Ein Schnittstelle für dynamische Objektstrukturen für Entwurfsanwendungen

Peter Kolbe, Dirk Ranglack, Frank Steinmann

Hintergrund

In den einzelnen Phasen der computergestützten Bauwerksplanung werden unterschiedliche Progamme eingesetzt. Ein reibungsloser Datenaustausch wäre wünschenswert, ist aber heute noch Illusion. Die Standardisierung eines Bauwerksmodells blieb bislang erfolglos. Deshalb wurden in einigen Projekten alternative Wege beschritten, um allgemeingültige Lösungen im Bereich der Gebäudemodellierung anzubieten. Prinzipiell lassen sich dabei folgende Gemeinsamkeiten festellen:

- Die Menge der Objekttypen eines Anwendungsgebietes kann nicht immer vom Programmierer festgelegt werden. Deshalb muß die Software in der Lage sein, diese Strukturen durch den Programmanwender beschreiben zu lassen.
- Der Zugriff auf Objekte, die erst durch den Anwender definiert wurden, erfolgt über eine Menge von Methoden. Diese Methoden werden Laufzeitsystem oder Objektsystem genannt. Sie können im Bereich der objektorientierten Programmierung vereinheitlicht werden. Zu einem Laufzeitsystem gehören Methoden zum Definieren von Objekttypen, zur Analyse der Typstruktur sowie zum Erzeugen und Bearbeiten von Objekten.
- Programme, die mit Hilfe des Laufzeitsystems anwenderdefinierte Objektstrukturen bearbeiten, benötigen eine stets wiederkehrende Funktionalität für ihre Oberfläche (Nutzerinterface). Dazu gehören: Objekte eines bestimmten Typs erzeugen, Eigenschaften eines Objekts bearbeiten, Objekte löschen, Objekteigenschaften auswerten und Objekte grafisch oder alphanumerisch darstellen.

Datenbestände eines bestimmten Formats können durch eine Anwendung, die auf einem Laufzeitsystem basiert, erstellt werden. Um mit einer zweiten Anwendung auf diese Daten zugreifen zu können, kann das Laufzeitsystem der ersten Anwendung verwendet werden. Demzufolge würde die Austauschbarkeit der Laufzeitsysteme verschiedener Anwendungen den Zugriff auf fremde Datenbestände erlauben - ein Datenaustausch wäre somit prinzipiell möglich.

Angespornt durch diese Vision begannen die Entwickler von drei Projekten eine Schnittstelle für solche Laufzeitsysteme zu definieren. Um die Tragweite einer solchen Schnittstelle zu zeigen, sollen die drei Projekte kurz vorgestellt werden. Das Forum für die Definition der Schnittstelle ist der „Arbeitskreis Objekte“ (Kurzbezeichnung AKO). Weitere Informationen befinden sich auf den WWW-Seiten des Arbeitskreises [WWW97].

FlexOB [WEST97]: Im Rahmen der Entwicklung von Werkzeugen zur Unterstützung der frühen Phasen des architektonischen Entwurfs wurde das System FlexOB (Flexibles Objektsystem) geschaffen. Hierbei werden objektorientierte Modellierungstechniken aus dem Bereich der Wissensverarbeitung eingesetzt, um Entwurfswerkzeuge zu realisieren. Dem Anwender stehen zwei Arten von Editoren zur Verfügung. Einerseits kann abstraktes Entwurfswissen des Architekten in Form von Typhierarchien und deren Attributierung abgelegt werden. Andererseits können konkrete Bauwerksentwürfe durch Instanziierung der zuvor definierten Typen erstellt werden. Ein konkretes Anwendungsprogramm ist FunPlan [HÜB95]. Als methodisches Werkzeug dient es zur Erstellung von Funktionsplänen beim funktionalen Entwurf eines Bauwerks.

