# Entwurfsbeschreibung der Kundenbetreuer-Erweiterung für eine Kooperationsplattform

## I. Allgemeines

Das zu entwickelnde Produkt soll als modulare Erweiterung in bestehenden OLAT-Instanzen, insbesondere im KOOP-Prototyp, das Tätigkeitsfeld des Kundenbetreuers abbilden. Dafür werden Adress- und Kontaktverwaltungsfunktionalitäten benötigt. Außerdem muss die bestehende Plattform um eine administrative Akte erweitert werden, in welcher firmeninterne Daten zu einem User, etwa einem Kunden oder seinem Betreuer, gespeichert werden können.

## II. Produktübersicht

Die zu realisierende Erweiterung soll die Arbeitsabläufe eines Kundenbetreuers, welche dieser bisher über die ADAG-Plattform durchführte, in einer Kooperationsplattform ermöglichen. Dafür müssen zwei unterschiedliche Tätigkeitsstränge implementiert werden: die Verwaltung von Nichtmitgliedern und die von Barterkunden. Alle hierbei durchgeführten Aktivitäten sowie Anmerkungen oder sonstige Informationen werden in einer Historie gespeichert, die einer Adresse oder einem Kontakt zugeordnet ist.

Bei der Verwaltung von Nichtmitgliedern geht es in erster Linie um das Management der Adressdatenbank und die Gewinnung neuer Kontakte. Dies kann in die Bereiche der Adressakquise, der telefonischen Kontaktaufnahme und des Besuchs untergliedert werden. Für letzteres werden zudem Ausdrucke aus dem jeweiligen Datensatz benötigt.

Die Betreuung der Barterkunden erfordert ebenso Telefongespräche und Besuche. Bei Telefonaten wird jeweils nur ein Historiendatensatz angelegt, für ein Treffen benötigt der Kundenbetreuer wieder einen kompletten Ausdruck der entsprechenden Kundendaten. Außerdem ändert er gegebenenfalls bestimmte Einträge oder (Vertrags-)Bedingungen, wenn sich durch den Besuch Veränderungen ergeben haben.

Es ist zudem möglich, Historieneinträge mit einem Wiedervorlagedatum zu versehen, welches im Kalender eingetragen wird und zum entsprechenden Zeitpunkt automatisch den gewünschten Eintrag wieder anzeigt. Der Kalender beinhaltet des Weiteren alle Termine – etwa Telefonate oder Besuche, welche der Kundenbetreuer mit Kontakten oder Kunden vereinbart aber noch nicht in der Historie eingetragen hat.

Außerdem wird vom Produkt eine administrative Akte zur Verfügung gestellt, in welcher als Gegenstück zur OLAT-Identity die Geschäftsdaten eines Users gespeichert werden. Diese ist modular erweiterbar und kann im Rahmen des vorliegenden Produkts zwei unterschiedliche Nutzergruppen aufnehmen: den Kundenbetreuer und die Kunden.

# III. Struktur- und Entwurfsprinzipien für das Gesamtsystem

**Grundlegendes zur Modularität:** Die gesamte Erweiterung teilt sich in zwei Bereiche auf, und zwar in die administrative Akte – „*Record*“ – und ein Bündel von modular in einen OLAT-Kurs integrierbaren Funktionalitäten zur Adress- und Kontakt-bzw. Kundenverwaltung – zusammengefasst in den Kursbausteinen „*BBAddressManagement*“ und „*BBContactManagement*“. Es wird also innerhalb der Implementierung nach Funktionen getrennt, die sich entweder nur auf die Adressen oder auf Kontakte beziehen. Die Entität „Barterkunde“ aus der bisherigen Plattform würde dabei zukünftig insbesondere auch ein „*Contact*“-Objekt im *Record* umfassen, sodass sich alle Aufgaben, die sich beim Kunden auf den Kontakt beziehen, auch über den *ContactManagement*-Baustein erledigen ließen. Ein neuer Baustein „*BBCustomerManagement*“ wird perspektivisch allerdings benötigt, um beispielsweise die Transaktionen und Kontostände der Kunden für den Kundenbetreuer verfügbar zu machen, da diese nicht mehr durch *Contact* gekapselt sind.
Die Datenobjekte als auch die Kursbausteinkomponenten orientieren sich dabei wieder strikt am MVC-Prinzip in OLAT und der entsprechenden Strukturierung in Datenklasse, Controller und Manager sowie Hilfsobjekte (etwa des Views: TableModel und zugehörige Controller).

