in einer Hauptgruppe mit dem Schiffsbau zusammengefasst.

Für den Datenaustausch im Bauwesen wurde die ISO/DIS 10303-225 (AP 225) "Building Elements Using Explicit Shape Representation" [14] entwickelt. Das AP 225 ermöglicht den Datenaustausch von 3D-Gebäudemodellen. Die Gebäudemodelle werden dabei durch Bauteile dargestellt. Schwerpunkt ist die korrekte Übertragung der 3D-Geometrie und der Struktur von Gebäuden [15], wie z. B. ihre Einteilung in Bauabschnitten, Geschossen und Bauteilgruppen. Das AP 225 sieht die Geometrie als Hauptträger der Informationen, während die Semantik der übertragenen Elemente durch die Zuweisung einer textuellen Beschreibung festgelegt wird. Das AP 228 "Building Services: Heating, Ventilation and Air Conditioning" [16] ist ein STEP-Anwendungsmodell zur Modellierung technischer Gebäudeausrüstung, speziell heiz- und raumlufttechnischer Anlagen. Das AP 228 wurde im Rahmen des ATLAS [17] Projektes entwickelt. Der Modellierungsumfang von AP 228 beschränkt sich auf die Beschreibung der Anlagetopologie einschließlich der Rohrnetze sowie einer energetisch-thermischen Bewertung der Anlagetechnik. Eine einheitliche Betrachtung über den gesamten Gebäudelebenszyklus wurde nicht durchgeführt.

Zur normativen, formalen Beschreibung der Produktmodelle bietet STEP die Spezifikationssprache EXPRESS. EXPRESS wird speziell zur Modellierung komplexer und wissenschaftlicher Daten auf objektorientierte Weise verwendet. Dabei werden die Prinzipien wie Zerlegung, Vererbung und Kapselung beachtet. Eine detaillierte Beschreibung von EXPRESS wird in [13], [18] und [19] gegeben. Um Schemaspezifikationen für Menschen leichter verständlich und überschaubar zu machen, wurde EXPRESS-G [13] als Möglichkeit zur graphischen Darstellung von Produktmodellen entwickelt. Damit können sowohl Informationen über den hierarchischen Aufbau von Modellen als auch Zusammenhänge innerhalb eines Modells abgebildet werden. Auf EXPRESS aufbauend existiert eine Vielzahl weiterer Gebäude- und Anlagemodelle, welche im Rahmen verschiedener internationaler und nationaler Programme entstanden sind. COMBINE 2 [20] versuchte, mit dem Integrated Data Model (IDM) ein integriertes Produktmodell zu etablieren, in das oder aus dem andere Sichten gewonnen werden können. Durch einen Ausbau des IDM auf alle Bereiche wäre jedoch die Komplexität des Modells nicht mehr handhabbar gewesen. In Finnland wurde mit dem RATAS Projekt [21] das Ziel verfolgt ein Gebäude-Produkt-Modell zu entwickeln, das einer rechnergestützten Planung im Bauwesen dienen soll. Der Lebenszyklusgedanke

sowie die besonderen Aspekte der Heiz- und Raumlufttechnik bleiben bei den beiden Modellen ebenfalls unberücksichtigt.

2.2.2 Industrieallianz für Interoperabilität IAI

Die Industrieallianz für Interoperabilität (IAI) [6], ist eine Gruppe von Softwareherstellern und Industrieunternehmen der Baubranche, die den Austausch von elektronisch gespeicherten Daten im Bauwesen vorantreibt. Dazu entwickelt sie das Industry Foundation Classes (IFC) Objektmodell und greift dabei auf Techniken zurück, die sich im Rahmen von STEP bewährt haben. Dazu gehören die Grundsätze der Produktbeschreibung, geometrische und topologische Darstellungen, die Modellierungssprachen EXPRESS und EXPRESS-G und das Physical File Format. Über STEP hinausgehend wird die Verfügbarkeit von weiterentwickelten Datenbanken und Softwarewerkzeugen, welche objektorientierte Datenstrukturen zulassen, berücksichtigt.

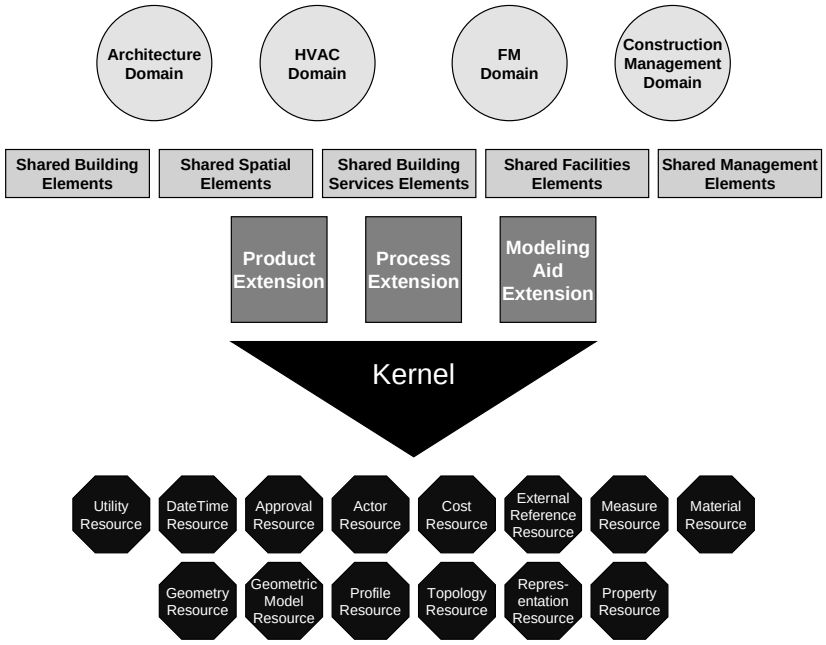


Abbildung 3: IFC Objektmodell Architektur[3]

Die erste Version, die in kommerzieller Software umgesetzt wurde, ist die Version IFC 1.5.1 [7]. Mit dieser Version können die wesentlichen Objektdaten des Rohbaus und der Raumaufteilung übertragen werden. Die Version IFC 1.5.1 setzt sich aus unabhängigen Ressourcenmodellen, einem objektorientierten Kernmodell und Modellerweiterungen für Architektur, TGA-Planung, Facility Management und Tragwerksplanung zusammen (Abbildung 3). Die Ressourcenmodelle beschreiben Geometrie, physikalische Einheiten und Hilfsobjekte zur allgemeinen Nutzung. Das Kernmodell legt Basisobjekte, ihre Attribute sowie Beziehungen zwischen diesen Objekten fest. Das Kernmodell wird

[3] Quelle: [7]

ausschließlich von darauf aufbauenden Modellerweiterungen verwendet. Die Modellerweiterungen legen Objekte, Attribute und Beziehung der jeweiligen Domäne fest. Damit die Bauwerksdaten über den eigentlichen Rohbau hinaus übertragen werden, wurde die Version IFC 1.5.1 erweitert. Die erweiterte Version trägt die Bezeichnung IFC 2x [22]. IFC 2x beruht auf einem neuartigen Plattformkonzept, das die Erweiterung des stabilen IFC Kernmodells um spezielle Fachanforderungen zulässt. Insbesondere die energetisch-thermische Bewertung der Anlagetechnik benötigt diesen erweiterten Datenumfang. Mit einer ersten Implementierung von IFC 2x ist Ende 2001 zu rechnen.

Ein weiterer Schritt, den die IAI im Rahmen von IFC 2x vollzogen hat, ist die Hinwendung zu eXtensible Markup Language (XML) [23] als alternatives Definitions-, Austausch- und Entwicklungsformat. Die neue ifcXML Spezifikation bietet den Leistungsumfang von IFC 2x zur Entwicklung eines XML-basierten Standards für das Bauwesen.

2.2.3 Datenaustauschverfahren der VDI

Die VDI-Gesellschaft Technische Gebäudeausrüstung hat sich zum Ziel gesetzt, die Regelung des Produktdatenaustausches im rechnergestützten Planungsprozess auf nationaler Ebene festzulegen. Zu diesem Zwecke wurden eine Vielzahl der Richtlinien entworfen und zum Teil auch verabschiedet. Die VDI-Richtlinien greifen ebenfalls auf EXPRESS bzw. EXPRESS-G sowie das STEP Physical File Format zurück.

Für den Teilbereich eines digitalen Gebäudemodells liegt seit Anfang 1998 der Entwurf der VDI-Richtlinie 6021 Blatt 1 "Datenaustausch für die thermischen Lastberechnungen von Gebäuden“ [24] vor. Ziel der Richtlinie ist es innerhalb des klar abgegrenzten Gebietes der Datenbereitstellung zur thermischen Lastberechnung von Gebäuden, ein Datenaustauschverfahren zu erarbeiten, das den Austausch der notwendigen Informationen aus dem CAD-Programm des Gebäudeplaners zur Nutzung in den Berechnungsprogrammen des Haustechnikers regelt. Die zu berechnenden Lasten werden im wesentlichen durch die klimatischen Bedingungen am Gebäudestandort, durch das Gebäude selbst und die Nutzung beeinflusst. Hierzu wurde ein thermisches Gebäudemodell entwickelt, das vom angewandten Rechenverfahren, Norm, Richtlinie oder Simulationsprogramm unabhängig ist. Das hierarchisch strukturierte Gebäudemodell beruht auf den Vorgängermodellen entwickelt von ISYBAU [25], Hirschberg [3] und Hinkelmann [19]. Die erwähnten Gebäudemodelle beinhalten jedoch keine Beschreibung der Geometrie, weshalb eine genaue Abbildung der Strahlungsvorgängen nicht möglich ist. Die Ergebnisse dieser Richtlinie sind über das IAI Projekt BS-4 [26] in IFC 2x eingeflossen.

Daten der Anlagetechnik sollen zukünftig nach der derzeit in Bearbeitung befindlichen VDI 6027 Blatt 2 "Anforderungen an den Datenaustausch von CAD-Systemen; Anlagetechnik" [27] ausgetauscht werden. In der Richtlinie wird zwischen Anlagekomponenten und Anlagenetzen unterschieden. Für Anlagekomponenten werden die lokale Zuordnung, die Beschreibung der Produktdaten, der Auslegedaten sowie weitere, für spezielle Anwendungen (z.B. Simulationsrechnung) erforderliche Daten festgelegt. Gleichzeitig werden die Voraussetzungen geschaffen, diese Daten über

verschiedene Lebenszyklusphasen eines Gebäudes zu erfassen, zu ergänzen und zu aktualisieren. Für die Netze werden die allgemeine Netzstruktur und die weitergehende Netzbeschreibung festgelegt. Aus der Netzstruktur geht hervor, welche Komponenten in welcher Richtung im betroffenen Netz angeordnet sind. Die Abbildung von vermaschten Netzen ist ebenfalls möglich. Beim Entwurf dieser Richtlinie wurde die aktuelle Entwicklung des IFC Modells beachtet.

Für den Austausch von Produktdaten für Anlagekomponenten der Heiz- und Raumlufttechnik werden seit 1994 eine Vielzahl von Richtlinien [28] entwickelt. Die Produktbeschreibungen beinhalten sowohl technische Daten als auch einfache Geometriedaten und erlauben die Bildung von Produkthauptgruppen und Varianten. Die Beschreibungen sind hierarchisch fein strukturiert und kommentierbar. Gleichzeitig wird eine ‚TGA-Nummer' zur Kennzeichnung von Produkten eingeführt. Diese 36-stellige Nummer dient der Zuordnung von Datensätzen innerhalb der hierarchischen Datenstruktur. Mehrere Produkte mit jeweils eigener TGA-Nummer können durch Verweise auf Zubehör logisch miteinander verbunden werden. Bisher sind Richtlinien für Heizungsarmaturen [29] und Wärmeerzeuger [30] und Entwürfe von Richtlinien für Pumpen [31], Luftdurchlässe [32] und Heizkörper [33] erschienen.

2.3 Werkzeuge zur Unterstützung von Planung und Betrieb

Um die zukünftigen Anforderungen einer bedarfsorientierten Planung und des bedarfsorientierten Betriebs zu erfüllen, ist der Einsatz von software-technischen Werkzeugen notwendig. Die Planungsschritte entsprechend den Gebäudelebenszyklusphasen Vor-, Entwurfs- und Ausführungsplanung sollten mit visueller Unterstützung am CAD-System realisiert werden und im Sinne eines CAE (Computer Aided Engineering) zur Berechnung und Auslegung einer Anlage führen. Hierbei nutzt der Planer den dreidimensionalen CAD-Entwurf des Architekten, indem er seinen Anlagenentwurf direkt in das CAD-Modell einträgt. Neben Auslegedaten können dann zusätzliche Planungsinformationen z.B. Produktdaten für weitere Planungsschritte bereitgestellt werden. Die Optimierung und Abstimmung der geplanten Anlage mit dem Gebäude und seiner Nutzung bzw. die Ableitung von Regelstrategien geschieht durch eine gekoppelte Gebäude- und Anlagesimulation.

Diverse Werkzeuge sind in Bezug auf jeweiligen Planungsstadium in [34] beschrieben. Dort werden der Zweck des Werkzeuges, die zur Bewertung erforderlichen Daten sowie das zu erwartende Ergebnis genannt. Die meisten Werkzeuge, die bei der Bewältigung der Planungsaufgaben eingesetzt werden, weisen einige Nachteile auf:

- CAD-Systeme sind in der Regel große monolithische Anwendungen, die mehrere hundert Mannjahre Entwicklungszeit benötigten und über Jahrzehnte hinweg gewachsen sind. Entsprechend ist der Aufwand, der auf Seiten der Anwender in Schulungen investiert werden muss, groß.
- In der Gebäudetechnik geht trotz der Notwendigkeit einer langen Datenhaltung und -bearbeitung ein großer Teil der Informationen nach der

abgeschlossenen Planung verloren (siehe auch Abbildung 1 Informationsbrüche im Bauprozess).

- Die Verbindung von Gebäudeplanung (Architekt) und technischer Gebäudeausrüstung sollte sich auf der Basis gemeinsamer Daten vollziehen. Gemeinsame Datenmodelle sind aber nur ansatzweise vorhanden.
- Berechnungsprogramme sind als software-technische Insellösungen im Planungsprozess integriert. Dies führt ebenfalls zu Informationsbrüchen, inkonsistenten Daten und häufig auch Fehlern.
- Der Datenaustausch zwischen den beteiligten Werkzeugen findet in der Praxis bisher in Form eines dateibasierten Datenaustausches mit Hilfe von anwendungsbezogenen Direktkonvertern oder neutralen Dateiformaten statt. Der Datenaustausch wird dabei meist in mehreren Iterationsschleifen durchgeführt und ist oft mit Datenverlusten, Konvertierungsfehlern sowie Unterbrechungen des Planungsprozesses verbunden.

Wird das während der Planungsphase entstandene Gebäude- und Anlagemodell kalibriert, kann es zusätzlich als Referenzmodell zur Inbetriebnahme und Betriebsüberwachung verwendet werden. Dies ist für eine Unterstützung der bedarfsorientierten Betriebsführung notwendig. Ihr Ziel ist es, den Aufwand, der zur Erbringung des tatsächlich geforderten Nutzens nötig ist, möglich gering zu machen. Dabei sind die wichtigsten Einflussgrößen die Betriebsführung, der Anlagezustand und die Vermeidung gewöhnlicher Fehler (Fehlererkennung und -diagnose). Werkzuge, welche ein effektives Energiemanagement auf dieser Basis unterstützen, sind zur Zeit auf der Schwelle der Kommerziellisierung. Das in Abschnitt 5.5 beschriebene Visual Energy Center (VEC) ist ein Beispiel dafür.

2.4 Bewertung der Ausgangssituation und Schlussfolgerungen

Mit der Methode der Bedarfsentwicklung stehen dem Ingenieur und seinen Partnern – Architekt und Betreiber – Verfahren zur Verfügung, mit denen heiz- und raumlufttechnische Anlagen auf bisher nicht gekannte Art geplant und betrieben werden können. Besonderer Wert wird dabei erstmals auf die Bewertung der Anlagetechnik gelegt. Der Ausgangspunkt der Bewertung sind die vom Nutzer gestellten Anforderungen an die Behaglichkeit im Raum. Damit die gestellten Nutzenanforderungen erfüllt werden können, ist meistens eine Anlage einschließlich Regelung erforderlich. Wird die Anlage so betrieben, dass dem Raum mehr Energie zu- bzw. aus ihm abgeführt als benötigt wird, ist dies als zusätzlicher, unnötiger Aufwand zu bewerten.

Die bedarfsorientierte Planung erfordert die Bereitstellung von software-technischen Werkzeugen, mit welchen abhängig vom Planungsstadium, Bewertungen im Bereich heiz- und raumlufttechnischer Anlagen durchgeführt werden können. Die bereitzustellenden Werkzeuge sollten keine Insellösungen darstellen, sondern in Anwendungen integriert werden, welche für bestimmte Fragestellungen und Benutzerbedürfnisse frei konfigurierbar sind. Die nötige Flexibilität kann nur dann erreicht werden, wenn solche Anwendungen aus Komponenten aufgebaut werden, wobei

unterschiedliche Komponenten in den verschiedenen Lebenszyklusphasen zum Einsatz kommen. Das hierzu erforderliche Datenmodell zur Beschreibung des Gebäudes und seiner Anlage sollte Gültigkeit über den gesamten Lebenszyklus haben. Dazu muss es in der Lage sein, je nach Stand der Planung unterschiedliche Informationen zu liefern.

Nicht nur die Anbindung an die Architekturprogramme sollte über das Datenmodell geschehen, sondern die Informationsdichte des Datenmodells muss so groß sein, dass auch die für Berechnungsprogramme der technischen Gebäudeausrüstung notwendigen Daten aus dem Datenmodell entnommen werden können. Die meisten Berechnungsprogramme benötigen eine detaillierte Abbildung der Räume, welche geometrische, topologische und semantische Informationen über die den Raum umschließenden Flächen beinhaltet. Darüber hinaus müssen die Informationen über die Lage des Raumes sowie die Beziehungen zu benachbarten Räumen im Datenmodell vorhanden sein. Daher muss das Datenmodell raum- bzw. zonenbasiert[4] sein.

Das Datenmodell sollte als eine selbständige Datenmodell-Komponente unabhängig von etwaigen Berechnungsprogrammen realisiert werden und primär für den Datenaustausch zuständig sein. Da sich die Anforderungen an die Daten während des Lebenszyklus eines Gebäudes ändern, genügt es aber nicht nur die Daten auszutauschen, sondern es müssen auch Informationen über die Bedeutung und Qualität der Daten zur Verfügung gestellt werden. Zusätzlich kann die Datenmodell-Komponente weitere Funktionalitäten zur Unterstützung rechnergestützter Prozesse anbieten, wie z.B. die für Bewertung erforderliche Variantenbildung durch das Klonen oder einen Persistenzmechanismus für eine dauerhafte Speicherung der Daten.

Die Berechnungsprogramme zeichnen sich dadurch aus, dass sie in bestimmten Phasen des Gebäudelebenszyklus benötigt und daher nur bei Bedarf aktiviert werden. Sie sollen daher als funktionale Komponenten im System angebunden werden, so dass sie ihre Funktionalität als Dienste zur Verfügung stellen. Die Berechnungskomponenten sind zustandslos und ihre Lebensdauer beschränkt sich auf jeweils eine Berechnung. Die Berechnungsergebnisse sind meistens Zeitreihen. Werden die Zeitreihen als Sammlungen nach der Zeit geordneter, numerischer Werte betrachtet, so ist es ohne Bedeutung, ob ihre Werte Ergebnisse einer Messung oder einer Rechnung sind. Dies eröffnet neue Perspektiven für den Einsatz des Datenmodells während des Betriebs einer Anlage. So können etwa Vergleiche zwischen Soll- und Ist-Werten auf Basis von Simulationen und Messungen durchgeführt werden.

Durch eine klare Trennung der Komponente Gebäude- und Anlagedatenmodell von den Berechnungskomponenten entsteht auch im Sinne moderner Informationstechnik ein Komponentensystem, das dem Gebäudelebenszyklus entsprechend konfiguriert werden kann. Dabei müssen die bestehenden Berechnungsprogramme nicht durch neu entwickelte Berechnungskomponenten ersetzt werden, sondern die existierenden Altlastsysteme (Legacy Systems) können über sog. Adapter weiter genutzt werden.

---

[4] Ein logische Gruppierung benachbarter Räume mit vergleichbaren thermischen Eigenschaften wird als thermische Zone bezeichnet.

Damit diese Ideen umgesetzt werden, wird in dieser Arbeit ein komponentenbasiertes Framework zur Bewertung heiz- und raumlufttechnischer Anlagen vorgeschlagen. Das Framework basiert auf der Komponententechnologie von Microsoft. Diese Wahl wird im folgenden Kapitel begründet. Das Framework selber wird in Kapitel 4 vorgestellt. In Kapitel 5 wird gezeigt, dass und wie auf das Framework aufbauend eine Vielzahl unterschiedlicher Anwendungen entwickelt werden kann.

3 Komponentenbasierte Softwaresysteme

Die Idee, Systeme durch das Zusammensetzen kleiner Teile zu bauen, gehört zu den Grundlagen der Ingenieursdisziplinen. Seit Parnas [35] wird dieses Verfahren im Software Engineering funktionale Dekomposition genannt. Dieses funktionale Herunterbrechen ist ein wichtiges Mittel zur Verbesserung der Handbarkeit und zur Erhöhung der Übersichtlichkeit eines Systems. Es ermöglicht im Ingenieurwesen, dass bereits bestehende Teile wiederverwendet werden und die Herstellung einzelner Teile von Drittanbietern übernommen werden kann.

Im Bereich der Softwareentwicklung wurden die Vorteile der funktionalen Dekomposition bereits Ende der 60 Jahre erkannt. Auf der ersten großen Konferenz zum Thema Softwareentwicklung, der „1st NATO Conference on Software-Engineering“ im Jahr 1968, formulierte McIllroy die Bedeutung einer Komponentenindustrie wie folgt: „My thesis is that the software industry is weakly founded, in part because of the absence of a software component industry“ [36]. Spätestens seit damals gab es das Bestreben, Softwaresysteme in modulare Einheiten zu zerlegen. Der gegenwärtige Stand dieser Entwicklung, eine Vision der Wiederverwendung von Codeteilen, wird als **die komponentenbasierte Softwareentwicklung** bezeichnet. In dieser Vision werden Anwendungen durch das Zusammensetzen kommerziell gefertigter, plattformunabhängiger Komponenten erstellt (der „LEGO-Stein Art“ der Softwarekonstruktion).

3.1 Komponenten

Die Definition dessen, was unter einer Komponente zu verstehen ist, ist in der Literatur nicht einheitlich. Für eine Diskussion verschiedener Ansätze sei auf Szyperski [8] verwiesen. Im allgemeinen wird unter einer Komponente eine für sich selbst stehende Einheit verstanden, die eine bestimmte Funktionalität über wohldefinierte Schnittstellen verfügbar macht, vermarktbar ist und mit anderen Komponenten zu einem Softwaresystem kombiniert werden kann.

Die Spezifikation der Schnittstellen und die Implementierung von Komponenten erfolgen unabhängig voneinander. Die Schnittstelle legt nicht nur ein Serviceinterface fest, das die Komponente selber zur Verfügung stellt, sondern sie definiert auch Dienste, welche die Komponente von ihrer Umgebung erwartet. Die Schnittstellen können mit Verträgen verglichen werden, welche jegliche Kontextabhängigkeiten einer Komponente von ihrer Umgebung vollständig beschreiben. Benötigt z.B. eine Komponente einen Persistenzservice, dann gehört diese Abhängigkeit auch zu der Schnittstelle dieser Komponente und muss explizit aufgeführt werden. Nur wenn Schnittstellen verbindlich spezifiziert sind, können sie von mehreren Herstellern unabhängig implementiert werden. Dies wiederum ist die Voraussetzung für die Entstehung nennenswerter Märkte für Komponenten.

Der Rahmen für die verbindliche Festlegung von Schnittstellen wird von Komponentenframeworks angeboten. Jede Softwarekomponente ist damit ein Element

eines Komponentenframeworks, das selber auf Kompositionsregeln einer Komponententechnologie aufsetzt (vgl. Abbildung 4). Obwohl jede einzelne Komponente eine eigene Funktionalität besitzt und individuell benutzt werden kann, liegt ihr wahrer Wert in der Fähigkeit, mit anderen Komponenten kombinierbar zu sein. Die Komponente muss daher immer im Zusammenhang mit der ihr zugrundeliegenden Softwarearchitektur des Komponentenframeworks betrachtet werden. Der Grundgedanke ähnelt dem der Tonarten bei der Komposition von Musik [37]. Komponenten werden – wie die Noten eines Notenblattes – kombiniert bzw. komponiert, mit der Absicht ein neues Gesamtwerk, eine Anwendung, zu schaffen. Eine Komponente gehört dabei immer zu einem bestimmten Gesamtsystem (Framework), wie auch einzelne Noten zu einer bestimmten Tonart gehören.

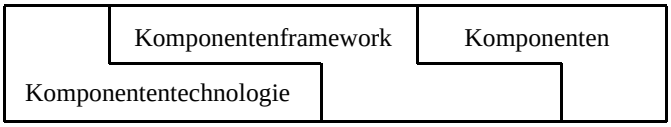


Abbildung 4: Komponenten, Komponentenframeworks und -technologie

Durch Komponenten wird weder ein neuer Objektbegriff eingeführt, noch werden die Objekte in ihrer bisherigen Bedeutung ersetzt. Unter Komponenten versteht man vielmehr eine neue Art, die Softwareentwicklung zu betreiben [38]. Das Ziel besteht darin, Komponenten unabhängig voneinander zu entwickeln, ohne Kenntnis ihres Quellcodes zu integrieren und ohne unerwünschte globale Effekte auf ihre Umgebung ersetzen zu können [8]. Eine gut strukturierte, flexible, objektorientierte Anwendung ist deshalb auch komponentenorientiert.

3.2 Komponententechnologien

Die Kompositionsregeln eines Komponentenframeworks hängen von einer Vielzahl der Faktoren ab. Dabei ist die Wahl der eingesetzten Komponententechnologie entscheidend (vgl. Abbildung 4). Die heute heranreifende Komponententechnologien (COM, CORBA, JavaBeans) bieten gute Bedingungen, um technologisch kompatible Komponenten zu bauen. Dieses Kapitel beschäftigt sich näher mit diesen Technologien. Dies geschieht zunächst unter dem Gesichtspunkt der Verteilung. Die Verteilung ermöglicht die Komposition von Komponenten, welche in eigenständigen Programmen – die auch auf unterschiedlichen Knoten eines Rechnernetzwerks plaziert sein können – ablaufen und erst im Zusammenspiel eine Anwendung ergeben. In Rahmen dieser Arbeit sind aus dem Bereich der Verteilung insbesondere die Arten von Schnittstellen, ihre Implementierungen und die Modelle, wie Komponenten über diese Schnittstellen mit anderen verbunden werden, von Interesse und werden deshalb vergleichend dargestellt. In der Praxis der komponentenbasierten Softwareentwicklung reicht der Aspekt der Verteilung allein nicht aus, um leistungsfähige Anwendungen zu bauen. Es sind zusätzliche Dienste, welche grundsätzliche Problemstellungen für die Entwickler

komponentenbasierter Systeme lösen, notwendig. Jede Komponententechnologie bietet dazu eine Reihe von Diensten an, die ebenfalls vergleichend beschrieben werden.

3.2.1 Komponententechnologie von Microsoft: COM, OLE und ActiveX

In diesem Kapitel wird das Microsoft Component Object Model COM vorgestellt. Dabei werden die fundamentalen Konzepte dieser Technologie betrachtet, indem untersucht wird, wie COM in einer komponentenbasierten Umgebung funktioniert. Das Hauptaugenmerk wird dabei auf die Funktionsweise von COM unter Microsoft Windows NT 4.0 gelegt. In Microsoft Windows 2000 ändern sich einige Dinge – diese Änderungen werden in der Schlussbetrachtung dieses Kapitels besprochen.

Verteilung

Zu den wichtigsten Bestandteilen der Verteilungsmechanismen von COM gehören Schnittstellen, die Implementierung der Schnittstellen und die logische Zusammenfassung von COM Objekten zu Apartments.

Eine primäre Eigenschaft von COM ist die Fähigkeit Funktionalität über transiente Schnittstellen nach außen zu exportieren. Auf der binären Ebene (siehe Abbildung 5, linke Seite) wird jede COM Schnittstelle durch einen Schnittstellenzeiger repräsentiert. Die Klienten greifen nur auf den Schnittstellenzeiger zu und niemals auf die eigentliche Implementierung. Der einzige Teil, der von COM spezifiziert wird, ist ein Zeiger, der im ersten Feld jedes Schnittstellenzeigers gespeichert wird. Dieser zeigt auf eine virtuelle Funktionstabelle, welche ihrerseits Zeiger auf eigentliche Schnittstellenmethoden hält.[5]

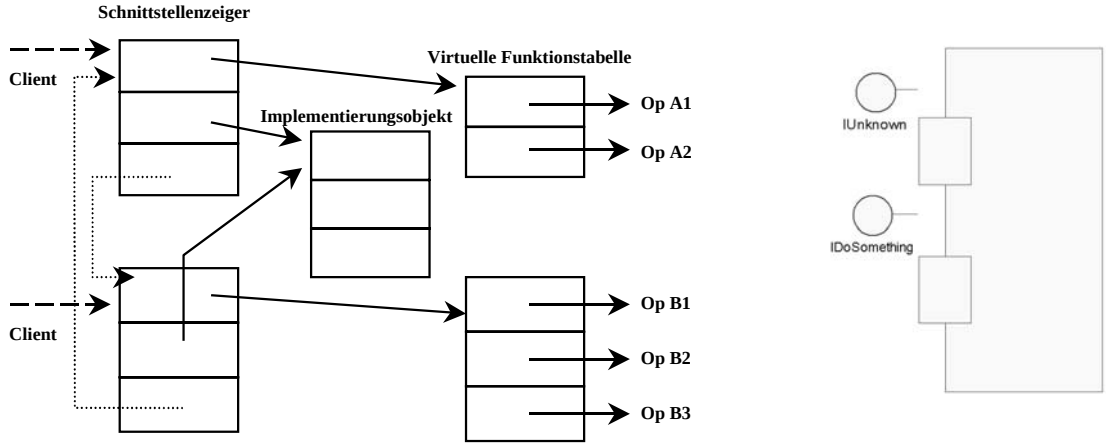


Abbildung 5: Binäre und Lollipop-Repräsentation einer COM Komponente

[5] Derselbe Mechanismus wird auch vom internen C++-Modell benutzt, um virtuelle Operationen zu implementieren [39]. Zu diesem Zweck werden die Funktionsaufrufe auf Objekte über virtuelle Funktionstabellen umgelenkt, was eine Bindung abstrakter Schnittstellen und ihrer konkreten Implementierung erst zur Laufzeit ermöglicht.

Eine COM Komponente kann beliebig viele Schnittstellen haben. Die linke Seite der Abbildung 5 zeigt eine COM Komponente, welche zwei Schnittstellen hat, repräsentiert durch entsprechende Schnittstellenzeiger. Die Implementierung dieser Schnittstellen wird von einem Implementierungsobjekt getragen. COM spezifiziert weder, wie viele Objekte sich an die Implementierung der Schnittstellen einer COM Komponente beteiligen sollen, noch, wie die Implementierung erfolgen soll. Das Implementierungsobjekt wird beim Aufruf der Schnittstellenmethoden an diese übergeben, womit sie einen Zugang zu den Objektdaten bekommen. Die gestrichelten Linien zwischen Schnittstellenzeigern werden intern benutzt, um die Navigation von einem Schnittstellenzeiger zum andern zu ermöglichen. Zum Navigieren besitzen COM Schnittstellen die *QueryInterface* Methode. Die *QueryInterface* Methode nimmt den Namen einer Schnittstelle, prüft, ob für die aktuelle COM Komponente eine solche Schnittstelle existiert und gibt bei Erfolg den entsprechenden Schnittstellenzeiger zurück. Auf der Ebene der COM Schnittstellen werden zur Benennung der Schnittstellen Interface Identifiers (IID) verwendet. Ein IID ist ein GUID (Global Unique Identifier), also eine 128-Bit-Zahl, die global eindeutig ist. COM bietet die *QueryInterface* Methode über die *IUnknown* Schnittstelle, die verbindlich von jeder COM Komponente unterstützt werden muss. Die Schnittstelle heißt *IUnknown*, weil der Klient, wenn er einen Zeiger davon enthält, nichts über die Komponente weiß, die sich hinter dem Zeiger verbirgt. Die rechte Seite der Abbildung 5 zeigt die sog. Lollipop-Notation einer COM Komponente mit zwei Schnittstellen, der obligatorischen *IUnknown* und der eigenen *IDoSomething*. Neben der *QueryInterface* deklariert die Schnittstelle *IUnknown* zwei weitere Methoden *AddRef* und *Release*, welche die Lebensdauer der Komponente kontrollieren. Zu diesem Zweck benutzt die Komponente einen Referenzzähler, der von den Methoden *AddRef* und *Release* um eins erhöht bzw. verringert wird. Erreicht der Referenzzähler Null, so entfernt sich die Komponente von selbst.

Die Trennung der Schnittstelle und Implementierung ist für COM fundamental. COM legt eine eigene Sprache für die Beschreibung von Schnittstellen fest, die Microsoft Interface Definition Language MIDL. Die MIDL beschreibt den Aufrufkontext einer Komponente so genau, dass er überall eingerichtet werden kann, wo er benötigt wird. Ist die Schnittstelle einer COM Komponente festgelegt, dann muss sie mit derer Implementierung verknüpft werden. COM Klassen, auch Co-Klassen genannt, sind Code, welcher COM Schnittstellen implementiert. Eine einfache COM Klasse kann mehrere Schnittstellen implementieren. Ein vollständiges ActiveX-Steuerelement implementiert mehr als ein Dutzend COM Schnittstellen. Die Instanzen der COM Klassen werden als COM Objekte bezeichnet.

COM Klassen sind immer mit Klassenobjekten gepaart. Ein Klassenobjekt ist eine Art Metaklasse der COM Klasse. Die Klassenobjekte sind statisch und global. Ihre Lebensdauer beginnt gleichzeitig mit der des COM Servers und überschreitet die Lebensdauer der COM Objekte, die sie repräsentieren. Diese Langlebigkeit macht die Klassenobjekte zum idealen Ort, um statische Daten zu speichern oder eine statische Schnittstelle zur Erzeugung der COM Objekte zu implementieren. Die Erzeugung der COM Objekte wird durch die Implementierung der Schnittstelle namens *IClassFactory* realisiert.