Das Fatima-Projekt [KOPF97]: In diesem Projekt wurde ein Integrationskonzept für Datenbestände gesucht, deren Schemata beliebig definiert sein können. Es wurden Integrationswerkzeuge geschaffen, die die Analyse und den Verbund von unterschiedlich struktu-

Dr. Dirk Ranglack, Dipl.-Inform. Peter Kolbe, HOCHTIEF Software GmbH, Bereich Objektbewirtschaftung
Dipl.-Ing. Frank Steinmann, Bauhausuniversität Weimar, Bereich Wissens- und Informationsverarbeitung

rierten Modellen ermöglichen. Das Ergebnis des Verbunds ist ein logischer Gesamtdatenbestand, der Auswertungen, wie sie im Bereich des Facility Management typisch sind, erlaubt. Zur Umsetzung wurden Beschreibungs- und Implementierungsmethoden der STEP-Norm verwendet. Das Anwendungsprogramm Grobjekt dient als Informationssystem für Aufgaben der Flächenbewirtschaftung.

Kopernikus [RANG96]: In diesem Projekt wird die Datenbanktechnologie für Aufgaben des Facility Management genutzt. Durch die konzeptuelle Sprache EXPRESS sind kundenspezifische Datenmodelle beschreibbar. Die meist umfangreichen Datenbestände werden durch passende Datenbank-Management-Systeme verwaltet. Die Programmoberfläche von Kopernikus stellt eine Reihe von Darstellungsfenstern zur Verfügung, um Objekte in Formular-, Tabellen-, Pixelgrafik-, Vektorgrafik- oder in Hierarchie-Sicht darzustellen und auszuwerten. Datenmodellierung, Datenbanksystem und Programmoberfläche sind konsequent getrennt. Dadurch können unterschiedliche Datenbanksysteme eingesetzt werden, die Programmoberfläche ist unabhängig vom Datenbanksystem und sie paßt sich an die verwendeten Datenstrukturen an.

Alle drei Projekte nutzen die Objekttechnologie in computergestützten Planungs- und Entwurfssystemen als Paradigma für Modellierung, Speicherung und Modellaustausch. Der einheitliche Zugriff auf Objektstrukturen durch ein Laufzeitsystem wäre ein wesentlicher Schritt in Richtung durchgängiger Computerunterstützung im Bauwesen.

Um Funktionalität austauschen zu können, wurden Programmoberfläche und Laufzeitsystem entkoppelt. Dies geschieht durch Einführung einer Schnittstelle für den Objektzugriff. Diese Schnittstelle deklariert eine Menge abstrakter Datentypen. Ein spezielles Laufzeitsystem entsteht durch Implementierung der abstrakten Datentypen. Benutzt ein Anwendungsprogrammierer die Schnittstelle der abstrakten Datentypen, so entsteht ein Anwendungsprogramm. Wiederverwendung ist auf beiden Seiten der Schnittstelle möglich. Bei Einhaltung der Schnittstellenvereinbarungen kann das Laufzeitsystem oder die Anwendung ausgetauscht werden.

Was ist ein abstrakter Datentyp?

Um die Verbindung zwischen den Anwendern einer Klasse und ihren Implementatoren zu lockern, kann eine abstrakte Basisklasse eingeführt werden. Sie stellt eine Schnittstelle zu einem Konzept bereit, wofür eine Menge von Implementationen realisiert werden können. Die Vorteile der Abstraktion liegen auf der Hand. Die Implementationsabhängigkeit sinkt, wodurch sich die Wartungs- und Änderungsfreundlichkeit erhöht. Zu einem abstrakten Datentyp können eine Vielfalt von Implementationen angeboten werden, die sich auf verschiedene Nebenbedingungen wie Effizienz, Sicherheit oder Entwicklungszeit konzentrieren - ohne dabei die den abstrakten Datentyp benutzende Anwendung ändern zu müssen.

Die Deklaration des abstrakten Datentyps verzichtet auf Datenelemente und auf implementierte Methoden. Lediglich die Methodenköpfe werden festgelegt. Wären Operationen und Datenelemente vorhanden, würde die Implementation eingeschränkt werden. Die Verwendung eines abstrakten Datentyps bedeutet jedoch das Hinausschieben dieser Festlegung. Der abstrakte Datentyp führt zu einer sauberen Trennung zwischen Schnittstelle und Implementierung.

