UNIVERSITÄT KOBLENZ · LANDAU

EU-Schadensbericht Back-Office

Bachelorarbeit

zur Erlangung des Bachelors
im Studiengang Informationsmanagement

vorgelegt von

Alexander Rippert

Betreuer:	Diplom-Informatiker Stefan Stein, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik
Gutachter:	Prof. Dr. J. Felix Hampe, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik

Koblenz, im Juli 2010

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

ja☐ nein☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

ja☐ nein☐

Koblenz,den .... Unterschrift

German Summary / Deutsche Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein Back-Office für die elektronische Version des Europäischen Schadensberichtes erstellt. Es wurden bereits in anderen Arbeiten ein mobiler Client, welcher auf einem Windows Mobile Handy läuft, sowie ein Polizei Client erstellt. Diese greifen auf das Back-Office zu, um Daten, wie z.B. die Autodaten (Automarke, der Typ, das Baujahr und Bilder eines 3D-Modells des Autos) zu einem bestimmten Kennzeichen oder die Personendaten des jeweiligen Autobesitzers zu erhalten. Der mobile Client sendet zudem die Unfallakte an das Back-Office, damit die Daten über einen Unfall in diesem abgespeichert und weiter bearbeitet werden können.

Ziel der Arbeit war es ein erweiterbares, modulares System zu entwickeln, welches später um weitere Module ergänzt werden kann, um neue Funktionen bereitstellen zu können. Diese Module können jeweils beliebige Daten in einer Datenbank abspeichern und diese von der Datenbank auch wieder abfragen, sowie verändern, ohne dass das relationale Schema der Datenbank verändert werden muss. Diese Funktionalität wird von dem Kernsystem bereitgestellt.

Als Teil dieser Bachelorarbeit wurden fünf Module entwickelt, die alle auf dem Kernsystem aufbauen und verschiedene Funktionen für unterschiedliche Zielgruppen bereitstellen:

Das Modul für den mobilen Client stellt einen Webservice zur Verfügung, über den der mobile Client Daten über Fahrzeuge sowie über die Fahrzeughalter abfragen kann sowie den Unfallbericht als XML-Datei an das Back-Office übermitteln kann.

Das Modul für den Polizei Client stellt einen Webservice bereit, über welchen Daten über Fahrzeughalter abgefragt werden können.

Das Modul für die Versicherung besteht sowohl aus einen Webservice, der eine Integration in die Softwareinfrastruktur des Versicherungsunternehmens ermöglicht, als auch aus einer Webanwendung, über die ein Mitarbeiter einen Unfallvorgang bearbeiten kann. Dabei kann er alle nötigen Daten sehen und bearbeiten sowie Schäden beurteilen und festlegen, wie hoch die Kosten für eine Reparatur der Schäden sind und ob die Versicherung für den Schaden aufkommt.

Das Modul für den Autobesitzer verfügt, genauso wie das Modul für die Versicherung, über einen Webservice sowie eine Webanwendung. Der Webservice ermöglicht auch hier eine Integration in bereits existierende Systeme. Über die Webanwendung kann der jeweilige Autobesitzer die Daten über einen Unfall ansehen, sowie den Bearbeitungsstatus erkennen. Er kann auch Autowerkstätten auswählen, die dann die Schäden seiner Autos beurteilen dürfen.

Das Modul für die Autowerkstatt besteht aus einem Webservice sowie einer Webanwendung. Ein Mitarbeiter kann die Schäden zu einem Unfall in das System einpflegen sowie die Kosten zu dessen Reparatur angeben.

Zudem gibt es noch eine Weboberfläche für Administratoren des Systems, über die die Benutzer des Systems, die Automodelle, sowie Versicherungen und Autowerkstätten verwaltet werden können.

Das System besteht aus einem Kernsystem, welches aus der Entity-Framework Schicht inklusive der Datenschicht und der eigentlichen Datenbank, sowie der Geschäftsschicht besteht (siehe Abbildung 0.1). Dieses Kernsystem stellt ein abstraktes Objektmodell sowie Methoden zur Verfügung, die es ermöglichen, Daten abzuspeichern, zu verändern und abzufragen. Es stellt auch sicher, dass alle Änderungen versioniert (protokolliert) werden.

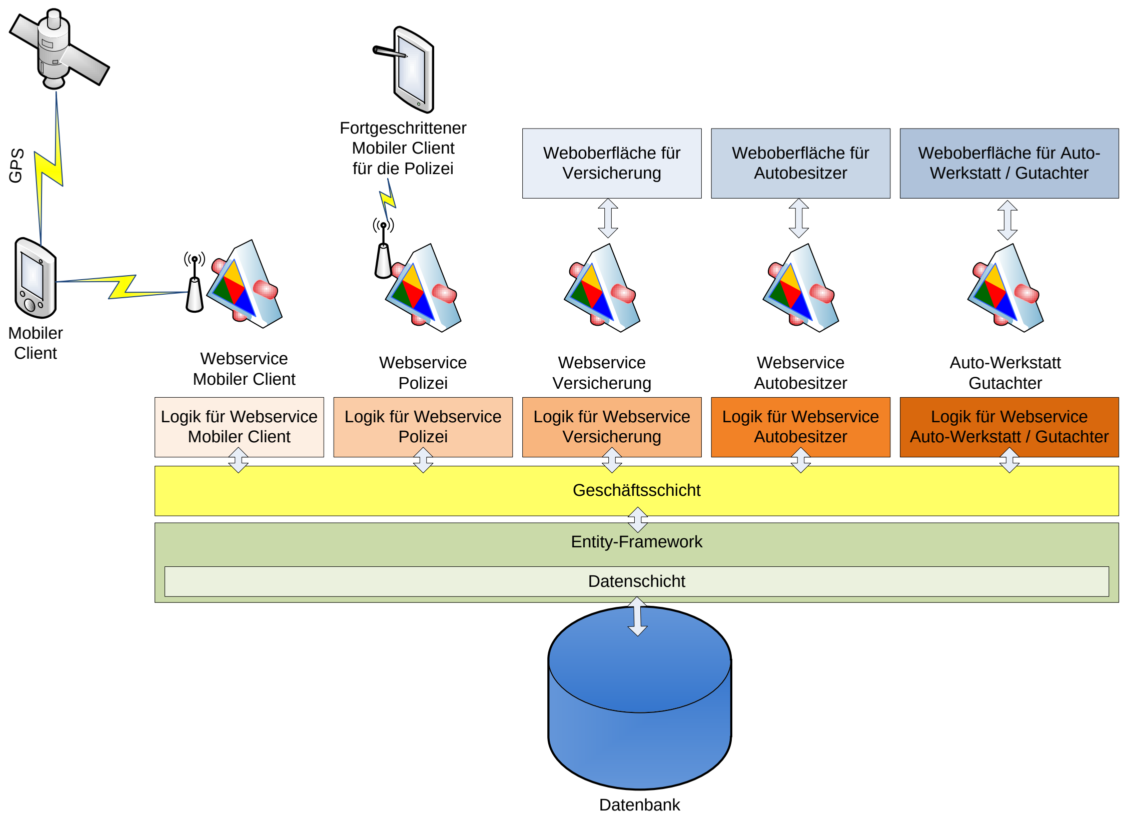

GPS
Mobiler Client
Fortgeschrittener Mobiler Client für die Polizei
Weboberfläche für Versicherung
Weboberfläche für Autobesitzer
Weboberfläche für Auto-Werkstatt / Gutachter
Webservice Mobiler Client
Webservice Polizei
Webservice Versicherung
Webservice Autobesitzer
Auto-Werkstatt Gutachter
Logik für Webservice Mobiler Client
Logik für Webservice Polizei
Logik für Webservice Versicherung
Logik für Webservice Autobesitzer
Logik für Webservice Auto-Werkstatt / Gutachter
Geschäftsschicht
Entity-Framework
Datenschicht
Datenbank

Abbildung 0.1: Aufbau des Systems

Daten werden in sogenannten Container-Objekten, welche Entities enthalten, abgespeichert. Container-Objekte haben jeweils einen Namen und fungieren als Behälter für die einzelnen Werte, die als Entity-Objekte abgespeichert werden. Entity-Objekte können jeweils als Parent weitere Entity Objekte enthalten und können daher eine Referenz zu alten, veränderten Daten enthalten (siehe Abbildung 0.2). Entity-Objekte können beliebige Daten abspeichern.

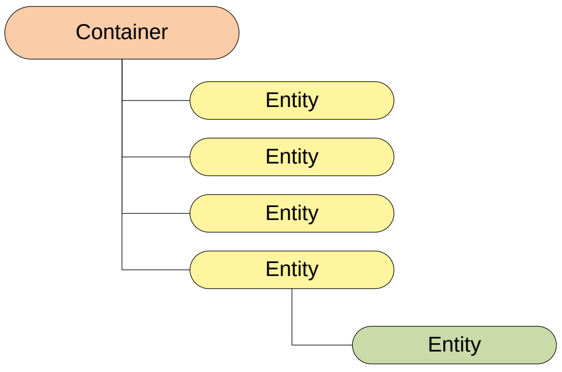


Abbildung 0.2: Container Entity Modell

UNIVERSITÄT KOBLENZ · LANDAU

EU-Accident-Report Back-Office

Bachelor Thesis

in order to obtain a bachelor degree
in the degree program Information Management

provided by

Alexander Rippert

Supervising Tutor:	Diplom-Informatiker Stefan Stein, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik
Reviewer:	Prof. Dr. J. Felix Hampe, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik

Koblenz, July 2010

Table of Contents

Contents

Table of Figures

Table of Listings

Tables

1. Introduction

The amount of cars on Germany's roads has been increasing during the last decades. There had been 41.183.594 cars according to the ADAC in Germany in 2008. There were 3.148.163 new car registrations in 2007. 3.566.122 motorbikes existed in 2008 [ADAC09]. In addition to cars and motorbikes there are also trucks on our roads.

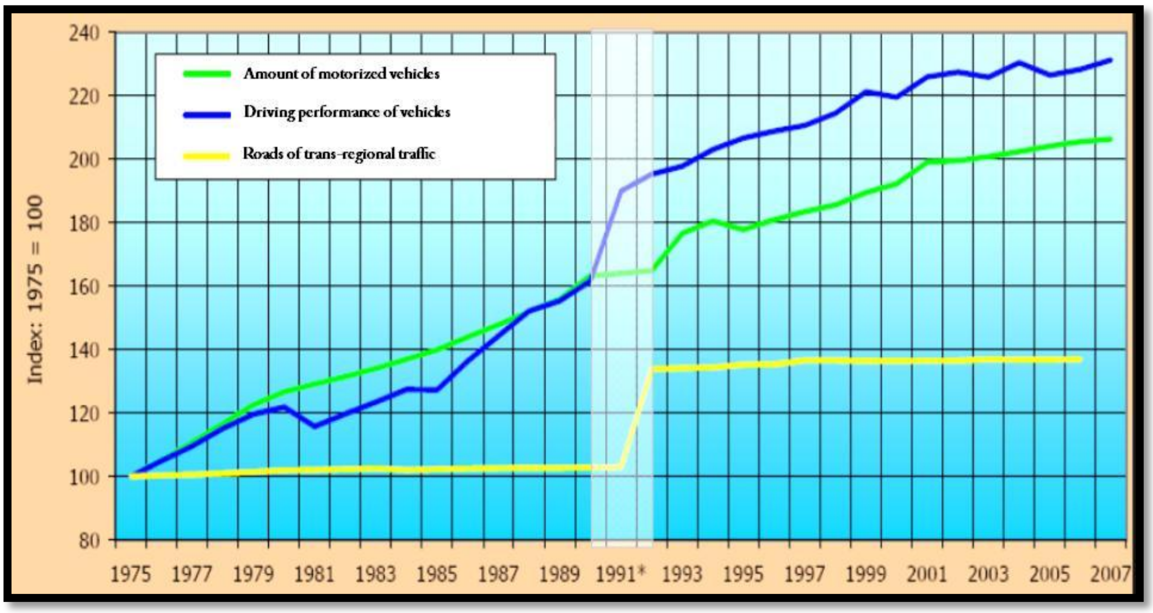


Figure 1.1: Development of the road network and load on roads in Germany [ADAC09]

Figure 1.1 shows that the amount of motorized vehicles is continually rising while there are not significantly more roads being build (the high rise of roads in 1990/1991 is due to the opening of the Berlin Wall and the Union of Western- and Eastern-Germany and does therefore not indicate a high rise of new roads being build). This means that there is an ever growing amount of vehicles on the roads and the gaps between vehicles is getting smaller.

