6. Implementierung

Die Implementierung stützt sich auf die Designphase. Dieses Kapitel beschreibt die eingesetzte Implementierungstechnologie, sowie die Umsetzung des Conceptual Designs, Navigational Designs und des Abstract Interface Designs. Weiterhin werden einige Implementierungskonzepte erläutert.

6.1 Einleitung

Die Implementierung eines Prototyps folgte dem ersten Coneceptual und Navigational Design. Er besaß keine Funktionalität aber stellte die erste Oberfläche sowie Navigationsmöglichkeiten dar und wurde parallel zu dem Abstract Interface Design erstellt. Dies gab dem Kunden die Möglichkeit, sich die Applikation vorzustellen und seine Arbeitsabläufe daran zu testen. Auch beim späteren Umsetzen diente diese abgenommene Vorlage als Grundstein für die Applikation.

Die Vorgehensweise während der Implementierung war nach Modulen geordnet:

- Projektvorschlag
- Projekte
- Personen

- Accounts
- Berechtigungen
- Erinnerungen
- Reporting

In dieser Reihenfolge wurden die Module implementiert. Von der Implementierungsdauer ist das Projektmodul das zeitaufwendigste, dicht gefolgt von dem Personen und Accountmodul. Diese zwei Module sind eng miteinander verknüpft, wohingegen die anderen Module mehr oder weniger eigenständig sind. Im Design wurde nach der Methode OOHDM vorgegangen. Drei Designphasen wurden durchlaufen: Conceptual Design, Navigational Design und Abstract Interface Design. Die Umsetzung der einzelnen Phasen ist nachfolgend beschrieben. Als Implementierungssprache wurde Englisch gewählt. Deswegen werden in nachfolgenden Diagrammen die englischen Übersetzungen der in der Analyse und im Design verwendeten Begriffe benutzt.

6.2 Implementierung des Conceptual Designs

Im Conceptual Design wurden die einzelnen Klassen und ihre Relationen zueinander definiert. In der Implementierung diente das Conceptual Design als Grundlage für das Datenbankdesign, welches hier beschrieben werden soll.

Die einzelnen Klassen des Conceptual Schemas wurden in Tabellen der Datenbank umgewandelt. Die Verknüpfungen zwischen den Klassen wurde in 1-n, 1-1, bzw. n-m Relationen einer relationalen Datenbank überführt. Dies impliziert die Einführung von Verknüpfungstabellen. Außerdem mußten die Attribute um Primärschlüssel und Fremdschlüssel erweitert werden. Das Enterprise Object Framework, die Middleware zwischen Applikation und Datenbank, unterstützt auch Vererbung. Es existieren mehrere Möglichkeiten, um Vererbung auf einer relationalen Datenbank abzubilden. Aus Leistungsgründen wurde die Eintabellenvererbung genommen. Hierbei werden alle Attribute aller Klassen einer Vererbungshierarchie in einer Tabelle gespeichert. Für eine gegebene Klasse werden die nichtgebrauchten Attribute mit Nullwerten gefüllt. Da die einzelnen

Klassen der Vererbungshierarchie nicht stark in den Attributen untereinander abweichen, kommt es trotzdem zu keinem großen Verlust an Datenbankplatz. Bei dieser Vorgehensweise ist für die Restaurierung eines jeglichen Objekts nur ein Zugriff auf die Datenbank notwendig.

Im Folgenden ist das Datenbankschema als Entity-Relationship Modell dargestellt. Die Diagramme stammen von dem Entwicklungswerkzeug Enterprise Object Modeller, der im Kapitel 6.5.3 kurz beschrieben ist.

Exemplarisch wird die verwendete Notation an Abbildung 6–1 erläutert. Eine Klasse besteht aus zwei Teilen: den Attributen und den Relationen. To-One Relationen sind mit einem Einfachpfeil markiert wohingegen To-Many Relationen mit einem Doppelpfeil markiert sind. In der Klasse sind alle Relationen aufgeführt. Für die Relationen zu Klassen die in dem aktuellen Diagram sichtbar sind, werden die Relationen auch graphisch durch Verbindungslinien dargestellt. Leider bietet der Diagrammodus des Enterprise Object Modellers keine Möglichkeit für die Darstellung der Vererbungshierarchien. Deshalb wird auf die Vererbung nur im Beschreibungstext eingegangen.

Alle Primärschlüsselattribute sind mit einem Schlüssel gekennzeichnet. Nicht alle aufgeführten Attribute sind auch Teil der Klasse. Nur die mit einer Raute markierten Attribute sind Klassenattribute. So werden z.B. Fremdschlüssel und Primärschlüssel nicht in die Klasse übernommen.

Abbildung 6–1 zeigt das Datenbankschema der Projektverwaltung. Es entspricht bis auf die hinzugekommene Verbindungsklasse *SalesPerCategory* dem Conceptual Design. Das Modul Notizen ist mit in dem Diagramm enthalten. Die Vererbungen, welche im Conceptual Design modelliert wurden, können hier nicht gezeigt werden. So erbt *Comment* von *Note*.