Oft ist es zweckmäßig, die Implementation in einer dynamisch zu linkenden Bibliothek (*dynamic link library* bzw. *shared library*) anzubieten. Dadurch besteht für den Programmanwender die Möglichkeit, durch den Erwerb einer neuen Bibliothek die Funktionalität eines Programms zu ändern, ohne das Programm neu binden zu müssen.

Das Konzept des abstrakten Datentyps ist die grundlegende Softwaretechnologie für die Schnittstelle des Laufzeitsystems. Der Mechanismus der Vererbung kann bei konkreten Typen verwendet werden, um die Schnittstelle zu erweitern. Im Gegensatz dazu ist die Ableitung (Vererbung) von abstrakten Datentypen Grundvoraussetzung, um eine Implementierung zu realisieren.

Die AKO-Schnittstelle

Die Schnittstelle besteht aus einer Menge abstrakter Klassen und einigen wenigen, bereits implementierten Klassen (Bild 1). Für die Bereitstellung eines Laufzeitsystems müssen Ableitungen der abstrakten Klassen gebildet und implementiert werden. Die AKO-Schnittstelle teilt sich in drei Arten von Klassen:

- Schemabeschreibung
- Modellbearbeitung
- Datentypen

Die Klassen zur Schemabeschreibung enthalten die Funktionalität zur Erzeugung von Anwendungsklassen

sowie deren Slots (Verallgemeinerung von Objekteigenschaften und Objektbeziehungen) und zur Analyse der dadurch erzeugten Typstruktur. Die Klassen zur Modellbearbeitung erlauben die Manipulation von Objekten einer Datenbank. Die Datentypen beinhalten die Basistypen des Laufzeitsystems.

Schemabeschreibung

Zur Schemabeschreibung gehören die drei Klassen `MetaSchema`, `MetaClass` und `MetaSlot`. Die Funktion `CreateMetaSchema()` erzeugt ein `MetaSchema`-Objekt, wodurch ein Schema repräsentiert wird. Dieses `MetaSchema`-Objekt kann beliebig viele Objekte der Klasse `MetaClass` enthalten. Jedes dieser `MetaClass`-Objekte enthält wiederum beliebig viele `MetaSlot`-Objekte. Ein `MetaClass`-Objekt beschreibt eine Anwenderklasse. Zwischen den Anwenderklassen können Vererbungsbeziehungen bestehen. Mehrfachvererbung kann von einer Implementation unterstützt werden.

Ein `MetaSlot` beschreibt ein Attribut eines Anwenderklasse. Das Konzept des Slots (vergleiche Facetten-Konzept der Frames aus dem Bereich Wissensverarbeitung [PUPPE91]) reicht über eine einfache Attributdefinition hinaus. Der Slot ist durch seinen Namen definiert und kann beliebig viele Facetten enthalten. Die Facette "default-value" ist stets vorhanden. In der Schemadefinition können weitere Facetten für einen Slot definiert werden. Beispiele hierfür sind Grenzwerte, Maßeinheiten, Kommentare oder Texte zur Eingabeaufforderung.

Modellbeschreibung

Zur Modellbeschreibung gehören die Klassen `Database`, `Transaction`, `ImpInstance`, `Slot` und `Instance`. Die Funktion `CreateDatabase()` konstruiert ein `Database`-Objekt.

Die Klasse `Database` kapselt den Zugriff auf Datenbanken oder Dateisysteme, die im folgenden gemeinsam als Datenserver bezeichnet werden. Ein `Database`-Objekt dient als Verbindung von Anwendungsprogramm und Datenserver.