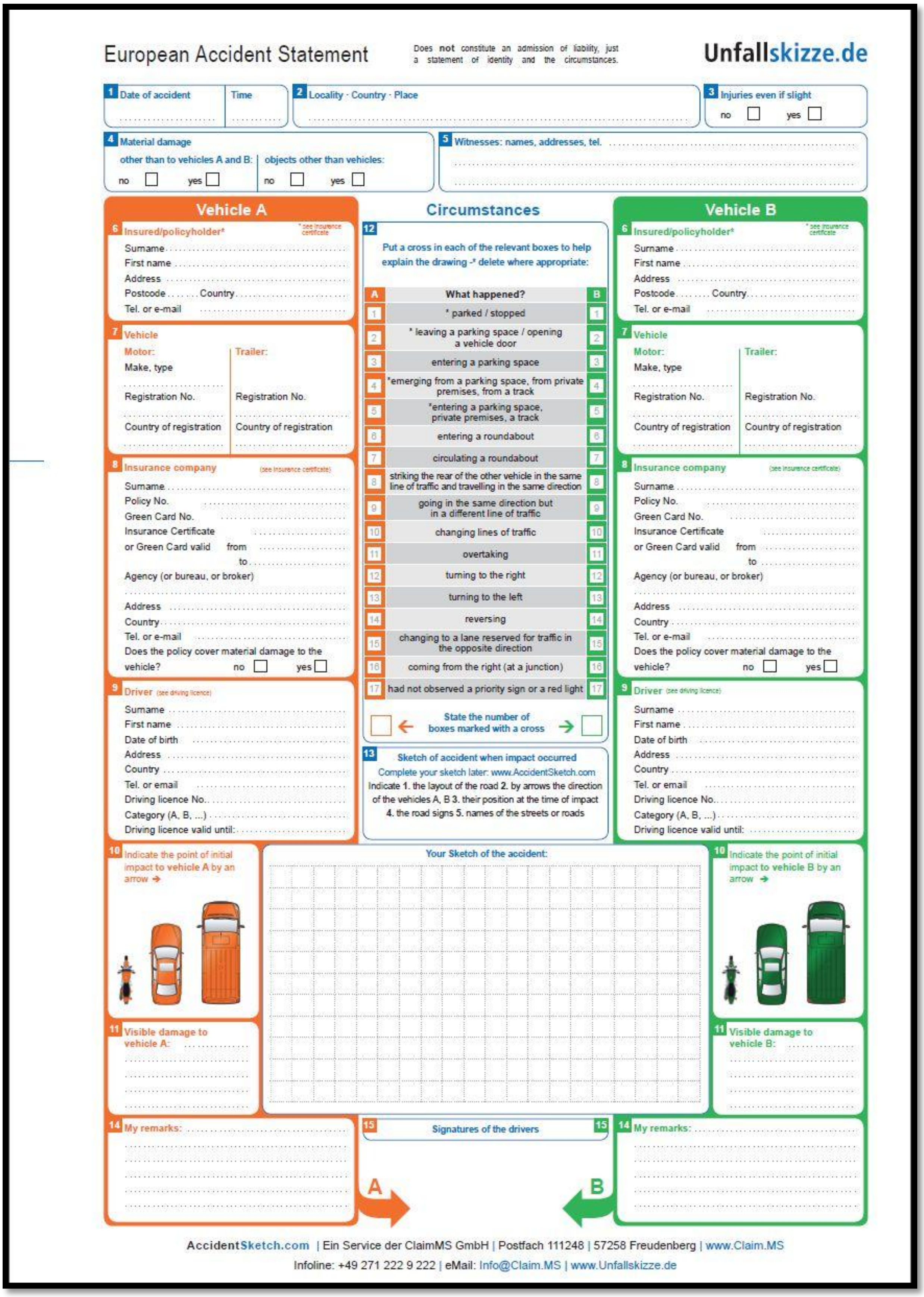
This inevitably leads to a higher risk of making an accident.

1.1 Motivation

If an accident happens, it should be as easy as possible to create a report about the accident. In a lot of cases it is not necessary to call the police. In those cases the people involved in the accident have to create the report by themselves and send it to their insurance company. The insurance company will then process the accident case. It is of interest that this process is as cheap as possible.

1.1.1 The European Accident Report

In order to make it easier to document an accident the European Accident Report has been created. This is a standardized form for recording an accident case. It contains fields, which the user can fill out, for all data about an accident that the insurance company needs in order to be able to process the case.
The form can be seen in figure 1.2.



European Accident Statement

Does **not** constitute an admission of liability, just a statement of identity and the circumstances.

Unfallskizze.de

1 Date of accident / Time	2 Locality · Country · Place	3 Injuries even if slight: no ☐ yes ☐
4 Material damage — other than to vehicles A and B: no ☐ yes ☐ — objects other than vehicles: no ☐ yes ☐	5 Witnesses: names, addresses, tel.	

Vehicle A

6 Insured/policyholder* (*see insurance certificate)
Surname
First name
Address
Postcode .... Country
Tel. or e-mail

7 Vehicle
Motor: Make, type; Registration No.; Country of registration
Trailer: Registration No.; Country of registration

8 Insurance company (see insurance certificate)
Surname
Policy No.
Green Card No.
Insurance Certificate or Green Card valid from .... to ....
Agency (or bureau, or broker)
Address
Country
Tel. or e-mail
Does the policy cover material damage to the vehicle? no ☐ yes ☐

9 Driver (see driving licence)
Surname
First name
Date of birth
Address
Country
Tel. or email
Driving licence No.
Category (A, B, ...)
Driving licence valid until:

10 Indicate the point of initial impact to vehicle A by an arrow ➔

11 Visible damage to vehicle A:

14 My remarks:

Circumstances

12 Put a cross in each of the relevant boxes to help explain the drawing -* delete where appropriate:

A	What happened?	B
1	* parked / stopped	1
2	* leaving a parking space / opening a vehicle door	2
3	entering a parking space	3
4	*emerging from a parking space, from private premises, from a track	4
5	*entering a parking space, private premises, a track	5
6	entering a roundabout	6
7	circulating a roundabout	7
8	striking the rear of the other vehicle in the same line of traffic and travelling in the same direction	8
9	going in the same direction but in a different line of traffic	9
10	changing lines of traffic	10
11	overtaking	11
12	turning to the right	12
13	turning to the left	13
14	reversing	14
15	changing to a lane reserved for traffic in the opposite direction	15
16	coming from the right (at a junction)	16
17	had not observed a priority sign or a red light	17
☐ ←	State the number of boxes marked with a cross	→ ☐

13 Sketch of accident when impact occurred
Complete your sketch later: www.AccidentSketch.com
Indicate 1. the layout of the road 2. by arrows the direction of the vehicles A, B 3. their position at the time of impact 4. the road signs 5. names of the streets or roads

Your Sketch of the accident:

15 Signatures of the drivers 15

A B

Vehicle B

6 Insured/policyholder* (*see insurance certificate)
Surname
First name
Address
Postcode .... Country
Tel. or e-mail

7 Vehicle
Motor: Make, type; Registration No.; Country of registration
Trailer: Registration No.; Country of registration

8 Insurance company (see insurance certificate)
Surname
Policy No.
Green Card No.
Insurance Certificate or Green Card valid from .... to ....
Agency (or bureau, or broker)
Address
Country
Tel. or e-mail
Does the policy cover material damage to the vehicle? no ☐ yes ☐

9 Driver (see driving licence)
Surname
First name
Date of birth
Address
Country
Tel. or email
Driving licence No.
Category (A, B, ...)
Driving licence valid until:

10 Indicate the point of initial impact to vehicle B by an arrow ➔

11 Visible damage to vehicle B:

14 My remarks:

AccidentSketch.com | Ein Service der ClaimMS GmbH | Postfach 111248 | 57258 Freudenberg | www.Claim.MS
Infoline: +49 271 222 9 222 | eMail: Info@Claim.MS | www.Unfallskizze.de

Figure 1.2: Paper form of the European Accident Report [Unfa09b]

It can be used in accidents in which two parties are involved. Both parties can write down what has happened and what damages exist. It also includes space to create a simple drawing of the accident. Using this standardized form ensures that no important data is missing in the report. It also helps to build trust because the form will be accepted by insurance companies. It is much easier to document an accident with this form than if a person needs to start with an empty page. Another advantage is that insurance companies can more easily work with the reports because they include all necessary data in a standardized form. This reduces costs.

The paper form of the European Accident Report has also got some drawbacks. Only about 1 in 15 people had the form in their car due to a survey by Nadine Gille [Gill08]. Not all people have a writing that is easily readable. This is especially true if people are nervous, which they are after an accident. This makes it difficult for the insurance company to read the form. It also leads to misunderstandings. In the paper form it is possible to make wrong statements. E.g. it is possible to answer yes and no at the same time to the question "Material Damage other than to vehicles A and B". It is also possible to make contrary statements on the left and right side of the form. The place for the sketch of the accident is very small and therefore it is difficult to draw a good sketch. Another problem is that not all people are good drawers and therefore sketches might not be easily readable. Damages cannot be documented in detail. It is only possible to show where damages are. The insurance company will later need a more detailed report to process the accident case.

Another big disadvantage of the paper based form is that the insurance companies need to manually insert the data from the forms into their computer systems. Some of those forms are already scanned and inserted into those computer programs via writing recognition, though, this only works if the writing is well readable. In a lot of cases insurance employees need to read and manually insert the data into their programs. This is a very costly process. Insurance companies have an interest in decreasing those costs.

1.1.2 The Electronic Version of the European Accident Report on a Mobile Client

Because of these disadvantages of the paper based European Accident Report an electronic version has been developed that runs on a mobile phone. This has a lot of advantages. The program guides the user through the form on a step-by-step basis, so that the he does not forget writing down important data. The user can get help in each step. This is important, because the user will probably not use the program very often. Making contrary statements is not possible. Therefore the user can no longer answer yes and no at the same time. Another big advantage is that the user can use GPS in order to figure out his location. The mobile client also helps the user create a sketch. He automatically gets a map of his current location. He can then insert the cars that were involved in the accident.

The user does not have to enter all data. E.g. his name, address and car information can be retrieved from a database. The insurance companies can insert this data when a new contract is made. This makes the process easier and helps avoiding errors.

A second client has been developed that runs on a tablet PC and can be used by the police to document an accident.

1.1.3 Server System

As mentioned above, the mobile clients have already been developed and it is therefore possible to create a report of an accident from a mobile phone. This does not only help the users by making documenting an accident easier, but also helps the insurance companies reducing costs, because they do no longer have to manually process paper-based reports. In order to create a paperless workflow it is necessary to have a server system that can receive and save the data from the accident reports that have been created on the mobile client. It should be possible to save, change and view data about accident cases via this server system. Such a system would eliminate the need to manually input the data from reports into a system and therefore help reduce costs.

The system should also provide a web interface for insurance company employees. Via this interface it is possible to see all data about an accident case. The mobile client provides an easy way to mark where car damages are on the car by showing an image of a car. The user can also make photos of the damages. This helps insurance companies understand how an accident has happened. Because the insurance employees can more easily reconstruct an accident case, they can also see which damages are not results of this accident case. This helps insurance companies identify insurance fraud.

The system could also provide a web interface for car owners. The car owner can then log in to this website and see all his accident reports and can also the current processing state of the accident reports. This reduces calls to the insurance companies from customers who want to know if their accident cases have already been processed.

Because of all those advantages such a server system is being implemented in this bachelor-thesis.

1.2 Bachelor-Thesis Project Task

The goal of this bachelor-thesis is to create a central system, which insurance companies can use to process their accident cases.

It should provide a web service interface that can receive the accident documents from the mobile client. It should also offer a user interface for insurance companies, which those companies can use to add data about their customer's cars, as well as process their accident cases.

The system should also have web services via which it is possible to access data about cars from the mobile client and from the police client.