Abbildung 6–1 Datenbankschema Projektverwaltung

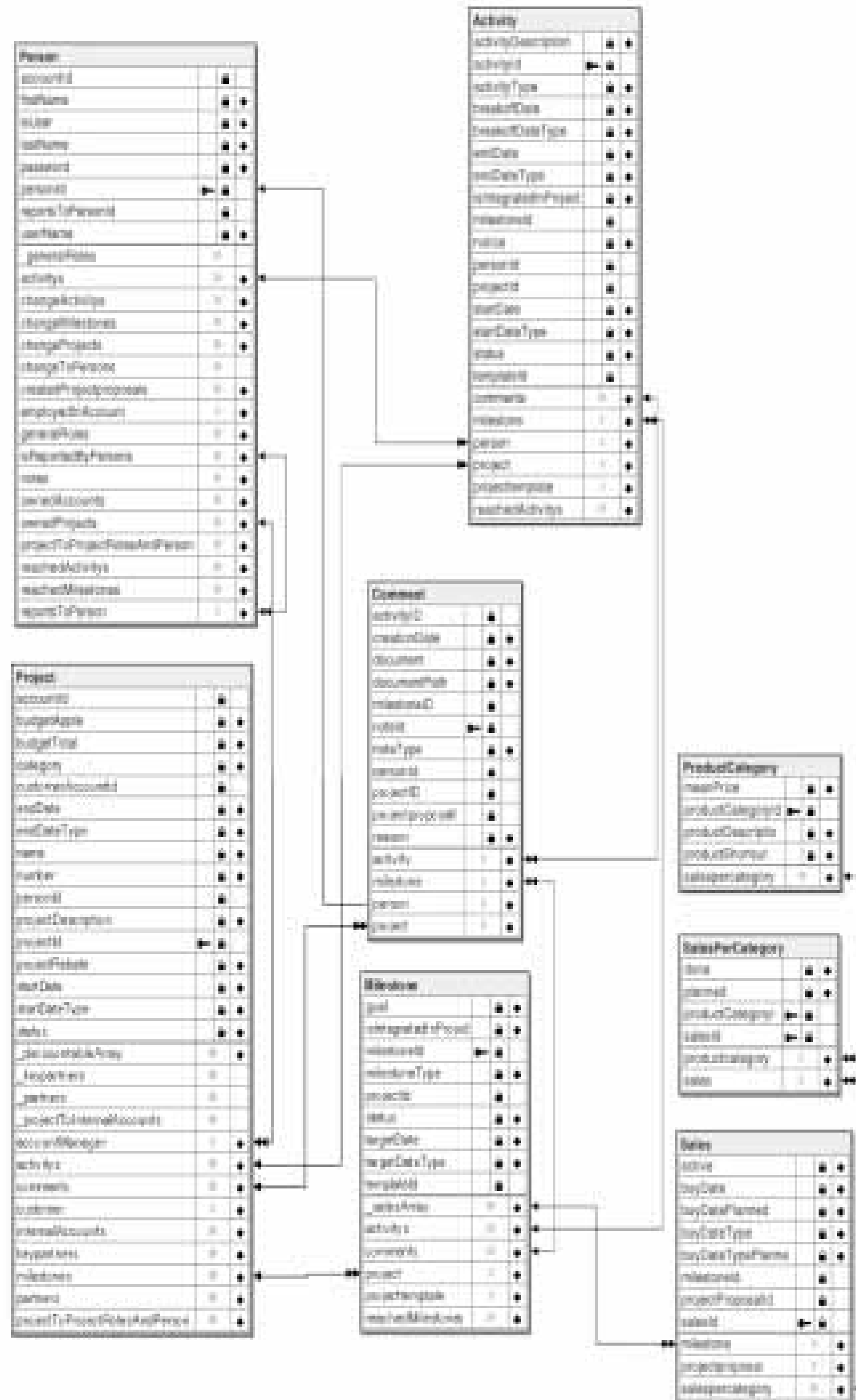


Abbildung 6–2 Datenbankschema für Accounts/Projekt

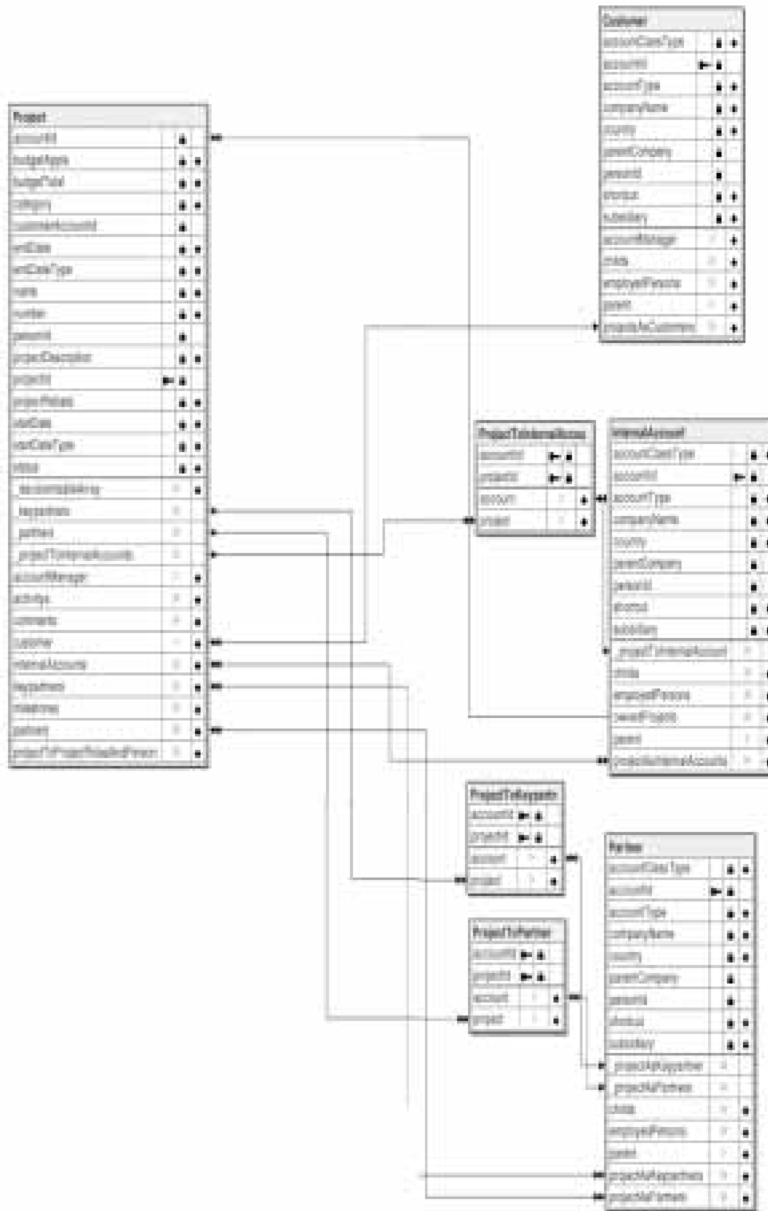


Abbildung 6–2 zeigt die Integration von Accounts und Projekten. Die drei Klassen *Kunde*, *interne Abteilung* und *Partner* sind alles Subklassen von *Account* (hier nicht gezeigt). Sie werden verbunden über jeweils eine Verbindungsklasse (*ProjektToInternalAccount*, *ProjectToKeypartner*, *ProjectToPartner*), wobei ein Partner in zwei verschiedenen Bezügen zum Projekt stehen kann, nämlich als Keypartner oder als Partner.

In Abbildung 6–3 ist das Datenbankschema für das Conceptual Schema der Personen angegeben. Auch hier sind wiederum Verbindungsklassen eingefügt worden (*ProjectToProjectRoleAndPerson*, *PersonToPersonRole*, *PrivilegeToCategoryAndRole*). Die Verbindungen, die als n-m Relationen mit zwei Doppelpfeilen markiert sind, werden eigentlich über die angezeigten Verbindungen über die Linkklassen realisiert. Z.B. hat die Klasse *Persons* keine direkte Verbindung zu der Klasse *PersonRole*. Diese wird über die Linkklasse *PersonToPersonRole* realisiert. Trotzdem ist in dem Diagram die Verbindung eingezeichnet. In diesem Datenbankschema befinden sich auch die zwei tertiären Relationen zwischen *Persons*, *Projekt* und *Projektrolle*, sowie zwischen *Projektrolle*, *PrivilegienCategory* und *Privilegien*, die wiederum über eine Linkklasse realisiert sind. Die Klassen *Rolle* (nicht eingezeichnet), *Projektrolle* und *Personenrolle* bilden eine Vererbungshierarchie mit der Klasse *Rolle* als Wurzel.