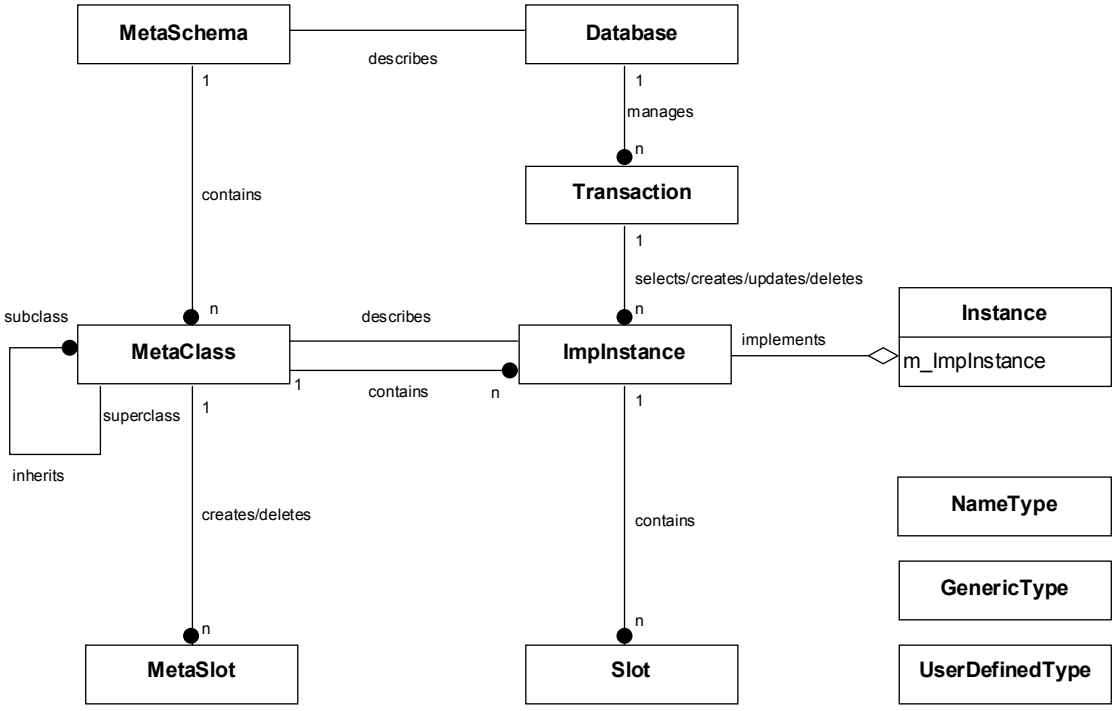

Bild 1: Klassendiagramm der AKO-Schnittstelle

Von einer Datenbank können vier grundlegende Operationen ausgeführt werden:

1. Laden von Instanzen aus der Datenbank.
2. Löschen von Instanzen in der Datenbank.
3. Update von Instanzen in der Datenbank.
4. Einfügen neuer Objekte in die Datenbank.

Die Verbindung zwischen Datenserver und Instanz übernimmt ein `Transaction`-Objekt. Eine Transaktion kann zu einem bestimmten Zeitpunkt immer nur einem Zweck dienen. Die beabsichtigte Operation wird durch eine der Methoden `From()`, `Delete()`, `Update()` oder `Insert()` der Klasse `Transaction` ausgewählt.

Objekte der Klasse `ImpInstance` repräsentieren eine Ausprägung eines Datenserver-Objekts im Hauptspeicher. Über die Metaklasse können ihre Attribute ermittelt werden, insofern sie der Anwendung nicht bekannt sind. Somit kann in der Anwendung mit den gleichen Klassen- und Attributbezeichnungen gearbeitet werden, wie sie im Datenserver gespeichert sind.

Die Klasse `Slot` steht im Verhältnis zur Klasse `MetaSlot`, wie `ImpInstance` zu `MetaClass`. Ein Slot ist die Ausprägung eines Attributs, wobei das Attribut neben dem Wert weitere Facetten besitzen kann. Im `MetaSlot` sind die Informationen über die Struktur des Slots abgelegt. Die Struktur dieses Slots ist dann für alle Instanzen einer Anwenderklasse gleich.