Der COM Server ist ein binäres Modul, in dem eine oder mehrere COM Komponenten enthalten sind. Eines der Hauptmerkmale von COM ist, dass zwei fundamentale Modelle unterstützt werden:

- Ein prozessinternes Modell (DLL), bei denen der Klient und das Objekt denselben Adressraum nutzen.
- ein prozessexternes Modell (EXE), bei denen der Klient und das Objekt in unterschiedlichen Adressräumen ausgeführt werden.

Ein weiteres Merkmal von COM ist, dass Klienten prozessinterne Objekte genau so aufrufen können, wie entfernte Objekte. Die Fernbearbeitungsschicht ist wohldefiniert und für den Klienten transparent. Hierzu stellt Microsoft das Distributed Component Object Model DCOM zur Verfügung, ein Netzwerkprotokoll höherer Ebene. Die Implementierung von DCOM [40] stützt sich dabei auf das Distributed Computing Environment (DCE) der Open Software Foundation, einem Standard für entfernte Prozeduraufrufe (DCE-RPC) [41]. Da DCOM einen objektorientierten Ansatz verfolgt, hat Microsoft objektorientierte RPCs (ORPCs) auf Basis der DCE-RPCs entwickelt. ORPCs unterstützen sowohl verbindungsorientierte als auch verbindungslose Kommunikationsprotokolle. Zielobjekte lassen sich dabei über sog. Bindungsinformationen adressieren. Die Bindungsinformation setzt sich aus dem verwendeten Transportprotokoll, der Hostadresse und der Portnummer zusammen.

Ein Apartment ist eine logische Gruppierung von COM Objekten, welche konkurrierende Eigenschaften haben. Demzufolge befinden sich COM Objekte, die in der Lage sind ihre internen Daten von konkurrierenden Zugriffen zu schützen, in einem Typ von Apartment, während die Objekte, die dazu nicht in dieser Lage sind, sich in einem anderen Apartmenttyp befinden. Die beiden unter COM am häufigsten verwendeten Apartmenttypen sind: das Single Threaded[6] Apartment (STA) und das Multithreaded Apartment (MTA). Bei einem STA ist das Threading-Verhalten so gestaltet, dass immer nur ein Ausführungspfad im Apartment existiert. Dennoch kann ein Server Multithreading implementieren, indem mehrere STAs erzeugt werden. Da ein STA nur einen Ausführungspfad besitzt, besitzen mehrere STAs demzufolge mehrere Ausführungspfade. Versuchen mehrere Ausführungspfade mit demselben STA zu kommunizieren, stellt COM sicher, dass ein Ausführungspfad nach dem anderen bedient wird.

Wie der Name Multithreaded Apartment (MTA) schon impliziert, kann dieses mehrere Ausführungspfade beinhalten. Daher kann maximal ein MTA pro Prozess existieren. Die MTA Architektur ist so konstruiert, dass COM einen Thread-Pool mit mehreren Ausführungspfaden unterhält. Wenn ein Klient eine Methode eines Objekts aus einem MTA aufruft, schaut COM im Thread-Pool nach, ob ein freier Ausführungspfad zur Verfügung steht. Ist das der Fall, wird diesem Ausführungspfad der Aufruf zur eigentlichen Ausführung der Objektmethode weitergegeben. Dies bedeutet, dass mehrere

[6] Engl.: thread – der Faden. Gemeint ist hier der Ausführungspfad eines Programmes.

Aufrufe zur selben Zeit das MTA Objekt erreichen können. Es liegt im Aufgabenbereich von COM dafür zu sorgen, dass immer genügend freie Ausführungspfade für die Methodenaufrufe im Thread-Pool zur Verfügung stehen. Da ein MTA Objekt Methodenaufrufe von mehreren Ausführungspfaden zur gleichen Zeit erhalten kann, sind die Inhalte der Instanzvariablen ohne geeignete Maßnahmen in diesem Umfeld nicht mehr sicher. Somit stellt eine saubere Synchronisation der Variablenzugriffe eine wichtige Grundlage für das einwandfreie Funktionieren der MTA Objekte dar. Für die Synchronisation der Ausführungspfade benutzt COM die gleichen Threading-Mechanismen wie das Win32-System.

Ein weiteres, mit dem Microsoft Transaction Server eingeführtes Modell, das Rental Threaded Apartment, erlaubt – ebenso wie das MTA Modell – mehreren Ausführungspfaden das Apartment benutzen zu können. Der Unterschied zum MTA Modell ist, dass der Ausführungspfad, der das RTA betritt, eine apartmentweite Sperre anfordert. Durch diese Sperre verhindert der aktuelle Ausführungspfad, dass andere Ausführungspfade das Apartment konkurrierend benutzen können.

COM Dienste

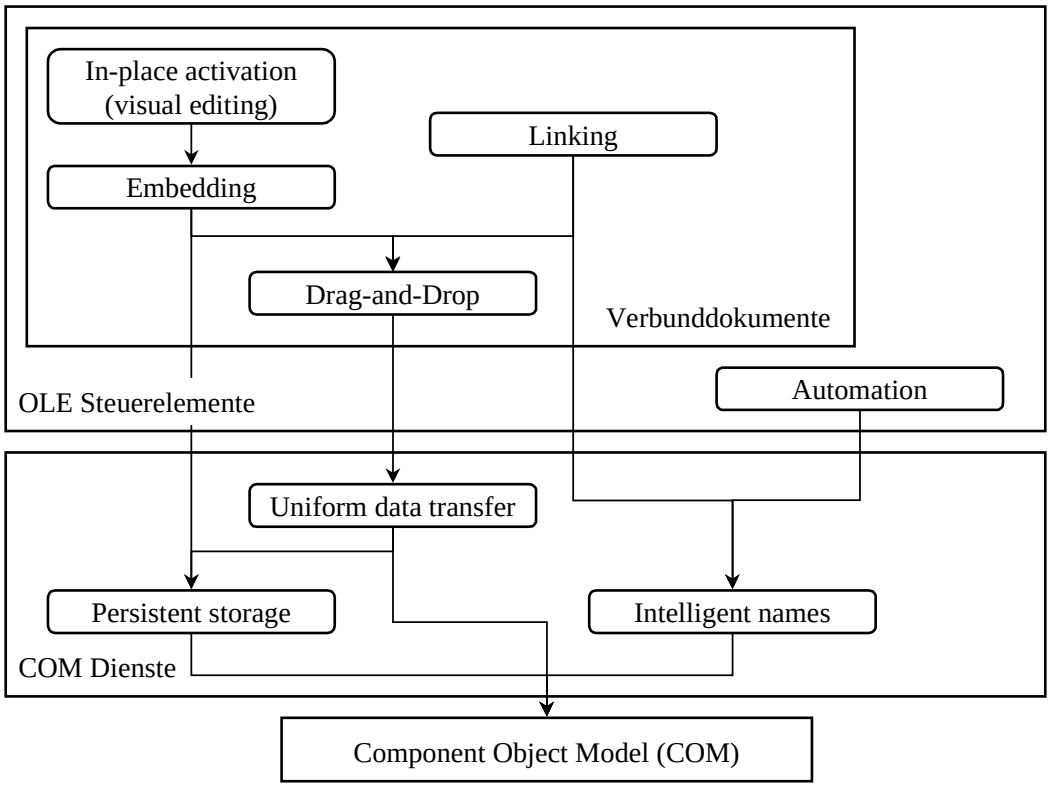


Abbildung 6: OLE-Steuerelemente als Grundlage von ActiveX

Auf COM aufbauend, ursprünglich zur Unterstützung der Verarbeitung von Verbunddokumenten, wurde von Microsoft das Object Linking and Embedding-Protokoll (OLE-Protokoll) [42] entwickelt. In einem OLE Verbunddokument können die

(persistenten) Daten eines Objektes, auf zweierlei Art abgelegt werden (vgl. Abbildung 6): durch Einbettung (embedding) oder durch Verknüpfung (linking). Bei der Einbettung von Objektdaten werden diese in das Verbunddokument kopiert und dort bearbeitet. Bei der Verknüpfung von Objekten werden dessen Daten außerhalb des Verbunddokumentes abgelegt. Innerhalb des Verbunddokumentes wird also lediglich eine Verknüpfung (meist ein Dateipfad) auf eine weitere Datendatei gespeichert.

Das OLE-Protokoll spezifiziert eine Vielzahl der Schnittstellen, welche von Objekten realisiert werden müssen. Solche Objekte werden als OLE Steuerelemente bezeichnet. Die OLE Steuerelemente können in einer Vielzahl verschiedener Umgebungen eingesetzt werden. Dementsprechend treten sie in vielen verschiedenen Formen auf, so beispielsweise als Textboxen, Schaltflächen, Uhren, Audio- und Video-Player etc. Ursprünglich implementierten die OLE Steuerelemente nahezu das gesamte OLE-Einbettungsprotokoll. Dabei wurden die folgenden Funktionen unterstützt:

- Die Automatisierungs- und Scripting-Dienste von OLE (*Automation*) erlauben die Steuerung der Objekte über Klienten, die in verschiedenen Programmier- oder Skriptsprachen implementiert sein können. Diese Steuerungsprogramme können eigenständige Programme darstellen, oder Bestandteil anderer Anwendungen sein, wie z. B. Makros für Microsoft Word oder Excel, die in speziellen Programmiersprachen geschrieben werden (z. B. Visual Basic for Applications VBA).

- Für die persistente Abspeicherung von Objekten sind zum einen spezielle Standardschnittstellen wie *IPersistFile* und zum anderen das Konzept der strukturierten Speicherung vorgesehen. Letztere ermöglicht die hierarchische Strukturierung einer Datei ähnlich einem Dateisystem. Mit Hilfe von sog. Monikern lassen sich die Instanziierung und Initialisierung von COM-Objekten in einem Schritt erledigen.

- Einheitlicher Datentransfer (*Uniform Data Transfer*) zwischen den einzelnen Komponenten, der für Drag-and-drop usw. genutzt wird.

- Die Objekte, die mit dem Dokument verknüpft sind, müssen benannt werden. Das OLE Konzept bietet eine Schnittstelle (*Intelligent Names*) an, um Verknüpfungsinformationen zum eingefügten Objekt zu ermitteln und eine Verbindung zu ihm herstellen zu können.

Als das Internet zu einem bestimmenden Faktor der weiteren Entwicklung wurde, entschied sich Microsoft, COM-basierte Elemente auch in Webseiten einzupflanzen. Nun wurde die Größe dieser Elemente, die sich aus der Menge vorgeschriebener Funktionen ergab, zu einem Problem. Microsoft nahm die Spezifikation der OLE Steuerelemente, änderte den Namen der OLE Steuerelemente in ActiveX Steuerelemente um und gab bekannt, dass alle oben eingeführten Funktionen optional seien [43]. Dies bedeutete, dass unter der neuen Definition für die ActiveX Steuerelemente nur die Anforderung bestand, dass ein Steuerelement auf COM basiert und die *IUnknown* Schnittstelle implementiert. Darüber hinaus besitzen die ActiveX Steuerelemente die Fähigkeit, eine bidirektionale Kommunikation abzuwickeln. Die bidirektionale Kommunikation bedeutet, dass ein

Steuerelement in der Lage ist, seine Klienten über Ereignisse zu informieren. Ereignisse sind nach außen gerichtete Schnittstellen, im Gegensatz zu den eingehenden Schnittstellen, den Eigenschaften und Methoden. Zur Implementierung der Ereignisschnittstellen stellt COM die Technik der Verbindungspunkte zur Verfügung. Um in einer bidirektionalen Kommunikation involviert zu sein, müssen die ActiveX Elemente die Schnittstellen *IConnectionPoint*, *IConnectionPointContainer* implementieren. Die Schnittstelle *IConnectionPoint* ermöglicht es dem Klienten Ereignisse zu abonnieren. Dazu verwenden sie den Verbindungspunkt *IConnectionPoint*, worüber eine Callback-Schnittstelle vom Klienten an den Server übergeben wird. Die Callback-Schnittstellen erhalten die Methoden, die im Fallen eines Ereignisses auf der Klientenseite aufgerufen werden. Zum Erwerben des Verbindungspunktes benutzt der Klient die zweite Schnittstelle *IConnectionPointContainer*.

In der industriellen Softwareentwicklung benötigen Entwickler außer der Bereitstellung einer Komponenteninfrastruktur höhere Dienste für ihre Anwendungen. Zwar kann Microsoft COM an dieser Stelle nicht mit dem breiten Fundus von Objektdiensten aufwarten, der beispielsweise CORBA auszeichnet (siehe Kapitel 3.2.2). Dafür sind die wichtigsten Dienste bereits heute verfügbar:

- Der Microsoft Transaction Server sorgt für die Abwicklung von Transaktionen über mehrere COM-Objekte und darüber hinaus für deren effiziente Ressourcenverwaltung.
- Während DCOM für die Kommunikation entfernte Methodenaufrufe mit synchroner Aufrufsemantik vorsieht, benötigen einige Anwendungen einen Mechanismus, bei dem sich nicht Methodenaufrufe sondern Nachrichten asynchron übermitteln lassen. Für diesen Zweck hat Microsoft den Microsoft Message Queue Server entwickelt.
- Zum Zugriff auf Datenbanken stehen mit ODBC (Open Database Connectivity) und OLE-DB (OLE Database) bewährte Lösungen bereit.

3.2.2 Komponententechnologie von OMG: CORBA und OMA

Die Object Management Group (OMG) ist mit über 900 Mitglieder die weltweit größte Vereinigung von Software-Herstellern und -Anwendern. Bei der Gründung 1989 hat man sich zum Ziel gesetzt, die Verbreitung der Objekttechnologie zu fördern und ein Architekturmodell für verteilte Anwendungen zu schaffen [44]. Da die OMG auf Programmiersprachen-, Implementierungs- und Plattformunabhängigkeit einen großen Wert legt, kam hierfür ein binärer Standard nicht in Frage. Seither sind eine große Zahl von Spezifikationen erarbeitet und veröffentlicht, in denen die Komponenten und Schnittstellen dieses Architekturmodells festgelegt worden sind.

Verteilung

Die Spezifikation der OMG für die Common Object Request Broker (CORBA) beschreibt, wie verteilte Objekte mit Hilfe eines ORB miteinander kommunizieren können. Ein Klient-Objekt kann die Dienste eines Server-Objekts nutzen, indem es sich

mit einer Dienstanforderung (Request) an den ORB wendet (vgl. Abbildung 7). Dazu muss dem Klient-Objekt lediglich die Objektreferenz des Server-Objekts bekannt sein. Der ORB ermittelt aufgrund der übergebenen Objektreferenz zunächst die Lokation des Server-Objekts, startet dort den eigentlichen Operationsaufruf und liefert am Ende dem Klient-Objekt das Ergebnis des Aufrufs zurück. Eine der wesentlichen Aufgaben des ORB ist demnach die Herstellung von Ortstransparenz, denn ein Klient hat keine Information darüber, auf welchem Rechnerknoten sich das Server-Objekt befindet.

Jeder Server-Objekt besitzt Zugang zu einem Objektadapter. Über ihn werden dem Server-Objekt grundlegende Dienste bereitgestellt. Zu den Aufgaben des Objektadapters gehört beispielsweise die Generierung und Interpretation von Objektreferenzen zu einer rechnerinternen Repräsentation eines Objektes. Um Objektreferenzen und Implementierung zueinander zuordnen zu können, benutzt der Objektadapter das Implementation-Repository, in dem sich alle Angaben darüber finden, welche Programme auszuführen sind, wenn eine Dienstanforderung für ein Objekt eintrifft. Außerdem gehört zu Aufgaben des Objektadapters die Überprüfung der Zugangsberechtigungen eines Klienten zu dem aufgerufenen Objekt und die automatische Aktivierung von Serverobjekten, falls für diese gerade keine Repräsentation im Hauptspeicher des Rechners existiert.

Damit die Kommunikation überhaupt stattfinden kann, müssen zwei Anforderungen erfüllt werden:

- Objektschnittstellen müssen in einer gemeinsamen Sprache beschrieben werden.
- Die gemeinsame Sprache muss auf äquivalente Konstrukte der benutzenden Programmiersprache übersetzt werden (Language-Mapping).

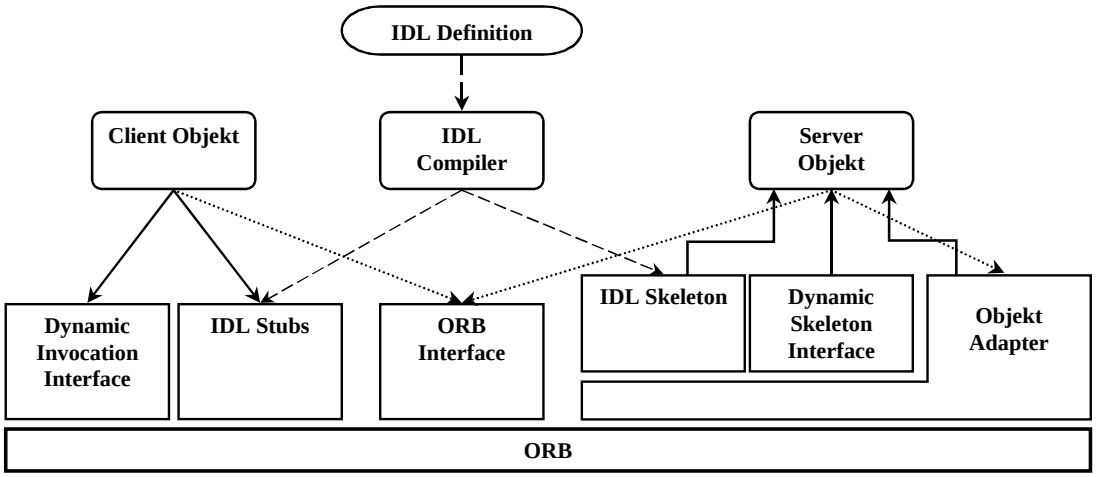


Abbildung 7: Informationsaustausch zwischen dem ORB und den Klienten bzw. den Servern.

Die normierte Sprache Interface Definition Language IDL stellt Mittel zur Beschreibung der Objektschnittstellen zur Verfügung. Bei der IDL handelt es sich um eine mit C++ verwandte Definitionssprache, welche ausschließlich Elemente zur Datenbeschreibung, jedoch keinerlei Anweisungskonstrukte enthält. Auch die Menge der verwendeten Datentypen ist in der IDL gegenüber Programmiersprachen nach dem Prinzip des kleinsten gemeinsamen Nenners eingeschränkt. Die IDL Beschreibung erfolgt auf einem konzeptuellen Niveau, denn keine Programmiersprache kann direkt etwas mit den IDL Definitionen anfangen. Diese werden mittels eines IDL-Compilers in Methoden- oder Funktionsaufrufe der jeweiligen Programmiersprache übersetzt und im Interface-Repository abgelegt. Wie den Konstrukten von IDL äquivalente Konstrukte einer Programmiersprache zugeordnet werden, ist im Dokument mit dem Namen Language-Mapping festgelegt. Es ist jeweils für eine Programmiersprache, denn in jeder Programmiersprache können andere Mechanismen zur Repräsentation von Schnittstellen, zum Aufruf von Operationen oder zum Anzeigen von Programmierausnahmen vorhanden sein. Dies ist nicht nur für objektorientierte Programmiersprachen wie C++, Java, Smalltalk und ADA, sondern auch für C und sogar COBOL und PL/1 der Fall.

Abbildung 7 zeigt, wie Informationen zwischen dem ORB und den Klient-Objekten bzw. den Serverobjekten ausgetauscht werden können. Zum einen besteht die Möglichkeit der Generierung von IDL-Stubs und IDL-Skeletons seitens des IDL-Compilers. Ein IDL-Stub hat die Aufgabe die Aufrufe für ein verteiltes Objekt aufzunehmen und mittels des ORB an das Zielobjekt weiterzuleiten. Die Aufrufe werden auf der Serverseite vom IDL-Skeleton entgegengenommen und direkt in Operationsaufrufe des Server-Objekts umgeleitet. Die IDL-Stubs und IDL-Skeletons sind eine statische Aufrufschnittstelle des CORBA-Systems und als solche erfüllen sie die Anforderung nicht, bei Bedarf Methodenaufrufe dynamisch zur Laufzeit zu bestimmen. Zu diesem Zweck sieht CORBA alternativ zu IDL-Stubs das Dynamic Invocation Interface (DII) vor. Das DII ermöglicht Server-Objekte zu benutzen, über deren Schnittstellen zur Zeit der Programmentwicklung noch keine Aussagen in Form einer IDL-Definition gemacht werden können. Welche Operationen vor einem, bis kurz vor seiner Benutzung völlig unbekannten Server-Objekt überhaupt angeboten werden, erfahren die Klient-Objekte direkt vom ihren Nutzer oder vom Interface Repository. Mit dem DII bekommen sie daher eine Schnittstelle zum CORBA-System, bei der sie erst zur Laufzeit mitteilen müssen, welches Server-Objekt angesprochen und welche seiner Operationen ausgeführt werden soll. Das Server-Objekt kann nicht anhand des empfangenen Aufrufes feststellen, für welche der beiden Möglichkeiten sich der Klient entschieden hat. Als Pendant zum DII, jedoch auf der Seite des Servers, stellt das Dynamic Skeleton Interface (DSI) eine Möglichkeit dar, Server-Objekte für Schnittstellen zur Verfügung zu stellen, die zur Zeit der Entwicklung des Servers noch unbekannt sind. Auf den ersten Blick erscheint es sicher etwas merkwürdig, einen Server programmieren zu wollen, ohne zu wissen, wie die in ihm enthalten Objekte aussehen. Die Lösung besteht aber darin, dass der Server dieses Wissen noch zur Laufzeit erhalten kann. Mit dem DSI muss ein Server eine Funktion bereitstellen, über welche ihm der ORB einen beliebigen Aufruf übergeben kann, ganz egal zu welcher Art von Objekten der Aufruf gehört. Eine ausführlichere Beschreibung der dynamischen Aufrufschnittstelle des CORBA-Systems findet sich in [45].

Das Fehlen eines binären Standards ermöglicht es Softwareherstellern individuelle CORBA-Implementierungen zu entwickeln [46], aber es hat zum Nachteil, dass die ORBs verschiedener Hersteller auf der binären Ebene nicht effizient zusammenarbeiten können. Die ORBs können zusammenarbeiten, wenn

- alle ORBs (neben einem eigenen internen) ein einheitliches Protokoll für die Kommunikation unterstützen. Wie ein solches Protokoll prinzipiell aussehen müsste, hat die OMG mit General Inter-ORB Protocol (GIOP) beschrieben .
- jeder ORB sein eigenes Protokoll verwendet. Es werden aber Brücken zwischen den Systemen geschaffen, in denen eine Protokollumwandlung stattfindet.

Die Rolle gemeinsamer Sprache zwischen ORBs kann das als Spezialisierung des GIOP entworfene Internet Inter-ORB Protocol (IIOP) übernehmen. Es beruht auf der im Internet üblichen Protokollfamilie TCP/IP und ist im CORBA-Standard 2.0 detailliert beschrieben [47]. Durch die Benutzung von Brücken erhält jeder ORB ein höheres Maß an Autonomie. Man unterscheidet dabei Vollbrücken, die direkt das Protokoll des einen ORB in das Protokoll eines anderen ORB umsetzen und Halbbrücken, die das Protokoll des einen ORB zunächst in ein allgemeines, weitverbreitetes Protokoll (z.B. IIOP) überführen und dann auf anderer Seite dieses allgemeine Protokoll in das spezifische Protokoll des Ziel-ORB übersetzen.

CORBA Dienste

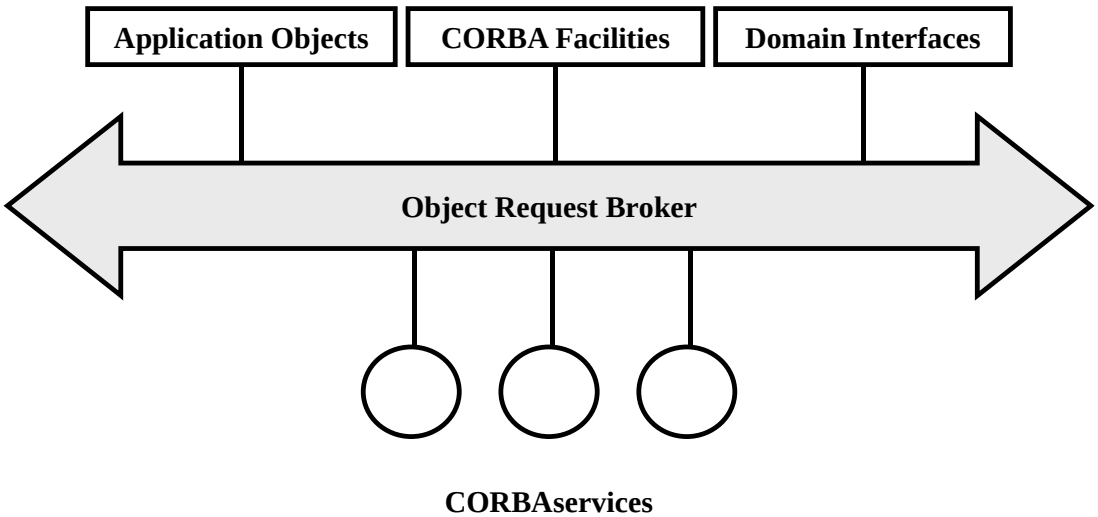


Abbildung 8: Das Architekturmodell – Object Management Architecture (OMA)

Der Fokus der Standardisierung der OMG beschränkt sich nicht nur auf die Kommunikationskomponente CORBA. Ein ORB stellt lediglich einen objektorientierten Mechanismus zum Aufruf entfernter Objektmethoden bereit. Wie schon ausgeführt,

reicht dies in der Praxis der komponentenbasierten Softwareentwicklung nicht aus. Dort sind zusätzlich Dienste notwendig. Deshalb entwickelte die OMG mit der Object Management Architecture (OMA), ein Architekturmodel, nach dessen Vorgaben sich komponentenbasierte Anwendungen leichter erstellen lassen [47]. Neben dem ORB sind in der OMA vier weitere Bereiche vorhanden: CORBAservices, CORBAfacilities, Domain Interfaces und Application Objects (Abbildung 8).

Als **CORBAservices** werden die elementaren Basisdienste der OMA bezeichnet. Gemäß der Definition in [48] handelt es sich um eine Sammlung von Diensten, die zur Implementierung und zum Betrieb eines verteilten Objektverwaltungssystems nützlich und notwendig sind. Jeder CORBAservice deckt nur einen eng abgegrenzten Funktionsbereich ab und ist hochgradig modular – im Idealfall ist jeder dieser Dienste unabhängig von den anderen und kann demnach eigenständig implementiert und genutzt werden. Dahinter steckt das bewährte Wiederverwendungsprinzip, bei dem aus monolithischen Applikationen immer mehr Funktionalität in spezialisierte, standardisierte Softwarekomponenten ausgelagert wird. Beachtung findet bei der Funktionspartitierung das Bauhaus-Prinzip, wonach jegliche funktionelle Überdeckung zu vermeiden ist.

Im Verlauf der Standardisierung der CORBAservices wurden seit 1992 in mehreren Phasen mittlerweile 15 Dienste in Form umfangreicher Spezifikation niedergelegt. Nachfolgend eine Liste von bereits verabschiedeten Diensten:

- Naming Service zum Auffinden von Objekten im Netz
- Event Service zur Behandlung asynchroner Ereignismeldungen
- Persistent Object Service zur langfristigen Speicherung von Objektzuständen
- Life Cycle Service zur Regelung der Erzeugung und Vernichtung von Objekten
- Concurrency Control Service zur Koordinierung von nebenläufigen Aktivitäten
- Externalization Service zum Ausschreiben von Objektzuständen in einen beliebigen Datenstrom
- Relationship Service zur Verwaltung der Beziehungen zwischen Objekten innerhalb einer ORB-Umgebung
- Transaction Service zur Realisation von flachen und geschalteten Transaktionen
- Query Service zur Manipulation von Objektmengen mit Hilfe gegebener Anfrageprädikate
- Licensing Service zur Unterstützung verschiedener Lizensierungsverfahren
- Property Service zur Auszeichnung bereits existierender Objekte mit zusätzlichen Eigenschaften, welche nicht zur eigentlichen Schnittstelle gehören (Metadaten)

- Time Service zur Synchronisation von Uhren
- Security Service zum Schutz von unerlaubter Benutzung
- Trading Object Service zum Auffinden der Objekte zur Laufzeit und damit einer späteren Bindung von Klient- an Server-Objekte.
- Object Collection Service zur Verwaltung von verschiedenen Typen von Objektmengen

Neben den verabschiedeten Spezifikationen sind Ausschreibungen für weitere Services in Vorbereitung. Sie umschließen Bereiche wie Archive Service, Backup/Restore Service, Internationalization Service, Logging Service, Recovery Service und Replication Service. Auch wenn viele CORBAservices von der gebotenen Funktionalität sehr attraktiv sind, ist doch ihre tatsächliche Verfügbarkeit in Form stabiler Produkte sehr eingeschränkt. Nur wenige Hersteller bieten zusammen mit ihrer ORB-Implementierung ein breites Spektrum passender CORBAservices an.

Erheblich mehr Funktionalität als CORBAservices, bieten die **CORBAfacilities**. CORBAfacilities sind Komponenten in der OMA, die auf einem hohen Abstraktionsniveau anwendungsnahe Funktionen realisieren. Diese Komponenten besitzen ein klar umrissener Funktionsumfang und können durch Konfigurationsmechanismen an verschiedene Aufgabenstellungen angepasst und somit in mehreren Anwendungen eingesetzt werden. Im Unterschied zu Domain Interfaces, die branchenspezifisch ausgerichtet sind, handelt es sich bei CORBAfacilities um horizontale, fachneutrale Komponenten. Die Beispiele für fachneutrale Dienste sind der Ausdruck von Dokumenten (Printing Facility), die Verwaltung von Verbunddokumenten (Compound Document Facility), aber auch Dienste der Benutzerschnittstelle wie Browsing oder Rendering. Die CORBAfacilities teilen sich gemäß ihrer in [49] beschriebenen Architektur in vier große Bereiche auf:

- User Interface Common Facilities beschreiben Schnittstellen, welche mit der Benutzerschnittstelle von Anwendungen, insbesondere mit Präsentationen von Informationen zu tun haben.
- Information Management Common Facilities befassen sich mit dem Aspekt der Datenhaltung.
- System Management Common Facilities umfassen administrative Dienste zur Verwaltung, Konfiguration und zum Betrieb verteilter Objektverwaltungssysteme.
- Task Management Common Facilities sind Infrastrukturdienste, welche den Benutzer bei seiner Arbeit unterstützen.

In früheren Entwicklungen der OMA wurden CORBAfacilities in horizontale (Horizontal CORBAfacilities) und vertikale (Vertical Market Facilities) unterschieden. Als horizontal wurden fachneutrale Dienste bezeichnet. Vertikale Dienste sind dagegen auf konkrete Bereiche spezialisiert, z.B. für industrielle Anwendungen oder Simulationen. In aktuellen Dokumenten wird diese Trennung aufgegeben, und vertikale Dienste werden nun

Domain Interfaces genannt. Momentan gibt es innerhalb des Domain Technology Committee Arbeitsgruppen zur sechs Fachbereichen: Business Object, Electronic Commerce, CORBAfinancials, CORBAmanufacturing, CORBAmed und CORBAtel. Einen wichtigen Erfolg konnte die OMG im Bereich des Fachgebietes Anwendungsentwicklung verbuchen, das von der Analysis and Design Platform Task Force bearbeitet wird. Dort wurde im November 1997 die Spezifikation für die Object Analysis and Design Facility [50] verabschiedet, bei der sich im wesentlichen um die Unified Modeling Language UML handelt. Die UML ist eine graphische Notation für die Visualisierung, die Spezifikation, das Konstruieren und die Dokumentation von Artefakten innerhalb eines Software-Entwicklungsprozesses [51]. Die UML gewinnt in der Softwareindustrie immer mehr an Bedeutung und wird auch in dieser Arbeit für die Modellierung und Darstellung verwendet. Eine ausführliche Darstellung der UML würde aber den Rahmen der Arbeit sprengen, so wird an dieser Stelle nur an die Literatur zum Thema UML [51], [52] und [53] verwiesen.

Hinter einem **Application Object** verbirgt sich ein konkretes Produkt oder ein System, wie es vom Endbenutzer gesehen wird. Die Application Objects lösen fachbezogene Aufgaben aus einer Anwendungswelt und können nicht ohne weiters in einem anderen Kontext wiederverwendet werden. Aus diesem Grund werden die Application Objects von der OMG weder spezifiziert noch standardisiert.

3.2.3 Komponententechnologie von Sun: Java und JavaBeans

Im Jahre 1995 wurde von der US-amerikanischen Firma Sun Microsystems [54] mit der objektorientierten Programmiersprache Java ein interessantes Konzept vorgestellt, welches den Benutzern ermöglichen soll, unabhängig vom Betriebssystem Programme zum Ablauf zu bringen. Diese Entwicklung wurde von WWW-Anbietern sofort aufgenommen, da dadurch Java-Programme auf sämtlichen Rechnern ausgeführt werden können, die über Java-fähige WWW-Browser verfügen. Java-Programme können zusätzlich wie herkömmliche Programme auch ohne WWW-Browser ablaufen, sofern im Betriebssystem das Java-Laufzeitsystem installiert ist.

Der Java Development Kit (JDK) setzt sich aus zwei Definitionskomponenten zusammen:

- einer virtuellen Maschine und ihrem Befehlsvorrat und
- einer an C bzw. C++ angelehnten, objektorientierten Programmiersprache [55], die über Klassenbibliotheken verfügt.

Java ist eine interpretierende Sprache, allerdings wird vom Interpreter, der virtuellen Maschine, nicht der Quellcode sondern ein Bytecode interpretiert. Gemäß der plattformunabhängigen Spezifikation arbeitet die virtuelle Maschine auf allen Plattformen und Hardwarearchitekturen gleich. Da eine Interpretation von Java Bytecode relativ langsam ist, kann ein sog. Just-in-Time (JIT) Compiler verwendet werden, der den Java-Code in die Maschinensprache der verwendeten Plattform übersetzt und in der Regel wesentlich schneller arbeitet als ein Interpreter. Außerdem besitzt Java ein Java-Native-

Interface-API zur Anbindung von Systemaufrufen oder binären Programmteilen, allerdings auf Kosten der Portabilität.