The system is implemented as a generic framework that includes basic functionality to write all necessary data into the database and to read this data from the database. It also ensures that all data is versioned, that means that it is not possible to delete or change data without being able to precisely see who did those changes and when changes have been made. It therefore implements an object model that can be used to work with the data from the database. This object model provides certain functionality to save, read, update and delete data, while ensuring versioning.

On top of this basic framework are modules which provide certain functionality for different application areas, e.g. there is a module that implements the web service for the mobile client, a web service for the police, a web service for insurance companies and a web surface for insurance companies. Those modules use the object model and its functionality from the basic framework (core system) that lies below it to access data from the database and to write and update data to the database.

Due to this modular architecture it is easily possible to extend the system by adding more modules or to change the functionality of the system by modifying single modules. The developer who modifies a module only needs to know how this module works, how the core system works and what data he wants to use. He does not have to know how all the other modules work. Therefore it is easily possible to add new functionality to the system, as for example integrating 3D-models of the cars. It is not necessary to change the core system or the database when adding new modules, because the core system can store any kind of data. This avoids accidentally breaking functionality of the system.

The focus of this bachelor-thesis is to create a modular framework that can be used by other projects to add additional functionality by either modifying the existing modules or adding new modules.

The modules contained in this bachelor-thesis are meant to be examples of how modules can be created. Therefore they are very simple and do not provide a lot of functionality. Developers can use those modules as a starting point for developing their own modules that can be used by the system. It is also easily possible to replace an existing module with a new one, as long as the data used by the module is the same.

1.3 Overview

An overview of the European Accident Report as well as of the project task of this bachelor thesis has already been given.

Chapter 2 describes existing tools and prototypes that make it easier to report an accident case. This includes the mobile clients that make it possible to fill in the form on a mobile phone.

Chapter 3 talks about important privacy requirements that need to be considered.

Chapter 4 gives an overview of the architecture of the system.

Chapter 5 describes the technologies and tools used to build the system.

Chapter 6 is a detailed description of the system.
It consists of several chapters that all describe certain parts of the system:

Chapter 6.1 is about the core system and describes how new modules can be created on top of the core system. It gives detailed examples that show how the functionality of this framework can be used to enhance the system.

Chapter 6.2 gives an overview of the part of the system used by administrators of the system.

Chapter 6.3 describes the web service for the mobile client.

Chapter 6.4 is about the web service for the police.

Chapter 6.5 describes the module for the insurance companies.

Chapter 6.6 gives an overview of the module for the car owner.

Chapter 6.7 describes the module for the car repair companies.

Chapter 7 summarizes the results of this bachelor thesis and talks about possible future enhancements of the system.

2. Current Situation

Today people can use the paper based form of the European Accident Report to document an accident. This was a big step forward. It provided a standardized form that contains fields for all data needed by an insurance company to process a case. But the paper based version of the European Accident Report is not without problems (See chapter 1.1 for more information about the advantages and disadvantages of this form).

2.1 Sketch Tool on the web

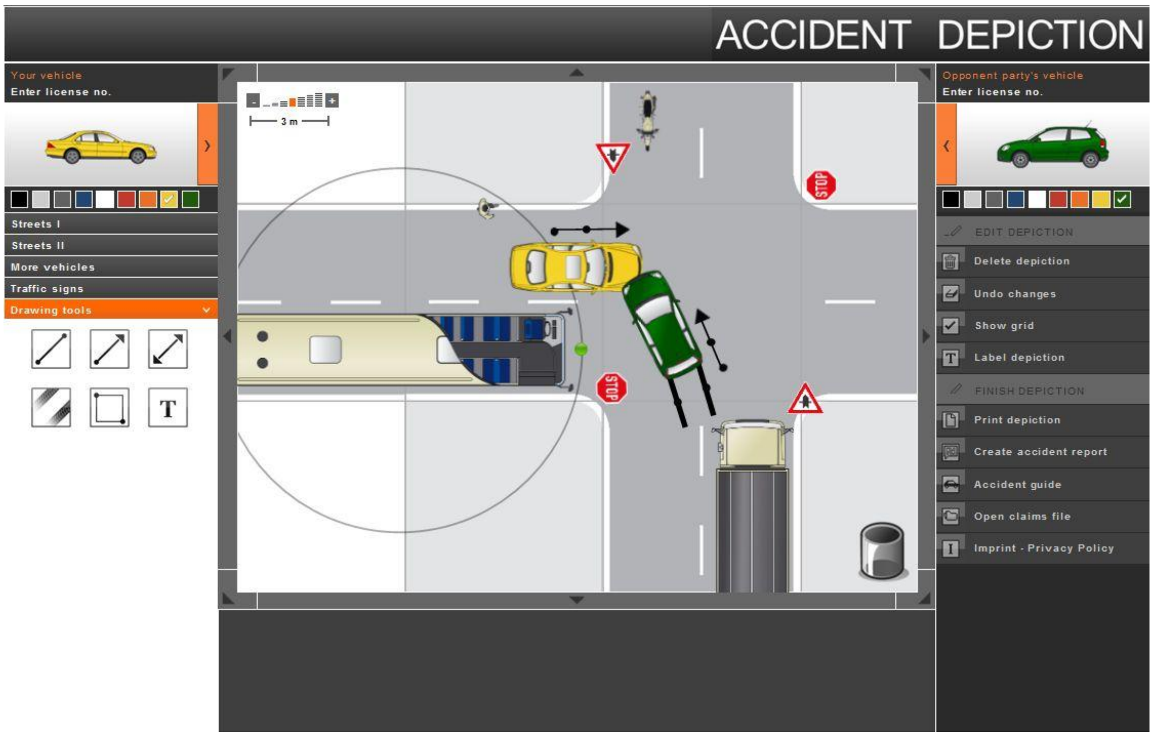


Figure 2.1: Online sketch tool that eases drawing of an accident sketch [Unfa09a]

Various projects exist that help the user fill out the form, for example an online tool which makes it easier for the user to draw sketches of an accident. See figure 2.1 for a screenshot of this application. The user can draw a road by choosing different road parts, add traffic signs, add the vehicles and persons involved in the accident and draw arrows to indicate the directions in which the vehicles were driving during the accident.

This tool helps insurance companies better understand how the accident happened. The tool is web based which makes it difficult to use it on a mobile phone, because the screen of this device is too small to display the tool. The tool can also only be used when the user is online. A person therefore either needs a notebook with internet access to draw the sketch, which is very unlikely. He can draw the sketch later, but then he has probably already forgotten some details of the accident at the time he creates the sketch.

2.2 Pre-filled forms of the European Accident Report

Another way of assisting the user is to use a partly filled out form. This is often the case when people get the form from their insurance companies. They fill out the customer’s name, address, the car’s information as well as the insurance company’s information.

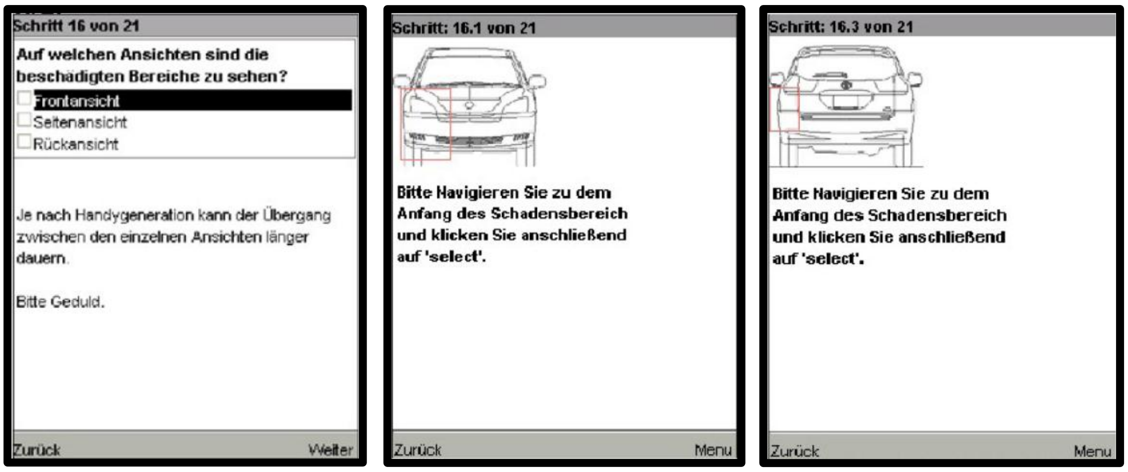
2.3 Processing the forms in the insurance companies

Today no technology exists which would make it possible to send the data from the European Accident Report to the user’s insurance company in a digital form. All forms are sent to the insurances in paper form. They can then be scanned and transformed into digital form via hand writing recognition or they need to be inserted into those systems in a manual way. If data is missing, the insurance company employee needs to either send the customer a mail or call him via telephone and request the missing data. Oftentimes the insurance company needs to wait for a detailed evaluation of the damages on a car. This analysis is also sent to the insurance company in an analog way. The insurance companies use their own software to process accident cases.

2.4 Mobile Client

2.4.1 First Prototype by Janek Klass and Tobias Knopp

Janek Klass and Tobias Knopp have developed a mobile client application at the University of Koblenz that makes it possible to document an accident on location [KlKn07]. It is implemented as a Java application that can be run on a mobile phone. The user can insert the required data into a step-by-step form. This ensures that all data is being inserted. It is also ensured that the user cannot make contradictory statements. For example the user can sometimes only select one option out of several options and therefore the risk of checking both “yes” and “no” at the same time is eliminated. Today most people carry a mobile phone with them. Therefore they would constantly have the form with them. This was not the case with the paper based form that a lot of people did not carry around with them. Another advantage of the system is that the data will be inserted in a digital form. This makes it easier to use the data in an insurance company, because the data does not have to be manually inserted into their system. Figure 2.2 shows screenshots of the mobile client. It can be seen how a person can create notes about damages on the car. They can select where the damage is.


Schritt 16 von 21

Auf welchen Ansichten sind die beschädigten Bereiche zu sehen?

Frontansicht

Seitenansicht

Rückansicht

Je nach Handygeneration kann der Übergang zwischen den einzelnen Ansichten länger dauern.

Bitte Geduld.

Zurück Weiter

Schritt: 16.1 von 21

Bitte Navigieren Sie zu dem Anfang des Schadensbereich und klicken Sie anschließend auf 'select'.

Zurück Menu

Schritt: 16.3 von 21

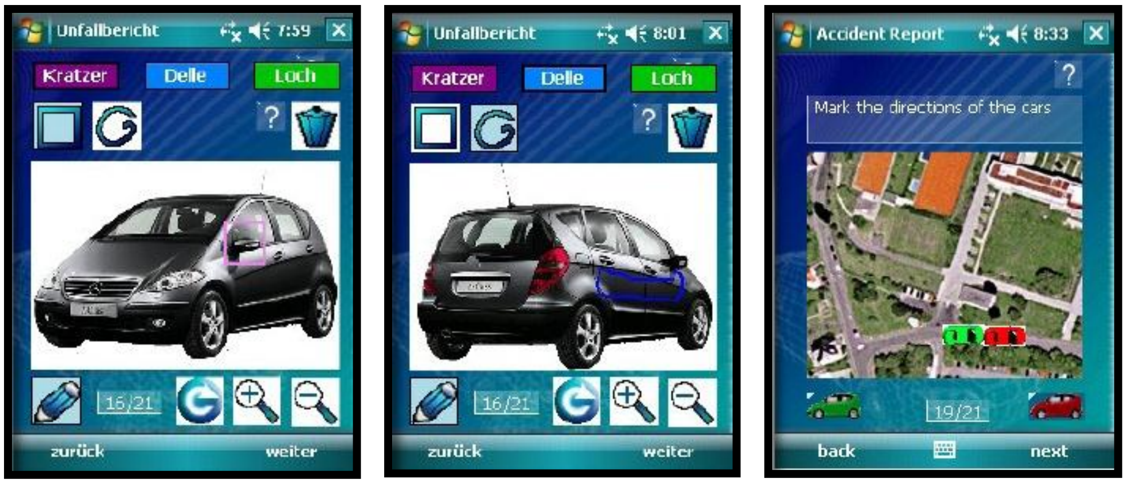
Bitte Navigieren Sie zu dem Anfang des Schadensbereich und klicken Sie anschließend auf 'select'.