Abbildung 6–3 **Datenbankschema Personen und Berechtigungen**

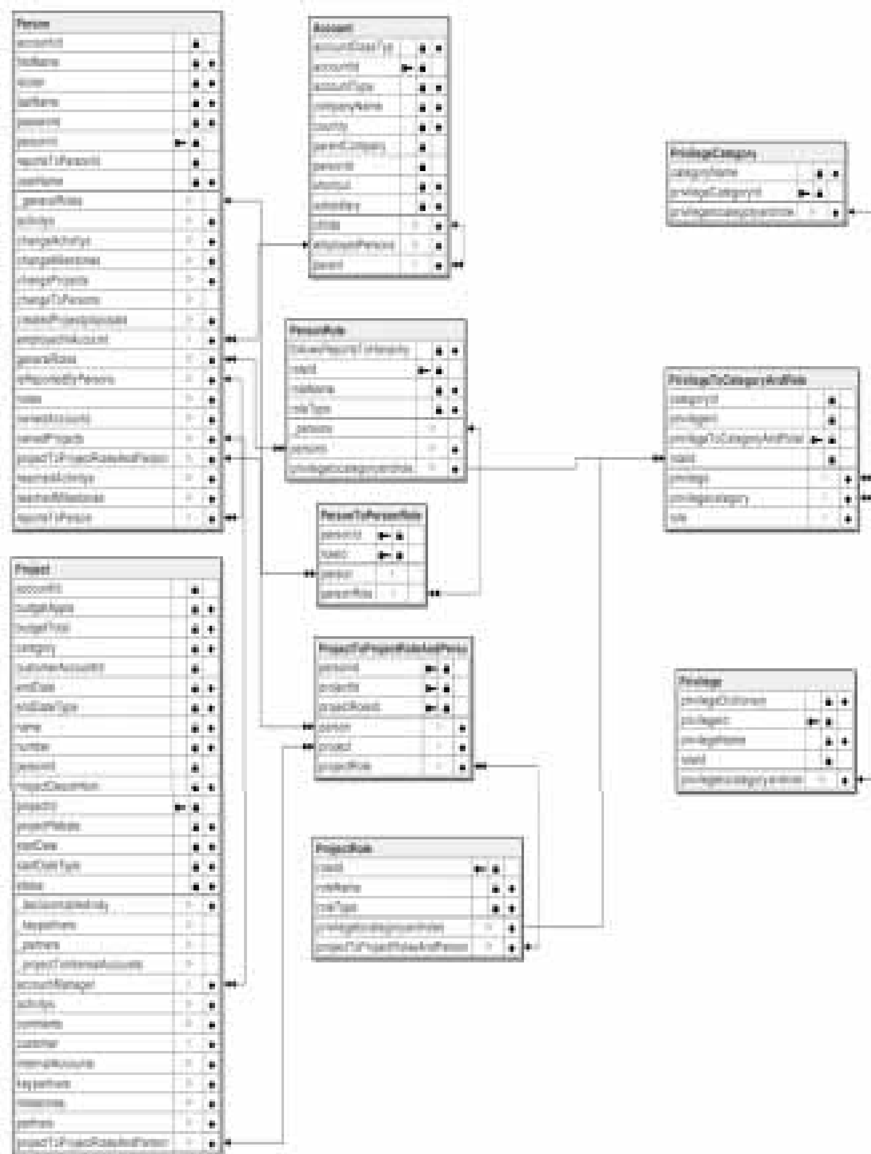
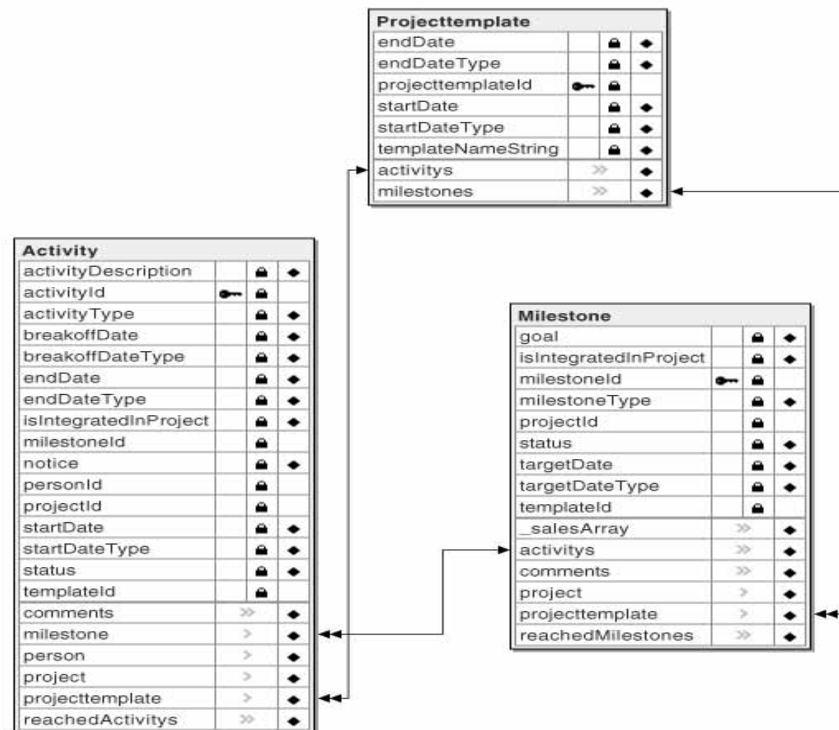


Abbildung 6–4 **Datenbankdesign Projekttemplate**



In Abbildung 6–4 ist das Datenbankdesign für die Projekttemplates gezeigt. Im Conceptual Schema ist eine Datumsangabe als ein Attribut modelliert. Jedoch existieren drei verschiedene Repräsentationen für ein Datum, je nach dem ob es sich um ein exaktes Datum, eine Monatsangabe bzw. eine Angabe der Kalenderwoche handelt. Dazu mußte für jedes Datum ein Datumstyp eingeführt werden, der angibt, ob es sich um ein exaktes Datum, eine Kalenderwoche oder einen Monat handelt.