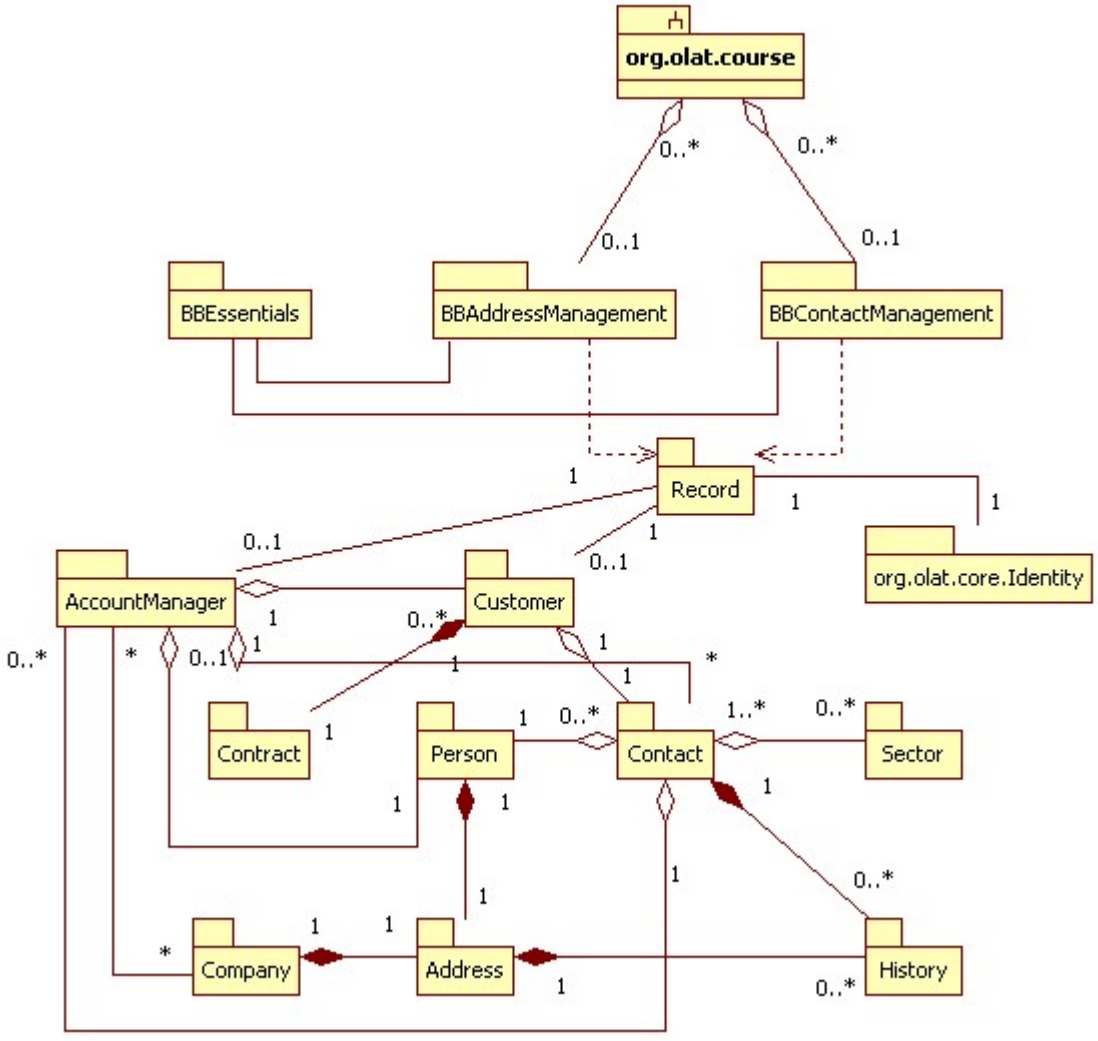



Abbildung 1: Überblick über das gesamte Produkt

**Grundlegendes zur Datenhaltung:** Die Records werden modular aus Datenpaketen zusammengestellt, die innerhalb einer Akte akkumulierbar sind und zusammenhängende Informationseinheiten, beispielsweise eine Historie oder eine Person, abbilden. Die Primärschlüssel sind dabei jeweils der Key des *PersistentObject*, von welchem alle Datenobjekte abgeleitet werden. Für diese Komponenten wird die bestehende Datenbank von OLAT um folgende Tabellen erweitert:

**REC_ADDRESS** (STREET, ZIPCODE, CITY, COUNTRY, EMAIL, FAX, PHONE, REFERENCED)
**REC_ADDRESSCONTAINER** (STREET, ZIPCODE, CITY, COUNTRY, EMAIL, FAX, PHONE, ADDRESS, ACCOUNTMANAGER)
**REC_COMPANY** (NAME, APPENDIX, WEBSITE, CATEGORY, COMMENT, ADDRESS, PARENT_COMPANY, REFERENCED)
**REC_COMPANYCONTAINER** (NAME, APPENDIX, WEBSITE, CATEGORY, COMMENT, ADDRESS, PARENT_COMPANY, COMPANY, ACCOUNTMANAGER, ADDRESSCONTAINER)
**REC_PERSON** (ADDRESS, FIRSTNAME, LASTNAME, TITLE, DATEOFBIRTH, ACTIVITY, SEX, MOBILEPHONE, CATEGORY, COMMENT, LANGUAGE, REFERENCED)
**REC_PERSONROLE** (PERSON, CONTACT, ROLE, REFERENCED)
**REC_CONTACT** (ACCOUNTMANAGER, COMPANY, BET, SEARCH, SEARCHKEYWORDS, INFORMINGMETHOD, STATUS, REFERENCED, ENTRYDATE)
**REC_SECTOR_CONTACT** (SECTOR, CONTACT)
**REC_SECTOR** (NAME, DESCRIPTION, REFERENCED)
**REC_HISTORY** (SALESTYPE, DATE, CAUSE, RESUBMISSION, RESULT, ANNOTATION, RESPONDENT, ADDRESS, CONTACT, ACCOUNTMANAGER, REFERENCED)
**REC_CUSTOMER** (CONTACT, CONTRACT)
**REC_CONTRACT** (PROVISION, ACTIVITY, TERMINATIONDATE, INACTIVITYPERIOD)
**REC_ACCOUNTMANAGER** (PERSON, ABBREVIATION)
**REC_RECORD** (IDENTITY, BODY_TYPE, ID)