Zurück Menu

Figure 2.2: The first prototype of a mobile client for the European Accident Report by Janek Klass and Tobias Knopp implemented as a Java application [KlKn07]

2.4.2 Second Prototype by Nadine Gille

A second client application has been developed by Nadine Gille [Gill08], who made it easier to use the application. The first prototype was only black and white and it was a very rudimentary application. The second prototype featured a modern user interface. The user can now ask for help in each step if he does not know how to use the application. The client application is now multi-lingual. As can be seen in Figure 2.3 the user can now see an image of a car and can use either a rectangle or a freehand pen to mark damages. He can also choose which kind of damage has occurred. The user can zoom into the car to make it easier to mark damages. He can also take pictures of the damages and add them to the report. The client application uses GPS to retrieve the current location of the person using the mobile phone. It then displays a map of this location onto which the user can place cars and indicate how the accident has happened. This replaces the traditional sketch.

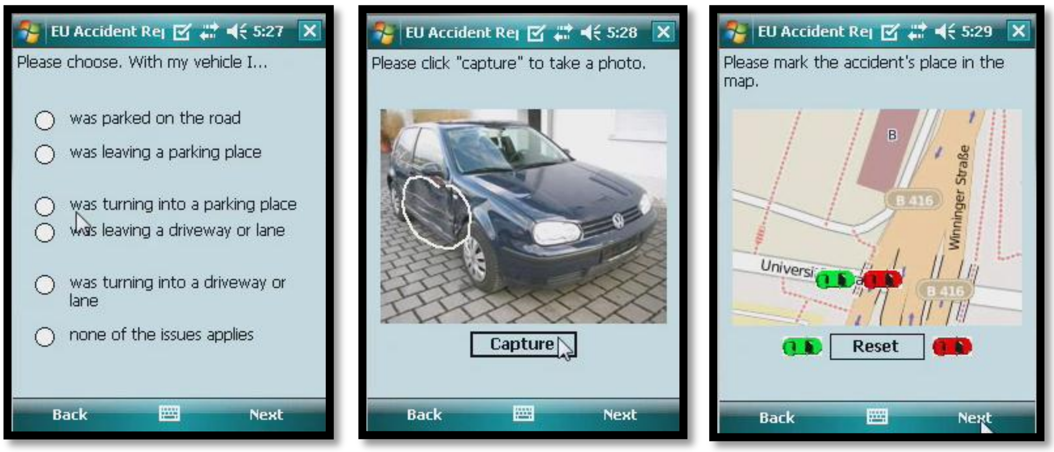


Mark the directions of the cars

Figure 2.3: Screenshots from the second prototype by Nadine Gille [Gill08]

2.4.3 Third Prototype by Stephan Arlt

A third prototype of the mobile client is being developed at the same time as this bachelor thesis is written [Arlt09b]. It will use the web service described in chapter 6.3 to retrieve data about drivers and cars and send the report to the back-office. This prototype uses a map in order to display the area where the accident happened instead of a satellite image, as used in the second prototype. This makes it easier for people to figure out where they are, because some things are not easily found on a satellite image (e.g. a road in a forest).



Please choose. With my vehicle I...

- was parked on the road
- was leaving a parking place
- was turning into a parking place
- was leaving a driveway or lane
- was turning into a driveway or lane
- none of the issues applies

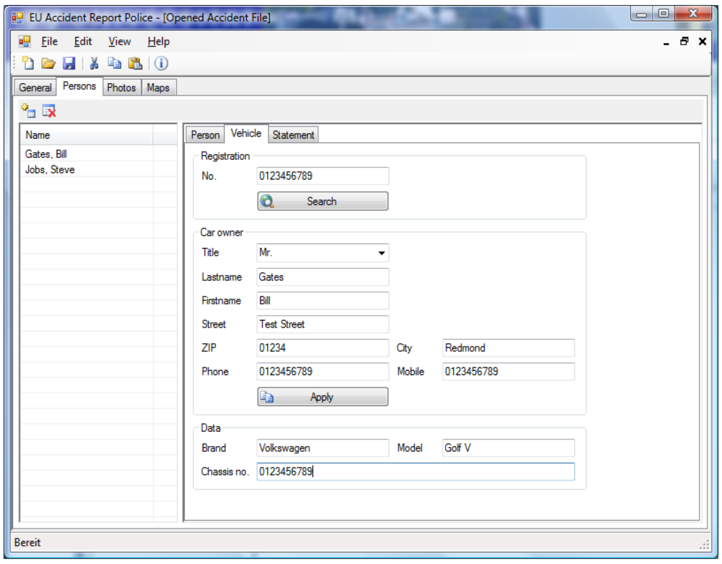
Please click "capture" to take a photo.

Please mark the accident's place in the map.

Figure 2.4: Screenshots from the third prototype by Stephan Arlt [Arlt09b]

2.4.4 Prototype for the Police by Stephan Arlt

In addition to the mobile client that can be used by car drivers, a second client that runs on a tablet pc has been created. This client can be used by the police to document an accident case [Arlt09a]. The police can insert all data needed to document an accident case into the application. The application can retrieve data about the drivers and the cars involved in the accident via the web service described in chapter 6.4. This reduces the amount of time needed to create an accident report. It also helps avoiding errors.


EU Accident Report Police - [Opened Accident File]
File Edit View Help
General Persons Photos Maps
Person Vehicle Statement
Search
Apply
Bereit
Name
Gates, Bill
Jobs, Steve

Registration

No.	0123456789

Car owner

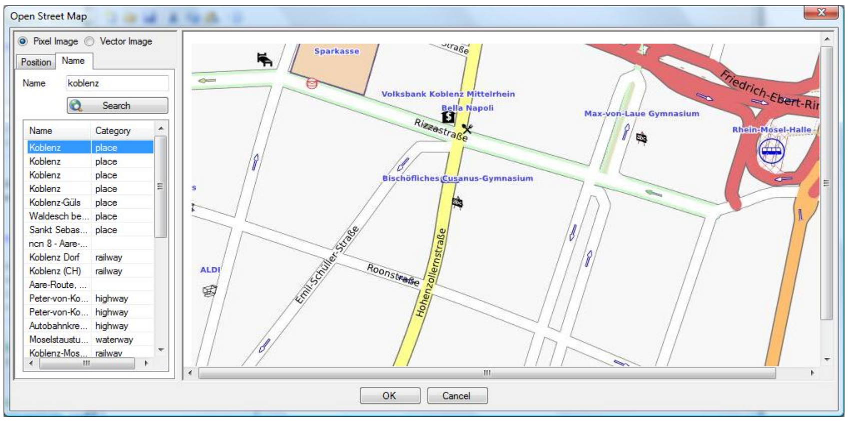
Title	Mr.		
Lastname	Gates		
Firstname	Bill		
Street	Test Street		
ZIP	01234	City	Redmond
Phone	0123456789	Mobile	0123456789

Data

Brand	Volkswagen	Model	Golf V
Chassis no.	0123456789		

Figure 2.5: Screenshot of the user interface of the police client: Adding a vehicle [Arlt09a]

The application is very flexible in order to make it possible to document complex accident cases. The user can add text and audio recordings to the report. This client is also being created at the same time as this bachelor thesis.



Name: koblenz

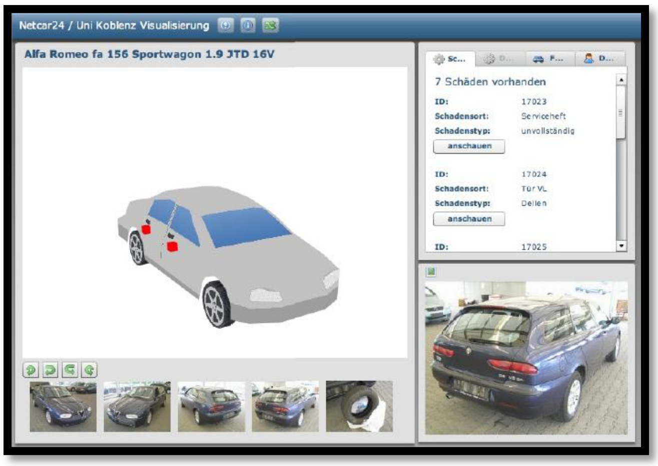
Name	Category
Koblenz	place
Koblenz	place
Koblenz	place
Koblenz	place
Koblenz-Güls	place
Waldesch be...	place
Sankt Sebas...	place
ncn 8 - Aare-...	
Koblenz Dorf	railway
Koblenz (CH)	railway
Aare-Route, ...	
Peter-von-Ko...	highway
Peter-von-Ko...	highway
Autobahnkre...	highway
Moselstaustu...	waterway
Koblenz-Mos...	railway

Figure 2.6: Screenshot of the user interface of the police client: Streets [Arlt09a]

These clients represent only the client side of the system. The task of this bachelor-thesis is to create a back-office system that can save the data created by the client applications and display them to an insurance company employee. The clients currently save the data as an xml-document locally on the mobile client.

2.5 3D-Damage Visualization by Thomas Lempa

Thomas Lempa has created an application that uses a 3D-model of a car to visualize information about its damages [Lemp08]. The 3D-model contains special markers on which the user can click in order to see more information about damages. It is also possible to see pictures of the damages on the car. This is a prototype that illustrates how a 3D-model could be used to ease to evaluation of a car's damages.


Netcar24 / Uni Koblenz Visualisierung
Alfa Romeo fa 156 Sportwagon 1.9 JTD 16V
Sc...
D...
F...
D...

7 Schäden vorhanden

ID: 17023
Schadensort: Serviceheft
Schadenstyp: unvollständig
anschauen

ID: 17024
Schadensort: Tür VL
Schadenstyp: Dellen
anschauen

ID: 17025

Figure 2.4: 3D-Visualization prototype for the European Accident Report and Netcar24 by Thomas Lempa [Lemp08]

Because this bachelor-thesis develops a base framework of the back-office and only implements a simple user interface for the insurance company, the 3D-model has not yet been implemented. This feature can be added in the future if needed.

3. Data Privacy

Insurance companies must adhere to German privacy laws. The use of personal data is therefore limited. The German privacy laws regulate, that someone may only collect the data that he needs to process a case. That means insurance companies may only save and use data, which they need in order to be able to process an accident case. They may not build and use a database containing data which they do not need for working on an accident case. If they collect and use more than the essential data needed to process a case they need the explicit written consent of the person for this data. This could be problematic, because this consent must be voluntarily. It therefore cannot be enforced for all insurance company customers. The privacy law also regulates that the data collected may only be used for the purpose that they had been collected for. They may not be used for other things. This means that e.g. an insurance company is not allowed to use their customer's address to send them advertisements about hotels, because the customers provided their address only for processing accident cases. This means that the system must ensure that the data is not used for purposes other than the ones that are needed by the insurance companies to process accident cases. [1]

This has several consequences for the system described in this bachelor-thesis. Only a minimum amount of data is collected from the customers including his name and contact information, which consists of his address and telephone numbers. The identity card number, passport and driver's license number are saved as well. The insurance companies also need to save the license plate number of the cars as well as information about the cars (brand, type and year of manufacture). This data is needed to be able to process an accident case. The identity card number, passport and driver's license number are only used by the police client. They cannot be seen by the insurance company. The system does not collect additional data (e.g. birthday, job, income) that is not needed by insurance company employees.

Data about an accident case can only be accessed by the insurance company of the car owner who was involved in the accident. Other insurance companies cannot see this data.

Only car repair companies who are chosen by the insurance company or the car owner can see the owner's data and evaluate damage.

No car owner can see other car owners' data.

Administrators of the central organization cannot access accident case data. They can only see the user's personal data, because they are responsible for maintaining this data.

The whole system is built in a way that ensures that only people who have got the right to see data can access this data. It is not possible to see private data of other people.

[1]Read more about Data Privacy in Germany at [Date09]

4. Architecture

A back-office for the Electronic Version of the European Accident Report has several requirements that need to be considered. First it must be possible to access all relevant data needed to process an accident case. Because not all car-owners have the same insurance, it is possible that a car accident case needs to be handled by more than one insurance company. Therefore it must be possible to access this data from all insurance companies of users affected by the accident. To avoid insurance fraud it may also be necessary to look up the accident history of their insurants.