Abbildung 6–5 **Datenbankmodell Projektvorschlag**

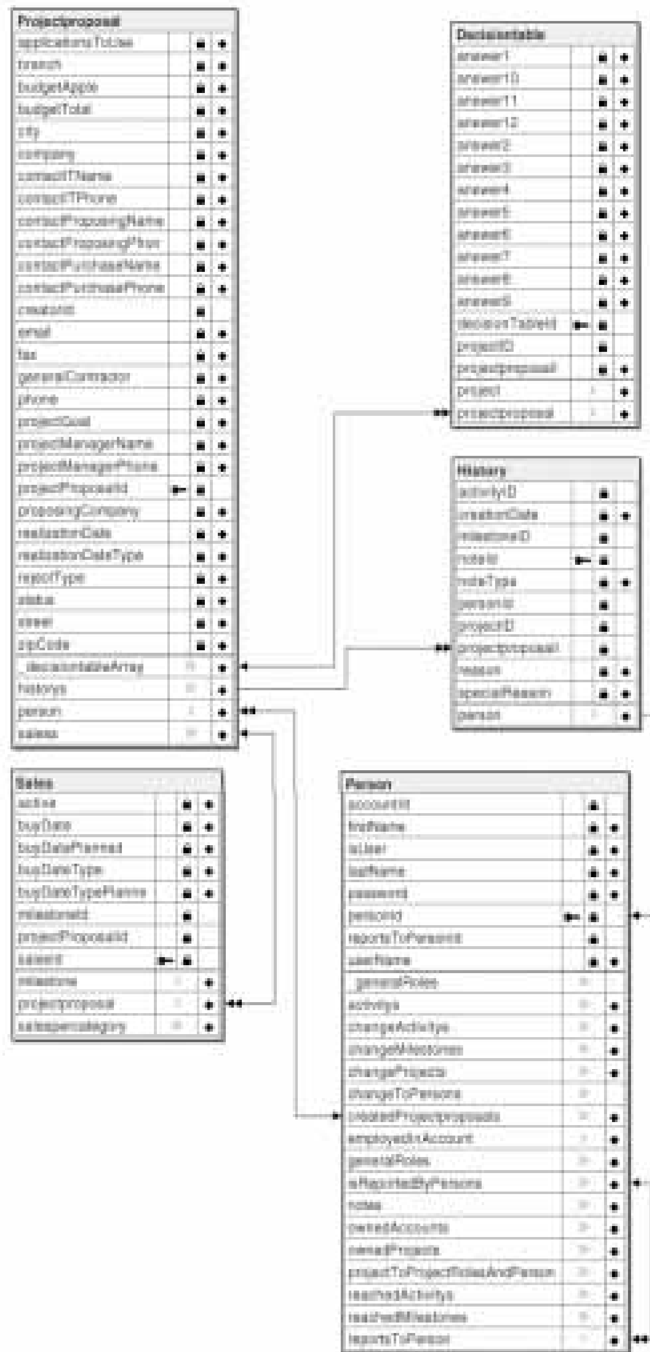


Abbildung 6–5 zeigt das Datenbankschema für den Projektvorschlag. Die Struktur der Verkäufe und der Entscheidungstabelle ist identisch zu der Projektverwaltung. Wenn ein *Projektvorschlag* in ein *Projekt* übergeht, werden die *Verkäufe* übernommen, einem *Milestone* zugeordnet und die *Entscheidungstabelle* mit dem *Projekt* verknüpft. *Verkäufe* in einem *Projektvorschlag* sind durch den Kontext natürlich Verkaufsannahmen, aber im Datenbankschema werden sie sowohl in der Projektverwaltung wie auch im *Projektvorschlag* durch eine Klasse *Verkäufe* repräsentiert.

6.3 Implementierung des Navigational Designs

In diesem Abschnitt wird gezeigt, wie das Navigational Design umgesetzt wurde.

Im Navigational Design wurden die Kontextklassen und ihre Beziehungen untereinander definiert. Wie in Abschnitt 4.3 beschrieben, gibt es sechs verschiedenen Kontextarten. In WebObjects ist eine Klasse vorhanden, welche die Funktionalität eines Kontexts sehr gut modelliert, die *WODisplayGroup*. Ihre Aufgabe ist es, Objekte einer Klasse anzuzeigen und zu verwalten. Hierzu gehören u.a. die Verwaltung einer Selektionsmenge, das Sortieren der angezeigten Objekte und die Auswahl der Objekte nach Suchkriterien. Außerdem besteht die Möglichkeit alle Objekte, die von einer *WODisplayGroup* verwaltet werden entweder nacheinander durchzunavigieren oder sie in disjunkte Gruppen von n Objekten aufzuteilen und dann durchzunavigieren. Deshalb wurde für die Implementierung die *WODisplayGroup* als Generalisierung eines jeglichen Kontexts ausgewählt. Nicht in allen Fällen kann jedoch eine *WODisplayGroup* alle Funktionalität eines Kontexts erfüllen. Im Folgenden soll aufgezeigt werden, wie die sechs Kontextarten zur Funktionalität einer *WODisplayGroup* passen und welche anderen Schritte notwendig sind, um einen Kontext in WebObjects abzubilden. Es sei hier erwähnt, daß die Klasse *NSMutableArray* auch über die Funktionalität Sortieren und Auswählen verfügt, jedoch im Gegensatz zur *WODisplayGroup* nicht direkt mit der Datenbank arbeitet.

- **Klassenbasierter Kontext**:
  Dieser Kontext enthält alle Objekte einer Klasse C, welche durch ein gemeinsames Attribut ausgewählt werden. Diese Eigenschaft erfüllt die *WODisplayGroup*. Die Fixierung eines Attributs kommt einer Einschränkung der Anzeigemenge gleich. Dafür muß eine einschränkende Bedingung (Qualifier) für die *WODisplayGroup* gesetzt werden, welche die Form: “attribut = wert” hat. Für Kontexte dieses Typs wird in der Implementierung der Name NC<Klassenname> verwendet.

- **Klassenbasierte Kontextgruppe:**
  Diese Gruppe von Kontexten wird durch die Enumerierung aller möglichen Werte eines gegebenen Attributs gebildet. Jeder Kontext der Gruppe zeichnet sich durch die Fixierung eines Wertes aus der Menge aller möglichen Werte für das gegebenen Attribut aus. Da ein Kontext aus der Kontextgruppe sich durch eine bestimmte einschränkende Bedingung (Qualifier) für das gegebene Attribut auszeichnet kann eine *WODisplayGroup* seine Kontextidentität während der Laufzeit ändern, indem die entsprechende Bedingung während der Laufzeit gesetzt wird. Demnach ist es möglich, die Kontextgruppe mit einer *WODisplayGroup* zu simulieren. Zusätzlich muß die Menge aller einschränkenden Bedingungen in der *WODisplayGroup* gespeichert werden. Diese Art von Kontext wurde im Programm nicht verwendet.

- **Link-basierter Kontext:**
  Diese Art des Kontexts wird von der *WODisplayGroup* defaultmäßig unterstützt. Anstatt die Anzeige von allen Objekten zu erlauben, kann auch spezifiziert werden, nur die Objekte für eine 1-n Relation X mit Masterobjekt Y zu erlauben. Hierfür wird das Masterobjekt und der Relationsname gesetzt. Kontexte dieses Typs wurden in der Implementierung als NC<Klassenname>Detail bezeichnet.