Die alten Daten des bestehenden Systems sollen dabei mittels eines einfachen Managers in die neue Datenbank portiert werden. Dieser liest aus der vorhandenen ADAG-Datenbank alle relevanten Tupel aus und speichert sie in den neuen Klassen.
Die Branchenzuordnung zu den jeweiligen Kontakten erfolgt über die Tabelle **REC_SECTOR_CONTACT**, die Zuordnung der Personen mit ihren Rollen über **REC_PERSONROLE**; die Akte wird über das Identity-Objekt mit dem jeweiligen Nutzer verknüpft.

Die Tabelle **REC_RECORD** nutzt ein many-to-any-Hibernatemapping, sodass die Anzahl von Arten der Klassen, welche im Rumpf des Records gespeichert werden können, beliebig erweiterbar ist.

**Grundlegendes zum dynamischen Modell:** Die Dynamik des Produkts ist durch die gegebenen Arbeitsabläufe eines Kundenbetreuers bestimmt. Neben Operationen, die einzeln durchgeführt werden – etwa die Aktualisierung einer bestimmten Adresse, lassen sich als generelle UseCases die nachfolgenden Prozessketten erkennen.

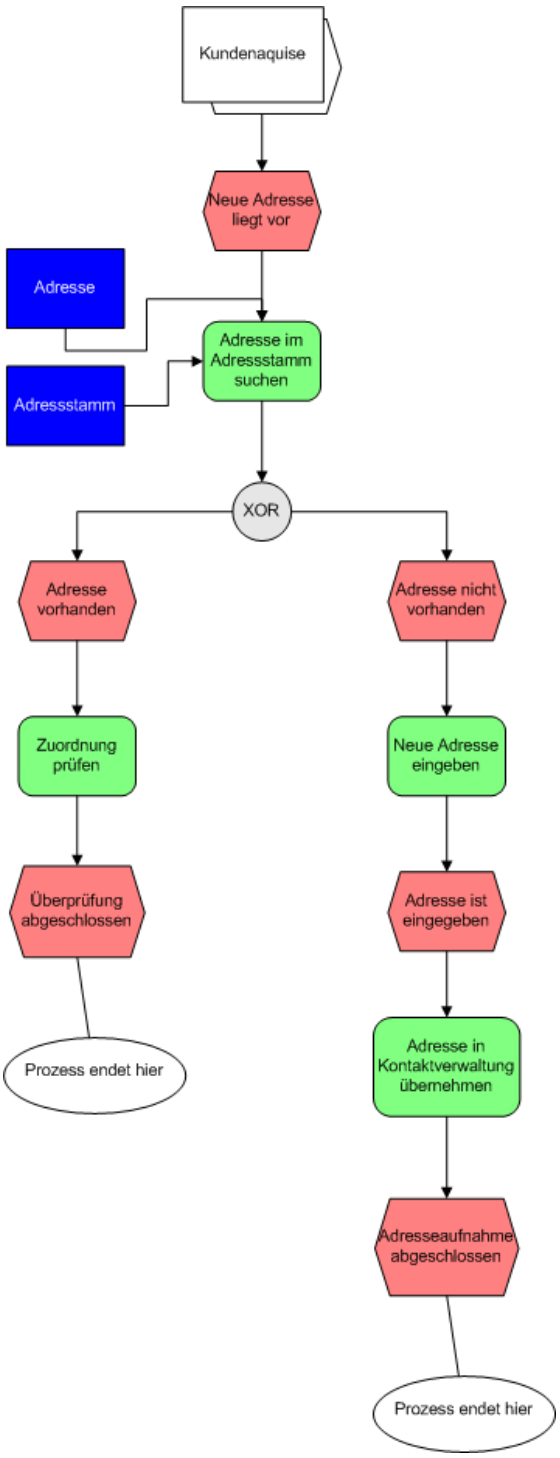



Abbildung 2: Verwaltung von Adressen

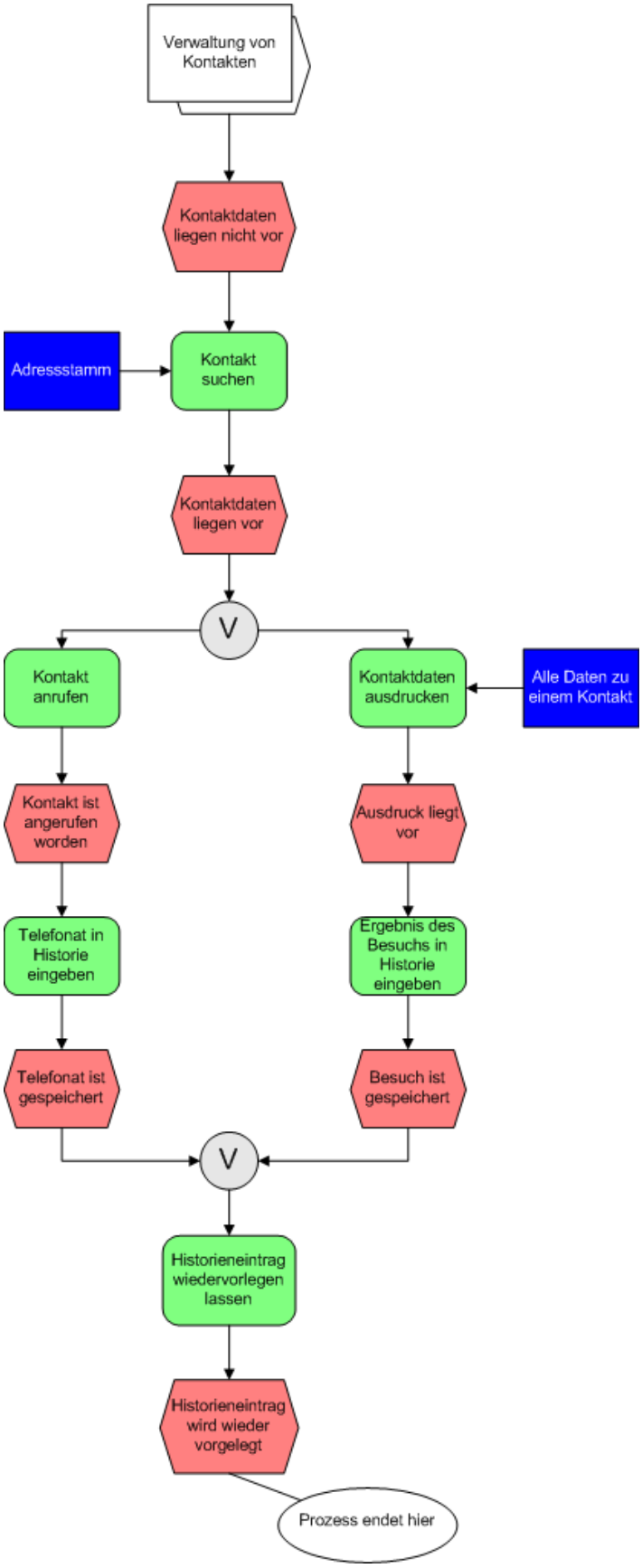
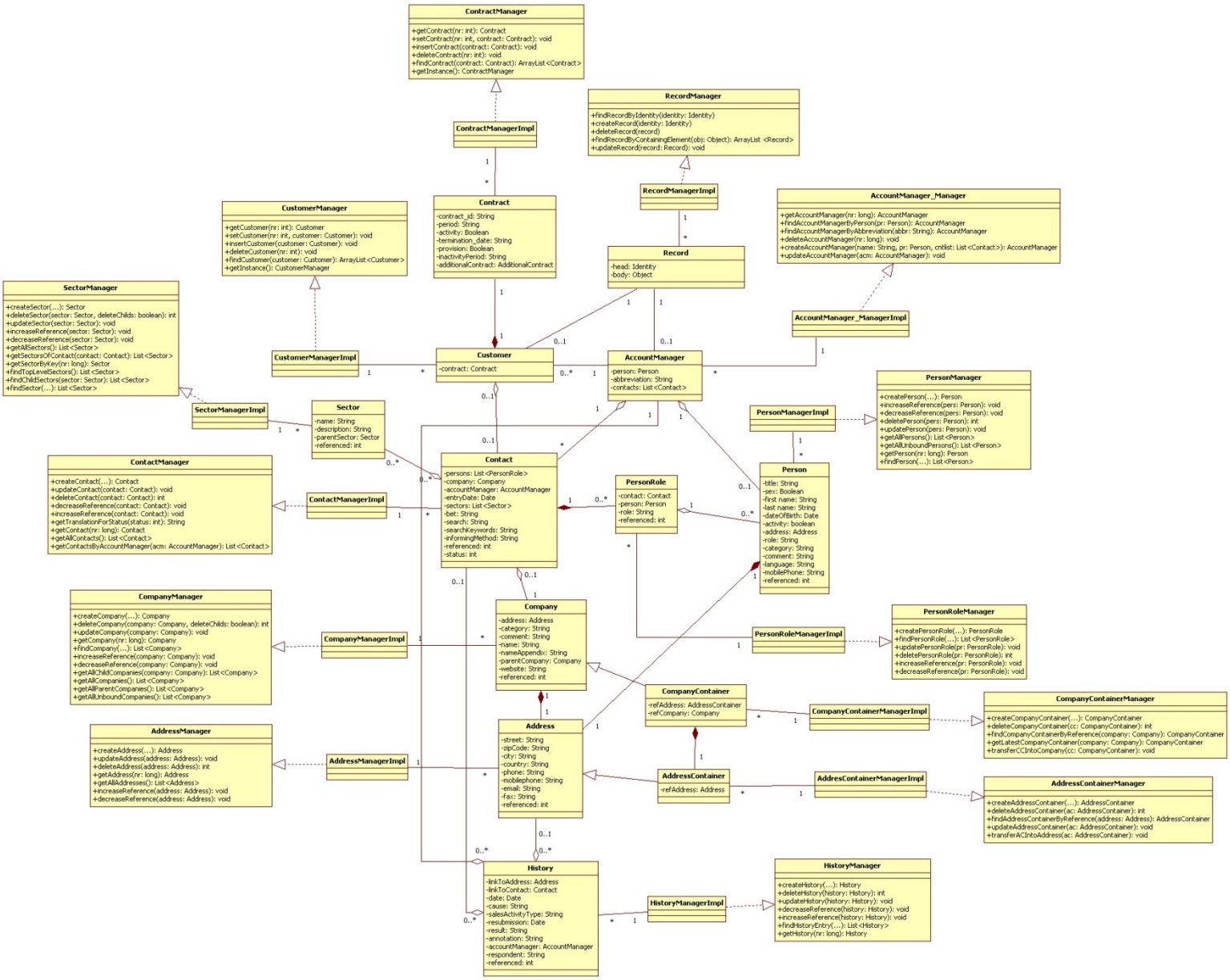