Second all changes made to data in the system must be logged. This assures that no person can change any data to manipulate an accident case. Changes might be possible out of several reasons, like for example when the car owner or car driver made wrong entries when using the mobile client due to a shock he had after the accident. It is also possible that an insurance employee makes a mistake when processing a case and needs to correct that mistake. Third most insurance companies already have an existing software infrastructure to process their accident cases. It should be possible to integrate the back-office of the Electronic Version of the European Accident Report into those systems and therefore possible to enable each insurance company to create an integrated workflow.

Third data privacy must be ensured. Each entity in the system must only have access to data relevant to solve a case. It must be prevented that a person can access all data at all times. For example an insurance company may not access personal data like an address of a person from another insurance company that is not involved into an accident processed by that company.

There are two basic variants of such a system:

4.1 Peer-to-Peer Architecture

The first is a peer-to-peer-like architecture. This means that each insurance company would implement their own system. All systems would need to communicate with all other systems of all insurance companies through a standardized protocol (see figure 4.1). This has several advantages. Data is only saved in the insurance company that processes an accident case. This ensures that other companies cannot access this data. Access to data is only granted when needed. Access can be granted for each case. This way it is easy to control which data can be read by which people. Another advantage is that, because the system will be implemented by each insurance company, each company can perfectly integrate the system into their existing software infrastructure. This makes it easier to comply with company policies, like for example where data may be saved. This way each insurance company is *in control* of their system.

There are also some disadvantages. Each company has its own unique system. Therefore a lot of different systems are used, written in different programming languages and running on different operating systems. This can make it difficult to connect all those systems with each other. By sending data from one system to another system, data might get lost due to encoding limitations, e.g. when converting a 64 bit integer to a 32 bit integer. Another problem is that duplicate data will be saved, because the data needs to be saved in each insurance company that needs to process an accident case. It is difficult to ensure that all this data will always stay up to date.

It is also difficult to handle people, who are car drivers but not car owners, because they do not appear in the insurance's database. In many cases the car driver is not the car owner, e.g. in a car lending company. It can also be difficult to maintain such a system. Each time a new insurance company is added to the system, all companies must update their system to connect to the new insurance company.

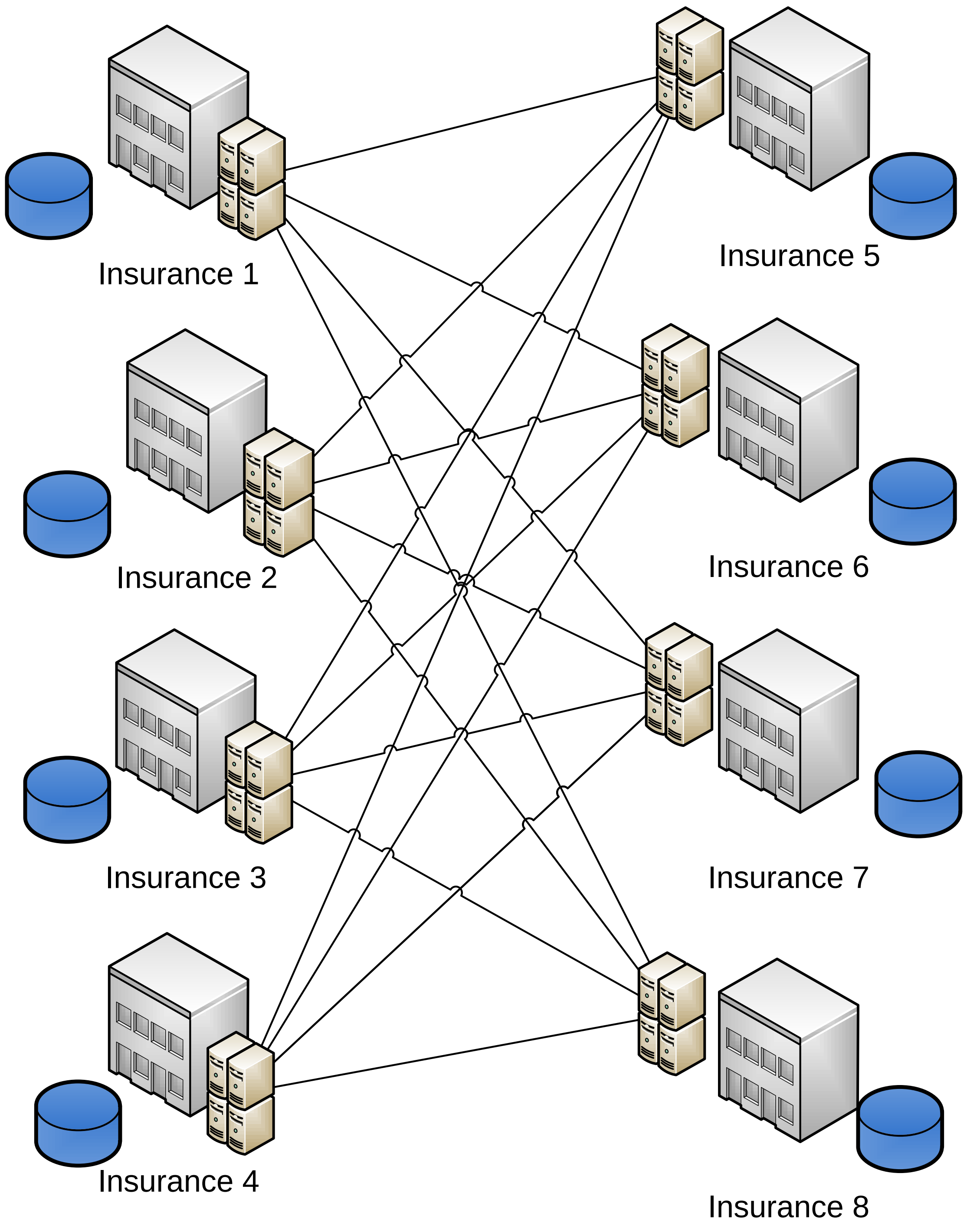


Figure 4.1: A Peer-To-Peer-like Architecture

4.2 Repository-Style Architecture

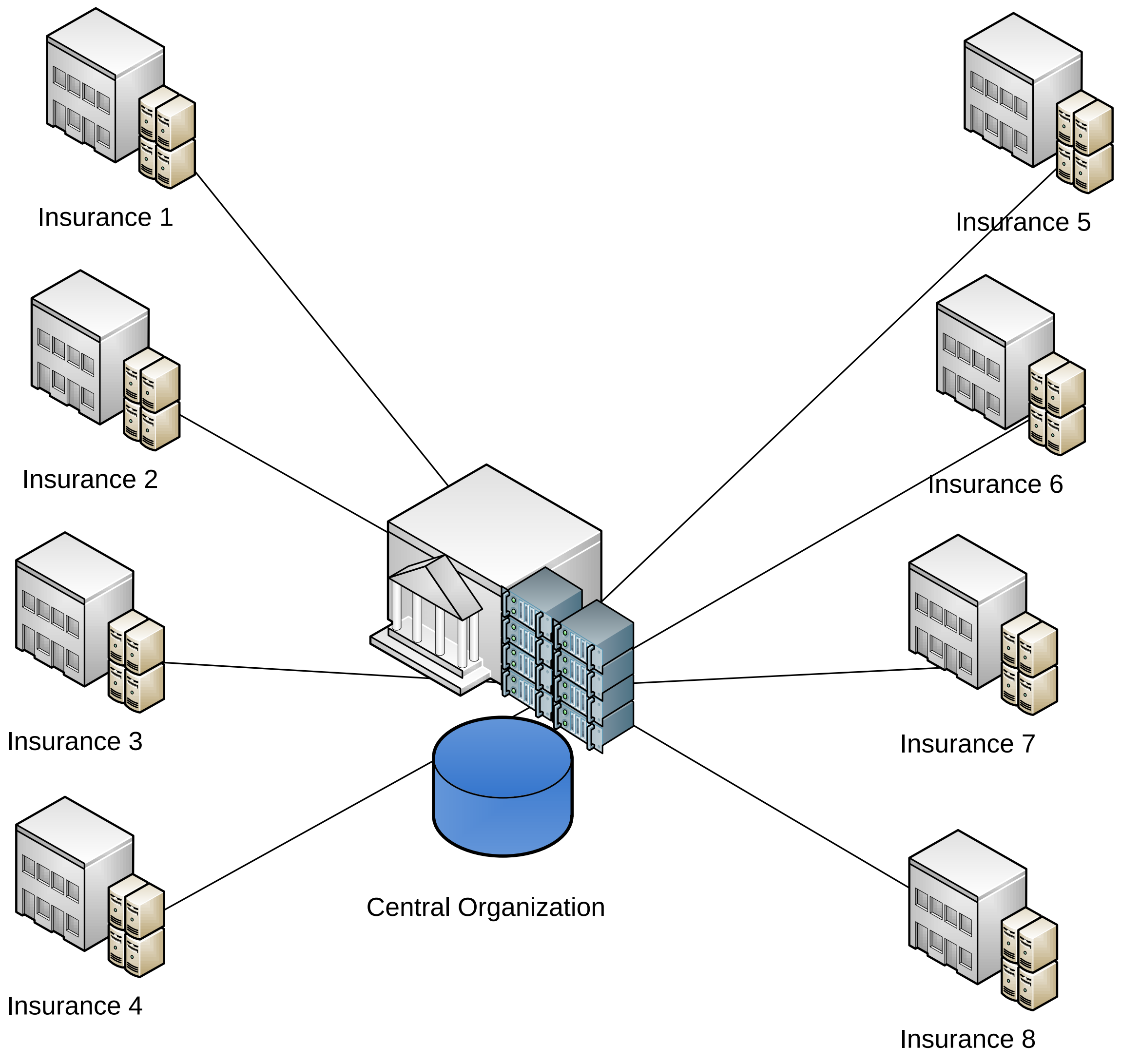


Figure 4.2: Repository-Style Architecture

The second version is a repository-style architecture. This means that there is a central database and a central software system that is being used by all entities (see figure 4.2). This implies that there is a separate organization which is responsible for maintaining the database. This organization can add and manage users. It can also add and manage insurance companies and other organizations that need to access the data saved in the database. All the organizations can access the data through a standardized protocol or through a web interface. Data access can be controlled through access rules. This way only organizations with proper rights can access certain data and data protection can be ensured. It is easy to administer and manage the system because all changes will only have to be made to a centralized system. Another advantage is that there are contact persons in the organization who can help insurance companies when they have problems related to the system.

This version has also got some disadvantages. There must be a centralized organization which is responsible to manage and maintain the system. It might be difficult to create such an organization.

All data will be saved in one big database system and therefore it is volatile to misuse because an attacker only needs to break into one system to get access to all data. If the data were saved in several systems an attacker would have to get access to all systems in order to get the data. It is also not as easy to integrate the system into the existing software systems of the software companies, because there would be one centralized workflow and one standardized data model.

The back-office of the Electronic Version of the European Accident Report is implemented in the second, repository-style architecture. This eases management and maintenance of the system. Another advantage is that it is possible to add people to the system who do not have a car insurance, like for example car drivers that drive cars owned by other people, e.g. parents or car lending companies. It also makes it easier to create a standardized data model that all companies will use. For example all car models can be inserted into this centralized database, therefore it is not possible that two insurance companies process the same accident case but get problems because the car model saved in their database is not the same, because they have two different data models for saving the car model. This also means that there is only one version of the mobile client software that will be used by all insurance companies. This client software will access the centralized back-office system. If the system was created in a peer-to-peer-style architecture, there would either have to be a standardized protocol via which the mobile client could access all the systems of all insurance companies related to the car accident or each insurance company would have to create their own software, which would make it difficult to use another person's mobile device to record an accident. It is also possible to access the data via a web interface without having to implement a software client for each insurance company. Another advantage is that the software needs to be implemented only once. Each insurance company can use the system. They can either access the data via standardized protocols and integrate the system into their existing software infrastructure or use the web interface to access the data. This greatly reduces costs because insurance companies do not have to implement their own systems. It would also be possible to use the centralized system to create detailed anonymous statistics of car accidents, because all data is saved in one database in opposition to saving all data inside of separate databases in insurance companies, where the data cannot be accessed for statistical purposes. This can be interesting for insurance companies as well as for research institutions. History has shown that it is possible to create a centralized organization that can manage the system. In Germany the Schufa [Schu09] was founded in order to record data about people's debt and to avoid fraud. This made it more secure for stores and banks to give a person a credit. The back-office of the Electronic Version of the European Accident Report could also help to protect the insurance companies from insurance fraud. Therefore they might be willing to pay some money for the service.