Verteilung

Zum Zwecke der Verteilung steht bei Java ein Remote-Method-Invocation-API (RMI) zur Verfügung. RMI ermöglicht eine einfache Implementierung von verteilten homogenen Java-Anwendungen. RMI verwendet wie CORBA das Klient-Server-Modell. Der Server registriert sich in der Registrierungsdatenbank (rmiregistry), über die der Klient die Objektreferenz des Servers ermitteln kann. Der Klient kann über einen Stub Methodenaufrufe an einen Server absetzen. Der Stub reicht den Methodenaufruf an den Remote Reference Layer weiter, welcher den entsprechenden Server lokalisiert und den Request über dessen Skeleton an ihn übergibt. Als Kommunikationsprotokoll wird das RMI Wire Protocol verwendet, das auf TCP/IP basiert. Der Klient-Stub und der Server-Skeleton werden über den Compiler (rmic) aus den Bytecode des Servers erzeugt. Abbildung 9 zeigt die Systemarchitektur des RMI, die in drei voneinander vollständig unabhängigen, eigenständigen Ebenen der Stub/Skeleton Layer, Remote Reference Layer und Transport Layer aufgeteilt ist.

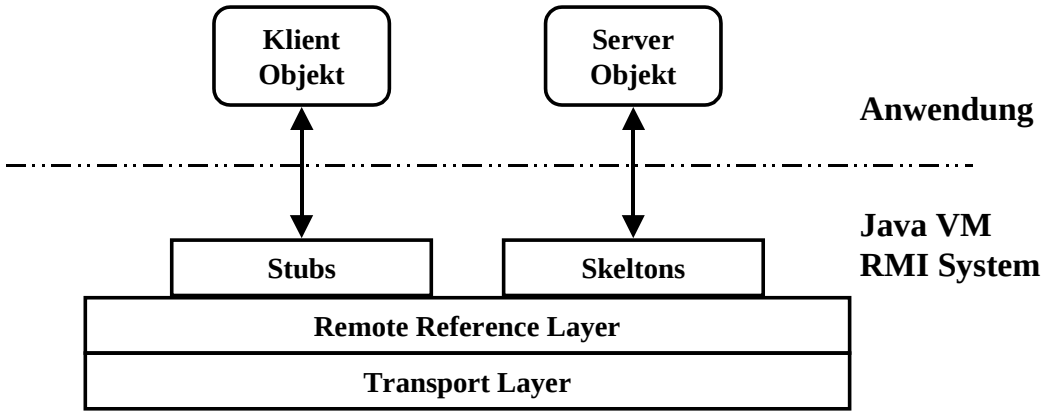


Abbildung 9: Systemarchitektur des Remote Methode Interface

Java Dienste

JavaBeans wurden von Sun als ein Komponentenmodell konzipiert [56], das Softwareentwicklern ermöglichen soll, wiederverwendbare Komponenten in Java zu erstellen. Endanwender können diese Komponenten einfach anpassen und zu Anwendungen zusammenfügen. Darüber hinaus kann ein JavaBean als eine funktionale Einheit mit einem bestimmten Verwendungszweck in einer Entwicklungsumgebung visuell verändert werden. Dazu muss das JavaBean die Manipulation des Aussehens und Verhaltens erlauben. Wird der Entwickler mit Wizards unterstützt, kann der Quellcode eines JavaBeans ziemlich umfangreich ausfallen, daher ist eine klare Trennung zwischen Entwicklungs- und Laufzeit erforderlich. So wird sichergestellt, dass ein JavaBean

ausgeliefert werden kann, ohne dass die für die Entwicklung bestimmten (und daher nicht mehr benötigten) Programmteile mitgeliefert werden müssen.

Die Eigenschaft der Plattformunabhängigkeit ist bei JavaBeans – wie bei Java – weiterhin vorhanden, um die Integration in vorhandene Komponentenmodelle unterschiedlicher Plattformen zu ermöglichen. Im folgenden sind die wichtigsten Begriffe, Eigenschaften und Mechanismen der JavaBean-Technologie erläutert [56]:

- *Properties* sind Attribute eines JavaBeans, die über ihren Namen angesprochen werden.
- *Methods* sind Methoden, welche von der Klasse, die das Bean implementiret, für andere Komponenten sichtbar gemacht werden.
- *Events* sind Ereignisse, welche von Komponenten verarbeitet werden.
- *Introspection* ist ein Prozess des Sichtbarmachens der Properties, Methoden und Events, welche eine Komponente unterstützt.
- *Reflection* ist die Fähigkeit, zur Laufzeit Informationen über die Felder, Konstruktoren und Methoden einer beliebigen Klasse zu erhalten.
- *Customization* ermöglicht die Anpassung der Attribute und des Verhaltens einer Komponente für eine konkrete Anwendung.
- *Persistence* ist ein Mechanismus zur Serialisierung, d.h. zum Speichern und Restaurieren des Zustandes eines Objektes. JavaBeans benutzen den portablen Mechanismus der Serialisierung von Java.
- *Multithreading* ist ein Mechanismus, welcher mehreren Programmausführungspfaden erlaubt, eine Komponente zu benutzen. JavaBeans benutzen den Multithreading-Mechanismus von Java.
- *Security* ist ein Sicherheitsmechanismus. JavaBeans bieten dieselben Sicherheitsmechanismen wie Java.

Eine etwas tiefergehende Einführung in die JavaBean-Technologie findet sich in [57] und [58]. Im Zuge der Weiterentwicklung von Java wurde das Java Development Kit ausgebaut. Im folgenden werden zwei Erweiterungen, die in Zusammenhang mit der JavaBean-Technologie stehen, kurz vorgestellt: JavaBeans Activation Framework (JAF) und InfoBus-System.

Zur Vereinheitlichung der Verwendung der JavaBean-Technologie spezifizierte Sun das JavaBeans Activation Framework (JAF) [59]. Das JAF stellt eine Ergänzung zum JavaBean-Komponentenmodell dar, die für solche Entwickler interessant ist, welche eigenständige Komponenten zum Editieren und Generieren von Daten entwickeln. Dieses Framework stellt ein Modell zur Verfügung, das nach [59] folgendes leistet:

- den Typ beliebiger Daten/Dateien/Dokumente zu bestimmen,
- den Zugriff auf die Daten zu kapseln,

- zu erkennen, welche Operationen für den Zugriff auf die Daten zur Verfügung stehen und
- diejenige Softwarekomponente zu instantiieren, die für die gewünschte Operation über den Daten erforderlich ist.

Durch die Verwendung des JAF wird die zukünftige Kompatibilität sowie die Partizipation an der Entwicklung von Dokument-Viewern und Editoren gewährleistet.

Schon kurze Zeit nach der Vorstellung der JavaBean-Technologie bemerkten die Entwickler von Sun eine Schwachstelle in ihrem Konzept. Man hatte keinen Mechanismus vorgesehen, der den Datenaustausch unter JavaBeans verschiedener Hersteller ermöglicht. Um dies zu ändern, wurde ein weiteres Softwarekonzept, der InfoBus [59], entwickelt, der es erlauben sollte, die Kommunikation zwischen JavaBeans auf einfache Art und Weise zu etablieren. Über einen InfoBus können die JavaBeans direkt miteinander kommunizieren. Voraussetzung dafür ist jedoch eine möglichst große Anzahl von vordefinierten Schnittstellen für den Zugriff auf verschiedene Datenstrukturen und –typen. Die am InfoBus angeschlossenen JavaBeans können Nachrichten versenden und empfangen. Sie werden einer der drei Kategorien zugeordnet: Datenproduzent (Producer), Datenkonsument (Consumer) oder Datenkontrolleur (Controller).

3.3 Komponententechnologien im Vergleich

Trotz ihrer architektonischen Übereinstimmung weisen die in vorangegangenen Kapiteln vorgestellten Komponententechnologien essentielle Unterschiede auf. Während sich CORBA für unternehmensweite Lösungen mit hohen Leistungsanforderungen vor allem im Backend-Bereich etabliert, kommen COM vorwiegend im Desktop-Bereich und Java im Internetumfeld zum Einsatz. COM unterstützt – als integraler Bestandteil des Windows-Betriebssystems – zwar mehrere Programmiersprachen, aber nur eine Betriebssystemplattform[7]. JavaBeans läuft auf nahezu allen Betriebssystemplattformen, kommt aber nur mit einer Programmiersprache zurecht. Die OMA – als vollständigstes und flexibelstes Modell – unterstützt zwar mehrere Programmiersprachen und Betriebssysteme, existiert aber zum Teil nur auf dem Papier.

Microsofts COM war aus Sicht des Autors im Jahr 2000 die am weitesten fortgeschrittene Komponententechnologie, die sich auch wirklich als eine solche bezeichnen durfte. Diese konzeptionelle Vorreiterrolle wird ergänzt durch die frühen Erfolge der Visual Basic-Komponenten – ohne die die Komponentendiskussion zweifelsohne nicht auf dem heutigen Stand angelangt wäre (vgl. [61]). Zudem wird die COM-Technologie durch einen hohen Marktanteil von Microsoft im Windows-Desktop-Bereich entscheidend gestärkt.

---

[7] Microsoft arbeitet seit längerem an einer Portierung von COM auf andere Plattformen. Portierungen für Solaris [62], OpenVMS [63] und Linux [64] sind schon verfügbar.

Das Basiskonzept der JavaBeans – geleitet von der Idee einer konzeptionellen Reinheit und einer klaren, leicht verständlichen Umsetzbarkeit in konkrete Technologie – leidet noch unter einer Unvollständigkeit bezüglich eines hinreichend elaborierten Komponentenmodells und kann daher zumindest in Bezug auf die komponentenbasierte Sicht, noch nicht mit COM mithalten, wo Konzepte wie Versionierung, Sicherheit und insbesondere Bildung von Aggregationen und Container bereits sehr gut berücksichtigt sind.

Die OMA der OMG ist eher als Dritter im Bunde zu sehen. Von der OMG gehen wichtige Impulse für die Entwicklung komponentenbasierter Softwaresysteme im allgemeinen aus. Da es sich um reine Spezifikations- bzw. Standardisierungsbemühungen handelt, ist auch eine Entkopplung von Konzepten und deren technischen Umsetzung gegeben – in diesem Zusammenhang ausdrücklich ein Vorteil, da dies einen gewissen Freiraum zur Gestaltung und Ausbreitung komponentenbasierter Softwaresysteme zulässt. Trotz Standardisierungsbemühungen lassen sich der ORB- und CORBAservice-Implementierungen verschiedener Hersteller keinesfalls problemlos miteinander verbinden [65] und der Wechsel von einer ORB-Implementierung zur anderen bedeutet einen großen Aufwand. In der Praxis führt dies dazu, dass Anwendungsimplementierungen auf CORBA-Basis an die jeweiligen ORB-Implementierungen gebunden sind [66]. Das Argument der Herstellerunabhängigkeit ist damit nur mit Einschränkungen gültig.

Für die Auswahl einer Technologie als Plattform für die Entwicklung eines komponentenbasierten Softwaresystems sind technische Details aber nur ein Teilaspekt. Dort zählt auch Investitionssicherheit, also die Frage, wie sich eine Plattform im Markt etabliert und welche Rolle sie in der kommerziellen Datenverarbeitung spielt. Es spricht vieles dafür, dass sich COM für kleinere bis mittlere Anwendungen auf PC-Basis durchsetzten wird. Für COM spricht auch noch eine gute Unterstützung durch Entwicklungswerkzeuge, Teilkompatibilität zu DCE und – nicht zuletzt – der niedrigere Preis. Theoretisch ist ein Nachteil für COM seine starke Plattform- und Herstellerbindung. Die Praxis vieler Anwendungsgruppen zeigt aber, dass diese Bindung schon aus anderen Gründen eingegangen wurde.

Der in diesem Kapitel vorgestellte Vergleich spiegelt den Entwicklungsstand der Komponententechnologie im Vorfeld dieser Arbeit wieder. Der Verglich diente als Entscheidungsgrundlage, COM als Basistechnologie zu wählen. Im Laufe der Arbeit hat jedoch eine Weiterentwicklung der Komponententechnologien stattgefunden, welche im folgenden Kapitel beschrieben wird.

3.4 Weiterentwicklung der Komponententechnologien

3.4.1 Von COM zu .NET-Plattform

Mit dem Betriebssystem Windows 2000 wurden COM und Microsoft Transaction Service (MTS) von der aufwärtskompatiblen COM+ Technologie abgelöst. COM+ ist eine erweiterte COM-Laufzeitumgebung, welche vorgefertigte Lösungen für viele allgemeine infrastrukturelle Probleme bietet, mit denen die Entwickler von Geschäftssystemen

konfrontiert werden. COM+ bietet gegenüber COM zwei signifikante Verbesserungen. Erstens, es ist die aktualisierte Version 3 von MTS, in die die vorhandenen Elemente von COM integriert wurden, um ein nahtloses Ganzes zu bieten. Zweitens COM+ umfasst vier neue Dienste, die den Funktionsumfang von COM wesentlich erweitern. Die neuen COM-Dienste sind:

- Warteschlangen zum asynchronen und zeitentkoppelten Versenden von Methodenaufrufen
- Ereignisdienst zur Übermittlung von Ereignissen
- In-Memory-Datenbank zur Zwischenspeicherung von Datenbanktabellen in der Mittelschicht einer Drei-Schichten-Architektur
- Lastverteilung zur automatischen Verteilung der Anforderungen zur Objekterzeugung auf mehrere Server in einem Pool.

Wegen der Aufwärtskompatibilität von COM+ können die unter Windows NT entwickelten COM Komponenten weiterhin genutzt werden. Der Unterschied liegt darin, dass COM Komponenten umfangreichere Unterstützung vom Betriebssystem erhalten, was sie beschleunigt und ihre weitere Entwicklung entscheidend erleichtert.

Zur Unterstützung im Bereich des Internets hat Microsoft eine völlig neue Strategie angekündigt. Diese Strategie umschreibt ein neues Architekturmodell und eine neue Entwicklungsplattform, die als Microsoft .NET bekannt ist. Kern der .NET-Plattform sind eine Laufzeitschicht, die für alle Programmiersprachen gemeinsam, und eine Art des Grundgerüsts für Anwendungen ist. Das .NET-Grundgerüst ermöglicht den Zugriff auf die gemeinsame Laufzeitschicht und bietet zudem eine ganze Reihe von den Diensten an. Die ersten Basiskomponenten der .NET-Plattform sind seit der Einführung von .NET Enterprise Server und Windows 2000 Datacenter Server einsatzbereit. Die Entwicklungswerkzeuge werden mit der Einführung von VisualStudio .NET und dem .NET Framework im Jahr 2001 zur Verfügung stehen.

3.4.2 Von JavaBeans zu Java 2 Enterprise Edition

Das Java 2 Enterprise Edition (J2EE) von Sun ist eine Spezifikation – inzwischen existiert eine Referenzimplementierung von Sun – für eine Gesamtarchitektur mit dem Anspruch, komplexe Geschäftsprozesse zu unterstützen. Dafür legt J2EE Konventionen fest, welche es erlauben, Verantwortlichkeiten und Rollen zuzuordnen, und so ein Vorgehensmodell für die Entwicklung komplexer Systeme einführen. Darüber hinaus stellt J2EE eine Ablaufumgebung und eine Vielzahl von unterstützenden Diensten bereit. Die wichtigsten sind der Naming und Directory Service zum Auffinden von Objekten und Dateien, Datenbankzugriffsdienste nach der JDBC-Schnittstelle, Einbindung von CORBA, Applets und JavaBeans, Schnittstellen zur Verwendung von elektronischer Mail, eine Schnittstelle zum Austausch von Nachrichten zwischen Komponenten, Transaktionsdienste sowie dynamische Webseiten.

Für die eigentlichen Anwendungen sind spezifische Komponenten zu entwickeln, sogenannte Enterprise Java Beans (EJB). Diese müssen eine allgemeine Schnittstelle zum

Erzeugen, Zerstören, Aktivieren und Deaktivieren bereitstellen. Der Code des EJB wird beim Kompilieren so aufbereitet, dass die J2EE-Laufzeitumgebung Komponenten bei Bedarf erzeugen und verwalten kann sowie auf die bean-spezifischen Schnittstellen zugreifen kann.

Die J2EE unterscheidet zwei Arten von Komponenten: Session Beans und Entity Beans. Session Beans sind einem Klienten direkt zugeordnet und sie existieren, solange der Klient mit dem System verbunden ist. Im Gegensatz zu Session Beans lassen sich Entity Beans durch mehrere Klienten nutzen und fungieren als persistente Komponenten, deren Lebenszeit nicht an die Dauer der Verbindung gekoppelt ist. Zu diesem Zweck ist Entity Beans ein Primärschlüssel zugeordnet, der eine eindeutige Identifikation von Komponenten gestattet und sich üblicherweise aus einer Kollektion von Datenfeldern zusammensetzt.

3.4.3 Von CORBA zu CORBA Component Model

Ziel der Spezifikation des CORBA Component Models (CCM) – CCM ist ein integraler Bestandteil der CORBA 3.0 Spezifikation – war von Beginn an die problemlose Kompatibilität mit Enterprise Java Beans. Im Gegensatz zu EJB können CORBA-Komponenten mehrere Schnittstellen, sogenannte Ports, spezifizieren. Über diese exportieren sie Funktionalität nach außen (Facetts) oder importieren externe Funktionalität (Receptacles). Zudem kann eine Komponente als Emitter Ereignisse an einen einzelnen Empfänger oder als Publisher sogar an beliebig viele Empfänger senden.

Neben den EJB-Komponentenarten Session Bean und Entity Bean definiert das CORBA-Komponentenmodell zusätzlich Servicekomponenten und Prozeßkomponenten. Erstere tragen keinen Zustand und sind transient; letztere besitzen einen persistenten Zustand und sind nur einem singulären Klient zugeordnet.

Zur Unterstützung der Implementierung stellt CORBA 3.0 die Component Implementation Definition Language (CIDL) zur Verfügung. CIDL ist ein CCM-spezifisches Artefakt, das kein Gegenstück in anderen Technologien wie COM+ oder EJB besitzt. Die Anwendung einer Komponentenimplementierungssprache stellt einen Zwischenschritt zwischen Schnittstellendefinition und Codierung der Implementierungsklassen dar. CIDL erlaubt es, die Beziehungen zwischen Schnittstellen und ihren Implementierungsklassen explizit zu benennen sowie Persistenzeigenschaften für die Komponenten zu spezifizieren. CCM ist damit die mächtigste, wenn auch die komplexeste Komponententechnologie, die zur Zeit verfügbar ist. Fraglich ist jedoch, ob CCM Marktrelevanz erlangen kann. Bisher scheinen ORB-Hersteller eher verhalten auf CORBA 3.0 zu regieren.

Zusammenfassend lässt sich festhalten, dass die Komponententechnologien langsam zu einem exakt ausgearbeiteten Arbeitsmodell heranreifen. Die zu beobachtenden Tendenzen, wie etwa die Schwerpunktverlagerung auf die Unternehmensunterstützung durch COM+ oder EJB, sowie auf die ganzen Anwendungsbereichen ausgelegten Komponentenframeworks der OMG verdeutlichen, dass es sich hier um mehr als nur eine vielversprechende Ergänzung der aktuellen Softwareentwicklung handelt.

3.5 Entwicklung von komponentenbasierten Softwaresystemen

Softwarekomponenten müssen in einer speziellen Art und Weise konstruiert werden, damit sie mit anderen Komponenten zu einer Anwendung zusammengefügt werden können. Das Zusammenfügen der Komponenten geschieht in einem Komponentenframework. Erst Frameworks verleihen Komponenten eine Bedeutung. Die Komponenten, die zu keinem übergeordneten Komponentenframework gehören und deshalb nicht mit anderen Komponenten verbunden werden können, stellen ein Widerspruch in sich dar.

Nicht alle Frameworks sind objektorientiert. Objektorientierte Frameworks benutzen oftmals Vererbung als Basisprinzip für die Anwendungsentwicklung. Vererbung ist eine typische Form von White-Box-Wiederverwendung. Bei dieser Wiederverwendungsform muss der Anwendungsprogrammierer meistens in den Frameworkcode hineinschauen, um das Framework angemessen und mit dem größtmöglichen Nutzen verwenden zu können. Schöckle [67] und Grohmann [68] zeigten, dass die Ableitung der neuen Klassen in der Regel nur dann möglich ist, wenn der Anwendungsprogrammierer intime Kenntnisse der Implementierung der Basisklassen besitzt. In Bezug auf moderne Frameworkentwicklung ([8], [37] und [38]) hat eine Trendwende eingesetzt. Es wird zunehmend versucht, sog. Black-Box-Frameworks zu entwickeln. In diesen Frameworks benutzt man vorrangig die Komposition als Form der Wiederverwendung, bei dem die internen Strukturen der eingesetzten Komponenten nicht bekannt sein müssen. Dieses Verbergen der Implementierungsspezifika macht die Black-Box-Frameworks für die Entwicklung komponentenbasierter Softwaresysteme besser geeignet. Hier liegt das Hauptaugenmerk auf der Spezifikation von öffentlichen Schnittstellen.

Jedem Framework liegt eine Softwarearchitektur zugrunde. Die Softwarearchitektur ist eine Sammlung von Entwurfskonventionen [38], welche die Basiseigenschaften der Schnittstellen und die Art und Weise der Komposition festlegen. Die Basiseigenschaften der Schnittstellen bestimmen, wie Komponenten miteinander gekoppelt werden und wie sie interagieren. Dazu gehört die Wahl der eingesetzten Komponententechnologie (vgl. Kapitel 3.2) sowie die Modelle zum Datenaustausch. Durch die Verwendung der Datenaustauschmodelle wird ein gemeinsames Verständnis über Gegenstände der betrachteten Anwendungsdomäne und deren fachliche Interpretation über Komponenten hinweg ermöglicht. Die Datenaustauschmodelle müssen, in Bezug auf die Anzahl der darin modellierten Begriffe, so umfangreich sein, dass sie eine gemeinsame fachliche Basis für alle eingesetzten Komponenten darstellen.

Der Grad der Wiederverwendung, der mit einem komponentenorientierten Ansatz erreicht werden kann, hängt nicht nur von der Wahl der passenden, wiederverwendbaren Komponenten ab, sondern auch von der Art und Weise wie Komponenten zusammengesetzt werden. Dieser Vorgang wird im Englischen als Gluing[8] oder Component Scripting [37] bezeichnet und wird durch eine Programmiersprache realisiert.

---

[8] Engl: to glue - kleben. Gemeint ist hier das "Zusammenkleben" der einzelnen Komponenten zu einem Gesamtsystem.

Prinzipiell kann jede Programmiersprache zum Aufbau der Anwendungen verwendet werden, sofern diese eine Schnittstelle zu der Sprache besitzt, in der die einzelnen Komponenten realisiert sind. Skriptsprachen eignen sich wegen ihres geringen Sprachumfangs zum Zusammenbauen und Ansteuern von Komponenten besonders gut. Die Skriptsprachen benutzen einen dynamischen Typmechanismus, bei welchem eine Typüberprüfung erst zur Laufzeit durchgeführt wird. Diese sog. Spätbindung ermöglicht eine generische Anbindung der Komponenten in eine Anwendung. Es werden nur die Komponenten angebunden, die für eine spezielle Fragestellung von Bedeutung sind. Die zielgerichtete Anbindung führt zur besseren Strukturierung der Arbeitsabläufe und eröffnet die Möglichkeiten, die Komponenten eines Arbeitsablaufes in Hinblick auf ein gemeinsames Ziel zu steuern, zu überwachen und zu koordinieren. Die Spezifikationen einzelner Arbeitsabläufe werden mittels einer Skriptsprache beschrieben. Diese Beschreibungen sind vom Anwendungscode unabhängig und werden zur Laufzeit interpretiert. Sie enthalten Konstrukte zur Beschreibung aller notwendigen Teilaufgaben, deren Reihenfolgebeziehungen und Ausführbarkeitsbedingungen sowie Angaben, welche Ressourcen während der Ausführung benötigt oder produziert werden.

3.6 Anforderungen an ein komponentenbasiertes Frameworks zur Bewertung heiz- und raumlufttechnischer Anlagen

Die Konstruktion einer Familie von Anwendungssystemen zur Bewertung heiz- und raumlufttechnischer Anlagen erfordert die Entwicklung eines Komponentenframeworks, in dem funktionale Komponenten geeignet integriert werden können. Neben den funktionalen Komponenten muss das Framework dazu eine eigenständige Komponente beinhalten, welche eine gemeinsame fachliche Basis zur Integration der Komponenten vergegenständlicht und den Datenaustausch zwischen Komponenten ermöglicht.

Die bisherige Diskussion hat gezeigt, dass während des Lebenszyklus eines Gebäudes Methoden mit verschiedenen Zielsetzungen eingesetzt werden müssen. Diese Methoden arbeiten auf Daten, die im Laufe des Lebenszyklus präzisiert werden, sich aber auch ändern können. Die Daten beschreiben in den verschiedenen Phasen stets dasselbe Gebäude. Die anzuwendenden Methoden sind ähnlich und erlauben mit Konkretisierung der Daten genauere Simulationen. Aus diesen Gründen ist ein dynamisches Datenmodell erforderlich, auf dem unterschiedliche aber doch ähnliche Anwendungssysteme operieren müssen. Dieses für alle Komponenten gemeinsame Austauschmodell wird das Gebäude- und Anlagedatenmodell genannt. Es besitzt Gültigkeit über den gesamten Lebenszyklus des betrachteten Gebäudes. Daher ist das Datenmodell so zu konstruieren, dass es sich fachlich erweitern lässt, ohne dabei verändert zu werden. Dieser Grundsatz muss dem von Meyer definierten Offen-Geschlossen-Prinzip [69] entsprechen. Nach diesem Prinzip soll das Datenmodell für Erweiterungen der Datenstruktur offen, aber gleichzeitig in seiner Verwendung geschlossen sein. Der konventionelle Widerspruch zwischen Offenheit und Geschlossenheit lässt sich durch die strikte Verwendung des objektorientierten Paradigmas [67] überbrücken. Außerdem soll das Datenmodell eine Reihe der Funktionalitäten anbieten, die eine leichtere Integration der Komponenten ermöglichen. Wichtige Funktionalitäten sind:

- Benachrichtigung über Ereignisse: Ereignisse ermöglichen das Propagieren der Informationen über Zustandsänderungen im Datenmodell
- Transparente Realisierung der Persistenz: Das Datenmodell ist dafür selbst verantwortlich, den Zustand seiner Objekte persistent zu machen.
- Variantenbildung durch das Klonen: Bei der Variantenbildung wird das Datenmodell zunächst kopiert und danach auf spezielle Gegebenheiten angepasst.

Den Anwendungen zur Bewertung von heiz- und raumlufttechnischen Anlagen liegen thermische Berechnungen zu Grunde, womit den funktionalen Komponenten die Berechnungskomponenten gleich zu setzen sind. Berechnungskomponenten sollen in unterschiedlichen Phasen des Gebäudelebenszyklus genutzt werden können. Die für die Berechnung benötigten Eingabedaten erhalten die Berechnungskomponenten aus dem Datenmodell. Aus den Eingabedaten und mit Hilfe von Berechnungsalgorithmen werden Ergebnisse in einem internen Modell gerechnet. Die Berechnungsergebnisse sollen als Zeitreihen abstrahiert werden. Die Abstraktion der Zeitreihen ermöglicht nicht nur eine einfache Verwaltung der Simulationsergebnisse sondern auch deren direkten Vergleich mit anderen Simulationen oder Messreihen.

Zur Verbindung zwischen den Komponenten sollte eine geeignete Skriptsprache verwendet werden. Sie sollte bei den Berechnungskomponenten für das Erreichen eines übergeordneten Zieles im Sinne eines Workflows und bei den Datenhaltungkomponenten für die Entwicklung einer Oberfläche zur anwendungsgerechten Dateneingabe eingesetzt werden können.

Im folgenden Kapitel wird vorgestellt, wie diese Anforderungen realisiert wurden. Dies bildet die Basis für eine Reihe von Anwendungssystemen (vgl. Kapitel 5), welche die wichtigen Phasen aus dem Lebenszyklus der technischen Anlage eines Gebäudes abdecken und beispielhaft die neuen Möglichkeiten einer integrierten Betrachtung der Gebäudetechnik aufzeigen.

4 Komponentenbasiertes Framework zur Bewertung heiz- und raumlufttechnischer Anlagen

In diesem Kapitel wird ein Vorschlag für ein komponentenbasiertes Framework zur Bewertung heiz- und raumlufttechnischer Anlagen während des gesamten Gebäudelebenszyklus vorgestellt. Das Framework basiert auf einer Komponentenarchitektur und einer klaren Trennung der Komponente Gebäude- und Anlagedatenmodell von den Berechnungskomponenten und der Benutzeroberfläche. Die Integration der Komponenten in eine Anwendung erfolgt durch die Verwendung einer Skriptsprache. Die Bereiche der Komponentenarchitektur werden hinsichtlich ihrer Art, ihrer Funktion und ihres Zusammenwirkens beschrieben.

4.1 Architektur

Als Basistechnologie wird die Komponententechnologie von Microsoft (COM) vorgeschlagen. Hauptargument für COM ist die hohe Verfügbarkeit von COM im Bereich der Desktop-Anwendungen, welche im Bereich der Gebäudetechnik dominierend sind. Außerdem vereinfacht COM die Kapselung von Altlasten durch seine binäre Natur und den Einsatz verschiedener Programmiersprachen. Dies erlaubt die Heterogenität der Berechnungsprogramme mittelfristig beizubehalten.

Anwendungen werden aus Komponenten aufgebaut und sind auch selber als Komponenten zu interpretieren. Das in Abbildung 10 umschließende Symbol einer Komponente stellt eine der möglichen Anwendungen dar, die durch die Einhaltung der Konventionen der Komponentenarchitektur aufgebaut werden können. Nach dem Gesagten setzt sich jede Anwendung aus einem einheitlichen Datenmodell und beliebig vielen Berechnungskomponenten zusammen. Die Daten für die Berechnungskomponenten werden über das Datenmodell ausgetauscht. Neben dem Datenmodell und den Berechnungskomponenten ist eine Anwendung noch mit einer anwendungsspezifischen Benutzeroberfläche auszustatten. Hieraus ergibt sich eine Drei-Schichten-Architektur, welche die Anwendung in eine Datenschicht, eine funktionale Schicht und eine Präsentationsschicht unterteilt. Die Vorteile einer Drei-Schichten-Architektur sind in [70] und [71] ausführlich behandelt.

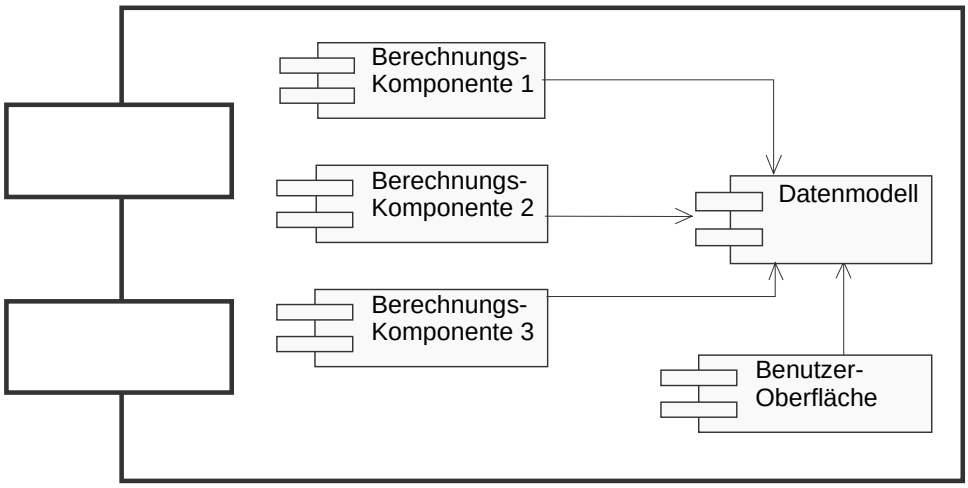


Abbildung 10: Anwendung auf Basis der vorgeschlagenen Komponentenarchitektur

Das Datenmodell (in Abbildung 10 rechts) verkörpert zunächst ein statisches Gebäude- und Anlagemodell. In erster Linie ist seine Aufgabe den Datenaustausch zwischen Komponenten zu ermöglichen. Das Datenmodell legt fest, welche Objekte zwischen Komponenten ausgetauscht werden und wie diese Austauschobjekte auszusehen haben. Die Aufgaben des Datenmodells gehen allerdings über den Datenaustausch hinaus, da es zur Beschreibung von Gebäuden und deren Anlagen während des gesamten Lebenszyklus dienen soll. Als Konsequenz sind dem Datenmodell andere Funktionalitäten zugeordnet, die es rechtfertigen von einer Komponente Datenmodell zu sprechen. Das Datenmodell ist hierarchisch aufgebaut. Die Komponente integriert einen Persistenzmechanismus auf XML Basis und ist in der Lage durch Initiieren von Ereignissen, interessierte Teilnehmer über eigene Änderungen zu benachrichtigen. Das Datenmodell ermöglicht die Untersuchungen diverser Entwurfsvarianten. Dazu existiert eine Funktionalität zum Anlegen gleichwertiger Kopien von Gebäude- und Anlagemodellen (Klonen).

Die Berechnungskomponenten legen etwa das dynamische Verhalten der Anwendung fest. Abbildung 10 veranschaulicht, dass alle Berechnungskomponenten auf ein und demselben Datenmodell arbeiten. Anhand der aus dem Datenmodell gewonnenen Gebäudeinformationen und mit Hilfe der in die Berechnungskomponente eingebetteten Algorithmen führen sie Berechnungen durch. Das Einbetten der Algorithmen erfolgt in einem internen Modell, welches für die spezielle Berechnung optimiert ist. Die Ergebnisse solcher Berechnungen sind meistens Zeitreihen, welche von speziellen Verwaltungskomponenten gehandhabt werden. Informationen über die Zeitreihen sowie die Zuordnung der Zeitreihen zu einem Gebäude werden im Datenmodell gespeichert. Somit weisen alle Berechnungskomponenten dasselbe Systemverhalten auf. Sie werden über eine generische OLE DB Schnittstelle in die Anwendung eingebunden.

Für die strukturelle Komposition der Komponenten und die Steuerung des Anwendungsablaufs sind Skriptsprachen besonders geeignet. Sie dienen als eine Art „Kleber“ für die Komponenten. Dies ermöglicht einen flexiblen Einsatz von Berechnungskomponenten erst zu dem Zeitpunkt, an dem sie im Gebäude-Entwicklungsprozess gebraucht werden. Weiterhin wird durch die Skriptsprachen die Entwicklung der Bedieneroberflächen möglich, die auf den aktuellen Datenbedarf reagieren. Solche Oberflächen können außerdem an individuelle Bedürfnisse unterschiedlicher Benutzergruppen einfach angepasst werden. Entsprechend der Konvention über die Anwendung von COM-Technologie wird Microsoft Scripting Technologie eingesetzt.