Abbildung 3: Verwaltung von Kontakten

# IV. Struktur- und Entwurfsprinzipien der einzelnen Pakete

## (a) Record




Das *Record*-Paket realisiert mittels verschiedener, aufeinander aufbauender Datenklassen das Prinzip einer administrativen Nutzerakte, welche die Datenbasis für die Funktionalitäten der Erweiterung darstellt. Die kleinste Dateneinheit ist dabei *Address*, welche eine rohe Adresse repräsentiert und wahlweise einer *Person* oder einer *Company* zugeordnet werden kann. Dabei wird sie innerhalb einer Person um deren spezifische Eigenschaften wie Geburtsdatum oder Geschlecht erweitert, in einer Firma erfolgt hingegen die Zuweisung des entsprechenden Firmennamens sowie eventueller Unterfirmen. Sowohl Adressen als auch Kontakte können mit Historieneinträgen (*History*) versehen werden, die die Interaktionen von Kundenbetreuern mit ihnen wiedergeben.

Nachdem der Kundenbetreuer mit einer Firma kommuniziert hat, und sich dadurch neue Daten wie etwa Branchenzuordnungen und Ansprechpartner ergeben haben, kann er eine Firma in einen

Kontakt (*Contact*) umwandeln. Dabei weist er eine Menge von Personen, Branchen und zusätzlichen Informationen wie dem Erfolgsstatus oder Benachrichtigungsmethoden zu, die in dem neuen Datenobjekt gespeichert werden. Wenn ein Kontakt Barterkunde wird, so schließt er einen Vertrag (*Contract*) ab, dieser wird dann zusammen mit dem *Contact*-Objekt in einer *Customer*-Klasse zusammengefasst, welche hierarchisch gesehen in der höchsten Ebene der Dateneinheiten liegt, und somit im Rumpf (*Body*) eines *Records* gespeichert werden kann. Außerdem wird diese Akte über den Kopf (*Head*) mit einem Systemuser per olat.org.core.Identity verknüpft, sodass der Zusammenhang zwischen privatem Nutzerprofil innerhalb der Plattform und administrativem Datensatz aus der Tätigkeit der Kundenbetreuer sichergestellt ist.

Ähnlich verhält es sich mit dem Kundenbetreuer selbst, der in einem *AccountManager*-Objekt abgebildet wird, das ebenfalls Body einer Akte sein kann. In dieser Klasse wird die Person des Kundenbetreuers mit seinem Kürzel und der Menge der von ihm betreuten Kontakte und Kunden verknüpft.