4.3 Structure of the System

The software-implementation of the back-office of the Electronic Version of the European Accident Report is built on top of a Microsoft SQL-Server 2008 database instance, in which all data about accident cases as well as all user data is stored in.

The ADO.NET Entity-Framework layer [Lerm09] is located on top of the database. The Entity-Framework creates an object model from the data in the database. These objects can be used to work with all the data used in the system. The Entity-Framework also enables CRUD-operations (Create, Read, Update and Delete) that make it possible to save, read, update and delete data to and from the database (for more Information about the Entity-Framework read section 5.2.3). The entity-framework layer acts as an access layer to the database. It therefore includes the data-access-layer.

On top of the entity-framework layer is the business layer (see figure 4.3). In this layer all the logic needed by the system is implemented. It acts as a further abstraction of the object model found in the entity-framework layer. Additional functionality is added in this layer that ensures that only users with appropriate privileges can access certain data and that no data can be directly changed or deleted. Changes to data happen in an indirect way via a functionality that archives all changes and therefore ensures that all changes can be reviewed in the future.

The entity-framework layer and the business layer form the core of the system. This core is a platform that provides all functionality for working with the data in the system. All other parts of the system are on top of this platform and use the functionality provided by the core. The core provides a user management system as well as functionality for data access. Encapsulating all functionality needed to work with the data in the system eases enhancing the system in the future. Those enhancements can be made via modules. A module is a certain set of functionality that is targeted toward a certain group of users, e.g. car owners. Each module is independent of all other modules in the system. This improves maintainability of the system, because the developers who make changes to a module only need to know how the module they want to change works and how to use the core. They do not need to know how all the other modules are implemented. Using the core is easy, because it can be used via APIs and therefore it is not necessary that the developer knows how the core is implemented. Changing a module cannot break other modules, because a module is only dependent on the core and not on other modules. Data can be shared between modules via Containers and Entities.

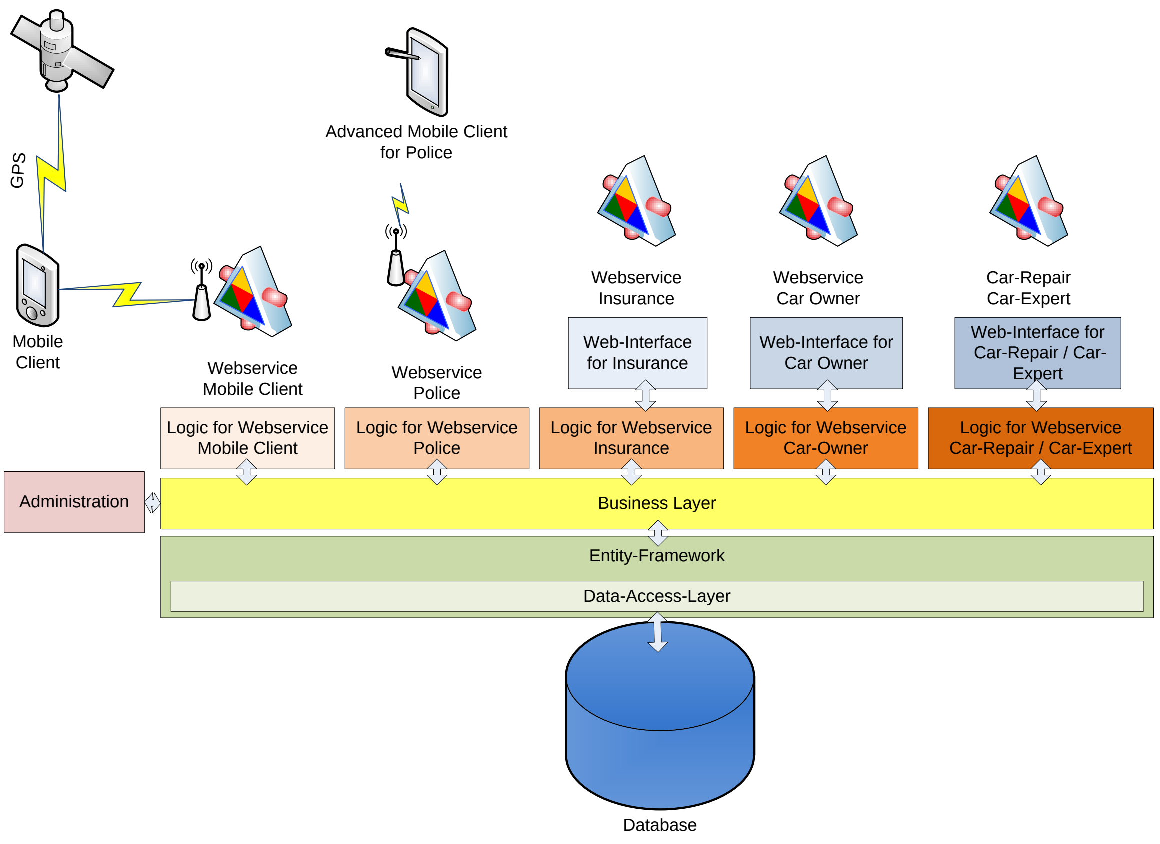
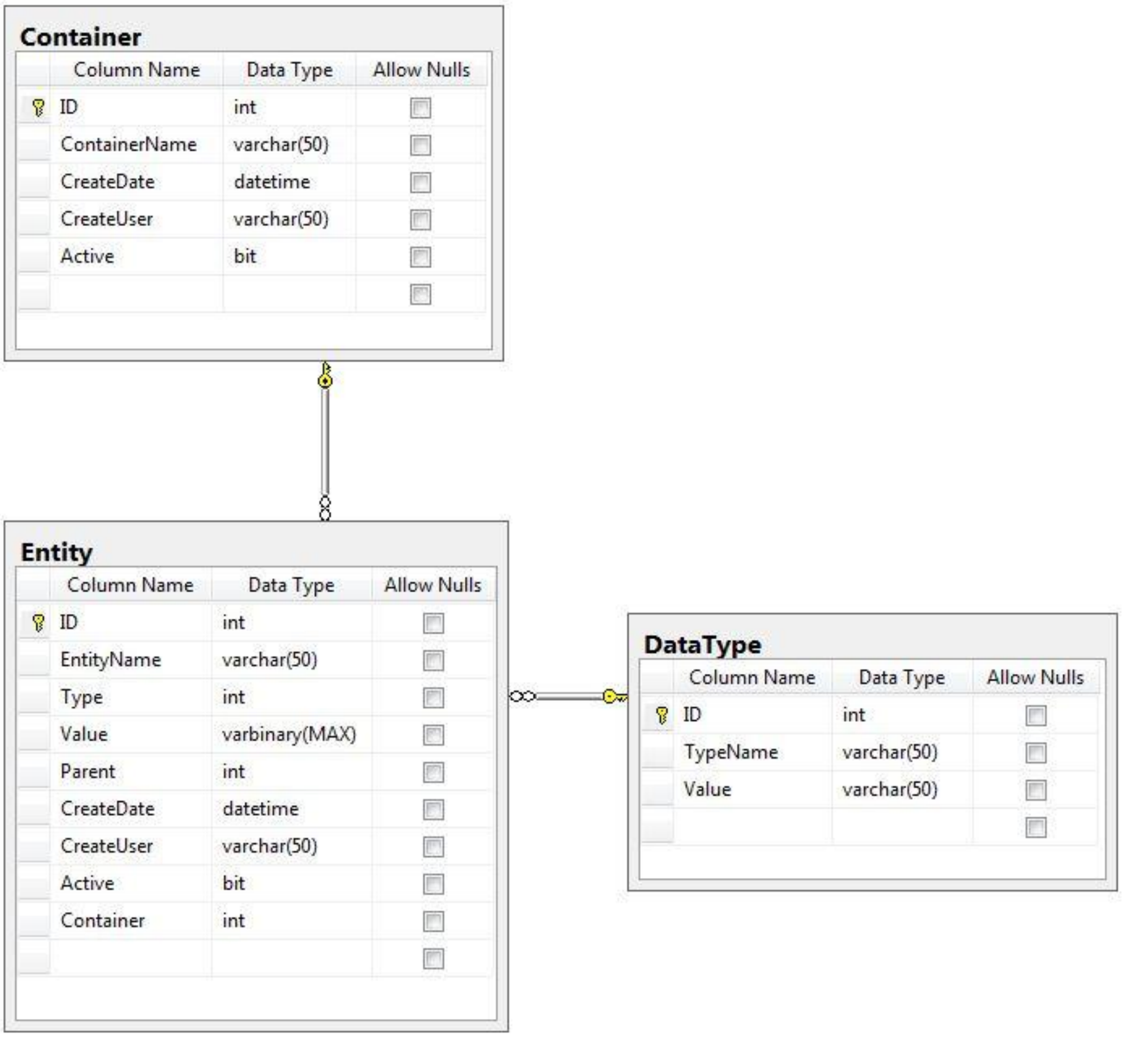

GPS
Advanced Mobile Client for Police
Mobile Client
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 4.3: Structure of the system

4.4 Structure of the Database

The database of the back-office of the Electronic Version of the European Accident Report consists of three tables. An Entity stores a simple atomic value. It has a name (EntityName) and an ID, through which it can be accessed. An Entity is of a certain type (e.g. Integer). This type is represented by the DataType table, which saves the type's name (TypeName) and its value. The value represents a .NET-Framework type. A module can use this value in order to convert the data saved in the Value property of the Entity into a .NET-Framework type (e.g. System.Int32). The data is saved as a varbinary in the database and can therefore save all kinds of data. The module accessing this data is responsible for converting the data into an appropriate data type so that it can use the data. The Parent property is used for versioning an Entity. If an Entity is changed it is not deleted in the database, but it is set to inactive (via the Active property) and the Parent property saves a link to this old Entity. Therefore it is always possible to access older versions of an Entity. If an Entity is deleted its Active property is set to false. No data is completely removed from the database.

The Container stores several Entities and therefore is similar to a table in a database. The Entities are like the columns of that table. The tables are only accessed via the core system. All modules use the core system to access the data in the database. No module directly uses these tables.



Container

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	ContainerName	varchar(50)	☐
	CreateDate	datetime	☐
	CreateUser	varchar(50)	☐
	Active	bit	☐
			☐

Entity

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	EntityName	varchar(50)	☐
	Type	int	☐
	Value	varbinary(MAX)	☐
	Parent	int	☐
	CreateDate	datetime	☐
	CreateUser	varchar(50)	☐
	Active	bit	☐
	Container	int	☐
			☐

DataType

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	TypeName	varchar(50)	☐
	Value	varchar(50)	☐
			☐

Figure 4.4: Database Schema

5. Technology

The server infrastructure for the back-office of the Electronic Version of the European Accident Report is using Microsoft technologies. All services run on the Internet Information Services (IIS) on a Windows XP computer. The system can easily be transferred to a Windows 2003 or Windows 2008 Server if needed.