- **Link-basierte Kontextgruppe:**
  In WebObjects ist diese Art von Kontextgruppe mit Hilfe einer Master-Detail Konfiguration von zwei *WODisplayGroups* zu implementieren. Die erste *WODisplayGroup* enthält hierbei die Masterobjekte der 1-n Relation und die zweite *WODisplayGroup* enthält die Detailobjekte der Relation. Es ist jedoch nicht möglich mit dieser Konfiguration mehrere Masterobjekte auszuwählen und dann durch die Detailobjekte zu navigieren. Eine Möglichkeit dies zu implementieren, ist die erste *WODisplayGroup* mit einem Array für alle Masterobjekte zu erweitern und die Detailobjekte in der zweiten *WODisplayGroup* per Hand zu setzen. Da alle Masterobjekte einen Verweis auf ihre Detailobjekte haben, ist das kein größerer Imple-

mentierungsaufwand. Die letztere Möglichkeit wurde jedoch im Programm nicht gebraucht.

- **Enumerierte Kontextklasse:**
  Hierfür eignet sich die *WODisplayGroup* nicht, da sie nur Objekte einer Klasse Anzeigen kann. Diese Kontextklasse muß unter Verwendung von Standarddatentypen implementiert werden. Ein Denkansatz ist die Klasse *Array*, die auch Sortierungen ihrer Elemente zuläßt und einen Enumerator definiert.

- **Dynamische Kontextklasse:**
  WODisplayGroup ist per se eine dynamische Kontextklasse in allen oben beschriebenen Varianten. Sie bietet die Methoden *insert* und *delete* an. Da die vorliegende Applikation sehr oft Objekte aus Kontexten löscht oder einfügt, ist es essentiell, daß die Implementierung der Kontexte dynamisch ist.

In OOHDM werden linkbasierte Kontexte für eine spezielle Relation definiert. Da in WebObjects jegliche Relation eingesetzt werden kann für die das Zielobjekt von der Klasse ist, die der Kontext verwaltet, ist nur eine linkbasierte Kontextklasse pro Datenbankklasse notwendig. Da auch die *WODisplayGroup* als klassenbasierter Kontext für die Einschränkung auf ein bestimmtes Attribut nur einen neuen Qualifier benötigt, wurde auch nur ein klassenbasierter Kontext pro Datenbankklasse implementiert.

6.4 Implementierung des User Interfaces

WebObjects unterstützt die Aufteilung einer Seite in Komponenten, die wiederum aus Komponenten aufgebaut werden können. Das Abstract Interface Design schlägt sich somit in der Aufteilung des Interfaces in Unterkomponenten wieder. Ein Beispiel der Überführung des Designs in die Implementierung gibt Abschnitt 6.6.2 auf Seite 104.

Das Abstract Interface Design teilt die Menüs schon in ein Hauptmenü und mehrere Untermenüs auf (Abbildung 5–14 auf Seite 75). Der Bildchirm wurde also in drei Bereiche mittels Frames unterteilt. Dabei wurde das Hauptmenü nicht oben sondern unten angelegt und jeweils ein Untermenü wurde auf der

linke Seite placiert. Die Menüs sind graphisch hervorgehoben durch einen eigenen Hintergrund. Umrandet von den Menüs befindet sich die Hauptarbeitsfläche, in der das aktuelle Fenster angezeigt wird. Da der Benutzer nicht im Hyperspace verloren gehen soll wurde im Menü zusätzlich eine Anzeige der zuletzt angezeigten Seiten implementiert. Die Anordnung der Menüs und des Hauptarbeitsbereichs wurde nach einer Studie einiger Webseiten nachempfunden. Auf multimediale, bessere Darstellungen wurde verzichtet, da sie sehr zeitaufwendig in der Erstellung sind und später hinzugefügt werden können. Des weiteren schaden sie der Effizienz (v.a. Java).

Weiterhin sollte die Benutzung möglichst durchgängig sein. Deshalb wurden verschiedene Icons ausgesucht (Tabelle 7–1), die verschiedene Operationen repräsentieren und diese durchgängig in der Applikation verwendet (z.B. der Stift bedeutet überall “Editieren”). Ein weiterer, viel benutzert Punkt sind die Detailansichten. Sie wurden mittels des speziellen Icons “Pfeil” realisiert. Siehe hierzu auch Abschnitt 6.6.3.

Es wird an dieser Stelle darauf verzichtet, die Interfacemasken abzubilden, da sie im Benutzerhandbuch im Abschnitt 7 schon dargestellt sind.

6.5 Die Implementierungstechnologien

6.5.1 WebObjects - ein Überblick

Als Implementierungstechnologie für das Projekt wurde WebObjects in der Version 4.0 eingesetzt. Für den Datenbankzugriff wird das Enterprise Object Framework (EOF) in der Version 3.0 benutzt. EOF ist die Datenbankmiddleware, welche über einen Adaptor auf eine relationale Datenbank zugreift. Es existieren Adaptoren für mehrere Datenbanken.

Die Entwicklungsumgebung ist für MacOS X Server und für Windows NT verfügbar. Die WebObjects Applikation kann jedoch auf einer viel breiteren Betriebssystembasis eingesetzt werden, da sie nur die Laufzeitumgebung ohne Graphical User Interface benötigt.

Für eine Webapplikation wird ein Webserver benötigt. WebObjects beinhaltet keinen Webserver. Es wird über die Webserverschnittstelle (CGI, NSAPI, IISAPI, WAI) an den jeweilig vorhandenen Webserver angeschlossen.

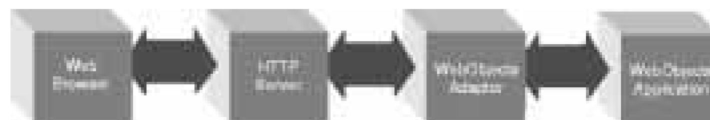
Weiterhin unterstützt WebObjects das Verteilen der Applikation auf mehrere Rechner und verwaltet die Rechnerlast (Load Balancing). Weiterhin kann mit dem Statistikmodul rudimentäre Analysen der Antwortzeit der Applikation erstellt werden.

Die unterstützten Programmiersprachen sind Java, Objective-C und WebScript. WebScript ist eine zur Laufzeit interpretierte Sprache mit Java-ähnlicher Syntax.