4.2 Komponente Gebäude- und Anlagedatenmodell

Bei der Entwicklung des Datenmodells ist wegen dessen zentralen Rolle in der Anwendung die größte Sorgfalt geboten. Das Datenmodell beschreibt die Domäne Gebäude und Gebäudetechnik. Der Entwurf des Datenmodells wurde stark von der aktuellen Entwicklung des IFC-Modells [7] geprägt. Das Datenmodell benutzt das IFC-Modell als Leitfaden für die Gebäudebeschreibung und erweitert dieses um eine Beschreibung heiz- und raumlufttechnischer Anlagen. Zudem ermöglicht das Datenmodell eine ganzheitliche Betrachtung des Gebäudes in allen Phasen des Lebenszyklus. Diese Erweiterungen wurden von der IFC-Modellierungsgruppe in das neue IFC-Modell vollständig übernommen (vgl. [22] und [72]). Aufgrund der internationalen Zusammenarbeit wurde Englisch als Basissprache gewählt.

Es wird zuerst die Breite und die Tiefe des Datenmodells beschrieben, um anschließend die vom Datenmodell bereitgestellten Funktionalitäten zu diskutieren. Für die Modellierung und Darstellung wird die Notation der Unified Modeling Language UML verwendet ([51], [52] und [53]).

4.2.1 Struktur

Das Datenmodell stellt ein objektorientiertes Gebilde zur Verwaltung von Gebäudeinformationen während des gesamten Gebäudelebenszyklus dar. Die kleinsten Einheiten des Datenmodells sind dementsprechend Objekte. Ihre Eigenschaften werden durch Objektattribute und Beziehungen zu anderen Objekten festgelegt. Ein Objekt bekommt Informationen über den Kontext seiner Benutzung von den mit ihm verbundenen Beziehungsobjekten. Beziehungsobjekte sind ein bewährter Ansatz für das Modellieren von Beziehungen zwischen Objekten (vgl. [73] und [74]). Sie erlauben das Aufbewahren von Beziehungseigenschaften beim Beziehungsobjekt selbst und trennen damit die Beziehungssemantik von Objektattributen. Die Objekte können auch Kollektionen gleichartiger Objekte besitzen, wenn sie aus diesen Objekten zusammengesetzt sind. Die Kollektionen erleichtern die Navigation unter den Objekten. Die Objekte sind durch zugehörige Klassen beschrieben. Die Klassen werden in einer hierarchischen Struktur zu Paketen zusammengefasst. Abbildung 11 zeigt die hierarchische Struktur des Datenmodells.

Jede Klasse im Datenmodell erbt von der Klasse *Root* aus dem gleichnamigen Paket. Die Klasse *Root* legt die Basiskonzepte für die Identifikation, Eigentümerschaft und Änderungsinformationen fest. Die Identifikation basiert auf einer 128-Bit-Zahl, die global eindeutig ist.

Das Modell enthält zunächst ein Paket mit Klassen, die einen Einstieg in das Projektmanagement ermöglichen. Die Klassen aus dem Paket *project management*, beschreiben allgemeine Projektdaten sowie die Daten, die das Projektteam, Projektphasen und projektbezogene Dokumente betreffen. Die Verwaltung dieser Daten wird durch die Klassen des Paketes *kernel* unterstützt.

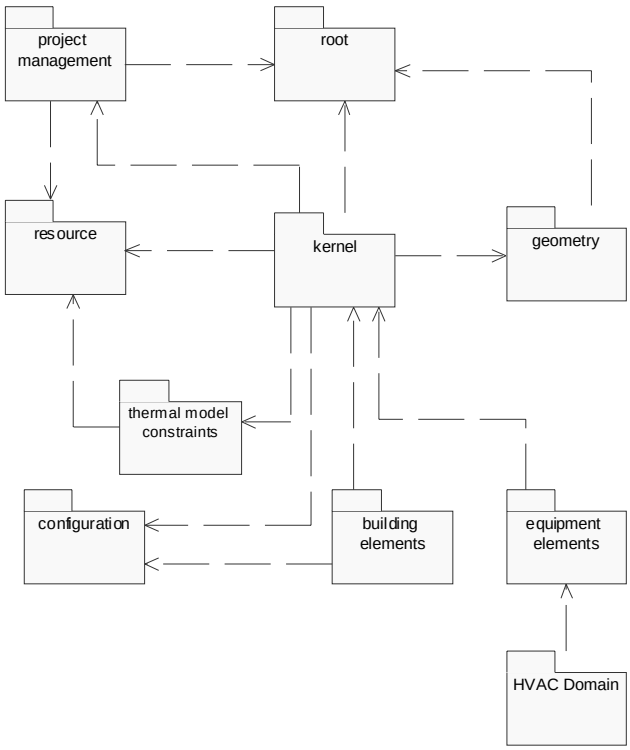


Abbildung 11: Hierarchische Struktur des Datenmodells

Das Paket *kernel* beinhaltet neben den abstrakten Konstrukten des allgemeinen Modellgrundaufbaus die Konstrukte eines raum- und zonenbezogenen Gebäudeabbildes. Um dies zu verstehen, soll zunächst der Grundaufbau des Datenmodells an Hand der Abbildung 12 beschrieben werden. Dazu wird zwischen Produkten und Systemen unterschieden. Produkte sind alle Artefakte des menschlichen Handelns. Zentrales Produkt dieser Arbeit ist das Gebäude. Systeme setzen sich aus Produkten zusammen, wobei ein System eine beliebige Anzahl unterschiedlicher Produkte beinhalten kann. Das Gebäude ist durch die Beziehungsklasse *BuildingSystemRelationship* mit den Systemen verbunden, die im Gebäude installiert sind. Die abstrakte Trennung zwischen dem Produkt Gebäude und den zugehörigen Systemen zieht sich durch das gesamte

Datenmodell. Alle mit dem Gebäude in Verbindung stehenden Klassen bilden das Gebäudemodell. Demgegenüber beschreiben die vom System abgeleiteten Klassen das Anlagemodell. Hierbei ist zu beachten, dass das Gebäudemodell die Randbedingungen für den Aufbau des Anlagemodells darstellt.

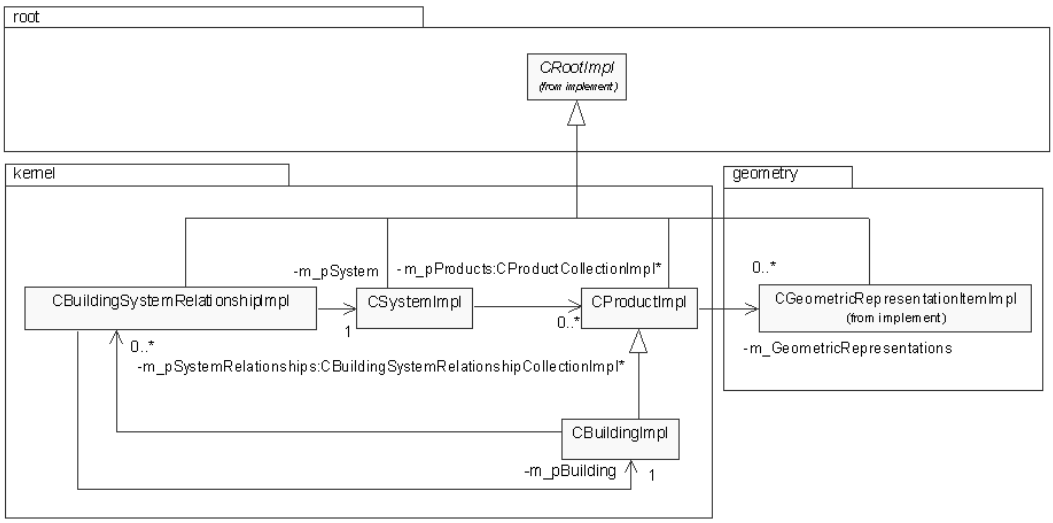


Abbildung 12: Grundaufbau des Datenmodells

Die Produkte können mehrere geometrische Repräsentationen haben. Die Ansammlung aller geometrischen Repräsentationen werden im Paket *geometry* modelliert. Das Paket *geometry* umfasst unter anderem die Repräsentationen eines kartesischen Punktes, eines Polygonzuges, eines Kreises, eines zylindrischen Körpers und einer Störkontur. Die Störkontur ist ein Hilfskonstrukt, welches einem Produkt zugeordnet werden kann, falls keine detaillierten Informationen über die Geometrie des Produktes vorhanden sind. Die Abbildung von Produktgeometrien ermöglicht, im Gegensatz zu den Austauschmodellen nach [3], [24] und [25], eine genauere Betrachtung von lokaler Strahlungsverteilung im Raum. Die Geometrie ist nur als eine der möglichen semantischen Ausprägungen des Produktes modelliert, und ist nicht der Hauptträger der Informationen, wie es im ISO Entwurf AP 225 [14] der Fall war.

Das Gebäudemodell schließt neben der geometrischen Ausprägung drei weitere Sichten ein:

- Physikalische Sicht (Bauteile)
- Raumbezogene Sicht (Topologie)
- Zonenbezogene Sicht (Nutzenanforderungen)

Die physikalische Sicht des Gebäudes beschreibt dessen Zusammensetzung aus Bauteilen (*Element* in Abbildung 13). Die Bauteile Fenster, Tür, Dach, Zwischendecke, Wand und abgehängte Decke stehen in der hierarchischen Struktur des Datenmodells unterhalb der Klassen *Element* und *BuildingElement*.

Alle Bauteile eines Gebäudes sind dem Gebäude als Elemente direkt zugeordnet. Die Konfigurationen dieser Bauteile haben fürs Gebäude eine lokale Bedeutung und sind im Paket *configuration* abgelegt. Somit ist es gewährleistet, dass die Konfiguration einer Bauteilgruppe nur einmal im Datenmodell (Redundanzfreiheit) vorhanden ist, unabhängig von der Anzahl der Bauteile, welche zu dieser Bauteilgruppe gehören. Sollte bei einer Variantenuntersuchung die Konfiguration der Bauteilgruppe geändert werden, dann ist diese Änderung entsprechend dem Lokalitätsprinzip nur an einer Stelle durchzuführen. Die Bauteilgruppen werden durch den Konfigurationstyp identifiziert. Die Trennung der Konfigurationsdaten schafft die Voraussetzung für eine flexible Anbindung von Bauteilkatalogen. Die Bauteilkataloge können, falls vorhanden, direkt aus dem Internet bezogen werden.

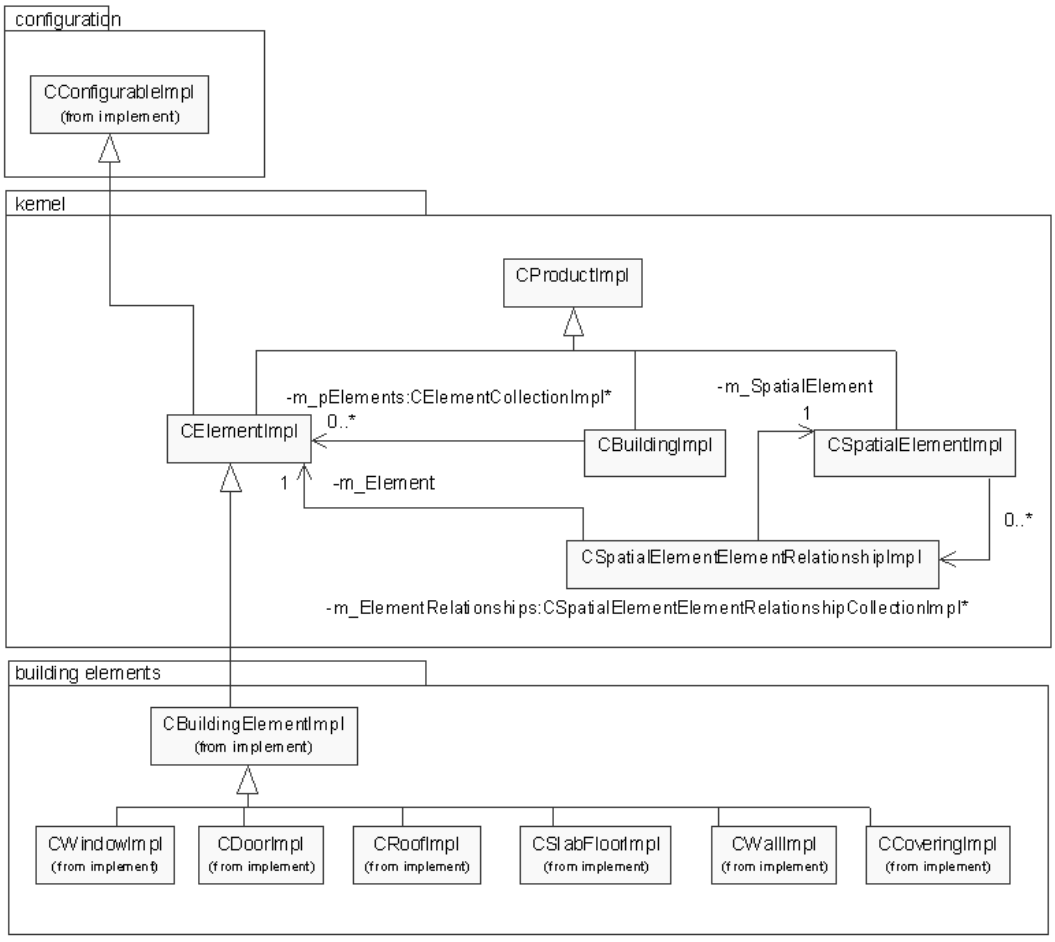


Abbildung 13: Physikalische Sicht auf das Gebäudemodell

Die Beziehungsklasse *SpatialElementElementRelationship* verbindet Bauteile mit den Räumen, da die Räume von der Klasse *SpatialElement* erben. Diese Verbindung erlaubt es, dass die physikalische und raumbezogene Sicht ineinander überführt werden können. Die raumbezogene Sicht stellt den Raum und seine topologischen Beziehungen zu

anderen Räumen in den Vordergrund (Abbildung 14). Der Raum (*Space*) ist ein abstraktes Gebilde, dargestellt in Form einer Fläche oder eines Volumens, das bestimmte Funktionalitäten innerhalb des Gebäudes erfüllt. Aus den Raum-Bauteil-Verbindungen *SpatialElementElementRelationship* werden die topologischen Beziehungen zwischen Räumen errechnet. Die topologischen Beziehungen sind durch das Vorhandensein begrenzender Bauteile gegeben. Ergänzend können die Räume im Gebäudemodell zu Stockwerken (*BuildingStorey*) zusammengefasst werden. Der Begriff Stockwerk als Hierarchiestufe oberhalb der Räume und unterhalb des Gebäudes ist eine Verallgemeinerung des Begriffes Geschoss. In der Gebäudeplanung werden insbesondere bei versetzter Bauweise für verschiedene Höhenlagen gleiche Geschoßbezeichnungen verwendet. Die Zuordnung eines Stockwerkes zu einer Höhenlage führt zur Eindeutigkeit. Die raumbezogene Sicht auf das Gebäude ergänzt durch Auflistung der Bauelemente wird meistens als Sicht des Architekten bezeichnet [75].

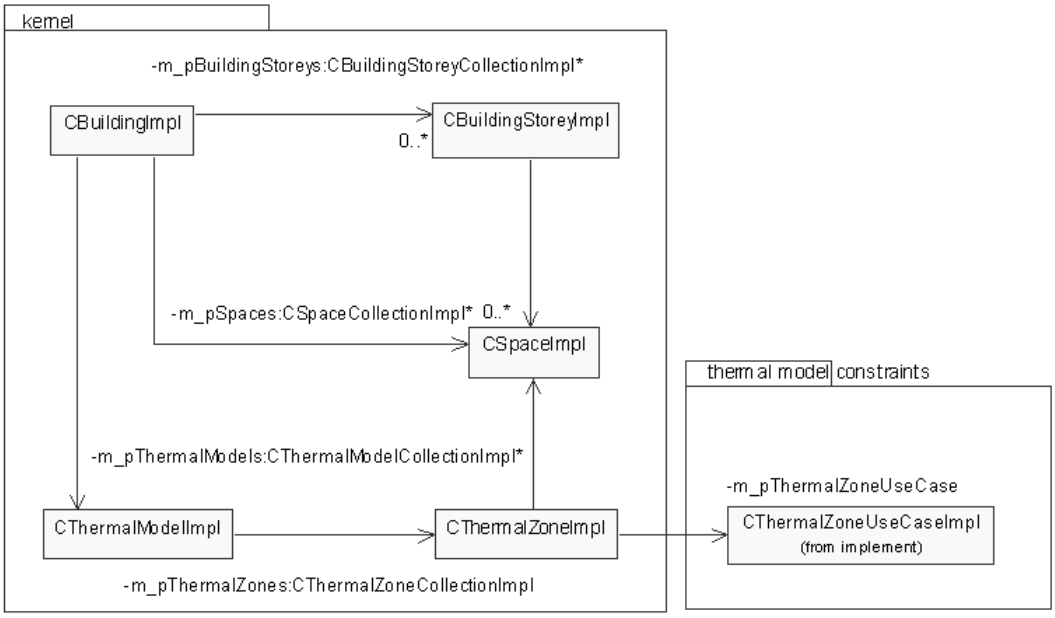


Abbildung 14: Raum- und zonenbezogene Sicht auf das Gebäudemodell

Für ingenieur-technische Berechnungen müssen anwendungsbezogene Modelle des Gebäudes erstellt werden. Der Ingenieur erweitert daher die Gebäudebeschreibung um eine neue Abstraktion und teilt das Gebäude je nach Fragestellung in Bereiche mit vergleichbaren Eigenschaften ein, die sog. Zonen. Eine Zone ist eine logische Zusammenfassung von unterschiedlichen Räumen, die eine geschlossene räumliche Einheit bilden. Handelt es sich bei den Zonen um Bereiche mit gleichen Nutzenanforderungen, so werden sie als thermische Zonen bezeichnet (Abbildung 14). Die Nutzenanforderungen werden im Paket *thermal model constraints* beschrieben. Das thermische Gebäudemodell, das aus thermischen Zonen besteht, wird als Aspektmodell thermische Lastberechnungen [3] oder kurz Zonenmodell [24] bezeichnet. Die thermischen Gebäudemodelle bilden die Basis für eine Bewertung des thermischen Verhaltens eines Gebäudes und seiner heiz- und raumlufttechnischen Anlagen.

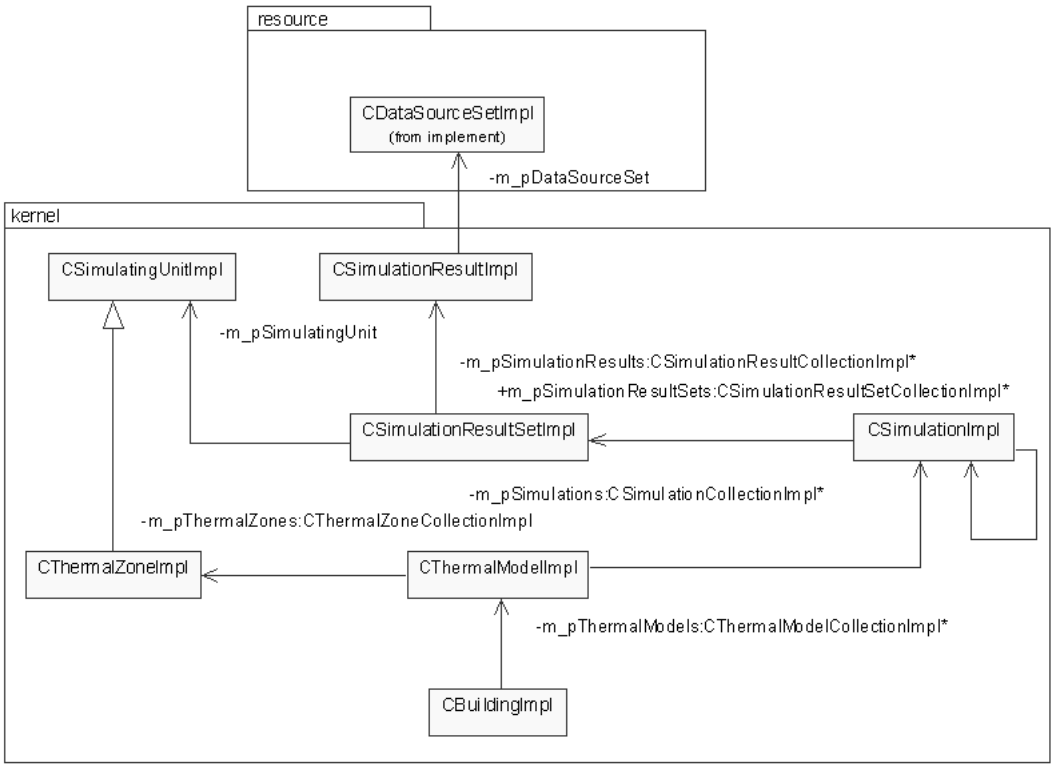


Abbildung 15: Gebäudesimulation

Die energetische Bewertung geschieht durch thermische Simulation. Die Struktur einer Gebäudemodellierung für eine thermische Gebäudesimulation ist in Abbildung 15 wiedergegeben. Steht das thermisches Modell für das Gebäude fest, dann kann eine Simulation für dieses Modell angelegt werden. Dabei kann ein thermisches Modell in beliebig vielen Simulationen benutzt werden. Jede Simulation legt Simulationsergebnisreihen *SimulationResultSet* fest. Eine Simulationsergebnisreihe ordnet einer Simulationseinheit (*SimulatingUnit*) ein Simulationsergebnis zu. Im allgemeinen wird als Simulationseinheit der Betrachtungsgegenstand einer Simulation bezeichnet. Er ist die kleinste Einheit eines Simulationsmodells. Im Falle einer thermischen Gebäudesimulation entspricht die thermische Zone der Simulationseinheit. Die Anzahl der Simulationsergebnisreihen für eine thermische Zone kann beliebig groß sein und hängt vom eingesetzten Berechnungsverfahren ab. Die Simulationsergebnisse werden als Zeitreihen abstrahiert. Eine Zeitreihe ist die chronologische Aneinanderreihung einzelner Zeitbereiche. Das Ende eines Zeitbereiches ist dabei gleichzeitig der Beginn des nächsten. Jedem Zeitbereich wird für jeden Betrachtungsgegenstand ein Wert zugeordnet. Der Wert kann einen Mittelwert über einen Zeitbereich oder aber einen Wert zu einem bestimmten Zeitpunkt im Zeitbereich (z.B. Messwert am Ende eines Zeitbereiches) repräsentieren. Seine Semantik wird über die Metadaten einer Zeitreihe dargestellt. Die zeitreihenbeschreibenden Metadaten werden als Instanzen der Klasse *DataSourceSet* vom Paket *resource* im Datenmodell abgelegt. Die Zeitreihen selbst werden außerhalb des Datenmodells verwaltet.

Simulationen können mit anderen Simulationen verbunden werden (Abbildung 15). Eine solche Verbindung ermöglicht einer Simulation sich auf die Ergebnisse einer vorgeschalteten Simulation zu beziehen. So bekommt beispielsweise eine Anlagesimulation den Zugriff auf Ergebnisse einer Gebäudesimulation. Dies ist eine Voraussetzung für die gekoppelte Gebäude- und Anlagesimulation.

Die abstrakte Klasse *Element* ist die Basisklasse nicht nur für die Gebäudebauteile sondern auch für alle Anlageelemente (Abbildung 16). Sie ermöglicht den Anlageelementen, mit gleichen Vorteilen wie bei den Bauteilen, den Zugang zu den herstellerabhängigen Daten (*configuration*). Die herstellerabhängigen Daten unterliegen der Normierung [27] und können direkt vom Hersteller bezogen werden. Dabei werden von Herstellern alle, zur rechnerischen Auslegung und graphischen Darstellung benötigte Daten, vollständig und in einer einheitlicher Form angegeben. Wiederum vereinfacht die Trennung der Herstellerdaten sowohl die Modellierung von Anlageelementen als auch die Berechnung, da z.B. das Kennfeld einer Pumpe vom Hersteller bereits mathematisch vollständig abgebildet ist.

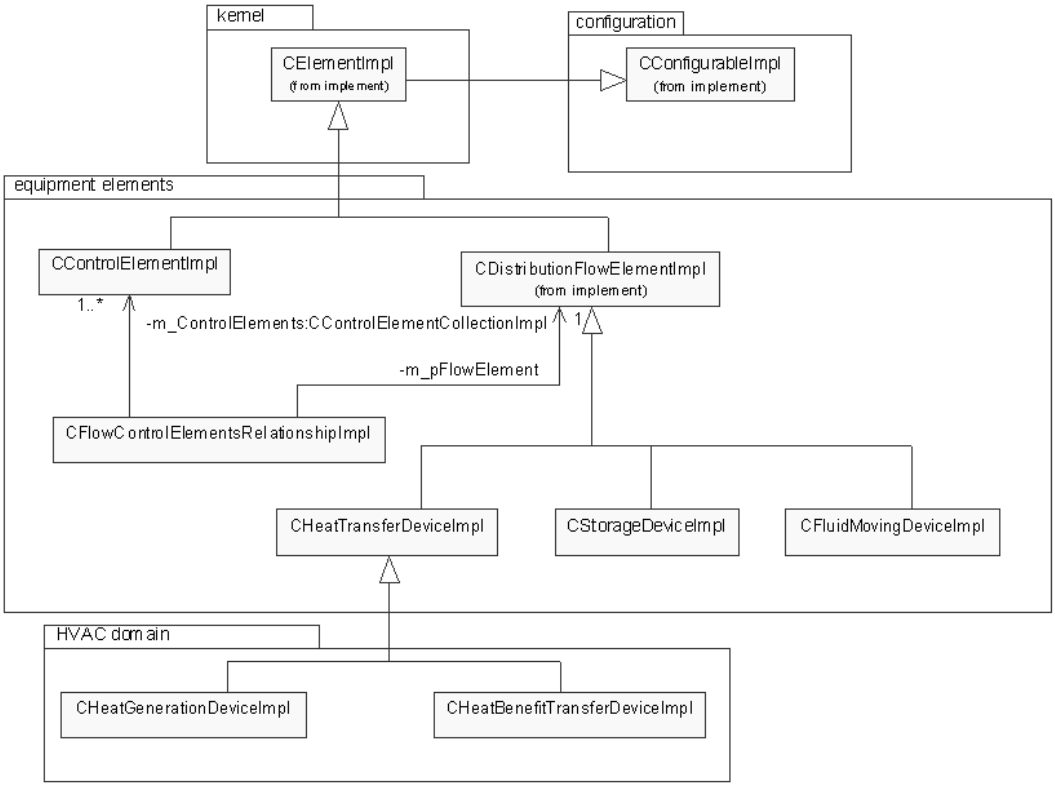


Abbildung 16: Beschreibung der Anlageelemente

Die Anlageelemente unterscheiden sich anhand der Prozesse, welche sie unterstützen (Abbildung 16). Für die Modellierung von heiz- und raumlufttechnischen Anlagen sind Prozesse des Energie- oder Materietransportes *DistributionFlowElement* und die Regelungsprozesse *ControlingElement* von besonderer Bedeutung. Der Transportprozess

umfasst alle Anlageelemente eines Distributionssystems mit der Aufgabe, den Transport von Energie oder Materie zu ermöglichen. Luft, Wasser oder Strom können Transportmedien sein. Der Regelungsprozess verallgemeinert Regelelemente eines Gebäudeautomationssystems, welche zur Regelung in Transportprozessen *FlowControlElementsRelationship* genutzt werden. Die Regelung wird getrennt behandelt, da ein Regelkreis zwei oder drei Bereiche eines Systems beeinflusst. So wirkt ein Thermostatventil beispielsweise zugleich unmittelbar auf die Heizwasserverteilung *FluidMovingDevice* und auf die Nutzenübergabe am Heizkörper *HeatBenefitTransferDevice*.

Unterhalb der Klasse *DistributionFlowElement* sind drei weitere Gerätegruppen modelliert, welche Prozesse der Wärmeübertragung *HeatTransferDevice*, Speicherung *StorageDevice* und Hydraulik *FluidMovingDevice* umfassen. Am Prozess der Wärmeübertragung beteiligen sich Anlagekomponenten, welche entweder die Energie in Heizwärme umwandeln oder die Wärmeübertragung durchführen. Dieser Prozess unterliegt einer weiteren Aufteilung in Prozeßbereiche der Wärmeerzeugung *HeatGenerationDevice* und der Nutzenübergabe *HeatBenefitTransferDevice*. Die zuletzt genannten Bereiche entsprechen den Systembereichen nach der Methode der Bedarfsentwicklung (siehe Kapitel 2.1). Der dritte Bereich der Methode der Bedarfsentwicklung, die Wärmeverteilung, setzt sich aus Rohrstücken, Armaturen, hydraulischen Geräten und Speichergeräten zusammen. Die Rohrstücke und Armaturen werden direkt von der Klasse *DistributionFlowElement* abgeleitet.

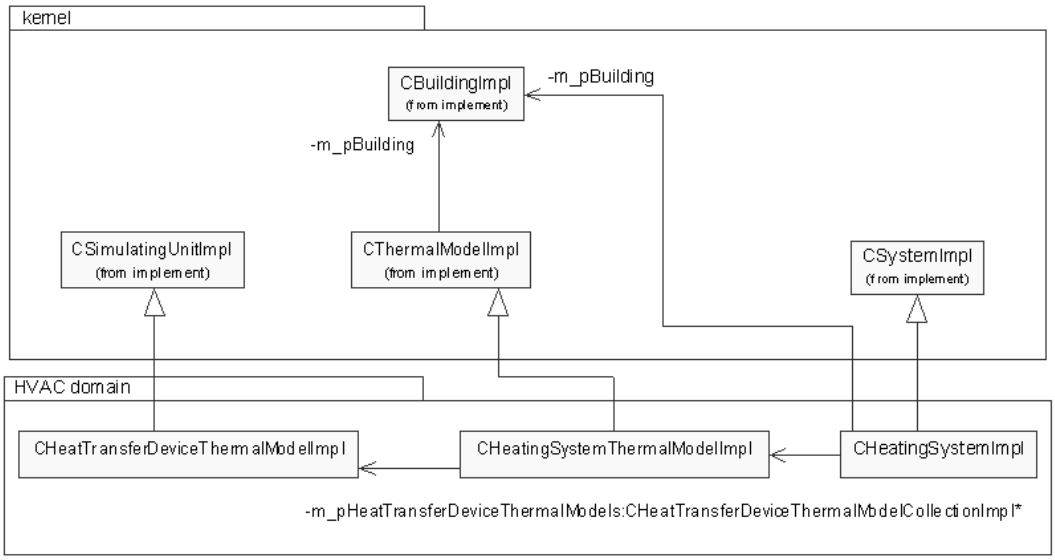


Abbildung 17: Thermisches Modell eines Heizsystems

Nach den Regeln des Datenmodell-Grundaufbaus (Abbildung 12) setzt sich ein System aus Produkten zusammen. Dies gilt auch für ein Heizsystem, welches aus Anlagekomponenten für Wärmeerzeugung, Wärmeverteilung und Nutzenübergabe aufgebaut ist. Damit eine thermische Anlagesimulation möglich wird, muss ein thermisches Modell für die Anlage analog zum Gebäudemodell festgelegt werden. Die

thermische Modellierung für die Anlage ist in Abbildung 17 dargestellt. Für jedes Heizsystem existieren thermische Modelle, welche wiederum aus Abbildungen von den zum Heizsystem gehörenden Wärmeübertragungsgeräten bestehen. Sie stellen die kleinste Einheit eines thermischen Anlagemodells dar, für die simuliert wird, und somit erben sie von der Klasse *SimulatingUnit.* Da die Anlagesimulation den gleichen Grundaufbau wie die Gebäudesimulation hat, benutzt sie dieselben Mechanismen für die Ergebnisverwaltung (Abbildung 15).

4.2.2 Modellierung - Heizkörper

Für die Modellierung heiz- und raumlufttechnischer Anlagen wurde in der hier vorgestellten Version des Datenmodells exemplarisch am Beispiel ein Warmwasser-Heizsystem behandelt. Modelliert wurde im Bereich der Nutzenübergabe Heizkörper und Warmwasserfußbodenheizung und im Bereich der Wärmeerzeugung Heizkessel, Wärmepumpe und Solarkollektor mit zugehöriger Regelung. Um die Tiefe des Datenmodells zu beschreiben, wird die Modellierung eines Heizkörpers als Beispiel gewählt. Der abzubildende Heizkörper soll die Daten für die Wirtschaftlichkeitsberechnungen nach der alten [76] und neuen [77] Fassung von VDI 2067 und für eine thermische Simulation mit TRNSYS [78] bereitstellen. Das Heizkörpermodell für die TRNSYS-Simulation baut auf dem von Bach und Claus [79] erstellten Schichtenmodell auf. Abbildung 18 zeigt die Klassenstruktur, die notwendig ist, um einen Heizkörper mit einem Thermostatventil zu modellieren. Sie wird im folgenden detaillierter beschrieben.