Die Klasse `Instance` nimmt eine Sonderstellung im Laufzeitsystem ein. Der Anwendungsprogrammierer kann hiervon seine Anwendungsklassen ableiten und erbt somit deren Methodenvorrat, beispielsweise zur persistenten Speicherung von Anwenderobjekten. Die Möglichkeit der Ableitung kann auch dem Anwender zur Laufzeit zur Verfügung gestellt werden, beispielsweise durch einen Schema-Editor. Die Klasse `Instance` bildet somit die Verbindung zwischen dem Laufzeitsystem und der Hierarchie der Anwenderklassen des jeweiligen Anwendungsgebietes. Die Klasse `Instance` ist nicht abstrakt sondern sie bezieht ihren Methodenvorrat vollständig von der Klasse `ImpInstance`.

Datentypen

Zur Schnittstelle gehören die drei Datentypen `NameType`, `GenericType` und `UserDefinedType`. Der `NameType` ist der Bezeichner für alle zu vergebenden Namen innerhalb der Schemadefinition. Er ist bereits implementiert. Der ebenfalls implementierte `GenericType` wurde zur einfachen Handhabung für alle vordefinierten Datentypen eingeführt. Er kann jeden vordefinierten Typ annehmen. Insbesondere generische Anwendungsprogramme, die vor dem Attributzugriff den Attributtyp erfragen müssen, profitieren hiervon. Die Menge der vordefinierten Datentypen ist in Tabelle 1 zusammengestellt.

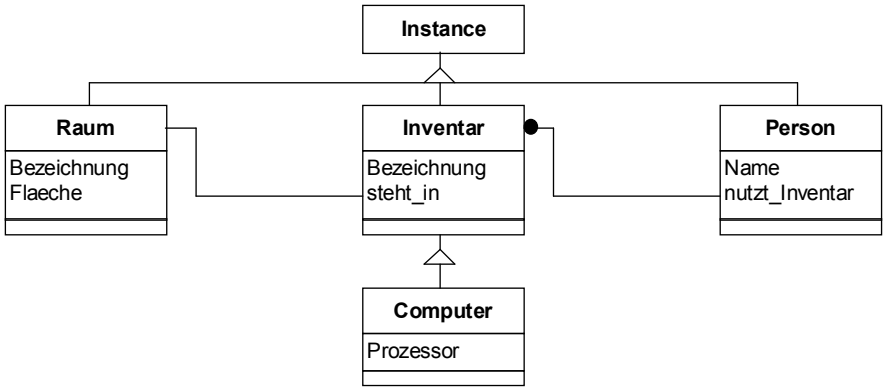

Bild 2: Einfache Klassenhierarchie für eine Facility-Management-Anwendung

Konzeptueller Typ	C++ Datentyp	Bedeutung
`BOOLEAN_TYPE`	`bool`	Wahrheitswert
`SHORT_TYPE`	`short`	Ganzzahl
`LONG_TYPE`	`long`	
`FLOAT_TYPE`	`float`	
`DOUBLE_TYPE`	`double`	

STRING_TYPE	char*	Zeichenkette
INSTANCE_TYPE	Instance*	Objekt
BOOLEAN_AGGR	bool*	Liste von Wahrheitswerten
SHORT_AGGR	short*	Liste von Ganzzahlen
LONG_AGGR	long*	Liste von Ganzzahlen
FLOAT_AGGR	float*	Liste von Fließkommazahlen
DOUBLE_AGGR	double*	Liste von Fließkommazahlen
STRING_AGGR	char**	Liste von Zeichenketten
INSTANCE_AGGR	Instance**	Liste von Objekten
USER_DEFINED	UserDefinedType	anwendungsspezifischer Datentyp
UNKNOWN_TYPE	-	-