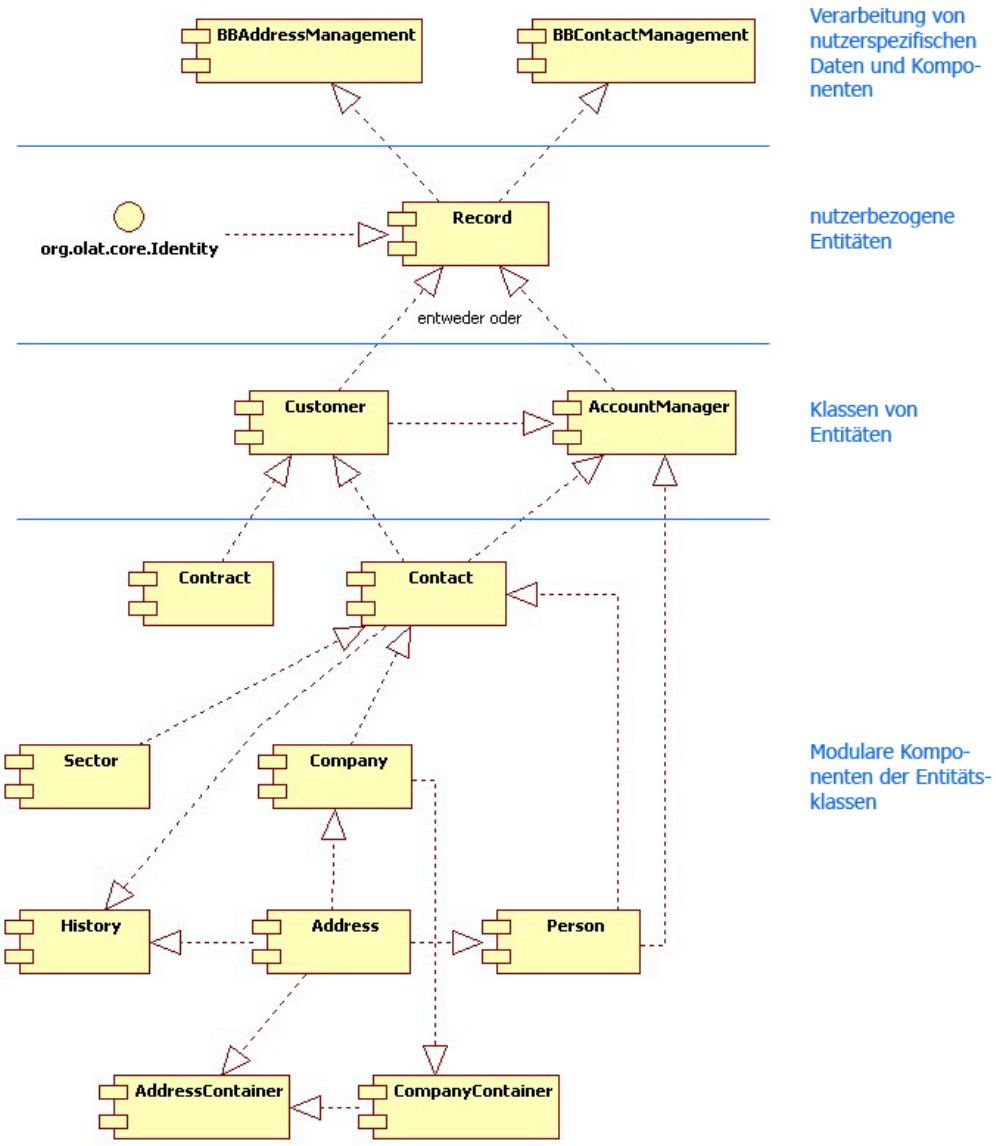



**Abbildung 4: Schichtenstruktur der Daten im Record**

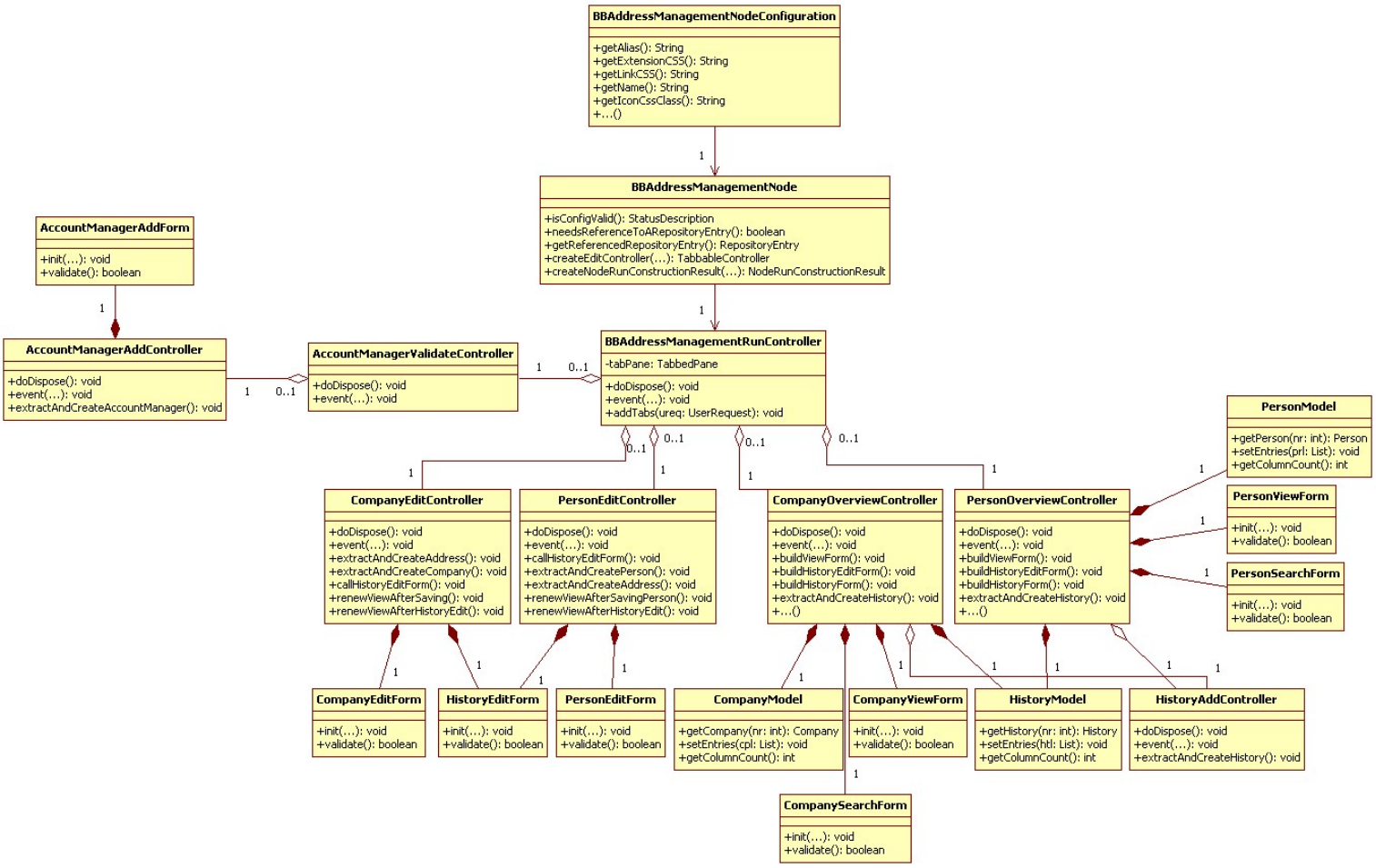
Die so entstehende Schichtenstruktur erlaubt dabei in modularer Weise leicht das Einbringen neuer Nutzertypen für die Akte, etwa eines Mitarbeiters oder einer Kassenstelle. Hierfür müssen lediglich in analoger Weise alle unterliegenden Datenverbände entsprechend einer Baumstruktur hinzugefügt werden.