5.1 Microsoft SQL Server 2008

All data associated with the European Accident Report is saved in Microsoft SQL Server 2008. This is a relational database management system. It consists of the Database Engine Services which are responsible for saving the data and retrieving it. It is possible to query data by using Structured Query Language (SQL) or by using Common Language Runtime (CLR) Languages, for example C#. SQL Server 2008 also includes services for creating reports (Reporting Services, Analysis Services, Data Mining), data transformation (Integration Services), synchronization (Sync Framework) and Messaging (Service Broker). In this bachelor thesis Visual Studio 2008 Team Suite and Microsoft SQL 2008 Server Management Studio were used to create the tables for the system.[2]

5.2 Microsoft .NET-Framework 3.5 SP1

The .NET-Framework is a programming environment developed by Microsoft. It consists of the CLR (Common Language Runtime) which is responsible for the execution and compilation of the source code. Similar to Java the CLR provides an intermediate language (IL) which is executed at runtime. A lot of different languages exist which can be used in the .NET-Framework, for example C#, Visual Basic.NET, C++. All of those languages are compiled into the intermediate language and executed at runtime. The .NET-Framework also provides a rich library consisting of thousands of classes and methods that help the programmer achieving his goals.[3]

The following diagram gives an overview of the .NET Framework:

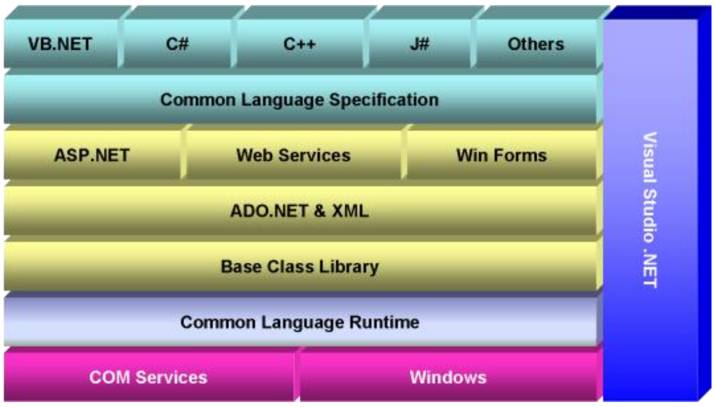


Figure 5.1: .NET-Framework [Trai09]

[2] To learn more about Microsoft SQL Server 2008 read: [BKSK06], [BSWK06], [DoKo03], [DrRa06], [BenG09], [Micr09a], [Stan09]

[3] To learn more about the .NET Framework read: [LoSt02], [Nort09], [Plat04], [Rich06], [Schw05]

The .NET-Framework 3.0 and 3.5 are enhancements of the 2nd Version of the Framework. The subsequent graphic displays the different parts of the .NET-Framework Version 3.5:

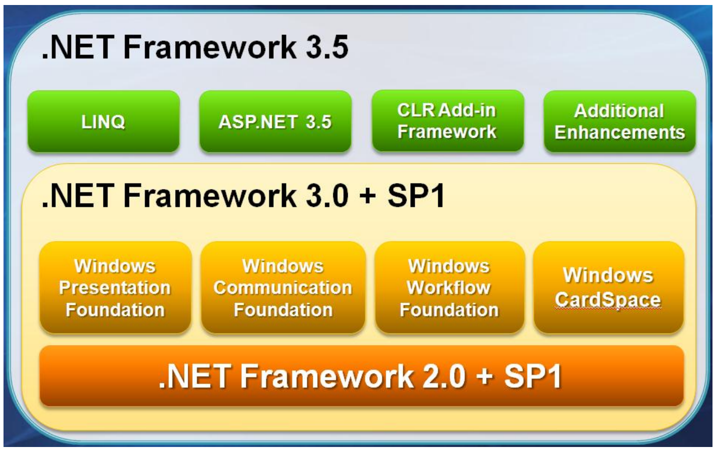


Figure 5.2: Overview of the .NET-Framework 3.5 [CONR07]

5.2.1 ASP.NET 3.5

ASP.NET is the successor of ASP (Active Server Pages). ASP.NET is a server side programming environment for creating interactive internet applications. It is part of the .NET-Framework and leverages its capabilities. Programming is possible with languages that run on the CLR (Common Language Runtime). In this bachelor-thesis C# is used for programming all ASP.NET pages.

ASP.NET uses xml-markup, which will be parsed at runtime and converted into HTML (Hypertext Markup Language) to create a user interface. Though it is possible to combine markup and code in one file, it is recommended to use code behind files for storing all code. Therefore code will be saved in separate files. This makes it possible to separate functionality from representation, which makes it easier to maintain the application later, because it is possible to change the design of the application, e.g. by a designer, without touching the functionality.

The functionality from the code behind files is saved into dlls (dynamic link libraries) at runtime out of performance reasons. ASP.NET also provides rich functionality concerning user and membership management and security. The membership-services are used by all server side applications in the electronic version of the European Accident Report to authenticate and authorize users. Access permissions to pages and directories can be configured in the web.config of the application. This configuration can be made per user or it can be role-based. The membership-services, too, handle state and session management. ASP.NET also includes functionality for creating a navigation system for the website. The hierarchy of the website is saved in an xml-file. It is possible to automatically create menus, a bread-crump navigation element and a sitemap from this file. When adding or deleting pages, only the xml file needs to be changed, all other pages do not need to be touched. This makes it easy to add, change or delete pages.[4]

[4] To learn more about ASP.NET read: [SSVG05], [Espo08], [LoMü03], [Prei05], [ASPN09],

5.2.3 ADO.NET Entity-Framework

The Entity Framework is a data access and modeling technology. It is built on top of ADO.NET and uses LINQ (Language Integrated Query) to access data from a data source. Enterprise applications are often implemented as a three-tier architecture. The lowest of those tiers is usually a data access layer that is responsible for accessing data from a certain data source and providing functionality for accessing this information through methods. The middle tier is the business logic layer that contains objects representing the data from the data source and methods that implement the functionality of the system. The third tier is the presentation layer that is responsible for displaying the data via a user interface. The ADO.NET Entity Framework makes it easier to create the business logic layer and access the data from the data source, which can for example be a database. In our case this is a Microsoft SQL 2008 Server. It is possible to create a data model for the business logic layer and a mapping via an xml file that maps the fields from the database to the fields in the objects. The Entity Framework than figures out how to access the data from the data source and enables CRUD (Create, Read, Update, Delete) operations. Therefore a developer does not have to implement a data access layer and can create his application in an object oriented manner by using the objects found in the business logic layer. LINQ to Entity is a technology that provides an easy way to query data. With LINQ to Entity it is possible to use SQL style syntax, like for example select, from, where statements, inside of a C# program. In contrast to SQL statements, which in C# are handled as strings, LINQ to Entity commands can be validated by the compiler which reduces programming errors. It also represents an easy and object oriented way to access and work with data.[5]

5.2.4 Windows Communication Foundation (WCF)

The Windows Communication Foundation is a programming framework for creating distributed systems. It provides a common environment for different kind of communication technologies like for example xml web services, TCP/IP, Microsoft Message Queuing (MSMQ) and named pipes. WCF is also extensible, that means it is possible to add new messaging technologies to the framework. The three main components of a WCF application are an address, a binding and a contract. The address represents the location where the service is saved. The binding defines how an application processes, sends, and receives messages. This could for example be through xml web services. The contract defines the endpoints in a receiving application. This means it determines which parts in the message will be saved in which fields in the respective .NET CLR type. WCF with xml web services has been used in the back-office of the electronic version of the European Accident Report for all messaging between different parts of the application. [6]

[5] To learn more about ADO.NET Entity-Framework read: [MSDN09], [Lerm09]

[6] To learn more about the Windows Communication Foundation read: [Gail04], [Plat04], [Smit07]

5.3 Tools

Several tools have been used to create the back office and the front ends of the electronic version of the European Accident Report. The following section gives an overview of those applications:

5.3.1 Microsoft Visual Studio 2008

Visual Studio is an integrated development environment from Microsoft. It is available in several Versions: Visual Studio 2008 Express, a free light weight version for beginners and hobbyists; Visual Studio 2008 Standard; Visual Studio 2008 Professional, which also contains some advanced features as well as some more wizards as well as a tool for creating and using unit tests; Visual Studio 2008 Team Developer, an edition that includes team work features; Visual Studio 2008 Team Test, a version for testing engineers; Visual Studio 2008 Database, a version for database experts; Visual Studio 2008 Architect, a version for Software Architects and Visual Studio 2008 Team Suite, a comprehensive version containing all the features of all the other versions. Visual Studio can be used to program .NET-applications in C#, Visual Basic and C++. It is also possible to create native C++ applications with Visual Studio. Through the use of plug-ins other programming languages like the functional programming language F# can also be used. Visual Studio can be used to develop for different target platforms through various project types like for example dynamic websites (ASP.NET, Silverlight), windows desktop applications (Windows Forms, WPF), internet communication (asmx-web services, WCF, .NET-Remoting), Windows services, plug-ins, Microsoft Office applications, database applications.[7]

All functionality of the electronic version of the European Accident Report has been created with Visual Studio 2008 Team Suite.

5.3.2 Microsoft Expression Web 2

Expression Web is a WYSIWYG (What You See Is What You Get) tool for creating websites. It contains graphical tools for creating web pages as well as text editors with intelli sense for HTML, CSS, PHP, JavaScript as well as ASP.NET. It also provides functions which help to publish a website on a server. Expression Web was used for designing the markup ASP.NET pages. The functionality in the code behind files where created with Visual Studio 2008 Team Suite.[8]

[7] To learn more about Microsoft Visual Studio read: [Hund06], [Micr09b]
[8] To learn more about Microsoft Expression Web read: [Micr09c]

6. Prototype Implementation

This chapter describes the prototype implementation of the back-office of the electronic version of the European Accident Report. It starts by describing the core system, which can be used by all modules of the system. It is also shown how a developer can program new modules by using the container-entity-model provided by the core system. The chapter then describes how the administration interface, the module for the insurance company, the car-owner and the car-repair company has been created. It also gives an overview of the web service for the mobile client and the police.

6.1 The Core System

The core system is the basement for all the modules used in the system. It was created in a way that makes it easy to enhance the system by adding new modules or changing exiting modules that sit on top of the core system. All modules use the core system to read, write and update data from the database. No module accesses the database in a direct way. They all use the API of the core system to work with data in the database. Therefore the core system is in full control over all data that is being used by the system.

The following diagram shows the structure of the Core System:

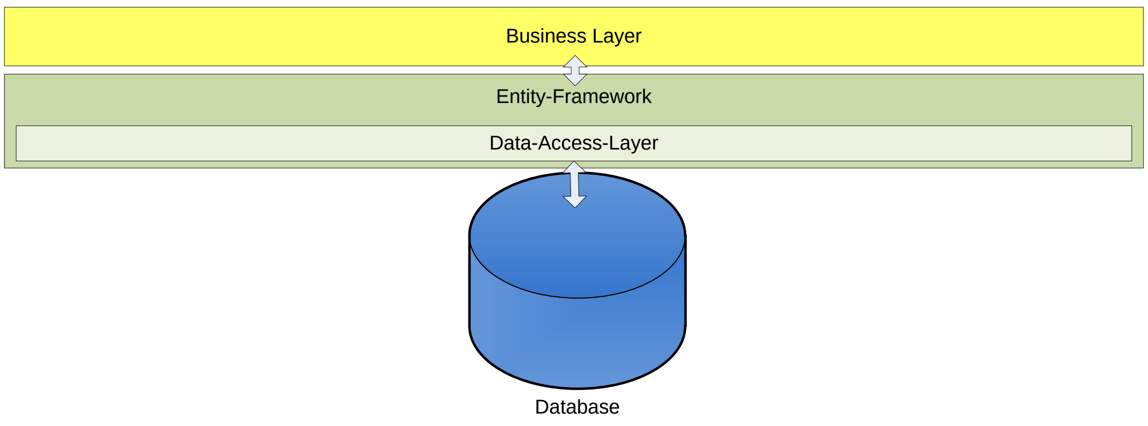


Figure 6.1: Overview of the core system

As can be seen in Figure 6.1 the core system consists of the database, the entity-framework layer and the business layer. The database stores all data used by the system. It consists of three tables in which all data is being saved (for more about the database see chapter 4.4 Structure of the Database).

6.1.1 The Entity-Framework Layer

The entity-framework layer is responsible for accessing the data from the database. It includes the data-access-layer that is being implemented by using the Microsoft Entity-Framework. The entity-framework layer consists of three parts: the Conceptual Model, the Mapping and the Storage/Logical Model. The Conceptual Model is defined in the Conceptual Schema Definition Language (CSDL) which is an XML-language that can be used to create a data model.