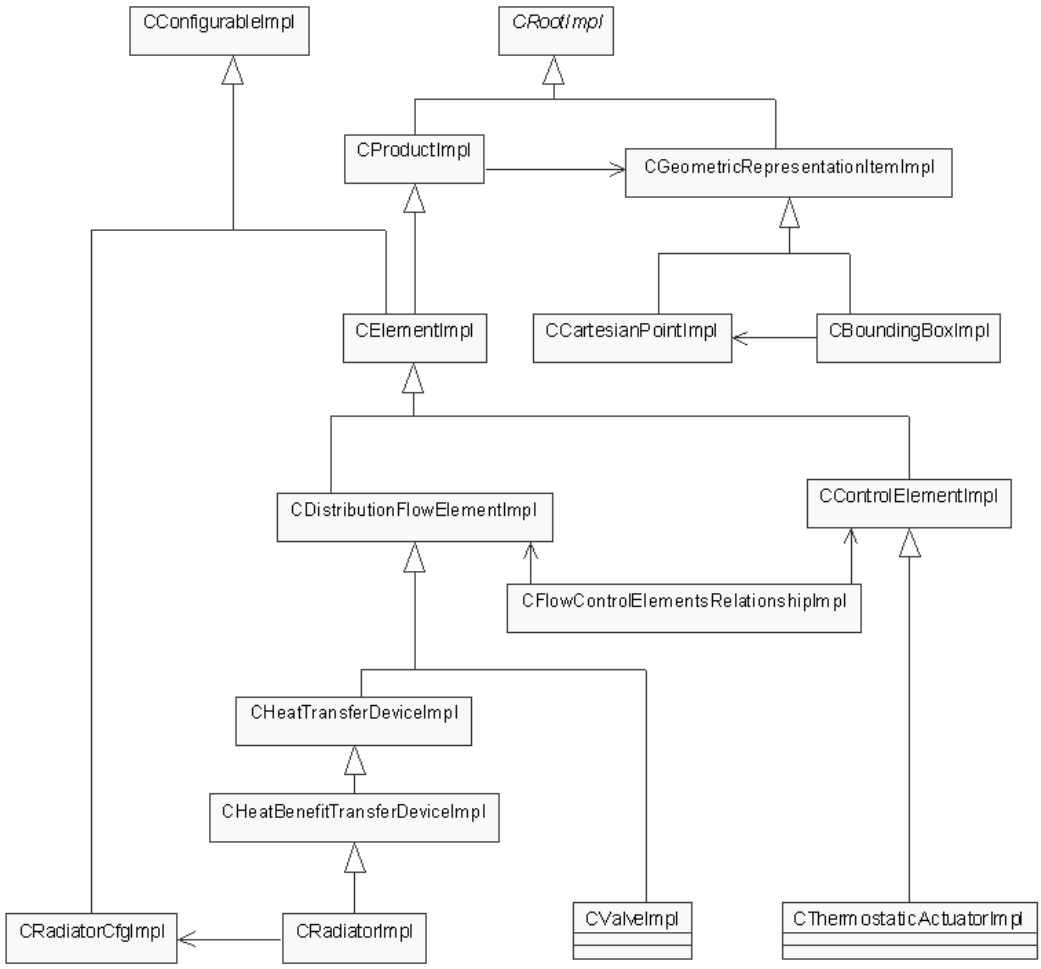


Abbildung 18: Heizkörperbeschreibung

Die Geometrie des Heizkörpers wird durch eine Störkontur beschrieben. Eine Störkontur (*BoundingBox)* ist nach [80] ein Quader, dessen Kanten parallel zu den Achsen des betrachteten Koordinatensystems verlaufen. Der Quader ist durch die Koordinaten vom unteren linken Eckpunkt und die in Richtung der Koordinatenachsen gemessenen Kantenlängen, beschrieben (Abbildung 19, links). Die korrespondierende Klassenstruktur ist auf der rechten Seite der Abbildung wiedergegeben. Eine Störkontur beschreibt einen Raum um den Heizkörper, in dem weder Wände noch Decken vorkommen dürfen. Dieser Raum wird etwa zur Vorkontrolle bei der automatischen Kollisionsüberprüfung während der Planung benötigt [33]. Da ein Produkt „Heizkörper“ mehrere geometrische Repräsentationen besitzen kann, können gleichzeitig, ergänzend zur Störkontur, weitere geometrische Beschreibungen des Heizkörpers in jedem gewünschten Detaillierungsgrad existieren. Die detaillierten Beschreibungen der Heizkörpergeometrie könnten etwa zur Simulation der lokalen Strahlungsverteilung im Raum benötigt werden. Diese ist z.B. für die Modellierung eines Thermostatkopfes wichtig, weil durch sie sein Regelverhalten beeinflusst wird.

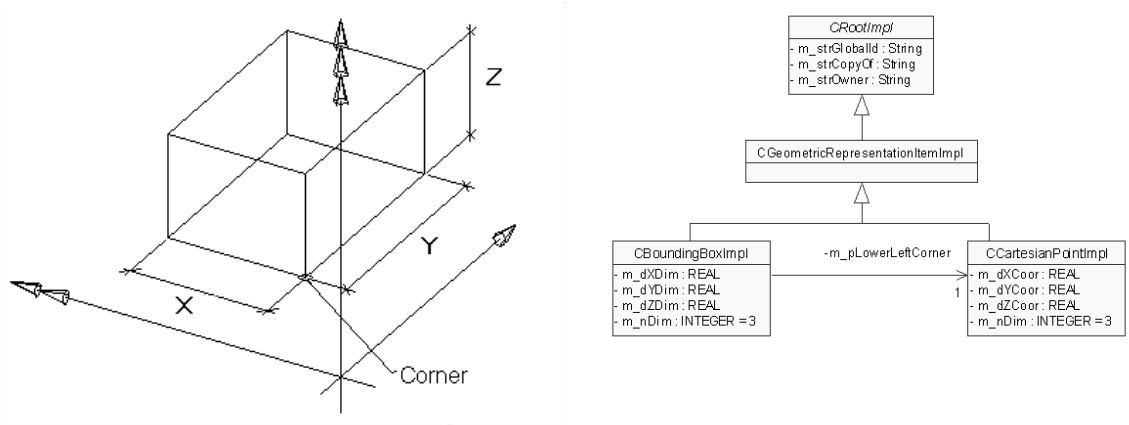


Abbildung 19: Störkontur

Neben der geometrischen Repräsentation sind die Angaben der Positionierung notwendig, um die Geometrie des Heizkörpers vollständig abzubilden. Für den Heizkörper wird ein eigenes, dreidimensionales Koordinatensystem durch die Nullpunktposition und zwei normierte Raumvektoren festgelegt [27]. Die Raumvektoren geben die Lage der X- und Y-Achse im Absolutsystem an. Der Raumvektor für die lokale Z-Achse ergibt sich aus dem Vektorprodukt des X-Vektors mit dem Y-Vektor. Innerhalb des lokalen Koordinatensystems können Punkte definiert werden, auf die Anschlüsse positioniert werden [33].

Eine vollständige Beschreibung des Heizkörpers ist gegeben, wenn die lokale Zuordnung, die Produkteigenschaften, Auslegeparameter und gegebenenfalls die Betriebsparameter angegeben sind [27]. Jeder Heizkörper ist einem Raum (*Space*) zuordenbar, so dass sich die lokale Zuordnung, wie in Abbildung 20 gezeigt, strukturell darstellen lässt. Die Produkteigenschaften sind in der Klasse *RadiatorCfgImpl* erfasst. Abbildung 20 zeigt nur einen Ausschnitt aus den Produktdaten, die für die oben genannten Berechnungsverfahren benötigt werden. Im allgemeinen sind Produktdaten von den Angaben der Hersteller abhängig. Durch die in Abbildung 18 beschriebene Trennung von den Produkt- und den projektbezogenen Daten ist eine flexible Anbindung von herstellerabhängigen Produktdaten gewährleistet.

Die Anbindung der Produktdaten erfolgt durch den Datenaustausch zwischen dem Datenmodell und den Datenbanken der Produkthersteller. Ein mögliches Austauschverfahren für die Produktdaten ist in VDI 3805 [28] festgelegt. Für die Produktgruppe Heizkörper ist eine detaillierte Beschreibung der VDI Richtlinie in [33] angegeben. Nach der VDI Richtlinie werden die technischen Daten eines Heizkörpers durch die Angabe des Herstellernamens und einer internen TGA-Nummer eindeutig gekennzeichnet. Bei Heizkörpern verweist die TGA-Nummer entweder auf ein eindeutig bestimmtes Produkt oder auf eine Baureihe, die Produkte unterschiedlicher Baulänge bzw. Sektorenzahl umfasst. Die Heizkörperhersteller stellen alle zum jeweiligen Heizkörper erforderlichen technischen Daten zur Auslegung des Heizkörpers und der zugehörigen Anlagen zur Verfügung. Dabei bezieht sich die Norm-Wärmeleistung immer auf Normtemperaturen von Vorlauf, Rücklauf und Raumluft. Die Bezugstemperaturen für

Vor- und Rücklauf unterscheiden sich zwischen DIN 4703 [81] und EN 442 [82]. Damit für den Fall späterer Änderungen der Bezugstemperaturen die Produktbeschreibung beibehalten werden kann, werden die Bezugstemperaturen explizit angegeben [83]. Ein alternativer Weg für den Austausch der Produktdaten wird im Rahmen des IAI Projektes XM-1 gezeigt [84]. Das Projekt beschäftigt sich mit der Anbindung externer Bibliotheken an das IFC-Modell. Die Beispiele für die externen Bibliotheken sind unter anderem Produkt- oder Preiskataloge. Der Datenaustausch erfolgt über das weit verbreitete XML-Format der W3C [23]. Mit dem XM-1 Projekt wird das Ziel verfolgt, den Produktherstellern die Nutzung des Internets und den Zugang zu e-business zu ermöglichen.

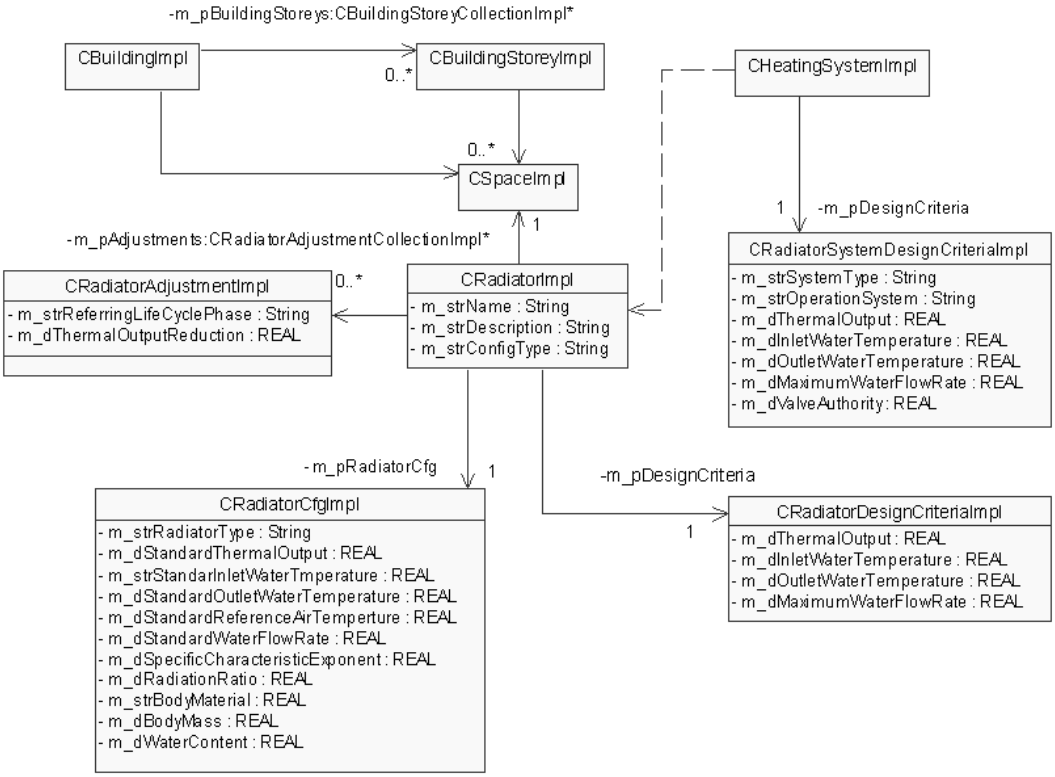


Abbildung 20: Vollständige Beschreibung eines Heizkörpers

Die Auslegeparameter werden durch die Klassen *RadiatorSystemDesignCriteria* und *RadiatorDesignCriteria* festgelegt. Dabei gelten die Auslegeparameter der Klasse *RadiatorSystemDesignCriteria* für alle Komponenten eines Heizsystems. Dagegen beziehen sich die Auslegeparameter der Klasse *RadiatorDesignCriteria* nur auf einzelne Heizkörper. Während der Auslegung werden die Ansichtsabmessungen, die Vorlauf- und Rücklauftemperatur, der Heizwasserstrom und die Leistung der betrachteten Heizfläche bestimmt. Wird das Heizsystem nach den Vorgaben der Bedarfsplanung (Kapitel 2.1) ausgelegt, so müssen zur Ermittlung der Auslegeparameter die Regeln von VDI 6030 [85] befolgt werden. Ziele sind neben der Deckung der Heizlast [86], die Heizflächen bei der Auslegung so anzuordnen, dass die Behaglichkeitsdefizite im Raum gezielt gemindert

oder beseitigt werden können. Die Behaglichkeitsdefizite können durch die Erwärmung äußerer Umfassungsflächen des zu beheizenden Raumes, die Kompensation der durch kältere Flächen hervorgerufenen Strahlungsdefizite und die Beseitigung der Falluftströmungen gemindert werden.

Die Klasse *RadiatorAdjustment* beinhaltet solche Daten, die in unterschiedlichen Phasen des Lebenszyklus verschiedene Werte einnehmen können. So umfasst etwa die Leistungsminderung (*ThermalOutputReduction*) Werte, welche sich aus der Einbausituation ergeben. Der Wert der Leistungsminderung kann während der Auslegungsphase vom Planer entsprechend den Vorgaben aus VDI 6030 entnommen werden. Gerät etwa die Heizkörpernische durch Ungenauigkeiten beim Bau kleiner als geplant, so stellt sich bei der Inbetriebnahme ein anderer Wert für die Leistungsminderung ein. Gleichermaßen kann in der Phase des Gebäudebetriebs die Leistungsminderung weiteren Änderungen unterliegen, welche durch vorgestellte Möbel oder durch eine nachträglich angebrachte Verkleidung hervorgerufen werden können.

Anlagekomponenten sind in der Regel über Netze miteinander verbunden und sind, unabhängig vom Medium, bezüglich der Netzstruktur gleich. Die Netzbeschreibung des Datenmodells orientiert sich an der Netzbeschreibung des IFC-Modells [87]. Mit ihr werden Netze auf drei Ebenen beschrieben. Die oberste Ebene beinhaltet die Topologie, die durch Kanten und Knoten gebildet wird. Die zweite Ebene bezieht sich auf Netzelemente. Dabei werden bereits Anschlüsse und Fließrichtungen unterschieden. Auf der dritten Ebene werden die Netzwerkkomponenten im Detail beschrieben. Für eine ausführliche Beschreibung der Abbildung von Netzen wird auf die Literatur zum Thema Netzabbildung z.B. [3], [26], [83] und [87] verwiesen. Die aktuelle Version des Datenmodells ermöglicht die Betrachtung einzelner Komponenten oder Komponentengruppen. Demzufolge wird die Netzstruktur nicht in vollem Ausmaße abgebildet. Die anstehende Weiterentwicklung des Datenmodells, etwa im Rahmen des neu aufgesetzten IAI-Projekts BS-8 [88], schließt eine vollständige Netzwerkabbildung mit ein.

4.2.3 Implementierung

Basierend auf der Komponententechnologie von Microsoft wurde das Datenmodell als ein COM-Server in C++ implementiert. Der COM-Server besteht aus mehreren COM-Objekten, welche Instanzen von den Klassen aus dem Datenmodell sind. Ein COM-Objekt implementiert mehrere COM-Schnittstellen. Abbildung 21 zeigt die Klassen, die für die Implementierung der COM-Schnittstellen *IUnknown*, *IDispatch*, *IEagRadiator* und *ISupportErrorInfo* von der Klasse *CRadiatorImpl* notwendig sind. Als Implementierungstechnik wurde die Active Template Library ATL von Microsoft [43] benutzt. ATL stellt Basisklassen zur Verfügung, welche allgemeine COM-Funktionalitäten kapseln und damit die Implementierung maßgeblich erleichtern. In Abbildung 21 unterscheidet man drei Bereiche:

- COM-Bereich mit der COM-Klasse *EegRadiator* und den COM-Schnittstellen *IUnknown*, *IDispatch*, *IEagRadiator* und *ISupportErrorInfo*,

- ATL-Bereich mit der ATL-Klasse *CRadiator* und den ATL-Basisklassen *CComObjectRootEx, IDispatchImpl*, und *ISupportErrorInfoImpl* und
- den Bereich der fachlichen Modellierung bestehend aus der zuvor beschriebenen Klasse *CRadiatorImpl*.

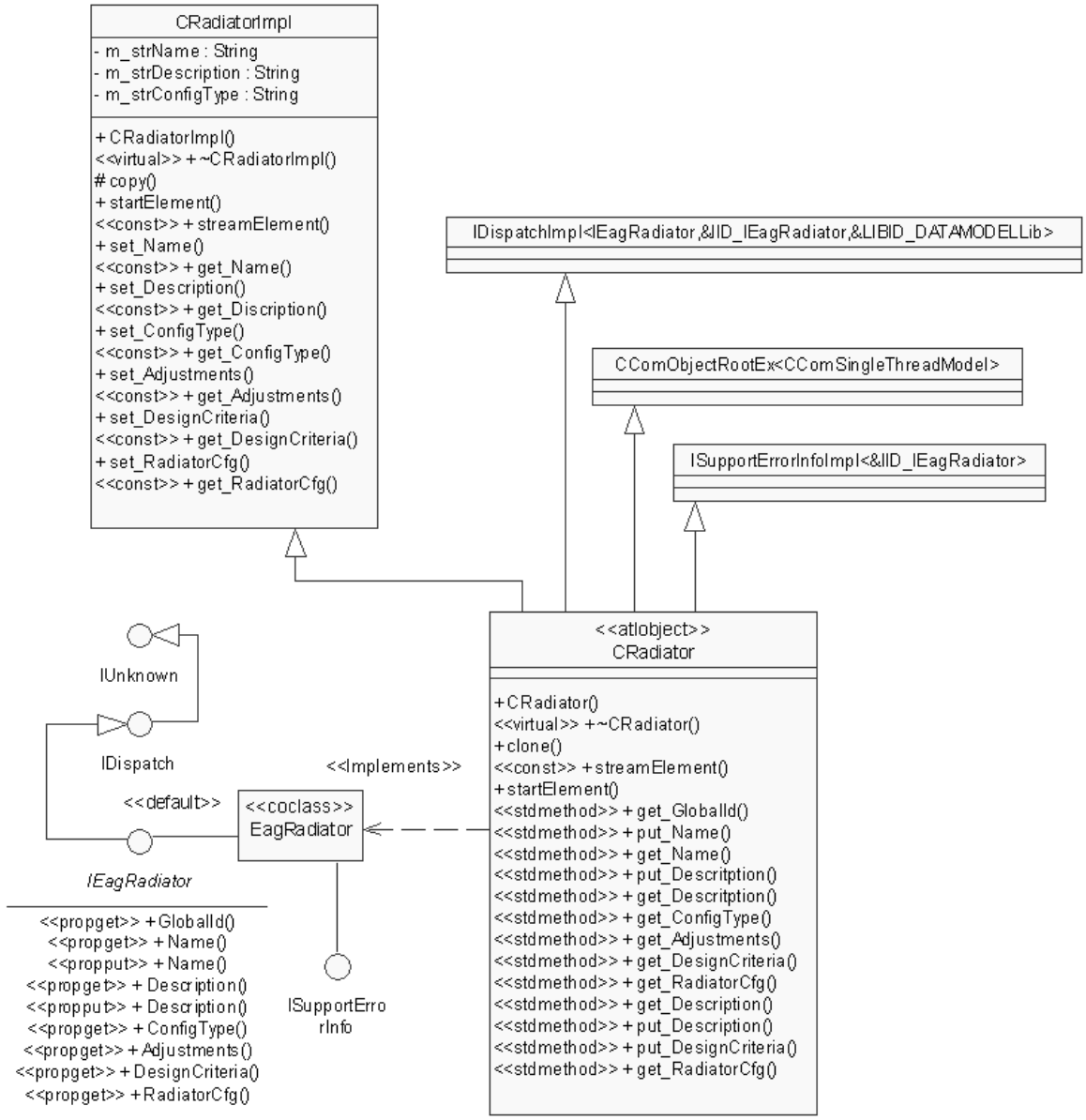


Abbildung 21: Implementierung der Klassen zur Beschreibung eines Heizkörpers

Die Funktionsweise von COM sowie die Grundbestandteile, aus denen COM besteht, wurde in Kapitel 3.2.1 eingeführt. Über die COM-Klasse *EagRadiator* sind die COM-Schnittstellen mit funktionierendem Code verbunden. Die COM-Schnittstelle *IUnknown*

muss von jedem COM-Objekt implementiert werden. *IUnknown* bietet Methoden zum Navigieren zwischen COM-Schnittstellen sowie Methoden für die Referenzzählung an. Die Schnittstelle *IDispatch* legt die Attribute und Methoden für externe Benutzer offen, welche zur Übersetzungszeit keinen Zugriff auf Typinformationen haben oder bei denen es sich möglicherweise um Skriptsprachen handelt. Die *ISupportErrorInfo* stellt sicher, dass Fehlerinformationen korrekt in der Aufrufkette propagiert werden. Im Fehlerfall wird einfach die Methode *Error* aufgerufen, die einen Fehlertext an den Klienten weiter leitet. Die Meldungen, die über die *ISupportErrorInfo* Schnittstelle gesendet werden, führen bei einem Zugriff über die COM-Schnittstelle zu einem Absturz des Klientenprogramms - es sei denn, man wertet auch auf der Klientenseite die Fehlerbehandlung aus. Die Schnittstelle *IEagRadiator* beinhaltet benutzerdefinierte Methoden, welche sich aus der fachlichen Modellierung ergeben haben.

Für gebräuchliche COM-Schnittstellen stellt ATL kleine und effektive Wrapper-Klassen zur Verfügung, ohne die Performanz einer Komponente einzuschränken. Die Wrapper-Klassen verbergen einen Großteil der internen Architektur eines COM-Objekts in nützlichen Abstraktionen. Diese Abstraktionen sind in den ATL-Basisklassen enthalten. Die Klasse *CComObjectRootEx* implementiert die Schnittstelle *IUnknown*. Darüber hinaus bietet sie durch die Benutzung der Threading-Klasse *CComSingleThreadModel* den Schutz vor konkurrierenden Zugriffen an. Die Klasse *IDispatchImpl* leistet auf Basis einer dualen Schnittstelle die Implementierung der Schnittstelle *IDispatch*. Als duale Schnittstelle wird eine Schnittstelle bezeichnet, deren ersten Funktionen, die *IDispatch*-Funktionen sind, gefolgt von benutzerdefinierten Schnittstellenfunktionen. Die Implementierung von *IDispatch* basiert auf der Typbibliothek, die das Objekt beschreibt. Um die Typbibliothek zur Laufzeit laden zu können, erhält *IDispatchImpl* die GUID *LIBID_DATAMODELLib* für die Typbibliothek als Vorlageparameter. Alle ATL-Objekte werden standardmäßig mit dualen Schnittstellen erzeugt. Die ATL unterstützt durch die Klasse *ISupportErrorInfoImpl* die Implementierung der Schnittstelle *ISupportErrorInfo*. Durch die Vererbungsbeziehung mit den ATL-Basisklassen enthält die ATL-Klasse *CRadiator* den gesamten anwendungsbezogenen Code, der benötigt wird, um ein funktionsfähiges COM-Objekt zu implementieren.

Da die COM-Technologie bzw. ATL keine direkte Implementierungsvererbung erlauben (ATL-Klassen können nicht voneinander erben) [43], und an dieser Stelle wegen des hierarchischen Aufbaus des Datenmodells nicht auf die Vererbung verzichtet werden kann, werden zwei Klassen für die Implementierung jedes Datenmodellelements benötigt. Für die Implementierung des Heizkörpers sind es die Klassen für die fachliche Modellierung *CRadiatorImpl* und die ATL-Klasse *CRadiator*. Die Klasse für die fachliche Modellierung *CRadiatorImpl* beschreibt, wie schon gesehen, alle für die Abbildung eines Heizkörpers benötigten Attribute und Beziehungen. Darüber hinaus legt sie über die Vererbung die Rolle des Heizkörpers im hierarchischen Aufbau des Datenmodells fest. Die Vererbung kann an der Stelle benutzt werden, da es sich bei *CRaditorImpl* (vgl. Abbildung 18, durch Vererbung verbunden mit *CHeatBenefitTransferDeviceImpl*) weder um eine COM- noch um eine ATL-Klasse handelt. Aus demselben Grund kann die Vererbung für die Kopplung der Klassen *CRaditorImpl* und *CRadiator* eingesetzt werden, wobei die ATL-Klasse *CRadiator*

lediglich für die Implementierung der notwendigen COM-Schnittstellen zuständig ist. Für sie besteht keine Möglichkeit von der ATL-Klasse *CHeatBenefitTransferDevice* zu erben.

4.2.4 Funktionalitäten der Komponente Datenmodell

Neben den von COM vordefinierten Funktionalitäten besitzen die Klassen des Datenmodells weitere Funktionalitäten zum Kopieren von Objekten und zur Speicherung bzw. Wiederherstellung von Objektzuständen aus Dateien.

Damit die Variantenuntersuchung ohne größeren Aufwand ermöglicht wird, ist es notwendig, die Elemente des Datenmodells zu klonen [67]. Mit dem Klonen ist eine Anfertigung von Kopien der Objekte oder Objektgruppen gemeint. Eine Umsetzung dieser Idee ist zum Beispiel die von Coplien [89] beschriebene Programmierung mit Exemplaren. Grundidee der Programmierung mit Exemplaren ist die Verwendung eines Objektes anstatt einer Klasse als Schablone zur Erzeugung weiterer Objekte desselben Typs. Diese werden nicht nur durch die Objekterzeugung allein, sondern zusätzlich durch das Kopieren des Schablonenobjekts erzeugt. Ein geklontes Datenmodellobjekt unterscheidet sich von seinem Original nur durch den Identifier. Sollte eine Variante des Objekts erstellt werden, dann ist lediglich erforderlich, die zu ändernden Attribute anzupassen. Die Methoden *clone* der Klasse *CRadiator* und *copy* der Klasse *CRadiatorImpl* führen das Klonen durch (Abbildung 21). Semantisch unterscheiden sich diese beiden Methoden dadurch, dass die Methode *clone* zuerst das neue Objekt erzeugt, um anschließend die Methode *copy* für das erzeugte Objekt aufzurufen. Die Methode *copy* kopiert die Werte der Attribute aus dem Original in das neu erzeugte Objekt. Als Rückgabeparameter der Methode *clone* wird das geklonte Objekt an den aufrufenden Klienten zurückgegeben. Die Anpassung der Attribute erfolgt durch Zugriffsmethoden des geklonten Objekts. Das geklonte Objekt enthält in seinem Attribut *IsCopyOf* den Identifier des Originals, aus dem es entstanden ist. Das Attribut *IsCopyOf* ist eine Eigenschaft der Klasse *CRootImpl*, die die oberste Basisklasse aller Klassen im Datenmodell ist (vgl. Abbildung 12).

Objekte des Datenmodells spielen in unterschiedlichen Phasen des Gebäudelebenszyklus jeweils eigene Rollen. Um auf die internen Zustände der Objekte in verschiedenen Gebäudelebenszyklusphasen zugreifen zu können, werden sie in einer Datei gespeichert. Die Datei liegt in Form eines XML-Dokumentes vor. XML (eXtensible Markup Language) ist eine Metasprache für die Definition eigener Auszeichnungssprachen ("Markup Language") [23]. Um XML einsetzen zu können, muss eine Sprache mit einem eigenen Vokabular und eigener Grammatik definiert werden. Das Vokabular für die Beschreibung der Domäne Gebäude und Gebäudetechnik ist durch Klassen-, Beziehungsrollen- und Attributnamen aus dem Datenmodell gegeben. Die Grammatik ergibt sich aus den Beziehungsklassen, die die hierarchische Struktur des Datenmodells beschreiben. Für den Einsatz von XML zur Implementierung des Persistenzmechanismus war ausschlaggebend, dass

- XML eine effiziente Abbildung von Objektstrukturen erlaubt,

- durch XML den Datenaustausch mit anderen Anwendungen, unter Voraussetzung der Verwendung gleicher Abbildungsvorschriften, ermöglicht wird,
- XML von W3C standardisiert wurde und
- XML auch für Menschen lesbar ist.

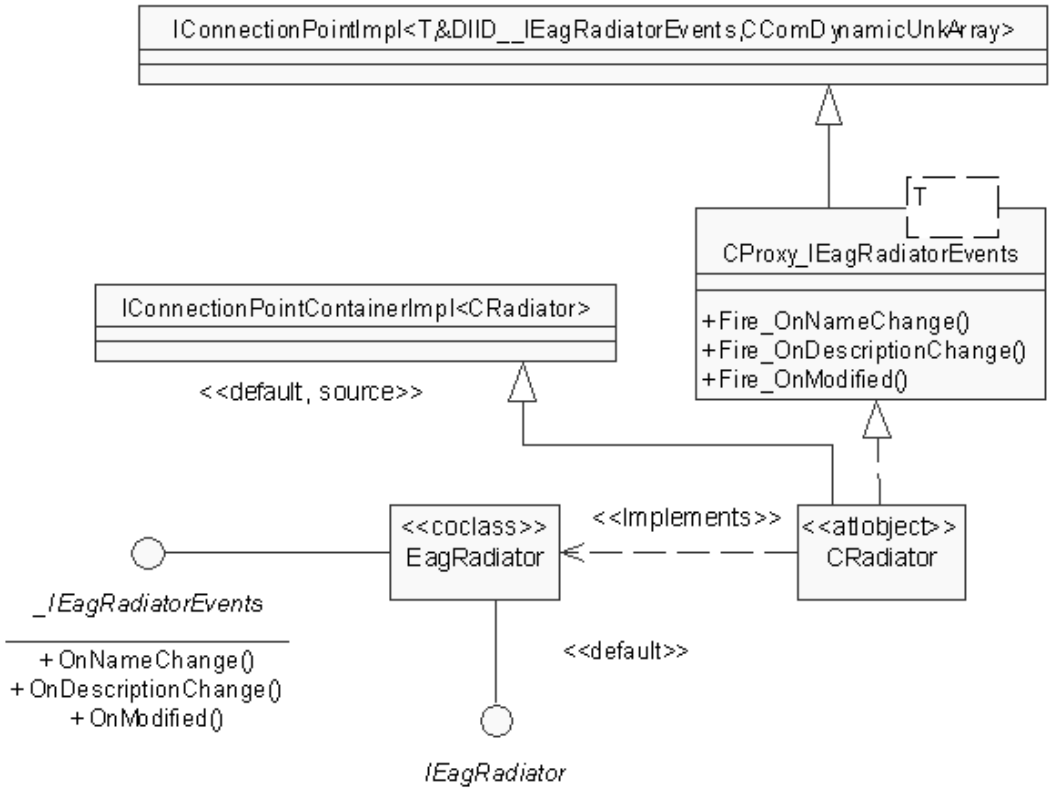


Abbildung 22: Ereignisschnittstellen des Datenmodells

Die Anwendungen, die XML-Dokumente verarbeiten, benötigen eine Programmierschnittstelle zum XML-Parser. Ein XML-Parser ist ein Programm, das eine XML-Datei liest und den Inhalt mittels einer vordefinierten Schnittstelle liefert. Es gibt zwei Arten von Schnittstellen: eine baum-orientierte und eine ereignis-orientierte Schnittstelle. Die baum-orientierte Schnittstelle Document Object Model (DOM) [90] ist ein W3C-Standard und ermöglicht den Zugriff auf die gesamte geparste Baumstruktur. Der Nachteil der Schnittstelle ist, dass das Dokument vollständig geparst werden muss, bevor man auf den Inhalt zugreifen kann. Um dieses Problem zu lösen, wurde in der XML-Gemeinde die ereignis-orientierte Schnittstelle SAX (Simple API for XML) [91] vorgeschlagen. Die Anwendung wird bereits während des Parsens des XML-Dokumentes über Ereignisse benachrichtigt, wenn eine Anfangsmarke, eine Endmarke usw. erkannt werden. So kann die Anwendung eine eigene interne Darstellung aufbauen oder Teile des

Dokumentes ignorieren. Um die SAX-Schnittstelle zu implementieren, benötigen die Klassen des Datenmodells jeweils eine Methode zum Herausschreiben des internen Zustands *streamElement* und eine zur Benachrichtigung über das Auftreten eines Ereignisses *startElement* (Abbildung 21). Außerdem sind alle primitiven Datentypen, die eine Abbildung des jeweiligen Datentyps auf XML ermöglichen, in den eigenen Klassen gekapselt.

Damit eine bidirektionale Kommunikation zwischen dem Datenmodell und dessen Klienten möglich wird, implementieren die Datenmodellklassen Ereignisschnittstellen. Ereignisschnittstellen werden von den Datenmodellklassen benutzt, um interessierte Klienten über eigene Zustandsänderungen zu benachrichtigen. Abbildung 22 zeigt die Ereignisschnittstelle *_IEagRadiatorEvents* der Klasse *CRadiator*. Die Ereignisschnittstellen werden vom Datenmodell festgelegt und von Klienten implementiert. Durch eigene Implementierungen der Ereignisschnittstellen werden die Klienten in die Lage versetzt, selbst zu bestimmen, ob und wie sie auf Änderungen im Datenmodell reagieren. Neben der Festlegung der Ereignisschnittstellen wird zusätzlich noch ein Mechanismus benötigt, durch welchen die Verbindung zwischen den Komponenten aufgebaut wird. Um in eine Verbindung involviert zu sein, müssen die Datenmodellklassen die Schnittstellen *IConnectionPoint, IConnectionPointContainer* implementieren (3.2.1). Hierfür bietet ATL Unterstützung durch einige Vorlageklassen und einen Satz vom Makros. ATL implementiert die Schnittstelle *IConnectionPointContainer* über die Vorlage *IConnectionPointContainerImpl* (Abbildung 22). Der Zweck dieser Schnittstelle besteht darin, den Klienten eine Methode zu bieten, um herauszufinden, ob ein Objekt eine aktuelle Ereignisschnittstelle unterstützt. *IConnectionPointContainerImpl* unterhält eine Auflistung von *IConnectionPoint* Schnittstellen. ATL implementiert die Schnittstelle *IConnectionPoint* über eine Vorlage namens *IConnectionPointImpl. IConnectionPointImpl* enthält die GUID, die den Verbindungspunkt angibt, und eine Auflistung der *IUnknown*-Schnittstellen, welche verwendet wird, um den Klienten zurückzurufen. *IConnectionPointImpl* ist die Basisklasse der Klasse *CProxy_IEagRadiatorEvents*, die Rückrufe an den Klienten erzeugt, indem sie eine *IDispatch*-basierte Schnittstelle kapselt, welche der Klient liefert.