Eine Besonderheit ist der Referenz-Zähler, den auf der Stufe der Entitätskomponenten jede Datenklasse enthält. Dieser gibt die Anzahl der logischen Referenzen auf die entsprechende Komponenteninstanz an, und soll verhindern, dass ein Objekt gelöscht werden kann, wenn es von anderen (eventuell später hinzugefügten) Bausteinen eingebunden wird. Dabei erspart der Counter den mühsamen Eingriff auf der Datenbankebene in Form von Fremdschlüsseln, und ermöglicht ein differenziertes, gegebenenfalls nur partiell auszuführendes Löschen von Objekten.

Weiterhin bestehen zu den *Address*- und *Company*-Komponenten sogenannte *Container*, welche Änderungen an bestehenden Objekten aufnehmen, wenn zu ihnen ein sie bindender hierarchisch höher liegender Baustein – in diesem Fall *Contact* – angelegt wurde. Damit wird sichergestellt, dass alle Kundenbetreuer im System Änderungen an Firmen in bestehenden Kontakten vornehmen können, dass diese Modifikationen aber nicht ungeprüft und damit ungeschützt übernommen werden. Der Inhaber des Kontaktobjektes kann jedoch Änderungen weiterhin direkt vornehmen.

Die einzelnen Datenklassen werden von Managern verwaltet, die das konkrete Suchen und Bearbeiten einzelner Objekte ermöglichen. Dadurch werden insbesondere auch Daten, die nicht über einen *Record* einem speziellen Nutzer zugeordnet sind (beispielsweise freie Kontakte), für diesen durch Bereitstellung der Funktionalität etwa in einem Kursbaustein einsehbar und nutzbar. Die Manager übernehmen ebenso die Verwaltung der Referenzzähler und das vollständige Löschen eines Objektes und aller darin gebundenen existenzabhängigen Unterobjekte (etwa *History*-Einträge).