Tabelle 1: Vordefinierte Datentypen

```
#include "ako.h"

void create()
{
  MetaSchema* pSchema = CreateMetaSchema( "simple_fm" );
  MetaClass* pRoom = pSchema->CreateMetaClass( "Raum" );
  pRoom->CreateMetaSlot( "Bezeichnung", STRING_TYPE   );
  pRoom->CreateMetaSlot( "Flaeche", DOUBLE_TYPE );
  MetaClass* pInventory = pSchema->CreateMetaClass( "Inventar" );
  pInventory->CreateMetaSlot( "Bezeichnung", STRING_TYPE );
  pInventory->CreateMetaSlot( "steht_in", INSTANCE_TYPE );
  MetaClass* pComputer  = pSchema->CreateMetaClass( "Computer" );
  pComputer->SetSuperClass( "Inventar" );
  pComputer->CreateMetaSlot( "Prozessor", STRING_TYPE );
  MetaClass* pPerson = pSchema->CreateMetaClass( "Person" );
  pPerson->CreateMetaSlot( "Name", STRING_TYPE );
  pPerson->CreateMetaSlot( "nutzt_Inventar", INSTANCE_AGGR );
  pSchema->Write( "simple_fm" );
}
```

Programmbeispiel 1: Erzeugung von Schema-Informationen

```
void analyse()
{
  MetaSchema* pSchema = CreateMetaSchema( "simple_fm" );
  pSchema->Read( "simple_fm" );
  // iterate all classes
  CountType iClsCnt = pSchema->GetMetaClassCardinality();
  MetaClass** ppClasses = new MetaClass*[ iClsCnt ];
  pSchema->GetMetaClasses( ppClasses, iClsCnt );
  for( CountType iCls = 0; iCls < iClsCnt; iCls++ )
  {
    MetaClass* pCls = ppClasses[iCls];
    cout << "\nKlasse: " << pCls->GetName();
    // iterate all slots
    CountType iSltCnt = pCls->GetMetaSlotCardinality();
    MetaSlot** ppSlots = new MetaSlot*[ iSltCnt ];
    pCls->GetMetaSlots( ppSlots, iSltCnt );
    for( CountType iSlt = 0; iSlt < iSltCnt; iSlt++ )
    {
      MetaSlot* pSlot = ppSlots[iSlt];
      cout << "\n\tAttribut: " << pSlot->GetName()
           << " (" << pSlot->GetTypeName() << " )";
    }
    delete [] ppSlots;
  }
  delete [] ppClasses;
}
```

Programmbeispiel 2: Analyse der Schema-Informationen

Durch das Konzept der Facetten ist eine erweiterte Attributdefinition möglich. Dennoch können Attributtypen lediglich einen der vordefinierten Typen annehmen. Abhilfe hierfür schafft die Einführung der Klasse `UserDefinedType`. Durch Ableitung einer eigenen Klasse von `UserDefinedType` besteht für den Anwendungsprogrammierer die Möglichkeit, neue Datentypen dem Laufzeitsystem hinzuzufügen, wie beispielsweise einen speziellen Matrixtyp oder einen Typ für Datum. Dieser neue Datentyp muß wie das Laufzeitsystem in einer dynamisch zu linkenden Bibliothek implementiert werden. Durch Aufruf der Funktion `CreateUserDataType()` fordert der Anwendungsprogrammierer das Laufzeitsystem auf, diese Implementierung zu laden. Durch diese Vorgehensweise ist es möglich, ein bereits implementiertes Laufzeitsystem um anwendungsspezifische Datentypen zu erweitern.