4.2.5 Weiterentwicklung des Datenmodells

Das Datenmodell soll über den gesamten Lebenszyklus eines Gebäudes genutzt werden. Daher muss es offen und erweiterbar sein (vgl. Kapitel 3.6). Bei der Erweiterung kann es sich sowohl um die Erweiterung bestehender Klassen als auch um das Hinzufügen neuer Klassen handeln. Der erste Schritt ist die Erweiterung im Bereich der fachlichen Modellierung. Da die fachliche Modellierung das objektorientierte Paradigma in vollem Ausmaße nutzt, stehen für Modellerweiterungen alle Mittel der Objektorientierung zur Verfügung [67]. Die wichtigsten Mittel sind die Verallgemeinerung und Abstraktion. Mit ihrer Hilfe war es möglich, das Datenmodell hierarchisch aufzubauen. Die Hierarchie ermöglicht es einerseits, die Komplexität zu reduzieren, da auf jeder Abstraktionsstufe nur die zu dieser Stufe gehörenden Klassen betrachtet werden müssen. Andererseits ist die Hierarchie auch wichtig für die Erweiterbarkeit, da im Datenmodell immer auf die

angemessene Stufe in der Hierarchie zurückgegriffen wird und so zusätzliche Spezialisierungen hinzugefügt werden können, ohne dass die vorhandenen Klassen geändert werden. Damit eine neue Klasse in das Datenmodell hinzugefügt werden kann, wird sie von einer vorhandenen Klasse des Datenmodells abgeleitet. Die auszuwählende Hierarchiestufe für die Ableitung ergibt sich aus den Anforderungen der fachlichen Modellierung. Wird die hinzugefügte Klasse Änderungen der logischen Gruppierung der Datenmodellelemente (Sichten) hervorrufen, dann muss die jeweilige Kollektion um die Behandlung der neu hinzugefügten Klasse erweitert werden. Die Verwendung der Kollektionen (Aggregation) ist ein weiteres Konzept zur verbesserten Erweiterbarkeit und einfacheren Handhabung, da auf diese Weise eine schrittweise Komplexitätsreduktion erfolgt.

Der nächste Schritt ist das Vorgehen bei der Schnittstellenänderung des Datenmodells. Einmal veröffentlichte Schnittstellen des Datenmodells sollten nicht mehr verändert werden, da es bereits Komponenten geben könnte, welche die Schnittstellen in ihrem Code verwenden. Wenn eine Schnittstelle einmal veröffentlicht ist, dann lässt COM nicht mehr zu, Änderungen an der Schnittstelle vorzunehmen, die ihre Syntax ändern oder die Programmlogik signifikant umwandeln würden. Um die gewünschte Erweiterung zu ermöglichen, muss eine neue Schnittstelle angeboten werden, während die alte weiterhin unverändert verfügbar bleibt. Ist die neue Schnittstelle einfach eine Obermenge der alten Schnittstelle, dann sollte die neue Schnittstelle so gestaltet werden, dass sie von der alten Schnittstelle erbt. Hieraus ergibt sich, dass die jeweilige COM-Klasse von nun an zwei Schnittstellen anbietet. Alte Klienten, die nur die Originalschnittstelle erwarten, würden durch die neue Schnittstelle nicht negativ beeinflusst werden. Sie würden nicht einmal bemerken, dass eine neue Schnittstelle hinzugefügt wurde. Neue Klienten wären in der Lage, der Funktionalität der beiden Schnittstellen auszunutzen, indem sie die *QueryInterface* aufrufen, um einen Zeiger auf jede der beiden Schnittstellen zu erhalten.

4.2.6 Das Datenmodell im INTERNET

Das Datenmodell wurde im Internet unter http://www.ike.uni-stuttgart.de/~www_wn/projects/datenmodell/index.html veröffentlicht. Durch die Veröffentlichung werden die Ergebnisse dieser Arbeit sowohl den Fachleuten für die weitere Entwicklung zur Verfügung gestellt als auch der internationalen Kritik ausgesetzt. Die weitere Entwicklung betrifft vor allem die Arbeit im Ausschuss der VDI Richtlinie 6027 Blatt 2 [27] und die Arbeiten im Rahmen der IAI Projekte BS-7 Performance Validation [72] und BS-8 IFC HVAC Extension Schemata [88].

4.3 Berechnungskomponenten und ihre Integration

Für die Planung und Betrieb heiz- und raumlufttechnischer Anlagen ist es erforderlich, die Berechnungskomponenten bereitzustellen, mit denen abhängig von jeweiliger Lebenszyklusphase, Bewertungen durchgeführt werden können.

4.3.1 Berechnungsverfahren für Nachweise, Auslegung und Bewertung

Berechnungsverfahren werden mit unterschiedlicher Zielsetzung angewandt. Der Wärmeschutznachweis nach WSchV'95 [92] muss als Nachweis für energiesparendes Bauen geführt werden. Bei der WSchV'95 wird ausschließlich das Gebäude in die Berechnung des Jahresheizenergiebedarfs mit einbezogen. Die Güte der Heizanlage und der Anlage zur Trinkwassererwärmung werden außer Acht gelassen. Die im Frühjahr 2002 erscheinende Energiesparverordnung ESPA [9] löst die WSchV'95 ab. Mit der ESPA fließt erstmals Anlagetechnik und Betriebsweise in die Bewertung des Endenergiebedarfes mit ein.

Die Berechnung der Heizlast nach DIN 4701 [86] und der europäischen DIN EN 12831 [93] sowie der Kühllast nach VDI 2078 [94] sind Auslegungsberechnungen. Nach der DIN 4701 bzw. der DIN EN 12831 wird die Leistung der Heizanlage dimensioniert. Werden Raumheizkörper zur Nutzenübergabe eingesetzt, kann die Auslegungsrichtlinie VDI 6030 [85] herangezogen werden. Nach ihr werden Raumheizkörper bedarfsorientiert, ausgelegt, d.h. es werden erweiterte Anforderungen z.B. an die Behaglichkeit in die Auslegung mit einbezogen.

Für die Berechnung des Energiebedarfs steht die DIN EN 832 [95] zur Verfügung. Diese Berechnungsvorschrift ist allerdings nur für die Berechnung des Heizenergiebedarfs in Wohngebäuden geeignet, da der Einfluss der Nutzung nur in eingeschränkter Form in die Berechnung mit eingeht. In einer deutlich erweiterten Form kann der Energiebedarf von Gebäuden mit der VDI 2067 [96] errechnet werden. Hiermit wird zunächst ein der angenommenen Nutzung entsprechender Referenz-Energiebedarf ermittelt. Je nach gewählter Anlagetechnik wird dem Referenz-Energiebedarf des Gebäudes ein Mehrbedarf der Anlagetechnik über eine Aufwandszahl zugeschlagen. Für die Nutzenübergabe bei Warmwasserheizsystemen liegt bereits Blatt 20 [77] vor, weitere Blätter folgen für die Nutzenübergabe von RLT-Anlagen sowie für Aufwandszahlen von Einzelheizgeräten.

4.3.2 Berechnungskomponenten

Das Verhalten einer Anwendung wird durch ihre funktionalen Komponenten festgelegt. Den Anwendungen zur Bewertung von heiz- und raumlufttechnischen Anlagen liegen thermische Berechnungen zu Grunde, daher sind den funktionalen Komponenten die Berechnungskomponenten gleich zu setzen. Da sich die Fragestellungen während des Gebäudelebenszyklus ändern, beschränkt sich der Einsatz einer Berechnungskomponente meistens nur auf eine Phase des Gebäudelebenszyklus. Dies hat zur Folge, dass eine Anwendung zur Bewertung von heiz- und raumlufttechnischen Anlagen in der Lage sein muss, eine große Anzahl von Berechnungskomponenten flexibel einzubinden. Außerdem werden die bestehenden Berechnungskomponenten durch Gewinnung neuer Erkenntnisse ständig verbessert, was eine Erweiterbarkeit der Anwendung voraussetzt. Um den Anforderungen der einfachen Einsetzbarkeit und Erweiterbarkeit gerecht zu werden, haben die Berechnungskomponenten ein eigenes internes Modell. Sie benutzen das Datenmodell für den Datenaustausch und besitzen eine Schnittstelle, welche einen universellen Ergebnisdatenzugriff ermöglicht. Schwierig wird dieses Vorgehen dann,

wenn Daten, welche eine bestimmte Berechnung benötigt, im Datenmodell unvollständig abgebildet sind oder sogar fehlen. Der Entwurf des Datenmodells impliziert daher die später zu verwendenden Berechnungsmethoden und ist besonders auf auch internationaler Ebene von essentieller Bedeutung für zukünftige Energiesparpolitik. Durch die Verwendung des Datenmodells und der Abstrahierung der Ergebnisverwaltung wird es möglich, wie schon früher ausgeführt (vgl. Kapitel 2.3), die Berechnungskomponenten zustandslos und mit einer Lebensdauer, welche sich auf jeweils eine Berechnung beschränkt, zu implementieren.

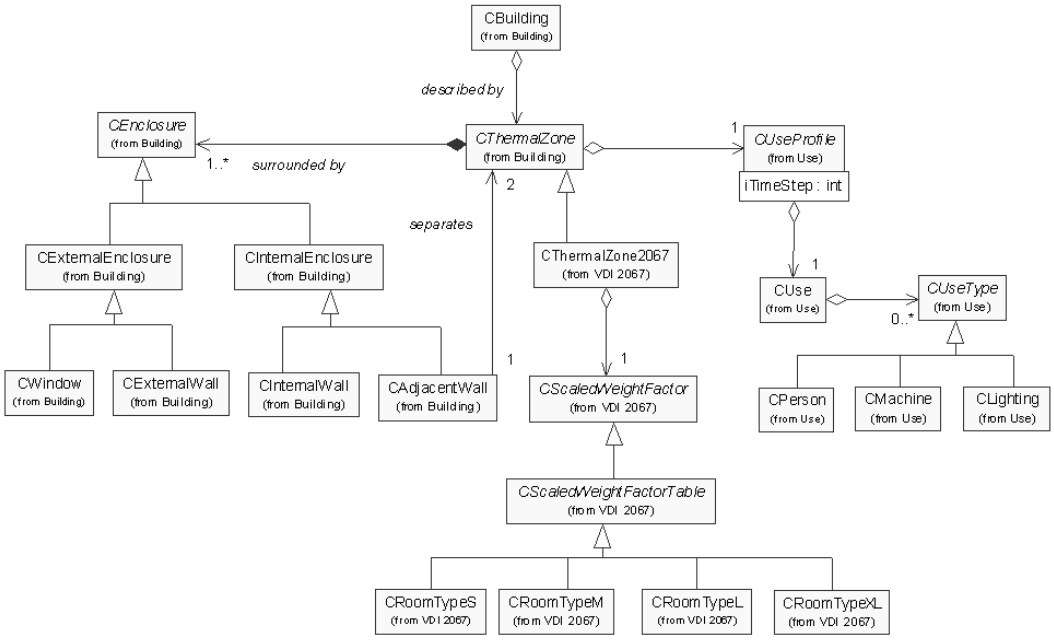


Abbildung 23: Internes Modell der Berechnungskomponente (*calcVDI2067)*

Die gebäude- und anlagebeschreibenden Eingabedaten bekommen die Berechnungskomponenten vom Datenmodell. Aus den Eingabedaten bauen sich die Berechnungskomponenten in Anlehnung an das Entwurfsmuster Erbauer [97] (siehe Anhang) das interne Modell auf. Das interne Modell ist jeweils für die spezielle Berechnung optimiert. Abbildung 23 zeigt beispielhaft das interne Modell der Berechnungskomponente für die Berechnung zur Bestimmung des Energiebedarfes von beheizten und klimatisierten Gebäuden nach VDI 2067 [96] und [98]. In das interne Modell der Berechnungskomponente sind notwendige Berechnungsalgorithmen entsprechend VDI 2067 eingebettet. Die Berechnung erfolgt in zwei Stufen. Zuerst wird für einen Grundnutzen der Energiebedarf als reine Gebäudeeigenschaft berechnet. In einem zweiten Schritt wird dann die spezielle Nutzung des Gebäudes berücksichtigt. Als Energiebedarf bezeichnet man die Energieströme, die über ein Jahr dem Gebäude zu- bzw. abgeführt werden müssen, um definierte Raumkonditionen zu halten. Als Ergebnis erhält man den Referenz-Energiebedarf für Heizen, Kühlen, Lüften, Be- und Entfeuchten. Um einen universellen Zugriff der Ergebnisdaten zu ermöglichen, besitzen die

Berechnungskomponenten eine standardisierte OLE DB Schnittstelle [99]. Diese Schnittstelle basiert vollständig auf COM. Man unterscheidet dabei im wesentlichen zwischen Anbietern und Konsumenten. Die Berechnungskomponenten als Anbieter stellen Ergebnisse zur weiteren Bearbeitung bereit. Der Aufbau von Ergebnisdaten ist durch eine chronologisch aufsteigende Aneinanderreihung der einzelnen Zeitbereiche gekennzeichnet (siehe Kapitel 4.3.3).

Die Vorgehensweise bei Zugriffen auf eine Berechnungskomponente ist immer gleich. Zuerst muss durch das Anlegen einer Datenquelle die Berechnungskomponente instanziiert werden. Als nächstes wird der Datenquelle eine Session zugeordnet. Die Session liefert den Kontext, in dem sämtliche Operationen auf der Datenquelle ausgeführt werden. Der Kontext beinhaltet die Verweise auf das zu verwendende Simulationsmodell, das durch das Datenmodell ausgetauscht wird. Über die Session kann ein Kommandoobjekt erzeugt werden. Das Kommandoobjekt bietet die Möglichkeit an, über Befehle Ergebnisdaten zu selektieren. Die Ergebnisse befinden sich in den Datensatzobjekten. Für jeden Datensatz wird ein Speicherbereich angelegt, in den die Ergebnisse für die weitere Verwendung kopiert werden. Das Auslesen der Ergebnisdaten erfolgt ausschließlich durch das Datensatzobjekt. Ein direkter Zugriff ist nicht möglich.

Die Kopplung der Berechnungskomponenten kann auf zwei verschiedene Arten erfolgen. Bei der Anbindung bereits bestehender komplexer Berechnungskomponenten, die nicht im Hinblick auf eine Nutzung über das Datenmodell entworfen wurden, geschieht die Anbindung über Implementierung anwendungsbezogener Adapter. Das Adaptermuster [97] lässt Komponenten zusammenarbeiten, die wegen ihrer unterschiedlichen Schnittstellen nicht dazu in der Lage wären (siehe Anhang). Dazu paßt der Adapter die Schnittstelle der Berechnungskomponente an die OLE DB Schnittstelle an und beliefert sie mit den Gebäudeinformationen aus dem Datenmodell. Die Informationen aus dem Datenmodell können über die von der Berechnungskomponente veröffentlichten Schnittstellen übertragen werden. Sollte die Berechnungskomponente solche Schnittstellen nicht zur Verfügung stellen, dann hat der Adapter die Aufgabe Eingabedateien zu erzeugen, welche von der Komponente eingelesen werden. Dagegen können zukünftige Berechnungskomponenten, die eine bestimmte Funktionalität anbieten und deren Schnittstellen speziell auf das Datenmodell und OLE DB Schnittstellen abgestimmt sind, direkt eingebunden werden.

4.3.3 Zeitreihen

Die Ergebnisse der Berechnungskomponenten sind in der Regel Zeitreihen. Zeitreihen sind Sammlungen numerischer Werte, geordnet nach der Zeit. Die Ergebnisse einfacher Berechnung sind häufig äquidistante Zeitreihen. Komplexere Berechnungsalgorithmen erlauben aber auch Zeitschrittweiten, die an die physikalischen Vorgänge angepasst sind (adaptive Löser). Zeitreihen werden auch zur Ordnung von Messwerten verwendet. Messwerte können infolge Messunterbrechungen eine nicht äquidistante Struktur aufweisen. Damit die Zeitreihen zur Modellierung sowohl der Berechnungsergebnisse als auch Messwerte genutzt werden können, müssen sie in der Lage sein, unterschiedlich lange Zeitintervalle zu behandeln. In Abbildung 24 wird eine Zeitreihe bestehend aus

zwei Zeitintervallen wiedergegeben. Jedes Zeitintervall wird durch die Angabe des Beginns und des Endes beschrieben, innerhalb dessen das Ergebnis Gültigkeit hat.

Die Eigenschaften der Zeitreihen sind Granularität, Regelmäßigkeit und Dichte. Die Granularität gibt den kleinsten Zeitbereich an, für welchen Werte existieren können. Liefert ein Messgerät beispielsweise jede Minute jeweils einen Wert, so hat die Zeitreihe die Granularität von einer Minute, unabhängig davon ob der Wert tatsächlich vorliegt oder nicht. Das Vorhandensein des Wertes wird als Regelmäßigkeit bezeichnet. Eine Zeitreihe ist regelmäßig, wenn Messwerte an jedem Messpunkt vorhanden sind. Ist es nicht der Fall, dann kann durch die Dichte prozentuell angegeben werden, wie viele Messwerte existieren. Folglich hat eine unregelmäßige Zeitreihe die Dichte, die kleiner ist als 100 %.

Die Zeitreihen ermöglichen Angaben zur Interpolation. Eine Interpolation ist sinnvoll, wenn die Werte aus kontinuierlichen Prozessen stammen und keine Sprünge zu erwarten sind. Im Gegensatz dazu ist die Interpolation bei diskreten Werten oder Werten, welche eine binäre Natur haben, nicht möglich. Ist die Interpolation erlaubt, dann kann für jede Zeitreihe ein Interpolationsfaktor und eine Interpolationsregel angegeben werden. Der Interpolationsfaktor legt die Anzahl der Segmente zwischen zwei existierenden Werten fest, für welche interpoliert wird. Die Interpolationsregel ist eine formale, mathematische Beschreibung der durchzuführenden Interpolation.

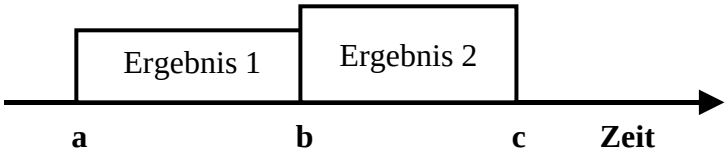


Abbildung 24: Zeitreihen

Zur Speicherung werden die Zeitreihen auf Datensätze abgebildet. Jeder Datensatz beschreibt durch drei Felder *TIME*, *VALUE* und *STATUS* jeweils einen Zeitpunkt. Das Feld *TIME* legt das Ende des Zeitintervalls fest. Das Zeitformat wird COM-konform als eine Gleitkomazahl dargestellt. Dabei ist der ganzzahlige Teil des Wertes die Anzahl der Tage seit dem 31.12.1899. Der Bruchteil repräsentiert den Bruchteil eines Tages. Das Feld *VALUE* ist der Wert des Rechen- oder Messergebnisses. Das dritte Feld *STATUS* ist eine 16-Bit Zahl, welche Informationen über die Natur und die Gültigkeit des Wertes angibt. Da das Ende eines Zeitbereiches gleichzeitig der Beginn des nächsten ist, wird der Wert am Beginn des ersten Zeitbereiches ignoriert. Er stellt lediglich der Beginn der gesamten Zeitreihe dar. Die Zeitreihe aus Abbildung 24 wird durch drei Datensätzen abgebildet, wobei der Wert am Zeitpunkt *a* ungültig ist und deshalb nicht betrachtet werden darf. Aus den Konformitätsgründen mit anderen Komponenten, hat der Wert am Zeitpunkt *a* jedoch dieselbe Größe, wie zum Zeitpunkt *b.*

4.3.4 Integration von Komponenten zu einer Anwendung

In Kapitel 4.1 wurde gezeigt, dass eine Anwendung aus der Komponente Datenmodell, verschiedenen Berechnungskomponenten und einer anwendungsbezogenen Benutzeroberfläche zusammengesetzt ist. In diesem Abschnitt soll gezeigt werden, wie sich Berechnungskomponenten miteinander verknüpfen lassen. Verschiedene Skriptsprachen bieten verschiedene Formen für die Integration der Komponenten. In dieser Arbeit wurde VBScript [100] verwendet. VBScript ist eine Untermenge von Visual Basic für Applikationen (VBA), der Programmiersprache aus Visual Basic und Microsoft Office. VBScript wurde unter der Annahme entwickelt, dass dem Programmierer eine Menge an COM-Komponenten zur Verfügung steht, die in erster Linie dafür gemacht sind, mit einander verbunden zu werden, um so eine neue Funktionalität gemeinsam anzubieten. VBScript verwendet ActiveX Skripting zur Kommunikation mit der Host-Anwendung [100]. ActiveX Skripting erspart der Host-Anwendung die Implementierung von speziellem Integrationscode für jede Skripting-Komponente. Darüber hinaus ermöglicht es einem Host die Übersetzung von Skripten, die Ermittlung und den Aufruf von Einsprungspunkten und die Verwaltung des Namenraums, der dem Entwickler zur Verfügung steht. Microsoft bietet Laufzeitunterstützung für VBScript an und arbeitet mit verschiedenen Internet-Gruppen an der Definition des ActiveX Skripting-Standards, um die Austauschbarkeit von Skripting-Modulen zu fördern. Unter anderem wird ActiveX Skripting im Microsoft Internet Explorer und im Microsoft Internet Information Server [101] verwendet.

```
Sub Calculate(mySimulations)

    myPos = mySimulations.FindFirst("CalculateBuildingPowerRequirement")
    if ( myPos > -1 ) then
        Set mySimulation = mySimulations.GetFirst(
        "CalculateBuildingPowerRequirement")
        Application.Calculate( mySimulation )
        mySimulation = Null
    end if

    myPos = mySimulations.FindFirst("CalculateGenerationEnergyEffort")
    if ( myPos > -1 ) then
        Set mySimulation =
        mySimulations.GetFirst("CalculateGenerationEnergyEffort")
        Application.Calculate( mySimulation )
        mySimulation = Null
    end if

    myPos = mySimulations.FindFirst("CalculateCosts")
    if ( myPos > -1 ) then
        Set mySimulation = mySimulations.GetFirst("CalculateCosts")
        Application.Calculate( mySimulation )
        mySimulation = Null
    end if

End Sub
```

Abbildung 25: Skript zur Berechnung von Anlagebetriebskosten nach VDI 2067

Abbildung 25 gibt ein Skript wieder, welches für die Berechnung der durch den Betrieb einer Anlage entstehenden Energiekosten nach VDI 2067 [96] benutzt werden kann. Die erste Berechnungskomponente namens *CalculateBuildingPowerRequirement* bestimmt den Gebäudeenergiebedarf. Die Zweite Komponente rechnet den Energieaufwand der

gesamten Anlage. Anschließend werden die bedarfsabhängigen Kosten von der Komponente *CalculateCosts* berechnet.

Es bietet sich an, für alle in einer Anwendung festgelegten Arbeitsabläufe Skripte abzulegen, die bei Bedarf einzeln aufgerufen werden können. Sollte die Anwendung um einen neuen Arbeitsablauf erweitert werden, dann bedarf es dazu nur der Anpassungen vorhandener Skripte und gegebenenfalls der Erstellung neuer Berechnungskomponenten, falls die vorhandenen die gewünschte Funktionalität nicht anbieten. Die anderen Teile der Anwendung bleiben von diesen Änderungen unberührt.

4.3.5 Graphische Benutzeroberfläche einer Anwendung

Die in Kapitel 4.1 beschriebene Drei-Schichten-Architektur trennt die Präsentationslogik der graphischen Benutzeroberfläche von der fachlichen Modellierung des Datenmodells und der funktionalen Schicht der Anwendung (Berechnungskomponenten). Vorteile dieser Trennung liegen darin, dass jedem Element des Datenmodells unterschiedliche graphische Repräsentationen zugeordnet werden können. Dies ermöglicht die Gestaltung von Benutzeroberflächen, die in der Lage sind, Elemente des Datenmodells in dem für die betrachtete Phase des Gebäudelebenszyklus erforderlichen Detaillierungsgrad abzubilden.

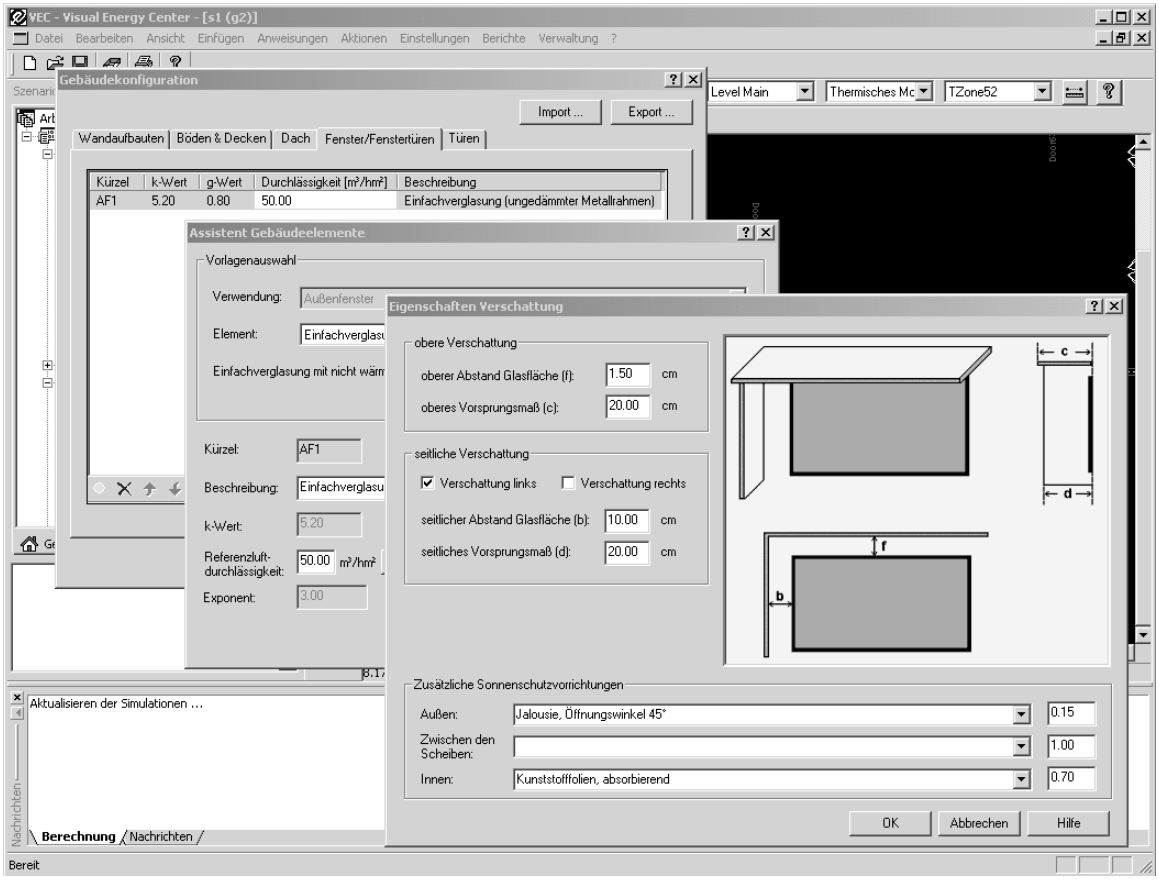


Abbildung 26: Graphische Benutzeroberfläche zur Dateneingabe

Die Benutzeroberfläche ist aus ActiveX-Steuerelementen aufgebaut. Die ActiveX-Steuerelemente sind vorgefertigte COM-Komponenten. Die Verwendung der ActiveX-Steuerelemente ermöglicht trotz der Komplexität einzelner Steuerelemente eine schnelle Entwicklung leicht erweiterbarer Benutzeroberflächen, die sowohl auf den aktuellen, von den Berechnungskomponenten vorgegebenen Datenbedarf reagieren, als auch den individuellen Bedürfnissen einzelner Benutzergruppen Rechnung tragen können. Abbildung 26 zeigt eine typische Benutzeroberfläche zur Eingabe der Gebäudedaten. Bei der Entwicklung der Benutzeroberfläche sind folgende Regeln zu beachten:

- fachliche Daten werden ausschließlich im Datenmodell gespeichert,
- der Zugriff auf fachliche Daten erfolgt über die vom Datenmodell veröffentlichen Schnittstellen,
- die vom Datenmodell generierten Ereignisse werden über den Mechanismus der Verbindungspunkte abgearbeitet,
- die von der Oberfläche ausgehenden Aktionen werden ausschließlich durch die Verwendung der in VBScript beschriebenen Arbeitsabläufe ausgeführt,
- benutzerbezogene Daten werden in der Registrierungsdatenbank gespeichert,
- Oberflächenelemente werden windowskonform angeordnet [102],
- zur Unterstützung der Mehrsprachigkeit werden alle textuellen Ausgaben in einer Ressourcendatei verwaltet.

5 Anwendungen

In Kapitel 4 wurde ein Komponentenframework zum Erstellen von Anwendungen zur Bewertung heiz- und raumlufttechnischer Anlagen beschrieben. Es wurde gezeigt, wie die Softwarekomponenten konstruiert werden müssen, damit sie mit anderen Komponenten zu einer Anwendung zusammengefügt werden können. Das hier beschriebene Framework ermöglicht das Zusammenfügen verschiedener Berechnungskomponenten, wenn sie das OLE DB Interface unterstützen und für den Austausch der Gebäude- und Anlagedaten das in dieser Arbeit entwickelte Datenmodell verwenden. In diesem Kapitel wird eine Reihe von Anwendungen vorgestellt, die während der Entwicklung des Frameworks entstanden sind. Diesen Anwendungen ist gemeinsam, dass sie auf einem zentralen Datenmodell aufbauen, mehrere Berechnungskomponenten je nach Fragestellungen enthalten, sowie verschiedene Benutzeroberflächen für unterschiedliche Nutzeranforderungen besitzen. Die Anwendungen repräsentieren verschiedene Arbeitsschritte im Lebenszyklus eines Gebäudes und weisen anschaulich nach, dass die in den vorherigen Kapiteln geforderte Wiederverwendbarkeit von Komponenten attraktiv umgesetzt werden kann. Während die ersten Anwendungen OPTIMA (Abschnitt 5.1) und RENSIM (Abschnitt 5.2) noch ein eigenes anwendungsspezifisches Datenmodell hatten, wurden die nachfolgenden Anwendungen unter Berücksichtigung der in OPTIMA und RENSIM gewonnenen Erkenntnisse und unter dem Einfluss der IAI-Standardisierungsbemühungen auf das in dieser Arbeit entwickelte gemeinsame Datenmodell aufgebaut.

5.1 OPTIMA

Im Rahmen des Forschungsprojektes INTESOL [4], wurde das Werkzeug OPTIMA ([19], [75] und [103]) weiter entwickelt. OPTIMA unterstützt die Modellierung von Gebäudedaten mit dem Ziel, eine Gebäudesimulation mit dem Simulationsprogramm TRNSYS kostengünstiger zu gestalten. Für die Gebäudesimulation benutzt TRNSYS den TYPE 56 [78], der die Berücksichtigung mehrerer thermischen Zonen ermöglicht. Eine erste Version von OPTIMA, die auf Unix basierten Workstations lauffähig war, wurde im Jahre 1995 von Hinkelmann [19] entwickelt. Abbildung 27 zeigt die Komponentenstruktur von OPTIMA. Aus Sicht dieser Arbeit ist OPTIMA aus vier Komponenten zusammengesetzt: *OPTIMA Data Model* (OPTIMA-Datenmodell), *OPTIMA GUI* (Benutzeroberfläche), *TRNSYS TYPE 56 Converter* und *Building Data Reader*.

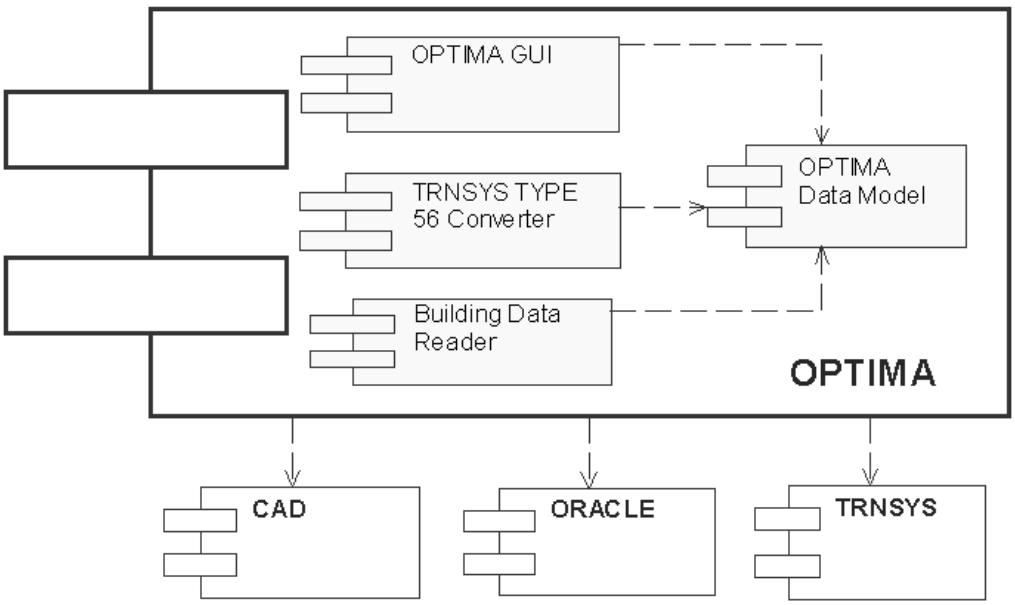


Abbildung 27: OPTIMA – Komponentenstruktur

Für die Modellierung werden die Gebäudedaten nicht vom Benutzer eingegeben, sondern aus einer von einem CAD-Programm erzeugten Datei direkt eingelesen. Zum Lesen der Daten verwendet OPTIMA die Komponente *Building Data Reader*, die das in ISYBAU [25] festgelegte Datenaustauschformat unterstützt. Aus der Perspektive des OPTIMA-Benutzers handelt es sich beim Einlesen der Austauschdatei im wesentlichen um einen Mausklick. Intern werden die Gebäudedaten in das OPTIMA-Datenmodell eingelesen und anschließend mit Hilfe des relationalen Datenbanksystems ORACLE 7 gespeichert. Sind die Gebäudedaten aus der CAD-Zeichnung übertragen, ist der OPTIMA-Benutzer in der Lage die thermische Zonierung durchzuführen. Dazu verwendet er die in Abbildung 28 wiedergegebene Benutzeroberfläche. Nachdem die thermischen Zonen eines Gebäudes festgelegt worden sind, kann durch die Auswahl der Zonen das zugehörige Zonenmodell des Gebäudes automatisch erstellt werden. Die Konsistenz des Modells wird geprüft und die Randbedingungen und die Kopplungsdaten wie z. B. die Temperatur angrenzender Bauteile oder der Luftaustausch zwischen benachbarten Zonen werden interaktiv erfragt (Abbildung 28, vordere rechte Teil). Schließlich werden vom Anwender Angaben zur Steuerung der von ihm generierten Simulation abgefragt. Alle Angaben sind auf TRNSYS TYPE 56 bezogen und legen unter anderem dessen Ausgaben fest. Zum Schluss wird aus den auf der Datenbank gespeicherten Daten die Eingabedatei für die Gebäudesimulation mit TRNSYS generiert.