### (b) BBAddressManagement

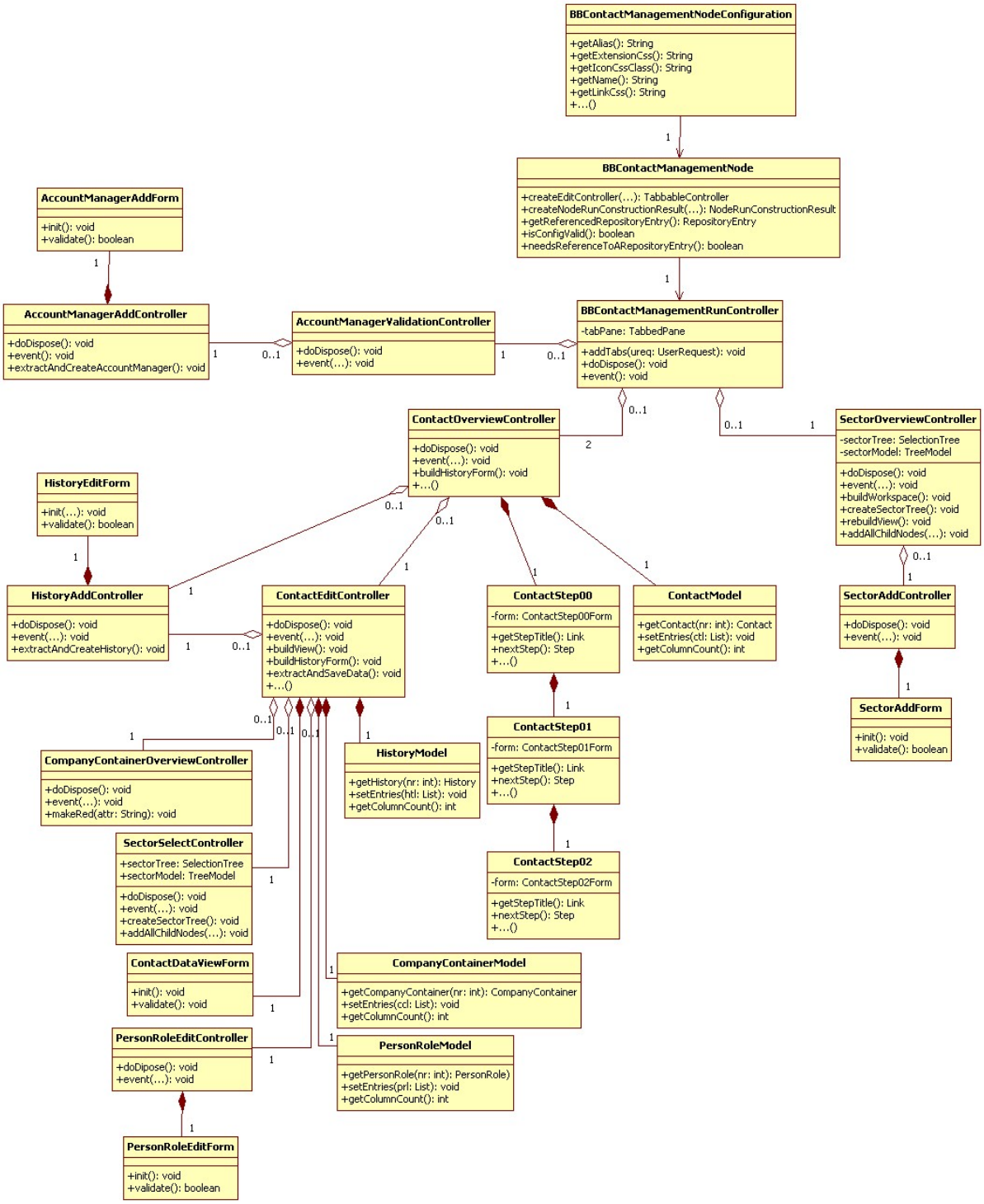



Der Kursbaustein *AddressManagement* orientiert sich strukturell strikt am Aufbau eines OLAT-Kursbausteins und soll dazu dienen, den Bestand an *Address*-Objekten aus der Aufgabensicht eines Kundenbetreuers zu verwalten. Dabei geht es vor allem darum, einen Überblick über alle Adressen und zugehörige Historieneinträge zu bekommen, die im System registriert sind. Dies geschieht über die *CompanyOverview*- und *PersonOverview*-Controller, welche den Datenbestand an Firmen und Personen tabellarisch anzeigen. Dabei kann der Kundenbetreuer gezielt nach bestimmten Datensätzen suchen und für jeden Datensatz auswählen, ob dieser bearbeitet, seine Historie angezeigt oder erweitert bzw. der Datensatz gelöscht werden soll.

Des Weiteren besteht über die *CompanyEdit*- und *PersonEdit*-Controller die Möglichkeit, neue Firmen und Personen mit eventuell vorhandenem *History*-Einträg zu speichern. Dabei kann einer Firma auch eine Stammfirma zugewiesen werden, sodass kompliziertere Firmenstrukturen mit Mutter- und Tochtergesellschaften abbildb sind.

Alle Controller benötigen zur Anzeige der Daten entweder Formulare oder entsprechende Tabellenmodelle, welche direkt an die Controller angebunden sind. Für den Datenzugriff werden außerdem Instanzen des *Address*- und *HistoryManagers* genutzt.

### (c) BBContactManagement




Der *ContactManagement*-Kursbaustein ermöglicht die Verwaltung von Kontakten. Hierbei wird über den *Record* eines Kundenbetreuers sein Stamm an Kontakten und Kunden ermittelt, die er zu betreuen hat. Diese sind dann über eine Überblickstabelle im *ContactOverviewController* zugänglich und in ihren Kontaktdaten durch den *ContactEditController* bearbeitbar. Der gesamte Kontaktdatenstamm wird über einen zweiten *ContactOverviewController* bereitgestellt, indem ein lesender Zugriff auf alle in der Datenbank vorhandenen Kontakte ermöglicht wird.

Des Weiteren lässt sich auch zu Kontakten eine (separate) Historie erstellen, welche über das Tabellenmodell *ContactModel* erreichbar ist. Zur Wiedervorlage des Historieneintrags wird ein entsprechender Datumseintrag im Historienobjekt gesetzt.

Liegen Änderungen an den Firmendaten eines Kontaktes durch andere Kundenbetreuer vor, so werden diese mit Hilfe des *CompanyContainerModel* zur Anzeige gebracht. Nach der Auswahl einer Änderung aus der Tabelle zeigt der *CompanyContainerOverviewController* diese an und hebt Modifikationen zu den aktuell gespeicherten Firmendaten farbig hervor. Der den Kontakt haltende Kundenbetreuer hat dabei die Auswahl, ob er die Änderungen übernehmen oder verwerfen will.

Zudem können in einem dreistufigen Wizard neue Kontakte hinzugefügt werden. Hierfür wird im ersten Schritt *ContactStep00* zunächst die Firma mit entsprechenden Branchen verknüpft, im zweiten Teil *ContactStep01* erfolgt eine Zuordnung von Ansprechpartner (Personen) mit ihren spezifischen Rollen im Unternehmen, und der dritte Teilschritt *ContactStep02* ermöglicht schließlich die Eingabe weiterer Daten wie Angebots- und Nachfrageinseraten und dem Kontaktstatus. Es ist dabei im System die Einschränkung implementiert, dass Kontakten jeweils nur die untersten Branchen zugeordnet werden können, Dopplungen zu vermeiden (etwa in der Untergliederung Autos, Autos / Große Autos).

Außerdem kann der Kundenbetreuer den gesamten Branchenbaum des Systems über diesen Kursbaustein bearbeiten. Dafür wird der *SectorOverviewController* verwendet. Er ermöglicht in einem Auswahlbaum die Selektion eines hierarchischen Knotenpunktes zu einer Branche, welche dann bearbeitet, erweitert oder gelöscht werden kann. Dazu wird der *SectorAddController* verwendet. Wenn allerdings eine Branche neue Unterbranchen erhält und damit selbst nicht mehr zugeordnet werden kann, veranlasst der Controller kein Löschen der alten Zuordnungen. Diese bleiben solange bestehen, bis sie durch den jeweiligen Betreuer neu gesetzt wurden.