Anwendung der Schnittstelle

Die Quelltextbeispiele in C++ demonstrieren die Anwendung des Laufzeitsystems. Bild 2 zeigt das in den Beispielen verwendete Schema aus dem Anwendungsgebiet Facility Management. Im ersten Beispiel wird das Schema zur Laufzeit des Programms definiert. Von der Klasse `Instance` werden die Anwendungsklassen abgeleitet, ihre Attribute beschrieben und in Beziehung gesetzt. Abschließend wird das Schema gespeichert, um es im folgenden zu verwenden. Das zweite Beispiel analysiert die Schemadefinition - eine Grundvoraussetzung für die Entwicklung schema-

```
void search()
{
  Database* pDB = ::CreateDatabase( "OCI" );
  pDB->Init();
  pDB->Login();
  Transaction* pTxn = pDB->CreateTransaction();
  pTxn->From( "Person" );
  pTxn->Where( "*" );
  GenericType xMeyer( "Meyer" ), xName, xInventory, xRoom, xBez;
  while( pTxn->MoreInstances() )
  {
    Instance* pPerson = new Instance( pTxn );
    pPerson->GetValue( "Name", xName );
    if( xName == xMeyer )
    {
      pPerson->GetValue( "nutzt_Inventar", xInventory );
      CountType nCount = xInventory.GetElementCount();
      Instance** ppInventory = new Instance*[ nCount ];
      ppInventory = xInventory.GetInstanceArray();
      for( CountType i=0; i<nCount; i++ )
      {
        Instance* pInventory = ppInventory[i];
        GenericType xRoom;
        pInventory->GetValue( "steht_in", xRoom );
        Instance* pRoom = xRoom.AsInstance();
        pRoom->GetValue( "Bezeichnung", xBez );
        cout << xBez.GetString() << "\n";
      }
      delete [] ppInventory;
    }
    delete pPerson;
  }
  pTxn->Destroy();
  pDB->Logout();
  pDB->Destroy();
}
```

Programmbeispiel 3: Zugriff auf Objekte

unabhängiger Programme. Im dritten Beispiel wird von einem bestehenden Datenbestand ausgegangen. Es werden die Objekte der Klasse "Raum" ausgegeben, in denen Inventar steht, welches von der Person namens "Meyer" benutzt wird.

Fazit

Ein Laufzeitsystem bietet den Zugriff auf Modelle und deren Meta-Informationen. Durch die AKO-Schnittstelle wird dieser Zugriff für verschiedene Systeme vereinheitlicht. Ziel ist die vollständige Entkopplung von Anwendungsprogramm und Laufzeitsystem, so daß beide ausgetauscht werden können. Durch die Verwendung des gleichen Laufzeitsystems in verschiedenen Programmen wird Datenaustausch möglich.

Diese Technologie zielt auf Programme ab, bei denen der Anwender direkte Kenntniss der Datenstruktur besitzt bzw. diese sogar festlegt. Dies sind vor allem Modellierungs- und Informationssysteme für frühe Entwurfsphasen oder die Bewirtschaftungsphase. Hier entstehen die Strukturen des realen Bauwerks und seiner Nutzung bzw. werden sie verändert. Deshalb sollten die Modellstrukturen auch innerhalb dieses Zeitraums und vom Anwender beschreibbar sein.

Laufzeitsysteme, die der AKO-Schnittstelle entsprechen, erlauben die dynamische Beschreibung von Modellen, die objektorientiert strukturiert sind. Das Slot/Facetten-Konzept stellt eine Erweiterung zum Modellieren mit Klassen und Attributen dar.

Wie die Implementierung der Schnittstelle umgesetzt wird, ist nicht festgelegt. Die Definition erlaubt jedoch den effektiven Einsatz von Datenbanken.

Literatur

[HÜB95]: Hübler, Kolbe, Steinmann: "Wissensbasierte Computerunterstützung der frühen Phasen des architektonischen Entwurfs, Teil I und II" in "Computer und Architektur - Computereinsatz in frühen Entwurfsphasen", Wissenschaftliche Zeitschrift der HAB Weimar, Heft 4/94, Weimar, 1994

[KOPF97]: Kolbe, Pfennigschmidt, Pahl: "Integration von Datenmodellen - Eine Technologie für Facility Management" in diesem Band

[PUPPE91]: Puppe: "Einführung in Expertensysteme", 2. Auflage, Springer-Verlag, Berlin/Heidelberg, 1991

[RANG96]: Ranglack: "Facility Management - der Weg von der CAD-Applikation zum Informationssystem" in "Der Facility Manager" 1. Quartal '96

[WEST97]: Wehner, Steinmann: "FlexOB - Entwicklungstools für dynamische, modellbasierte CAD-Systeme" in diesem Band

[WWW97]: http://www.uni- weimar.de/Bauing/biwv/ako.html