6.5.2 Struktur von WebObjects

WebObjects liegt zwischen Datenbank und Webserver. Es ist über die Webserverschnittstelle (z.B. CGI) an den Webserver angeschlossen und über EOF an die Datenbank. (siehe Abbildung 6–6)

Abbildung 6–6 **Position einer WebObjects Applikation**



Da das http Protokoll nicht statuserhaltend ist, muß WebObjects den Status sichern. Dies geschieht automatisch. Jedem Benutzer wird während seines Zugriffs auf die Applikation eine eindeutige Nummer zugeordnet (die Session ID), anhand derer WebObjects den Zustand der Applikation wiederherstellen kann.

WebObjects ist objektorientiert. Demnach wird auf der Applikationsebene nur mit Objekten gearbeitet. Eine Seite (oder auch Component) ist ein Objekt. Sie kann wiederum aus Unterkomponenten bestehen. Ein Menge von Grundkomponenten wie z.B. Eingabefelder, PopUp Buttons und Submit Buttons sind schon implementiert. Interessant ist es nun, aus diesen Grundkomponenten eigene,

wiederverwendbare, etwas größere Komponenten zu bauen und diese dann zu verwenden. WebObjects erlaubt eine Aufteilung der Seite in Subkomponenten, sie kann somit nach dem Baukastenprinzip erstellt werden.

Wie funktioniert nun die Generierung der Seiten? Eine Seite (Component) besteht aus folgenden Teilen:

- HTML Template: Es gibt die Grundstruktur der Seite an und enthält spezielle WebObjects Tags, die auf Objekte verweisen, welche im WebObjects Definitions File (wod) spezifiziert sind.
- Definition File: Es ist die Verbindung zwischen dem HTML Template und der Applikation. Hierin enthalten sind Definitionen von dem im HTML Template verwendeten Objekten. Die hier verwendeten Objekte sind entweder Grundkomponenten oder eigene Subkomponenten. Zu einer Definition gehören:

· Name des Objekts

· Name der Klasse des Objekts

· Attributslisten für die Verknüpfung von Programmvariablen mit den internen Variablen des Objekts

- Eine Klasse, die von WOComponent erbt. Diese Klasse ist in entweder Java, Objective-C oder WebScript zu programmieren.

Weiterhin nehmen an der Seitengenerierung noch die Objekte *Application* und *Session* teil.

Zum Verständnis ist es besser, erst die Generierung einer Seite zu betrachten. Auf den Anfragezyklus wird danach eingegangen. Auf die Anfrage einer Seite wird zuerst die zur Seite gehörende Klasse instanziiert. Danach wird das HTML Template und das Definition File geparsed und in einem Syntaxbaum überführt. An der Wurzel steht das gerade instanziierte Seitenobjekt. Die Knoten sind die im HTML eingelagerten speziellen WebObjects Tags. Für jedes Tag wird die Definition im Definitionfile nachgeschlagen, die dort spezifizierte Klasse instanziiert und zum Knoten hinzugefügt. Beginnend bei der Wurzel wird nun der

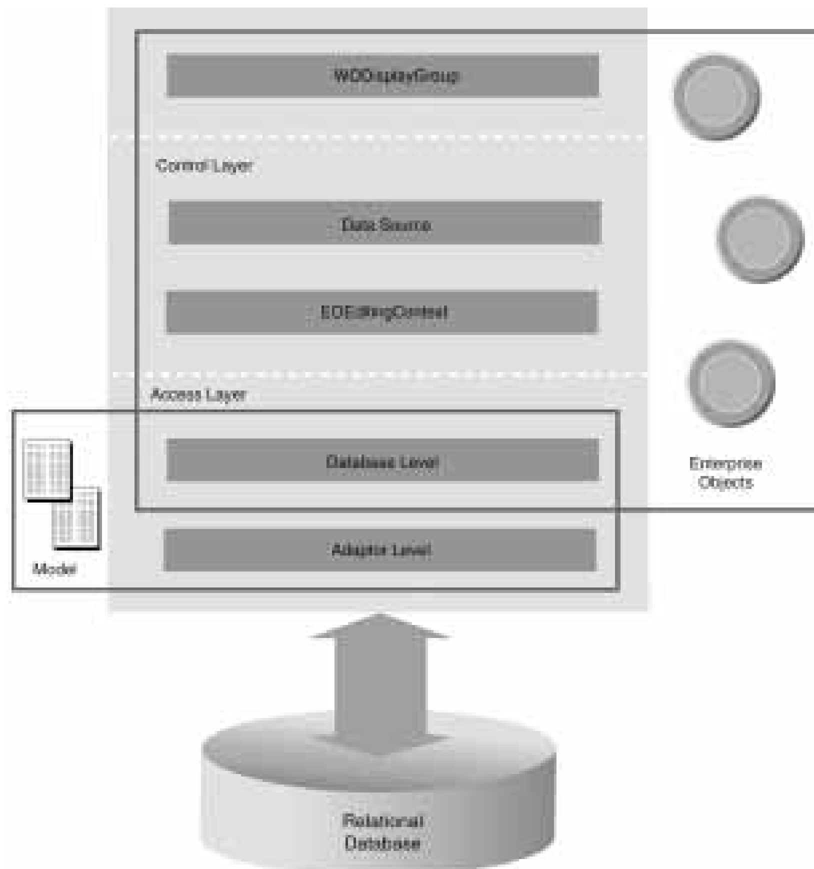
Befehl gegeben sich zu initialisieren, die Attributslisten, die im Definitionfile stehen abzuarbeiten, um danach das zugehörige HTML zu generieren. Damit ist die Generierung abgeschlossen.

Wie sieht nun der Anfang des Zyklusses aus? Ein Benutzeranfrage wird über den *WOAdaptor* an das *Application* Objekt als *Request* Objekt weitergereicht. Dieses findet nach der enthaltenen Session ID das zugehörige *Session* Objekt und reicht die Anfrage an das *Session* Objekt weiter. Die Session schaut nun in ihrem Speicher nach dem noch vorhandenen Baum (der von der vorigen Anfrage stammt). Falls das *Request* Eingabewerte des Benutzers beinhaltet, werden diese an das Wurzelobjekt weitergegeben. Anhand der Attributslisten weiß das Objekt, welche Werte für es bestimmt sind und reicht die Informationen an seine Kinder im Baum weiter die genauso verfahren. Zusätzlich zu Eingabewerten enthält ein Request immer ein Event (d.h. ein Link wurde geklickt, ein Submitbutton gedrückt oder ähnliches). Dieses Event ist einer bestimmten Subkomponente im Baum zugeordnet. In der Attributsliste aus dem Definitionsfile steht der Name der Methode, die bei diesem Event aufgerufen werden soll. Der Rückgabewert dieser Methode muß eine neue Seite in Form des Wurzelobjekts der neuen Seite sein. Um den vorher beschriebenen Mechanismus muß sich der Programmierer nicht kümmern. Er erlangt die Kontrolle durch den Methodenaufruf.

Wichtige Klassen für den Programmierer sind demnach die *Session*, die *Application* und die eigenen Subklassen von *WOComponent*.