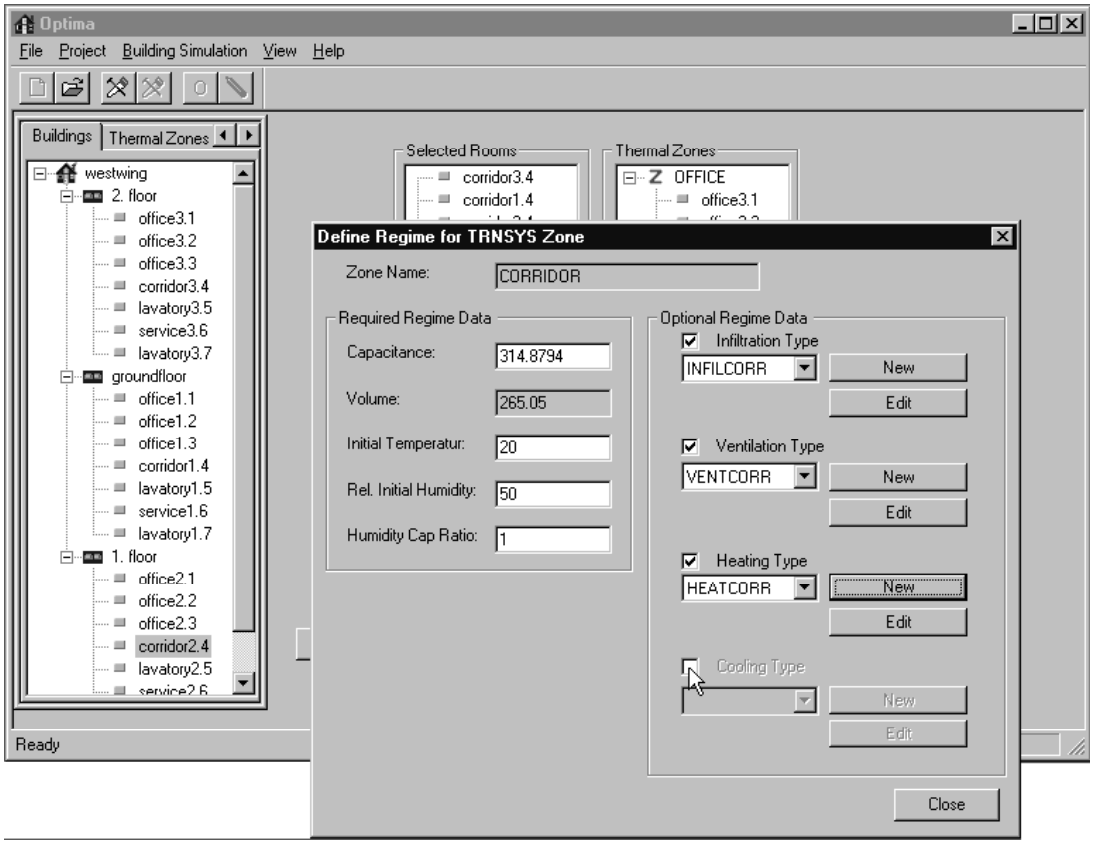


Abbildung 28: OPTIMA – Benutzeroberfläche

Die Anwendung des Programmes OPTIMA innerhalb der Beispielsprojekte aus den Arbeiten [104] und [105] hat gezeigt, dass ein Werkzeug, das auf einem zentralen dynamischen Datenmodell aufbaut, den Planungsprozess leistungsfähig unterstützen kann. OPTIMA ist geeignet, die Entscheidungsfindung im integralen Planungsprozess zu beschleunigen. Allerdings bedurfte es eines unverhältnismäßig großen Aufwandes, die Austauschdatei für Gebäudedaten im ISYBAU-Format mit Hilfe von VektorPlan3D [106] zu generieren. Für die Eingabe der Daten über VektorPlan3D mussten im Modell abzubildende Bauteile im CAD-Plan mit VektorPlan3D–Objekten nachgezeichnet werden. Durch die unsichere Datenhaltung innerhalb der VektorPlan3D-Anwendung und dadurch, dass es beim Schreiben der Austauschdatei zu Datenverlusten kommt, war die Eingabe sehr mühsam, zeitaufwendig und fehlerträchtig.

Mit dem in dieser Arbeit vorgeschlagenen Ansatz werden die Gebäudedaten aus den CAD-Architektenplänen direkt übernommen. Änderungen und Varianten können dann einfach in das während des Planungsprozesses fortgeschriebene gemeinsame Datenmodell eingebunden werden, um daraus das Modell zur thermischen Gebäudesimulation neu zu extrahieren. Derzeit wird diese Funktionalität nur von CAD-Programmen geboten, welche die IFC 1.5.1 [7] Schnittstelle unterstützen [107]. Deshalb ist als einer der nächsten Schritte in der Entwicklung der Komponente OPTIMA-Datenmodell eine Methode angestrebt, welche eine direkte Kopplung an IFC-Modell

ermöglichen soll. Die Grundlagen dafür wurden durch die Arbeiten im Rahmen des IAI Projektes BS-4 HVAC Loads Calculation [26] erarbeitet.

5.2 RENSIM

Im Rahmen des Renarch-Projekts wurde das Simulationsprogramm RENSIM [108] erstellt. Es wurde unter Einhaltung des in ISO EN 832 festgelegten Rechenalgorithmus zur Heizwärmebedarfsermittlung in Wohnhäusern implementiert. Die europäische Norm ISO EN 832 [95] basiert auf Energiebilanzen, die unter stationären Bedingungen ermittelt werden. Die Energiebilanzen werden auf monatlicher Basis und gemittelten Innen- und Außentemperaturen gerechnet. Dynamische Effekte werden lediglich mittels konstanter Faktoren berücksichtigt.

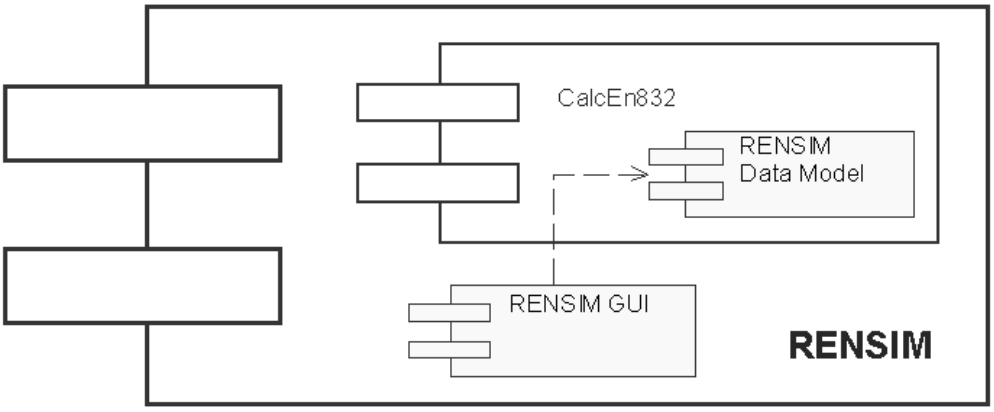


Abbildung 29: RENSIM – Komponentenstruktur

Die Anwendungsarchitektur (Abbildung 29) besteht aus der Berechnungskomponente *CalcEn832* und einer anwendungsspezifischen Benutzeroberfläche (*RENSIM GUI*). Die Berechnungskomponente beinhaltet ein internes Datenmodell (*RENSIM-Datenmodel*), der für die speziellen Bedürfnisse dieser Berechnungskomponente zugeschnitten ist.

RENSIM ist ein Lehr- und Lernprogramm. Es dient der Aus- und Weiterbildung von Architekturstudenten und Architekten im Bereich des energiesparenden Bauens. RENSIM ist durch dessen eigene, dem Anwendungszweck angepaßte Eingabeoberfläche (Abbildung 30), sehr leicht zu bedienen. Die Benutzer können ohne Einarbeitungszeit selbständig mit dem Programm arbeiten. Zur leichteren Vergleichbarkeit unterschiedlicher Gebäudeformen bzw. Gebäudekomplexe (z.B. Reihenhäuser) werden Gebäudebausteine als Gebäudegrundformen eingesetzt. Die Software baut auf Prinzipien des kognitiven Lernens auf. Der Benutzer muss keine vorgegebenen Lösungswege nachvollziehen, sondern kann sich im Programm frei bewegen. Dadurch unterscheidet sich dieses Simulationsprogramm von den früher sehr häufig angewandten Prinzipien der programmierten Unterweisung.

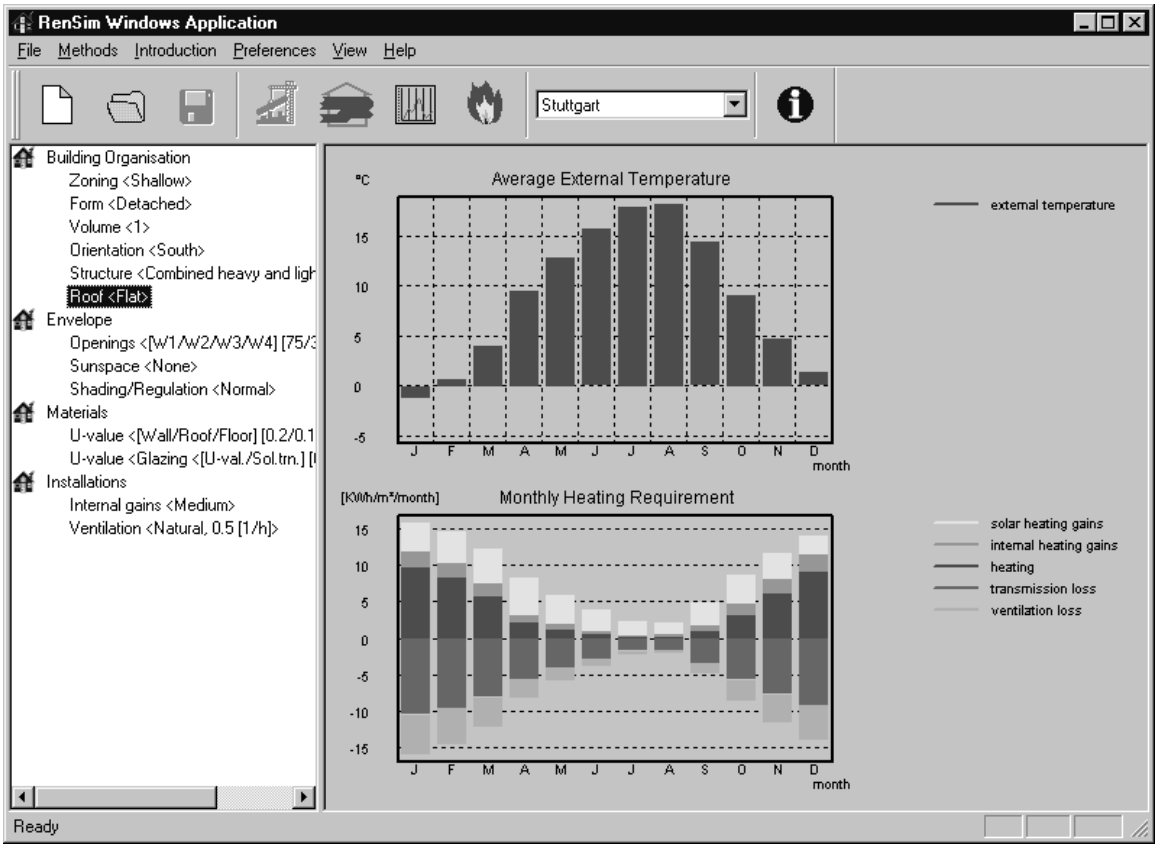


Abbildung 30: RENSIM – Benutzeroberfläche

Mit ISO EN 832 kann der Energiebedarf von Wohngebäuden für die Raumheizung und Trinkwassererwärmung berechnet werden. Damit kann RENSIM in sehr frühen Phasen des Gebäudeplanungsprozesses zur quantitativen Bewertung der Gebäude-Bauphysik herangezogen werden. Untersuchungen zum wärmetechnischen Verhalten von Heizanlagen und ihr Einfluss auf den Energiebedarf sind jedoch mit diesem Verfahren nicht möglich [2], da das dynamische Verhalten von Räumen und Heizanlagenkomponenten und die thermische Kopplung zwischen Raum, Anlage und Nutzer nicht abgebildet werden kann. Zur Bewertung heiz- und raumlufttechnischer Anlagen müssen daher Verfahren zur Berechnung der dynamischen Prozesse eingesetzt werden. Der Energiebedarf der Heizanlage kann anhand von Nutzungsgraden für die Wärmeübergabe im Raum, die Wärmeverteilung, die Wärmeerzeugung sowie die Regelung der Heizanlage erfasst werden. Die Ansätze zur Ermittlung der Nutzungsgrade sind jedoch noch nicht erarbeitet. Für die Bestimmung der Verlustgrößen hat die europäische Normungsorganisation CEN unter Vorhabensbezeichnung „WI 0228 013“ mit einem neuen Normenwerk [109] begonnen, dessen Fertigstellung für das Jahr 2001 geplant ist. Parallel dazu wird im Rahmen der neuen Fassung von VDI 2067 [110] ein Lösungsweg erarbeitet, bei dem Gebäude und Anlage anhand der Methode der Bedarfsentwicklung (vgl. Kapitel 2.1) zusammen bewertet werden. Es wird für die weitere Entwicklung von der Anwendung angestrebt, eine Komponente anhand der neuen Erkenntnisse zur Berechnung der dynamischen Prozesse im Raum zu entwickeln.

Die wichtigste Erkenntnis der vorangegangenen Entwicklungen war es zwischen einem gemeinsamen Datenmodell, das zum Datenaustausch verwendet wird, und einem für eine Berechnungskomponente internen Datenmodell, das für die jeweilige Berechnung optimiert ist, zu unterscheiden. Aus dieser Erkenntnis folgte die Entwicklung des in dieser Arbeit vorgestellten Frameworks, dem diese Trennung als architektonisches Prinzip zugrunde liegt. Deshalb wird an einer Neuimplementierung von OPTIMA und RENSIM nach den Vorgaben des hier entwickelten Frameworks gearbeitet.

5.3 SHK Energiesparcheck II

Am IKE wurde im Auftrag des Fachverbandes Sanitär-Heizung-Klima (SHK) Baden-Württemberg in Zusammenarbeit mit dem Steinbeis-Transferzentrum Energie-, Umwelt- und Reinraumtechnik das Simulationsprogramm SHK Energiesparcheck II zur wärmetechnischen Sanierung von Wohngebäuden und Heizanlagen entwickelt. Mit diesem Simulationsprogramm wird zukunftsorientierten Heizungsfachbetrieben ein Instrument an die Hand gegeben, mit dem Ein- und Mehrfamilienhäuser umfassend und herstellerunabhängig energetisch bewertet werden können. Die Energiesparpotentiale in Häusern und Wohnungen ergeben sich zum einen aus Wärmedämm-Maßnahmen am Gebäude und zum anderen aus Verbesserungen der Heizungstechnik.

Die Komponentenstruktur der Anwendung (vgl. Abbildung 31) besteht neben dem in dieser Arbeit entwickelten Datenmodell (*Data* Model) und der eigenen Benutzeroberfläche *SHK GUI* noch aus der Komponente *Calc4701* zur Berechnung der Heizlast in Anlehnung an VDI 4701 [86], der Komponente *Calc2067-2* zur Berechnung des Heizenergiebedarfes nach der alten Fassung von VDI 2067 Blatt 2 [76] und VDI 3808 [111] und der Komponente *CalcAnalysis* zur Berechnung der Betriebskosten, der Umweltbelastungen durch Schadstoffemissionen und des Primärenergieeinsatzes.

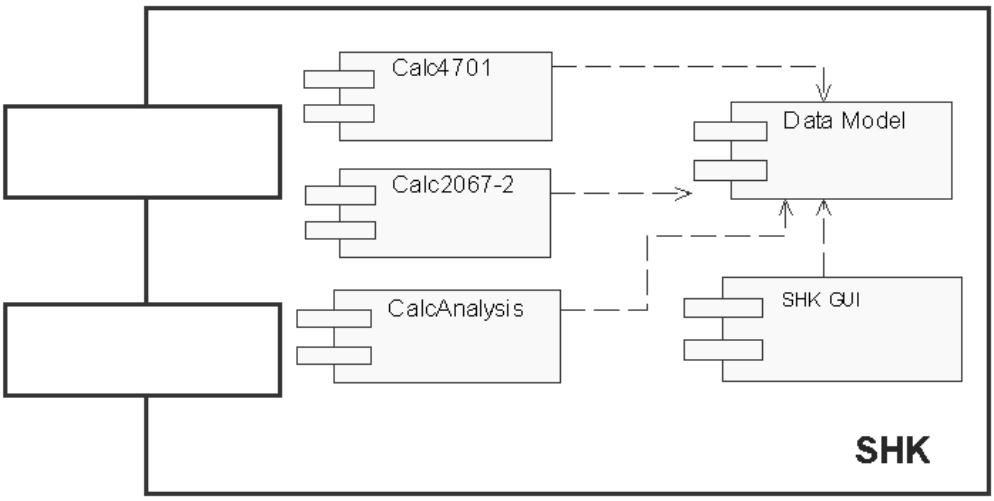


Abbildung 31: SHK Energiesparcheck II – Komponentenstruktur

Abbildung 32 zeigt die Benutzeroberfläche, mit der zunächst der Ist-Zustand des Gebäudes und der Heizanlage erfasst wird. Die dazu erforderlichen Daten werden vom Heizfachmann erhoben. Zur Ermittlung der Bauteil-k-Werte, dient eine Bauteil-Datenbank, in welcher die gängigsten Schichtaufbauten von Wänden, Decken und Dächern enthalten sind. Weiterhin wurden die bauphysikalischen Kenngrößen nach DIN 4108 Teil 4 [112] der einzelnen Aufbauten hinterlegt, um so den Benutzern möglichst eine einfache und schnelle Berechnung des k-Wertes zu ermöglichen. Er muss nur noch die Schichtdicke der vorhandenen Baumaterialien eingeben. Ist der Ist-Zustand vollständig erfasst, kann der Benutzer beliebig viele Varianten als Sanierungsmaßnahmen zur Energieeinsparung anlegen. In einer Variante kann zunächst die Durchführung einer Maßnahme berechnet werden. Sollen am Gebäude mehrere Maßnahmen durchgeführt werden, muss von der ersten Variante (z.B. verbesserte Außenwanddämmung) eine weitere Variante (z.B. Kesselsanierung) angelegt werden. Es können beliebig viele Sanierungsmaßnahmen berechnet werden. Das Simulationsprogramm ist anwendbar für Mehrfamilienhäuser mit bis zu 20 Wohneinheiten und vier Vollgeschossen und wird seit Januar 2001 vom Fachverband SHK vertrieben.

Abbildung 32: SHK Energiesparcheck II – Benutzeroberfläche

Durch die Auswertung der Vergleichsrechnungen zwischen der *alten* Fassung von VDI 2067 und der Simulation mit TRNSYS wurde deutlich [113], dass die Unterschiede zwischen den nach den verschiedenen Methoden errechneten Energiebedarfswerten verhältnismäßig gering sind, solange niedrige innere Lasten angesetzt werden. Mit Zunahme der Innenlasten werden die Differenzen jedoch immer größer. Die Ursache dafür liegt darin, dass beim Rechenverfahren nach VDI 2067 die Gewinne durch innere

Wärmequellen additiv berücksichtigt werden. Die Simulationsergebnisse zeigten weiterhin, dass bei der Beheizung mit Heizkörpern sowohl für den stetigen als auch für den unstetigen Regler der Heizenergiebedarf mit zunehmender Bauschwere abnimmt. Mit VDI 2067 hingegen wurde eine Zunahme errechnet. Dies gilt sowohl für die Betriebsweise Nachtabsenkung als auch Nachtabschaltung. Zur weiteren Verbesserung der software-technischen Unterstützung für die Bewertung heiz- und raumlufttechnischer Anlagen wurde die Entwicklung einer neuen Berechnungskomponente beschlossen. Dieser Komponente soll ein verbessertes Rechenverfahren, wie es die *neue* VDI 2067 [110] empfiehlt, zugrunde liegen, da wegen der höheren inneren Lasten und des zunehmenden Dämmstandards niedrige Heizenergiebedarfswerte eine immer bedeutendere Rolle spielen werden.

5.4 HOCHTIEF Energieberater

Die Aufgabe des Vorhabens HOCHTIEF Energieberater war die Entwicklung eines Beratungswerkzeuges, das mit wenigen Eingabeparametern (Hauptabmessungen, Gebäudekonstruktion, gebäudetechnische Ausstattung, klimatischen Randbedingungen) die während des Gebäudebetriebes zu erwartenden Energiekosten berechnet. Dabei wird eine interdisziplinäre Betrachtung angestellt. Der Zusammenhang zwischen optimalen Konstruktionsprinzipien (Fassadengestaltung, Fassadenausbildung, gewählte Materialien, Gläser etc.) und der technischen Gebäudeausrüstung hinsichtlich der Energiekosten eröffnet die Möglichkeit, mit einem Kunden eine ganzheitliche Betrachtung der Bauaufgabe durchzuführen und in einen intensiven Dialog einzutreten.

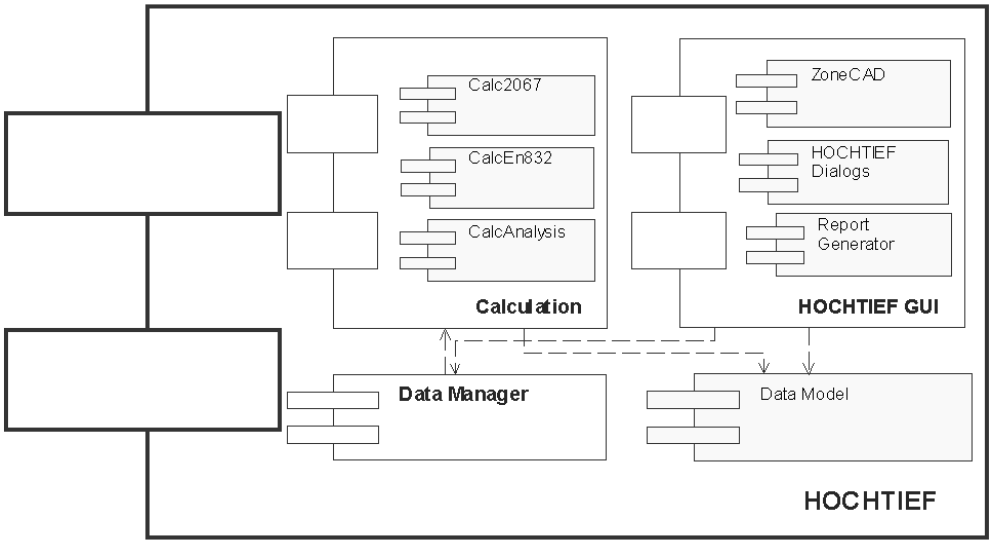


Abbildung 33: HOCHTIEF Energieberater – Komponentenstruktur

Die Anwendung HOCHTIEF Energieberater besteht aus den Bereichen: Datenmodell, Berechnungskomponenten (*Calculation*), Benutzeroberfläche (*HOCHTIEF GUI*) und der

Komponente zur Verwaltung der Simulationsergebnisse (*Data Manager*). Die Komponentenaufteilung wird in Abbildung 33 wiedergegeben.

Wie in der Anwendung SHK Energiesparcheck II ermöglicht das Datenmodell den Datenaustausch zwischen den Berechnungskomponenten und der Benutzeroberfläche. Der Bereich der Berechnungskomponenten umfasst die bereits beschriebenen Komponenten *CalcEn832* und *CalcAnalysis* (vgl. Kapitel 5.2 und 5.3), erweitert um die Komponente *Calc2067*, der das Berechnungsverfahren nach neuer Fassung von VDI 2067 [96] zugrunde liegt. Der mit diesem Berechnungsverfahren ermittelte Wärme- und Stoffstrom erfüllt zu jedem Zeitpunkt die Nutzenanforderungen. Das zeitliche Integral der Lasten – Jahresheiz- bzw. Jahreskühlenergiebedarf – wird dann als energetische Vergleichsgröße (Referenzenergiebetrag) für die nachfolgenden Prozesse herangezogen. Die Richtlinie gilt für alle Zonen des Gebäudes, für die Nutzenanforderungen eingehalten und denen zufolge Energieströme zu- bzw. abgeführt werden müssen.

Eine Besonderheit des HOCHTIEF Energieberaters ist der Einsatz der Komponente *Data Manager* zur Verwaltung der Simulationsergebnisse. Der *Data Manager* hat seinen Ursprung in einer Entwicklung. mit der im Rahmen des Projektes REUSE [114] zeitreihenbehafteten Messdaten erfasst und über das Internet dargestellt werden. Die Verwendung des abstrakten Datentyps Zeitreihen ermöglicht es dem *Data Manager*, die zeitreihenbehafteten Daten unabhängig von ihrer Herkunft (Simulation oder Messung) zu speichern und zu vergleichen. Dies eröffnet neue Perspektiven für den Einsatz des HOCHTIEF Energieberaters während des Betriebs einer Anlage. So können etwa Vergleiche zwischen berechneten (Soll-) und gemessenen (Ist-Werten) durchgeführt werden. Ferner ist es möglich, Basisoperationen auf Zeitreihen wie etwa für Auswertungen notwendigen Umrechnungen bereits durch den *Data Manager* erledigen zu lassen.

Die Beschreibung des Gebäudes muss dem Stand der Planung entsprechend erfolgen. Da es sich beim HOCHTIEF Energieberater um eine Anwendung handelt, die für die Planung realer, zum Teil sehr komplexer, Büro- und Schulgebäude eingesetzt wird, bildete die Dateneingabe für die Gebäudebeschreibung eine der Hauptherausforderungen des Projektes überhaupt. Hierbei ist von ganz unterschiedlichen Situationen auszugehen. Sie reichen von textuellen Beschreibungen (Baugesuch, Raumbuch über Pausen von CAD-Plänen) bis hin zu CAD-Plänen in elektronischer Form. Liegen CAD-Pläne in elektronischer Form vor, so werden in Zukunft die für die thermische Lastberechnung notwendigen Daten über die IFC-Klassen (Release IFC 2x) ausgetauscht. Liegen die Pläne jedoch noch nicht in elektronischer Form vor, so erlaubt die Benutzeroberfläche des HOCHTIEF Energieberaters die Zonierung des Gebäudes. Als Basisformen werden Formen zugelassen, die durch Polygone aufbaubar sind. Abbildung 34 zeigt einen Ausschnitt der Benutzeroberfläche des HOCHTIEF Energieberaters. Die Komponente *ZoneCAD* (Abbildung 34, rechts) bestimmt aus der geometrischen Anordnung die für die thermische Gebäudesimulation nötigen geometrischen Daten. Die Zonen sowie die sie begrenzenden Wände und deren Durchbrechungen, werden intern als Objekte behandelt, so dass ihnen neben den geometrischen Eigenschaften auch andere Eigenschaften (z.B. Nutzung, bauphysikalische Werte, usw.) mitgegeben werden können. Dadurch erhält die Benutzeroberfläche Funktionalitäten, die sonst von CAD-Programmen erwartet werden.

In früheren Phasen der Angebotserstellung sind diese aber nur rudimentär ausgebildet. Die Komponente *ZoneCAD* ersetzt demzufolge nicht ein CAD-Programm, unterstützt aber wohl die Ordnung von planerischen Gedanken.

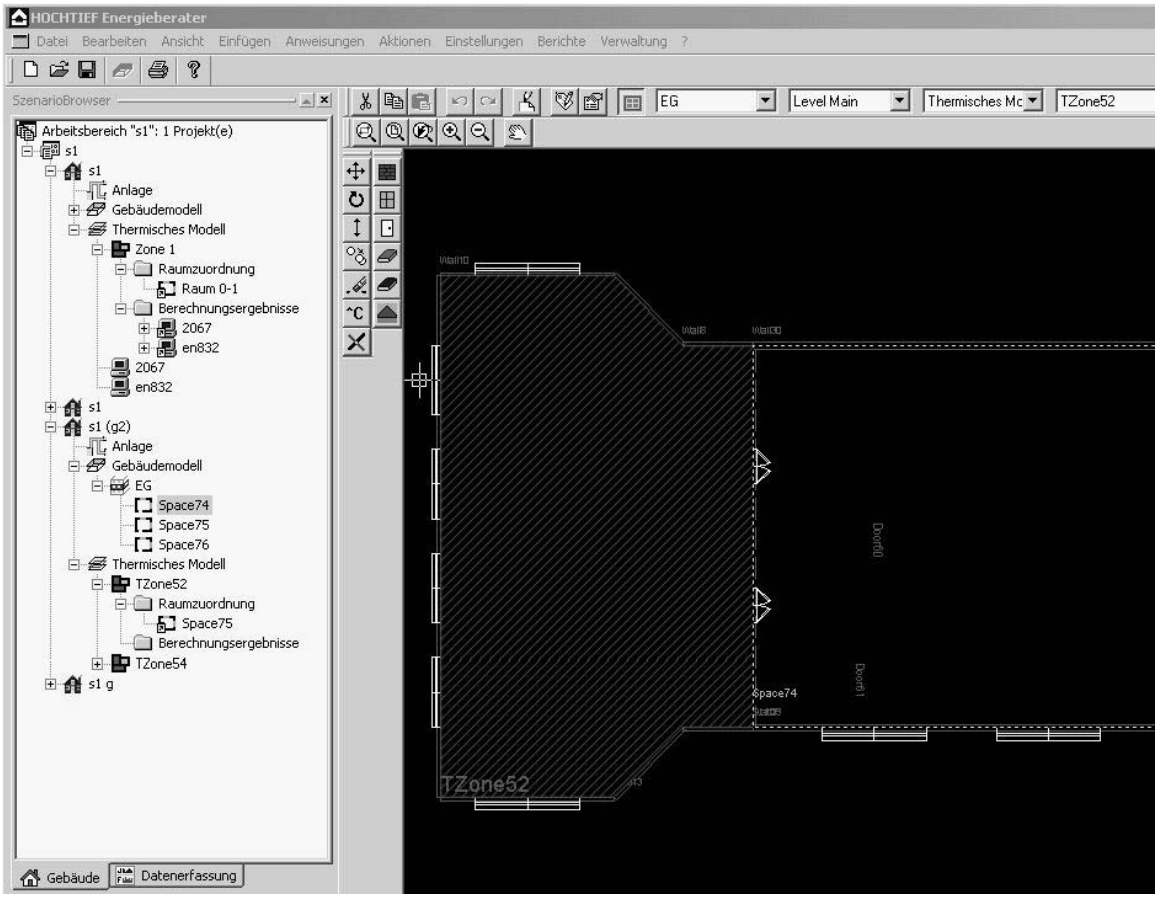


Abbildung 34: HOCHTIEF Energieberater – Benutzeroberfläche

Die Darstellung der Zonen, ihrer Eigenschaften und ihrer Verbindungen erfolgt nicht nur graphisch, sondern auch tabellarisch. Für die tabellarische Darstellung wurden verschiedene Sichten auf die Zone (Geometrie, Nutzung, Materialien, Anlagen) entwickelt, welche durch unterschiedliche Dialoge (*HOCHTIEF Dialogs*) dem Benutzer präsentiert werden. Darüber hinaus enthält die Benutzeroberfläche noch die Komponente *Report Generator* zur visuellen Darstellung aller für das Projekt relevanten Dokumente. Dokumente können entweder in Form von HTML-Berichten oder als Graphiken dargestellt werden.

5.5 VEC Visual Energy Center

In den vorangegangenen Abschnitten dieses Kapitels wurde gezeigt, dass das in Kapitel 4 vorgestellte Komponentenframework die Entwicklung einer Vielzahl unterschiedlicher Anwendungen ermöglicht. Diese Anwendungen zeichnen sich insbesondere dadurch aus,

- dass sie alle die Datenmodell-Komponente benutzen,

- dass sie bezogen auf die aktuelle Datenlage unterschiedliche Berechnungskomponenten und graphische Oberflächen enthalten und
- dass sie Daten und Berechnungskomponenten aus früheren Phasen im Lebenszyklus des Gebäudes konsequent nutzen.

Damit kann das Datenmodell in allen Phasen des Lebenszyklus eines Gebäudes genutzt werden. Die Datenerhebung findet nur einmal statt und ist nicht für jede Berechnung von neuem nötig. Auf der anderen Seite ist durch das offene Datenmodell die Bedeutung der Daten transparent. Sie können deswegen von verschiedenen Berechnungskomponenten benutzt werden und es wird möglich, ihre Konsistenz in Bezug auf die Berechnung zu prüfen. Der Einsatz von Simulationen zur Bewertung heiz- und raumlufttechnischer Anlagen wird dadurch wesentlich erleichtert und darüber hinaus kostengünstig durchführbar.

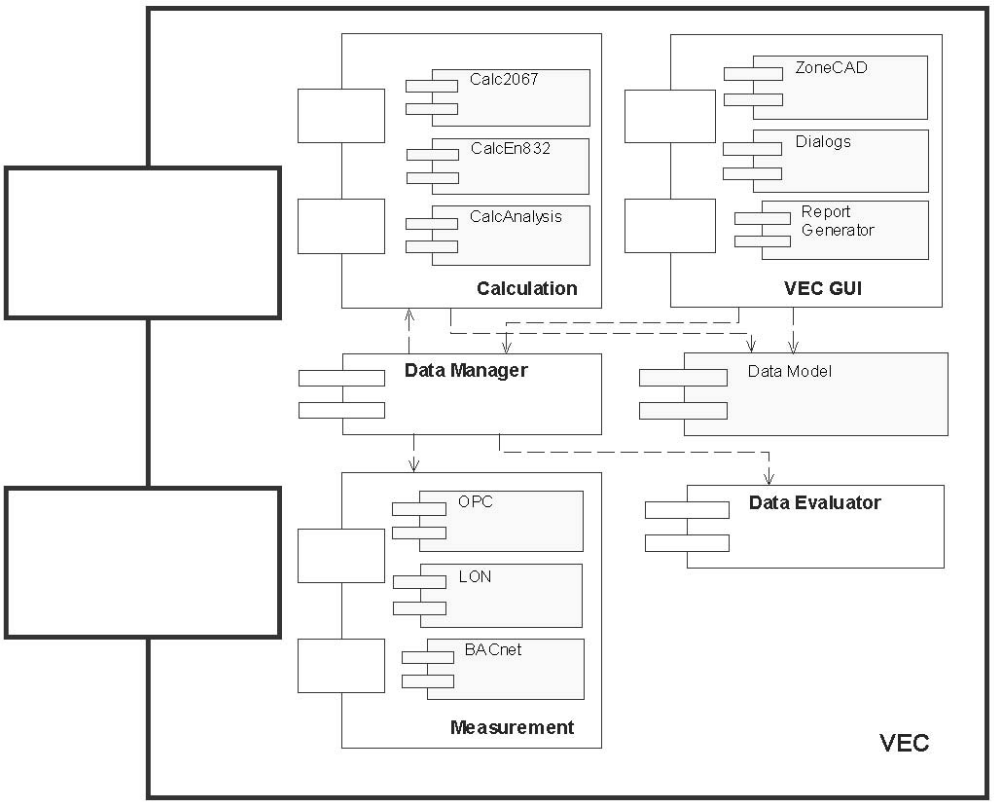


Abbildung 35: Visual Energy Center – Komponentenstruktur

Diese Erkenntnisse haben dazu geführt, dass aus der Universität Stuttgart eine Firma (Ennovatis GmbH) ausgegründet wurde, die auf Basis dieser Entwicklung die rationelle Nutzung von Energie in Gebäuden so attraktiv machen will, dass sie bei einer Vielzahl von Liegenschaften kostensenkend durchführbar wird. Sie entwickelt dafür auf Basis des

geschilderten Komponentenframeworks ein eigenes Produkt, das Visual Energy Center (VEC) [115]. Dieses Produkt konzentriert sich auf die Phasen Gebäude- und Anlagenentwurf und Anlagebetrieb. Durch den Entwurf des Gebäudes werden die Randbedingungen für den späteren Betrieb festgelegt (Energiebedarf). Der Entwurf der Anlage bestimmt den Aufwand. Wird der Entwurf von Gebäude und Anlage gemeinsam durchgeführt, so können im Sinne des bedarfsorientierten Vorgehens (vgl. Kapitel 2.1) die größten Erfolge bei der rationellen Nutzung von Energie erzielt werden.