6.5.3 Der Datenbankanschluß mit EOF

Abbildung 6–7 EOF Architektur



In Abbildung 6–7 ist die Architektur des Datenbankframeworks EOF dargestellt. Über ein Modell wird eine Abbildung von Tabellen zu Objekten definiert. Dieses Modell kann mit dem Entwicklungswerkzeug EOModeler bearbeitet werden. Zusätzlich zu den Tabellen nach Objekten Abbildung werden hierin auch die Relationen der einzelnen Objekte zueinander über Datenbankjoins definiert. Die Abbildung schließt auch die Übersetzung von internen Datenbanktypen zu den im Objekt vorhandenen Attributstypen ein. Das Ziel ist, sich nicht mehr um die Datenbank kümmern zu müssen, sondern mit dem auf dem EOControl Level angesiedelten objektorientierten API zu arbeiten.

Das einzige datenbankspezifische ist der Adaptor. Dieses Protokoll ist offen und es kann für jegliche Datenquelle ein Adaptor geschrieben werden. Auf Anfrage der oberen Frameworkschichten generiert der Adaptor SQL Aufrufe und verpackt die Resultate der Abfrage in Objekte. EOF hält die Objekte in einem Objektgraphen und notiert jegliche Änderungen, die vorgenommen werden. Der Programmierer arbeitet mit den Objekten der oberen Schicht des Frameworks.

Der EOEditingContext ist die verantwortliche Klasse, die den Objektgraphen verwaltet. Er repräsentiert eine Transaktionseinheit und notiert die Änderungen. Enterprise Objects (Datenobjekte) sind somit ganz normale Objekte mit dem kleinen Unterschied, daß sie zu einem EOEditingContext gehören und von ihm überwacht werden, wenn sie sich ändern. Der Benutzer dieser Objekte merkt jedoch nichts davon, daß sich EOF um die Persistenz kümmert. Der EOEditingContext bedient sich wiederum untergeordneter Klassen, um Datenbankobjekte zu holen und zu speichern. So wie auch Datenbanktransaktionen geschachtelt werden können, ist es auch möglich auch EOEditingContexts zu schachteln.

Zur Anzeige von Enterprise Objects gibt es die Klasse WODisplayGroup. Während sich der EOEditingContext um alle Objekte kümmert, beschränkt sich die WODisplayGroup auf eine Entität (Klasse), wie zum Beispiel die Anzeige von Personen. Sie bedient sich des EOEditingContexts, um Objekte zu holen und zu speichern. Die Aufgabe der WODisplayGroup ist:

- Vereinfachtes Interface des EOEditingContexts zur Verfügung zu stellen.
- Methoden zu Sortierungen und einschränkende Abfragen zu implementieren
- Möglichkeiten ein Suchresultat in Untermengen zu X Objekten anzuzeigen und durchzunavigieren
- Darstellung von Master-Detail Relationen wie z.B. Projekte zu Aktivitäten ermöglichen.
- Verwaltung einer Selektionsmenge des Benutzers.

Somit beschränkt sich die Interaktion mit dem Datenbankframework EOF hauptsächlich auf die zwei Klassen EOEditingContext und WODisplayGroup.

Für weitere Informationen kann die EOF Dokumentation von Apple herangezogen werden.

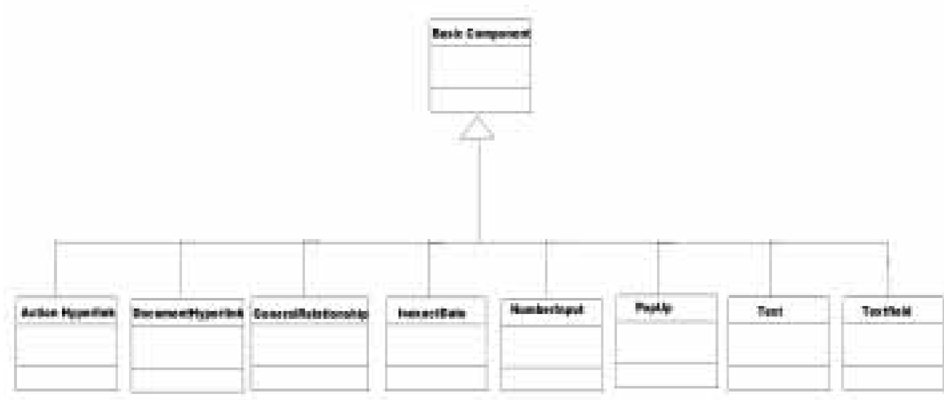
6.6 Einige Realisierungskonzepte

Bei der Realisierung haben sich einige Konzepte als sehr brauchbar erwiesen. Sie sollen hier kurz behandelt werden. Teilweise ist die Problematik nicht offensichtlich, sondern liegt vielmehr in WebObjects oder EOF.

6.6.1 Allgemeine Anzeigeelemente

Für die Anzeige und Eingabe verschiedener Attribute hat es sich angeboten aus den standardisierten Eingabe- und Ausgabeelementen angepaßte Ein- und Ausgabeelemente zu kreieren. Die Vererbungshierarchie dieser Komponenten ist in Abbildung 6–8 dargestellt. Sie haben ein Standard Interface, welches durch die Klasse *BasicComponent* definiert ist. Dazu gehört eine Variable isInput, welche definiert, ob das Element für die Eingabe oder Ausgabe dient, dazu das Datenobjekt und den Pfad zum Attribut. Der Hauptvorteil, der sich dadurch ergibt, ist die Konfiguration des Interfaces mit Attributslisten. Siehe dazu den nächsten Punkt.

Abbildung 6–8 **Vererbungshierarchie der Anzeigenelmente**



6.6.2 Konfiguration der Anzeige durch Attributslisten

Es gibt zwei Möglichkeiten, die Anzeigeelmente zu gestalten. Entweder man kodiert alle Anzeigen per Hand oder entwirft ein generisches Element, welches die Anzeige beliebiger Objekte realisiert und durch eine Attributsliste konfiguriert wird. Im Laufe der Implementierung hat sich gezeigt das ein Mittelweg das beste Maß ist.

In Abbildung 6–9 ist die Verwendung einer durch Attributslisten konfigurierten Anzeige dargestellt. In Abbildung 6–10 ist die zugehörige Attributsliste zu sehen. Der grau unterlegte Bereich stellt die Eingabe des Enddatums dar, der durch die Komponente InexactDate dargestellt wird. Die Methode, die an den "OK" Button gebunden ist, braucht nur das *isInput* Attribut der Komponente zu verändern und steuert so, ob die Eingabemaske für ein Datum erscheint (wie der grau unterlegte Bereich) oder die Anzeige, die neben dem Anfangsdatum steht und nur das Datum formatiert.