Das während der Entwurfsphase entstandene, digitale Gebäude- und Anlagedatenmodell kann im Laufe der Planung verfeinert und dann als sog. Referenzmodell für den Soll-/Ist-Vergleich zur Verfügung gestellt werden. Deswegen können während der Betriebsphase dieselben Berechnungsmethoden wie während des Entwurfs eingesetzt werden. Der eigentliche Verbrauch wird durch den tatsächlichen Betrieb (Nutzung) und die tatsächlichen Betriebsrandbedingungen (z.B. Meteorologie) bestimmt. Werden diese Berechnungsmethoden mit Daten des Referenzmodells verwendet, beschreiben sie das ideale Verhalten des Gebäudes. Abweichungen davon, erfasst über Messungen, weisen abgesehen von Messfehlern auf unverstandene Änderungen, meist Fehler im Betrieb hin. Daraus ergab sich, dass VEC gegenüber HOCHTIEF Energieberater vor allem um Komponenten zur Messdatenerfassung (*Measurement)* und Datenauswertung (*Data Evaluator*) erweitert werden musste. Dies ist in Abbildung 35 zu sehen.

6 Schlussbetrachtung

Es ist aus der Sicht der Gebäudetechniker unbestritten, dass die Gesamtbewertung von Gebäuden und Anlagetechnik zielführend im Sinne einer gewünschten Reduzierung der Energiekosten und des -verbrauches sein muss. Die energetische Bewertung ist wegen der Vielfalt der Gestaltungsmöglichkeiten für das Gebäude und der Vielzahl der Anlagekonzeptionen und Anlagekomponenten keine einfache Aufgabe. Zudem werden aus der Notwendigkeit eine einheitliche Betrachtung über den gesamten Gebäudelebenszyklus anzustellen, erhöhte Anforderungen (z.B. eine dynamische Betrachtungsweise) an die Bewertungsmethodik gestellt. Die erhöhten Anforderungen bewirken, dass unterschiedliche Berechnungskomponenten in verschiedenen Phasen des Gebäudelebenszyklus zum Einsatz kommen. Damit die Berechnungskomponenten benutzt werden können, müssen Gebäude und deren heiz- und raumlufttechnische Anlagen in sich und für die Berechnungskomponenten konsistent beschrieben werden. Die dazu nötigen Daten unterscheiden sich während der einzelnen Phasen des Gebäudelebenszyklus nicht nur in der Menge, sondern auch in ihrer semantischen Qualität. Eine Übertragung zwischen einzelnen Phasen und Programmen war daher bisher aufwendig und häufig mit Datenverlusten verbunden. Durch die Entwicklung eines Frameworks zur Bewertung heiz- und raumlufttechnischer Anlagen, das eine Komponente Datenmodell enthält, über die der Datenaustausch während des gesamten Gebäudelebenszyklus erfolgen kann, wird der Aufwand beträchtlich reduziert und Datenverluste bei der Übertragung und Konvertierung weitgehend vermieden.

In der vorliegenden Arbeit wurde ein Komponentenframework vorgestellt, mit dem eine Vielzahl von Anwendungen zur Bewertung heiz- und raumlufttechnischer Anlagen erstellt werden kann. Dem Komponentenframework liegt eine Architektur zugrunde, welche sich durch die Trennung von Daten und Berechnungskomponenten auszeichnet. Die Daten zur Beschreibung des Gebäudes und dessen heiz- und raumlufttechnischen Anlagen sind im Datenmodell abgebildet. Der Entwurf und die Implementierung des Datenmodells in Form eines COM Servers stellt einen der Schwerpunkte dieser Arbeit dar. In erster Linie ist es die Aufgabe des Datenmodells, den Datenaustausch zwischen Komponenten zu ermöglichen. Das Datenmodell legt fest, welche Objekte zwischen Komponenten ausgetauscht werden und wie diese Austauschobjekte auszusehen haben. Die Aufgaben des Datenmodells gehen jedoch über den Datenaustausch hinaus, da es zur einheitlichen Beschreibung von Gebäuden und deren Anlagen während des gesamten Lebenszyklus dient. Der Rahmen für die Entwicklung des Datenmodells ist durch das aktuelle IFC-Austauschmodell vorgegeben. Das Datenmodell erweitert dieses Modell um die für die gekoppelte Gebäude- und Anlagesimulation nötige Beschreibung heiz- und raumlufttechnischer Anlagen. Durch die Mitentwicklung des IFC-Modells gelang es, das Datenmodell nicht nur mit einer proprietären, sondern mit einer international normierten Schnittstelle auszustatten.

Das Komponentenframework ermöglicht das Zusammenfügen verschiedener Berechnungskomponenten. Die Voraussetzung dafür ist, dass sie das OLE DB Interface unterstützen und für den Austausch der Gebäude- und Anlagedaten das in dieser Arbeit entwickelte Datenmodell verwenden. Die Berechnungskomponenten liefern ihre

Ergebnisse im Form von Zeitreihen. Die Abstraktion der Zeitreihen vereinfacht die Verwaltung der Berechnungsergebnisse und ermöglicht deren direkten Vergleich. Berechnungskomponenten können als selbständige Komponenten vom Drittanbietern angeboten werden, da ihre internen Modelle meistens vom Gesetzgeber vorgegeben und dementsprechend nicht beeinflussbar sind.

Durch die in Kapitel 5 beschriebenen Anwendungen wurde die grundsätzliche Anwendbarkeit des in dieser Arbeit entwickelten Komponentenframeworks aufgezeigt. Durch dessen Einsatz war es möglich:

1. verschiedene Anwendungen mit einem vertretbaren Aufwand zu entwickeln
2. eingesetzte Komponenten wiederzuverwenden
3. Gebäude- und Anlagedaten in unterschiedlichen Detaillierungsstufen zwischen Komponenten auszutauschen
4. verschiedene Phasen im Lebenszyklus eines Gebäude zu betrachten
5. unterschiedliche Benutzergruppen anzusprechen.

Die Potentiale der entwickelten Anwendungen liegen darin, dass sie eine kostengünstigere Bewertung heiz- und raumlufttechnischer Anlagen ermöglichen, da etwa mehrere Varianten bei gleichem Zeitaufwand geplant werden können. Die Ergebnisse der durchgeführten Bewertung sind reproduzierbar, was dazu führt, dass die getroffenen Planungs- und Betriebsentscheidungen zu jedem Zeitpunkt nachvollzogen werden können. Zugleich werden durch die Befreiung von fehleranfälliger Routinearbeit die Ergebnisse zuverlässiger. Dies führt letztendlich zu einer Beschleunigung des Planungsprozesses und einer Erhöhung dessen Qualität. Schließlich werden die Planungsdaten über die Phase der Planung hinaus nutzbar. Sie bilden die Grundlage für die Abnahme der Anlage und die Bewertung ihres Verhaltens. Damit werden wichtige Voraussetzungen für eine bedarfsorientierte Betriebsführung und Einführung von modernen Methoden zur Fehlererkennung und -diagnose gelegt.

Die weitere Entwicklung des in dieser Arbeit vorgestellten Komponentenframeworks kann in zwei Richtungen erfolgen. Zunächst muss die semantische Modellierung im Datenmodell verbessert werden. Die inhaltlichen Aspekte der Datenmodellelemente beruhen auf der Annahme, dass deren Semantik und operationale Schnittstelle durch ein Normierungsgremium festgelegt werden. Diese Vorgehensweise ist nützlich und angemessen, solange es um allgemeingültige, universell verwendbare Komponenten geht. Das dahintersteckende Prinzip der Festlegung von Semantik versagt aber, sobald es um die Integration von Komponenten geht, für die es keine solche Absprache gibt (z.B. Integration vorhandener Berechnungssysteme). Wenn immer Softwarekomponenten unabhängig von einander und ohne gemeinsames konzeptionelles Modell entwickelt werden, entsteht eine *semantische Heterogenität*. Das Datenmodell in der aktuellen Fassung kann das Problem der semantischen Heterogenität nicht lösen. Mit der Etablierung einer großen Anzahl der standardisierten Schnittstellen im Datenmodell versuchen wir zur Zeit, einen Teil des Problems zu entschärfen. Dies wird durch die Entwicklung des Komponentenframeworks erleichtert, da eine Infrastruktur angeboten

wird, die einen universellen Rahmen zur Beschreibung des Gebäudes und dessen Anlage bietet.

Das Problem der semantischen Heterogenität lässt sich auf zwei unterschiedliche Arten lösen: Die erste Vorgehensweise beruht auf einer externen, manuellen Zusammenführung und damit letztendlich wiederum auf einer außerhalb des Systems stattfindenden Absprache über die Elemente und ihre Schnittstellen. Die vorgeschlagene Vorgehensweise hat aber einen statischen Charakter; die Integration erfolgt nicht zur Laufzeit und ist nicht transparent. Eine zweite Ansatz kommt der Vision autonomer, dynamischer, leicht integrierbarer Komponenten näher. Es wird versucht, Mechanismen bereitzustellen, mit denen eine semantische Beschreibung der Komponenten innerhalb des Systems ermöglicht wird. Ist dies der Fall, so kann die Integration automatisch und für den Endanwender transparent erfolgen.

Die zweite Entwicklungsrichtung des Komponentenframeworks betrifft den Aspekt der Weiterentwicklung der Anwendungen. Auf Grund der vorgestellten Architektur wird es möglich, weitere Phasen im Lebenszyklus eines Gebäudes zu erschließen. So könnte etwa ein intelligentes Fehlererkennungs- und Diagnosesystem Anlagefehler erkennen und lokalisieren und somit weitere Energieeinsparpotentiale aufdecken. Heutige Steuer- und Regelungssysteme für Gebäude können eine Vielzahl von Messpunkten aufnehmen und anzeigen. Der Umfang der anfallenden Daten übersteigt die Fähigkeit gewöhnlicher Bediener, diese Daten in allen notwendigen Details auszuwerten und zu verstehen. Dadurch werden Fehler nicht rechtzeitig erkannt oder aufgrund der komplexen Abhängigkeiten innerhalb des Systems falsch interpretiert. Dies hat zu Folge, dass auf Energiemanagement nahezu vollständig verzichtet wird. In den Folgearbeiten bleibt zu prüfen, wie Systeme zur bedarfsorientierten Betriebsführung, Fehlererkennung und Diagnose auf der Basis der hier beschriebenen Komponentenarchitektur zu entwickeln sind. Dabei liegen die Vorteile des hier vorgestellten Ansatzes darin, dass über das Datenmodell auch in der Betriebsphase auf ein umfassendes und bereits verifiziertes Wissen über das Gebäude und die Anlagen zurückgegriffen werden kann.

7 Literaturverzeichnis

[1] Wirtschaftsministerium Baden-Württemberg: Energiebericht '96. April 1997.

[2] Bauer, M.: Methode zur Berechnung und Bewertung des Energieaufwandes für die Nutzenübergabe bei Warmwasserheizanlagen. Dissertation, Mitteilung Nr.3, IKE-LHR, Universität Stuttgart, 1999.

[3] Hirschberg, R.: Rechnergestützte Planung heiz- und raumlufttechnischer Anlagen. Dissertation, IKE, Universität Stuttgart, 1995.

[4] INTESOL: Integrale Planung solaroptimierter Gebäude. Abschlussbericht, IKE 4-155, IKE, Universität Stuttgart, Dezember 2000.

[5] Domke, M.; Jensch, W.: Zwischenbericht des Verbundprojekts INTESOL (Berichtszeitraum Januar - Juni 1998). Ebert Ingenieure, 1998.

[6] Industrieallianz für Interoperabilität: http://www.iai-ev.de/.

[7] Industry Foundation Classes – Release 1.5.1, IFC Object Model for AEC Projects. International Alliance for Interoperability, 1997.

[8] Szyperski, C.: Component Software – Beyond Object-Oriented Programming. Addison-Wesley, 1998.

[9] Energiesparverordnung ESPA: Verordnung über energiesparenden Wärmeschutz und energiesparende Anlagetechnik bei Gebäuden. Bundesministerium für Verkehr, Bau- und Wohnungswesen, Referentenentwurf, Juni 1999.

[10] Madjidi, M.: Beitrag zur modellbasierten Überwachung und Optimierung des Betriebs heiz- und raumlufttechnischer Anlagen. Dissertation, Mitteilung Nr.2, IKE-LHR, Universität Stuttgart, 1996.

[11] Deutsches Institut für Normung DIN: Die Schnittstellen der rechnerintegrierten Produktion (CIM) – CAD und NC-Verfahrenskette. DIN-Fachbericht 20, Beuth-Verlag, 1989.

[12] Anderl, R.; Schilli, B.: Eine Schnittstelle zum Austausch integrierter Modelle, CAD-Datenaustausch und -Datenverwaltung – Schnittstellen in Architektur, Bauwesen und Maschinenbau. Beiträge zur Graphischen Datenverarbeitung, Springer-Verlag, 1988.

[13] Pilz, M.: Integrierte Informationstechnologie für Ingenieuranwendungen. Dissertation, Rechnerzentrum der Universität Stuttgart, Universität Stuttgart, 1997.

[14] ISO (1995) Product Data Representation and Exchange, Part 225: Building Elements using Explicit Shape Representation. Committee Draft, ISO TC184/SC4/WG3 N434, 1995.

[15] Haas, W.: CAD-Schnittstellen im Bauwesen, CAD-Datenaustausch und -Datenverwaltung. Springer-Verlag, 1988.

[16] ISO (1994) Product Data Representation and Exchange, Part 228: Building Services: Heating, Ventilation and Air Conditioning. Committee Draft, ISO TC184/SC4/WG3 N343, 1994.

[17] Böhms, M.; Storer, G.: ATLAS – Architecture, Methodology and Tools for computer-integrated large Scale Engineering. Proceedings JSPE-IFIP, WG 5.3 Workshop, 1994.

[18] ISO (1994) Product Data Representation and Exchange, Part 11: The EXPRESS Language Reference Manual. International Standard, ISO 10303-11, National Institute of Standardization, 1994.

[19] Hinkelmann, M.: Entwicklung eines Produktdatenmodells zur Unterstützung der integralen Planung von Gebäuden und ihrer heiz- und raumlufttechnischen Anlagen. Dissertation in Vorbereitung, IKE 4-153, IKE, Universität Stuttgart, 2000.

[20] Augenbroe, G.: Combine 2 – EU DG XII Joule. TU Delft, Final Report JOU2-CT92-0196, 1995.

[21] Enkovaara, E.; Salmi, M.; Sarja, A.: RATAS Project – Computer Aided Design for Construction. Building Book Ltd., 1988.

[22] Industry Foundation Classes – Release 2x, IFC Object Model for AEC Projects. International Alliance for Interoperability, 2000.

[23] Bray, T. et al.: Extensible Markup Language (XML) 1.0. W3C World Wide Web Consortium, (http://www.w3.org/TR/1998/REC-xml-19980210), 1998.

[24] VDI 6021 Blatt 1: Datenaustausch für die thermische Lastberechnung von Gebäuden. VDI-Verlag, Entwurf, Januar 1998.

[25] ISYBAU: Entwicklung und Einführung eines integrierten DV-Systems in Bauwesen. Hauptuntersuchung - BMBau, 1986.

[26] Building Services (BS-4) HVAC Loads Calculation. Process Definitions, IFC Domain Project Documentation (Draft 4), International Alliance for Interoperability, 1998.

[27] VDI 6027 Blatt 2: Anforderungen an den Datenaustausch von CAD-Systemen; Anlagetechnik. Arbeitspapier, Dezember 1999.

[28] VDI 3805 Blatt 1: Produktdatenaustausch in der TGA, Grundlagen. VDI-Verlag, August 1996.

[29] VDI 3805 Blatt 2: Produktdatenaustausch in der TGA, Heizungsarmaturen. VDI-Verlag, Oktober 1998.

[30] VDI 3805 Blatt 3: Produktdatenaustausch in der TGA, Wärmeerzeuger und Zubehör. VDI-Verlag, Februar 1998.

[31] VDI 3805 Blatt 4: Produktdatenaustausch in der TGA, Pumpen. VDI-Verlag, Entwurf, Januar 1998.

[32] VDI 3805 Blatt 5: Produktdatenaustausch in der TGA, Luftdurchlässe. VDI-Verlag, Entwurf, November 1997.

[33] VDI 3805 Blatt 6: Produktdatenaustausch in der TGA, Heizkörper. VDI-Verlag, Entwurf, November 1998.

[34] Haller, R.; Sucic, D.: Einsatz von Planungswerkzeugen in der integralen Planung. HLH, Dezember 1999.

[35] Parnas, D. L.: On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM, Februar 1972.

[36] Mili, H.; Mili, F.; Mili, A.: Mass-Produced Software Components. Buxton J.M., Naur P., Rendell B. (Hrsg.), Software Engineering Concepts and Techniques, 1968 NATO Conference on Software Engineering, Petrocelli/Charter, 1976.

[37] Nierstraz, O.; Lumpe, M.: Komponenten, Komponentenframeworks und Gluing. HMD-Heft zur Theorie und Praxis der Wirtschaftsinformatik, Nr. 197, September 1997.

[38] d'Souza, D.F.; Wills, A.C.: Objects Components and Frameworks with UML – The Catalysis Approach. Addison-Wesley, 1998.

[39] Lippman, S. B.: Inside the C++ Object Model. Addison-Wesley, 1996.

[40] Brown, N.; Hindel, Ch.: Distributed Component Object Model Protocol – DCOM 1.0. Internet-Draft, Microsoft Corporation, November 1996.

[41] Schill, A.: Das OSF Distributed Computing Environment – Einführung und Grundlagen. Springer-Verlag, 1993.

[42] Brockschmidt, K.: Inside OLE 2. Second Edition, Microsoft Press, 1995.

[43] Sheperd, G.; King, B.: Inside ATL, Microsoft Press, 1999.

[44] Object Management Group: Object Management Architecture Guide, Soley, R. .M (ed.). Third Edition, John Wiley & Sons, Juni 1995.

[45] Redlich, J.P.: Corba 2.0 – Praktische Einführung für C++ und Java. Addison-Wesley, 1996.

[46] Hornig, P: Die Kinder der OMA – Ein Überblick über CORBA-Implementierungen. HMD-Heft zur Theorie und Praxis der Wirtschaftsinformatik, Nr. 186, 1995.

[47] Object Management Group: Object Management Architecture. OMG Document AB/96-08-01, August 1995.

[48] Object Management Group: Object Services Architecture. Revision 6.0, OMG Document 92.8.4, August 1992.

[49] Object Management Group: Common Facilities Architecture. Revision 4.0, OMG Document 95-01-02, Januar 1995.

[50] Object Management Group: Unified Modeling Language Proposal. Submission to the OMG OA&D RFP-1 Version 1.1, OMG TC Documents ad/97-08-02 to ad/97-08-10, September 1997.

[51] Booch, G.; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1998.

[52] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[53] Fowler, M.; Scot, K.: UML Distilled – Applying the Standard Object Modeling Language. Addison-Wesley, 1997.

[54] Sun Microsystems Homepage: http://www.sun.com.

[55] Flanagan, D.: Java in a Nutshell. O'Reilly & Associates, 1996.

[56] Hamilton, G.: JavaBeans 1.01 Specification. Sun Microsystems, Juli 1997.

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

[58] O'Neil, J.: JavaBeans – Programming from the Ground Up. Osborne, 1998.

[59] Calder, B.; Shannon, B.: JavaBeans Activation Framework Specification. Sun Microsystems, 1997.

[60] Colan, M.: InfoBus 1.2. Specification, Sun Microsystems, Februar 1999.

[61] Udell, J.: Component Software. BYTE 5/94, Mai 1994.

[62] Microsoft: COM for Solaris,
http://www.microsoft.com/com/resources/solaris.asp.

[63] Microsoft: COM for OpenVMS,
http://www.openvms.compaq.com/openvms/products/dcom.

[64] Software AG: EntireX DCOM on Linux,
http://www.softwareag.com/entirex/download/free_download.html.

[65] Mittasch, C.; Schill, A.: Brokerage Service. Encyclopedia of Distributed Computing, Kluwer Academic Publishers, 1998.

[66] Schulze, W.: Workflow-Management für CORBA-basierte Anwendungen. Springer-Verlag, 2000.

[67] Schöckle, M.: Modellierung und Simulation komplexer technischer Systeme bei Verwendung des objektorientierten Paradigmas. Dissertation, IKE 4-152, IKE, Universität Stuttgart, Dezember 1999.

[68] Grohmann, A.: Entwicklung und Erprobung eines Dienstleistungskonzeptes zur Integration von Simulationen in die Kernreaktor-Fernüberwachung. Dissertation in Vorbereitung, IKE 4-155, IKE, Universität Stuttgart, Januar 2001.

[69] Meyer, B.: Object-Oriented Software Construction. Prentice Hall, 1988.

[70] Shaw, M.; Garlan, D.: Software Architecture – Perspective on an emerging discipline. Prentice Hall, 1996.

[71] Fowler, M.: Analysis Patterns – Reusable object models. Addison Wesley, 1997.

[72] Building Services (BS-7) HVAC Performance Validation. Process Definitions, IFC Domain Project Documentation (Draft 4), International Alliance for Interoperability, 2000.

[73] Sucic, D.: Generische Implementierung von Objekt-Assoziationen in verteilten Systemen. Diplomarbeit, 4 D-212, IKE, Universität Stuttgart, 1997.

[74] Rambough, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Objektorientiertes Modellieren und Entwerfen. Prentice Hall, 1993.

[75] Bauer, M.; Haller, R.; Sucic, D.: OPTIMA, a software tool generating building models for simulation programs from CAD drawings. Beitrag zur Konferenz System Simulation In Buildings, Dezember 1998.

[76] VDI 2067 Blatt 2: Berechnung der Kosten von Wärmeversorgunganlagen – Raumheizung. VDI-Verlag, Dezember 1986.

[77] VDI 2067 Blatt 20: Wirtschaftlichkeit gebäudetechnischer Anlagen – Energieaufwand der Nutzenübergabe bei Warmwasserheizungen. VDI-Verlag, August 2000.

[78] TRNSYS 14.2: A Transient System Simulation Program. Handbuch, Solar Energy Laboratory, Universität Wisconsin, 1996.

[79] Bach, H.; Claus, G.: Ermittlung des Nutzungsgrades von Heizungsanlagen. BMFT-FB-T-81-116, 1981.

[80] ISO (1992) Integrated Generic Resource, Part 42: Geometric and Topological Representation. Committee Draft, 10303-42, 1992.

[81] DIN 4703-3: Raumheizkörper; Begriffe, Grenzabmaße, Umrechnungen, Einbauhinweise. Deutsches Institut für Normung e.V., September 1988.

[82] EN 442-2: Radiatoren und Konvektoren – Teil 2: Prüfverfahren und Leistungsangabe. Deutsche Fassung, Juli 1996.

[83] Kothe, D.: Regeln für den Datenaustausch zwischen CAD-Systemen zur Planung von WW-Heizanlagen und Simulationsprogramm TRNSYS. Diplomarbeit, IKE 7-D-395, IKE, Universität Stuttgart, 1999.

[84] X-Domain, Cross Domain Model Extensions (XM-1) External Libraries. Process Definitions, IFC Domain Project Documentation (Draft 1), International Alliance for Interoperability, 1998.

[85] VDI 6030: Auslegung von freien Raumheizflächen – Grundlagen und Auslegung von Raumheizkörpern. VDI-Verlag, Entwurf, April 1999.

[86] DIN 4701: Regeln für die Berechnung des Wärmebedarfs von Gebäuden. Beuth-Verlag, 1947-1983.

[87] X-Domain, Cross Domain Model Extensions (XM-3) Core Model Extensions. Process Definitions, IFC Domain Project Documentation (Draft 1), International Alliance for Interoperability, 1998.

[88] Building Services (BS-8) IFC HVAC Extension Schemata. (http://eetd.lbl.gov/btd/iai/bs8), 2000.

[89] Coplien, J.: Advanced C++ – Programming styles and idioms. Addison-Wesley, 1992.

[90] W3C – The World Wide Web Consortium: Document Object Model (DOM). http://www.w3.org/DOM.

[91] Megginson Technologies: SAX: The Simple API vor XML. Version 2, http://www.megginson.com/SAX/index.html.

[92] Verordnung über einen energiesparenden Wärmeschutz bei Gebäuden (Wärmeschutzverordnung -WärmeschutzV-). Bundesgesetzblatt vom 24.8.1994.

[93] DIN EN 12831: Heizsysteme in Gebäuden – Verfahren zur Berechnung der Norm-Heizlast. Arbeitspapier, Deutsche Fassung, 2000.

[94] VDI 2078: Berechnung der Kühllast klimatisierter Räume (VDI-Kühllastregeln). Verein Deutscher Ingenieure, Juli 1996.

[95] ISO EN 832: Wärmetechnisches Verhalten von Gebäuden – Berechnung des Heizenergiebedarfs – Wohngebäude. Entwurf, Dezember 1995.

[96] VDI 2067 Blatt 11: Wirtschaftlichkeit gebäudetechnischer Anlagen – Rechenverfahren zum Energiebedarf beheizter und klimatisierter Gebäuden. Entwurf, VDI-Verlag, Juni 1998.

[97] Gamma, E.; Helm, H.; Johanson, R.; Vlissides, J.: Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software. Reading MA, Addison-Wesley, 1996.

[98] VDI 2067 Blatt 10: Wirtschaftlichkeit gebäudetechnischer Anlagen – Energiebedarf beheizter und klimatisierter Gebäuden. Entwurf, VDI-Verlag, Juni 1998.

[99] OLE DB Software Development Kit. http://www.microsoft.com/data.

[100] VBScript Development Kit. http://www.microsoft.com/vbscript/.

[101] Microsoft Internet Information Server – Die technische Referenz. Microsoft Press, 1998.

[102] Die Windows-Oberfläche – Leitfaden zur Softwaregestaltung. Microsoft Press, 1995.

[103] Schmidt, F.: OPTIMA – Neue Strategien zur Minimierung des Energieverbrauches in Gebäuden. Abschlussbericht, IKE 4-142, IKE, Universität Stuttgart, 1995.

[104] Schwede, D.: Integrale Planung der Gebäudetechnik für ein Bürogebäude abgestimmt auf Nutzungsanforderungen und gesamtenergetische Zielvorgaben. Diplomarbeit, IKE 7-D-396, IKE, Universität Stuttgart, 1999.

[105] Ibrahim, A.: Entwicklung eines Simulationmodells für das NWZ 2 der Universität Stuttgart. Diplomarbeit, IKE 7-D-405, IKE, Universität Stuttgart, 1999.

[106] VektorPlan3D Win. Gesellschaft für technische Softwareentwicklung und Vertrieb mbH, Dezember 1997.

[107] IAI: Marktreife Produkte nach IFC noch in diesem Jahr – Datenaustausch ohne Verluste angestrebt. Computerwoche, 13/98, 1998.

[108] Hirschberg, R.: Entwicklung eines Programmes zur Berechnung des thermischen Verhaltens von Wohngebäuden basierend auf EN 832. Diplomarbeit, IKE 4-D-222, IKE, Universität Stuttgart, 1999.

[109] Ehm, H.; Schettler-Köhler, H.: Von der Wärmeschutzverordnung zur Energiesparverordnung – Eckpunkte zur Fortschreibung der energiesparrechtlichen Vorschriften. Bundesbaublatt, Heft 11/97, 1997.

[110] VDI 2067 Blatt 1: Wirtschaftlichkeit gebäudetechnischer Anlagen – Grundlagen und Kostenberechnung. VDI-Verlag, September 2000.

[111] VDI 3808: Energiewirtschaftliche Beurteilungskriterien für heiztechnische Anlagen. VDI-Verlag, Januar 1993.

[112] DIN 4108 Teil 4: Wärmeschutz im Hochbau – Wärme und feuchtschutztechnische Kennwerte. Beuth-Verlag, November 1991.

[113] Bach, H.; Bauer, M.; Dipper, J.; Reichert, E.: Erarbeiten wissenschaftlicher Grundlagen für ein Richtlinienpaket zur energetischen Bewertung von Anlagen der Technischen Gebäudeausrüstung. Abschlussbericht, IKE 7-29, IKE, Universität Stuttgart, März 2000.

[114] Schmidt, F. et al.: REUSE – Rational Use Of Energy at the University of Stuttgart Building Environment. Technischer Abschlussbericht, IKE 4-151, IKE, Universität Stuttgart, März 2000.

[115] EXIST News: Die Ideenwetbewerb BuisinessChance 2000. Bundesministerium für Bildung und Forschung (BMBF), exist 05/ dez.00-feb.01, Dezember 2000.

Anhang

A Entwurfsmuster

Der Anhang gibt eine Übersicht über die in dieser Arbeit verwendeten Entwurfsmuster.

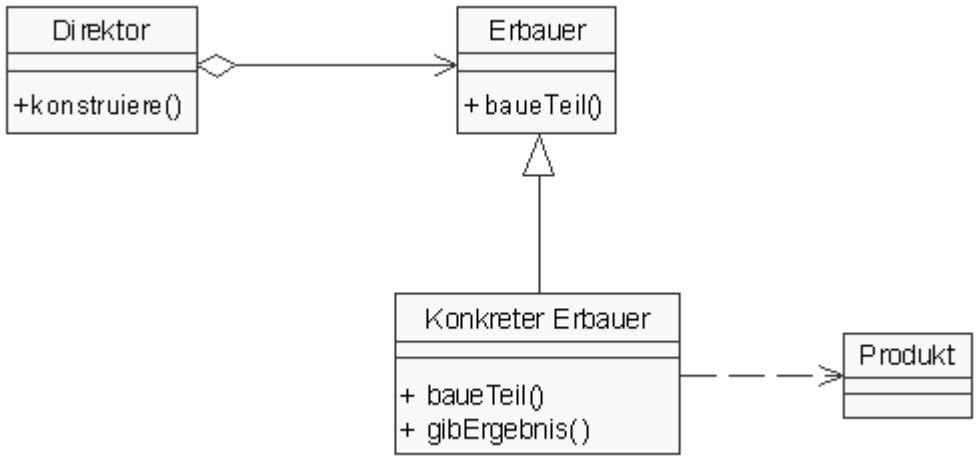
Erbauer

Ein objektbasiertes Entwurfsmuster.

Zweck

Trenne die Konstruktion eines komplexen Objektes von seiner Repräsentation, so dass derselbe Konstruktionsprozess unterschiedliche Repräsentationen erzeugen kann.

Struktur



Teilnehmer

- Erbauer – spezifiziert eine abstrakte Schnittstelle zum Erzeugen von Teilen eines Produktobjektes.
- Konkreter Erbauer – konfiguriert und fügt Teile des Produktes zusammen, indem es die Erbauerschnittstelle implementiert. Er definiert und verwaltet das von ihm erzeugte Repräsentation und bietet eine Schnittstelle zum Zurückgeben des Objektes.
- Direktor – konstruiert ein Objekt unter Verwendung der Erbauerschnittstelle.
- Produkt – repräsentiert das gerade konstruierte komplexe Objekt. Ein konkreter Erbauer erstellt die interne Repräsentation des Produktes und definiert den Prozess, durch den es zusammengesetzt wird. Das Produkt schließt die Klassen ein, welche die konstruierenden Teilen definieren. Dies umfasst die Schnittstellen, mit denen die Teile zum eindeutigen Resultat zusammengefügt werden.

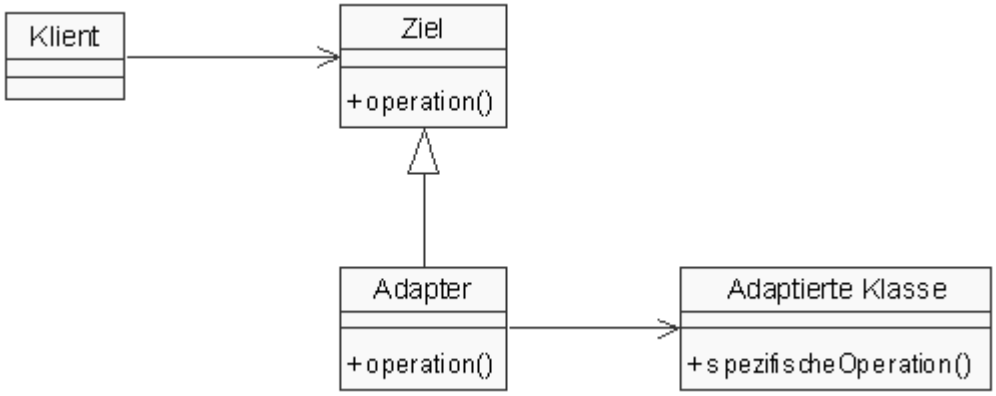
Adapter

Ein klassen- oder objektbasiertes Strukturmuster.

Zweck

Passe die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstelle an. Das Adaptermuster lässt Klassen zusammenarbeiten, die wegen inkompatibler Schnittstellen ansonsten dazu nicht in der Lage wären.

Struktur



Teilnehmer

- Ziel – definiert die anwendungsspezifische vom Klienten verwendete Schnittstelle.
- Klient – arbeitet mit Objekten, die der Zielschnittstelle entsprechen.
- Adaptierte Klasse – definiert eine existierende und anzupassende Schnittstelle.
- Adapter – passt die Schnittstelle der anzupassenden Klasse an die Zielschnittstelle an.

Institut für Kernenergetik und
Energiesysteme

Universität Stuttgart

Pfaffenwaldring 31

D-70550 Stuttgart