Abbildung 6–9 **Aktivität editieren**

Die Attributsliste in Abbildung 6–10 besteht für jedes Element aus einem Titel (*title)*, einer internen Bezeichnung des Attributs (*keypath)* und der Anzeigenkomponente. So ist der Titel für das Attribut *startDateValue* "Anfangsdatum" und die anzeigenden Komponente ist *InexactDate*. Gleicherweise lassen sich

die Werte für *endDateValue, description, activityType* und *status* aus der Attributsliste lesen. Eine Erweiterung der Anzeige um ein Attribut ist einfach durch Hinzufügen eines neuen Eintrags in der Attributsliste möglich.

Abbildung 6–10 Attributsliste für das Editieren der Aktivitäten

```
activityDetails = { config =
          ({ title = "Anfangsdatum";
                keypath = "startDateValue";
               component = "InexactDate"; },
          { title = "Enddatum";
                keypath = "endDateValue";
               component = "InexactDate"; },
          {   title = "Beschreibung";
              keypath = "description";
               component = "Text"; },
          {   title = "Aktivit&auml;tstyp";
              keypath = "activityType";
               component = "PopUp"; },
          {   title = "Status";
              keypath = "status";
               component = "PopUp"; }
            ) };
```

Folgende Komponenten benutzen Attributslisten, um Werte darzustellen:

- *__editDisplayAttributeRow*, welche ein Zeile in einer Tabelle mit dem jeweiligen Titel und dazugehörigen Attribut anzeigt. Außerdem ermöglicht es das Editieren des Attributs. Diese Komponente kommt in Aktivität editieren, Projektverwaltung Hauptseite und Milestone editieren vor.

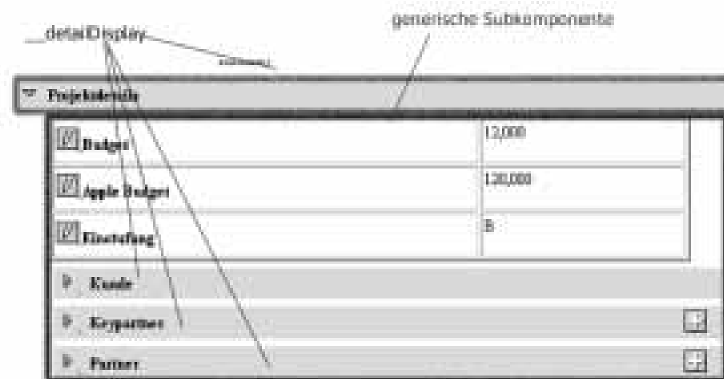
- *_detailTable, DetailRow, DetailRowDetail, __headersRow,* welche eine Tabelle mit Überschrift und den dazugehörigen Attributen anzeigt. Dies ist eine der meistgenutzten Komponenten. Sie kommt in Projektsuche, Projektverwaltung Hauptseite, Milestone editieren, Aktivität editieren, Erinnerungen anzeigen, Erinnerungsdetails anzeigen und der Personen- und Accountsuche vor.

- *__inputAttributeRow,* welche ein Attribut anzeigt, mit der Möglichkeit es zu editieren. Sie wird in der Hauptseite der Projektverwaltung und bei der Editierung von Milestones eingesetzt.

6.6.3 Detailanzeigen

Eine vielgenutzte Komponente ist die *_detailDisplay* Komponente. Sie wird bei der Projektverwaltung, beim Editieren von Milestones, beim Editieren von Aktivitäten, bei der Anzeige von Erinnerungen und der Anzeige der Details von Erinnerungen benutzt. In Abbildung 6–11 ist eine Benutzung der Detailanzeige in der Applikation exemplarisch herausgegriffen. Die Anzeige besteht aus einem Balken mit Titel und dem Pfeil für das An- und Ausschalten der Detailanzeige. Die Komponente ist so entworfen worden, daß die Detailansicht irgend eine andere Komponente sein kann. Alle Detailansichten haben eine gemeinsame Schnittstelle, um Informationen von der *_detailDisplay* zu bekommen. Deshalb erben alle Komponenten, die als Detailansicht eingesetzt werden sollen, von der Klassen *DetailDisplayRoot.*

Abbildung 6–11 Beispiel der Benutzung der Detailanzeige



In der Abbildung sieht man deutlich, daß die hier verwendete Detailansicht wiederum die Komponente *_detailDisplay* enthält. Es ist außerdem noch möglich der Komponente *_detailDisplay* ein Aktionsitem (in diesem Fall das “Plus” Bild) mit zugehörigem Methodenaufruf zuzuweisen.

6.6.4 Konsistenz der Datenbank sicherstellen

Ein wichtige Aufgabe ist, die Konsistenz des Objektgraphen, also die Konsistenz der Datenbank, zu gewährleisten. Das Problem stammt von den weitreichenden Navigationsmöglichkeiten, welche durch die Navigationshilfe gegeben sind. Als Anschauungsbeispiel stelle man sich den Vorgang des Hinzufügens eines Keypartners vor. Der Benutzer ist im Projekt Management Hauptfenster und möchte einen Keypartner hinzufügen. Die Suche schlägt fehl und er gibt ein neuen Keypartner ein. Das Programm kreiert ein neues Partner Datenbankobjekt. Während der Eingabe möchte er noch die Muttergesellschaft hinzufügen, die auch noch nicht in der Datenbank vorhanden ist. Auch hier wird ein Partner Datenbankobjekt erstellt. Wenn alles nach Plan läuft, müßte der Benutzer nun zweimal die Eingabe bestätigen, die Daten werden validiert und in die Datenbank eingetragen. Falls er aber zurück auf die Projekt Management Hauptseite mit Hilfe des Naivgationsmenüs zurückspringt, kommt es zu Inkonsistenzen. Das Programm hält nun zwei Datenbankobjekte, die nicht mehr gültig sind und deren Attribute nicht validiert sind.

Es muß also eine möglichst allgemeingültige Lösung gefunden werden, um die Konsistenz des Objektgraphen in solchen Fällen sicherzustellen. EOF unterstützt das Verwerfen von Änderungen in einem EOEditingContext, der einer Transaktion entspricht. Im Programm hat nun jede Seite, in der etwas eingegeben wird, ihren eigenen EOEditingContext, d.h. ihr eigenes Transaktionsfenster, das in den vorigen EOEditingContext geschachtelt ist. Die Seite merkt sich auch ihre Schachteltiefe. Wird auf die Seite zurücknavigiert, werden alle Änderungen in tiefer geschachtelten EOEditingContexts zurückgenommen. Das hat keinen Effekt, wenn die vorigen Seiten ihre Änderungen gespeichert haben. Falls das nicht der Fall ist, wie im obigen Beispiel, werden die so kreierten Objekte zerstört und die Konsistenz der Datenbank ist wieder hergestellt.