It is represented by the Entity-Framework as objects that can be used by the developer to work with the model. That means the developer does not directly access the database but uses the objects defined by the CSDL definition and created by the Entity-Framework. He can use them like normal .NET objects. Each object has methods for saving, updating and deleting objects in the database. The Entity-Framework loads the data from the database and transforms it into objects that can be used by the developer. A graphical representation of the Entity-Framework object model can be seen in figure 6.2. The object model consists of three classes: the DataType, representing a data type, the Entity, which saves atomic data and the Container which contains Entities. The model also represents the relationships between the different classes. It is possible to move from one object to another via navigation properties, e.g. from an object of type Entity to its Container object via the container navigation property. The Entity-Framework model mirrors the database model described in chapter 4.4. It is an object oriented representation of this database model.

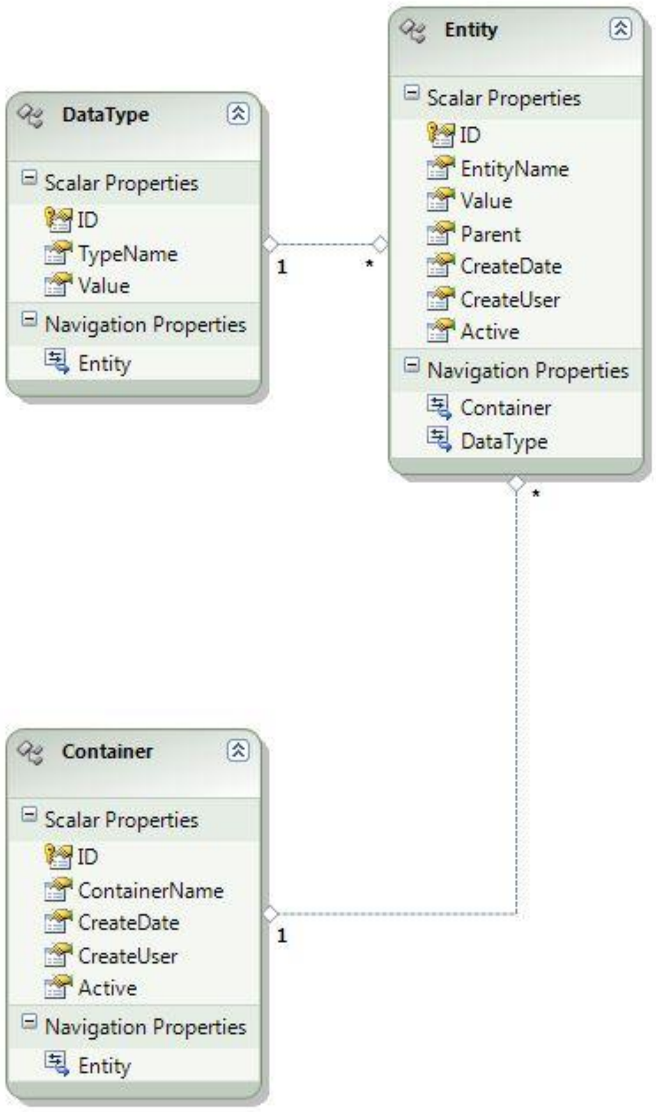
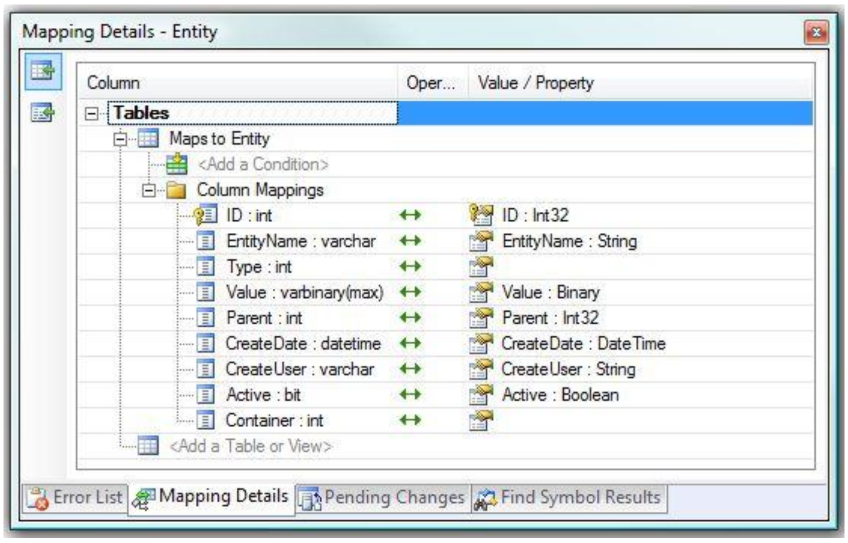


Figure 6.2: The Entity-Framework Model

The Storage/Logical Model is defined in the Store Schema Definition Language (SSDL) which is, as the CSDL, an XML-language. It creates a model that represents the database model. Every time the database changes, this model will be changed as well. The Entity-Framework uses this model so that it knows what the database model looks like. It uses this model to access the tables and fields from the database.

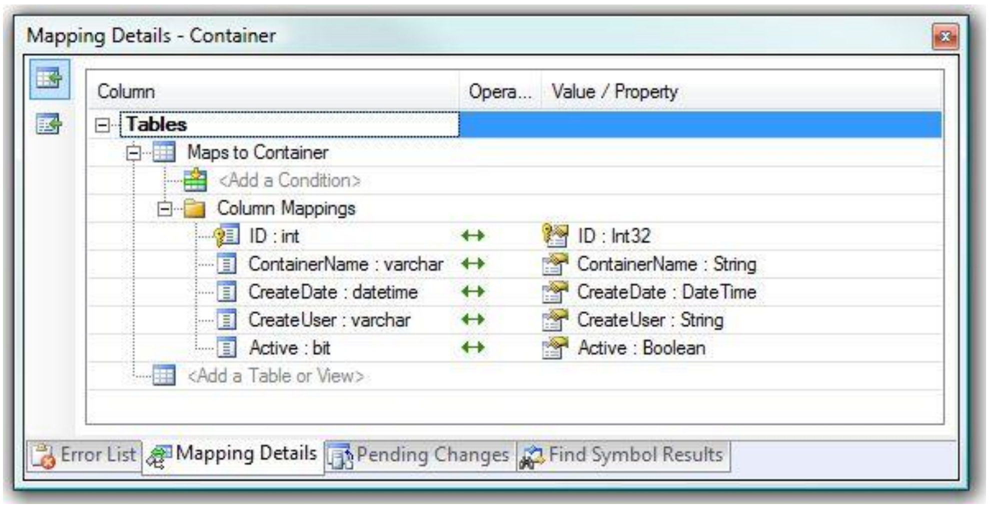
Since the Conceptual Model is an object oriented model that should make it easier for the developer to work with the database, this model can differ from the relational Storage/Logical Model that represents the database schema. In order for the Entity-Framework to connect classes and fields of the Conceptual Model with the tables and fields of the Storage/Logical Model a mapping must be defined. This mapping is being defined in the Mapping in the Mapping Specification Language (MSL) which is also an XML-Language. The Mapping tells the Entity-Framework which fields of the database map to which fields in the classes defined in the Conceptual Model.

Figures 6.3 – 6.5 show the mapping of the Entity, Container and DataType classes:


Mapping Details - Entity

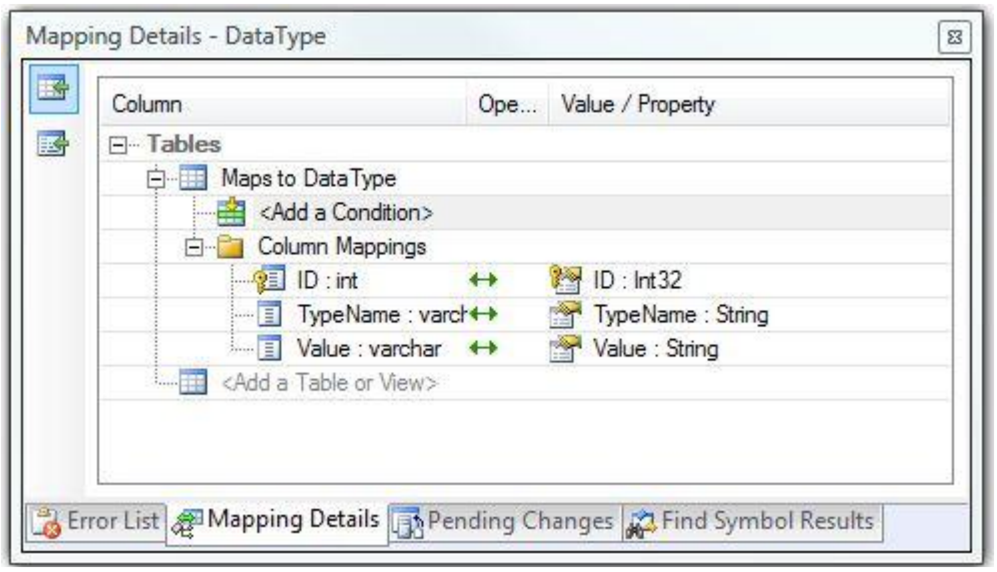
Column	Oper...	Value / Property
Tables		
Maps to Entity		
<Add a Condition>		
Column Mappings		
ID : int	↔	ID : Int32
EntityName : varchar	↔	EntityName : String
Type : int	↔	
Value : varbinary(max)	↔	Value : Binary
Parent : int	↔	Parent : Int32
CreateDate : datetime	↔	CreateDate : DateTime
CreateUser : varchar	↔	CreateUser : String
Active : bit	↔	Active : Boolean
Container : int	↔	
<Add a Table or View>		

Figure 6.3: Mapping Details of Entity


Mapping Details - Container

Column	Opera...	Value / Property
Tables		
Maps to Container		
<Add a Condition>		
Column Mappings		
ID : int	↔	ID : Int32
ContainerName : varchar	↔	ContainerName : String
CreateDate : datetime	↔	CreateDate : DateTime
CreateUser : varchar	↔	CreateUser : String
Active : bit	↔	Active : Boolean
<Add a Table or View>		

Figure 6.4: Mapping Details of Container


Mapping Details - DataType

Column	Ope...	Value / Property
Tables		
Maps to DataType		
<Add a Condition>		
Column Mappings		
ID : int	↔	ID : Int32
TypeName : varch	↔	TypeName : String
Value : varchar	↔	Value : String
<Add a Table or View>		

Figure 6.5: Mapping Details of DataType

As can be seen, the mapping contains information about which fields of the Conceptual Model (right side) are to be connected with which fields of the Storage/Logical Model (left side), which represents the database schema. The various data types shown in Figures 6.3 – 6.5 are saved in the Conceptual Model and in the Storage/Logical Model.[9]

Figure 6.6 shows how those three definitions (Conceptual Model, Storage/Logical Model and Mapping) work together to make it possible to use the object oriented model to work with the database:

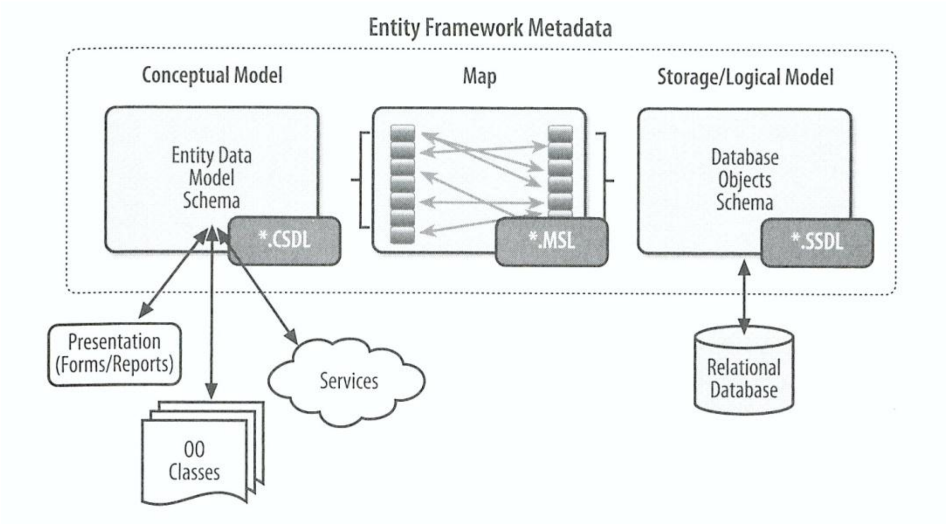


Figure 6.6: Entity Framework Metadata [Lerm09]

[9] To learn more about the inner workings of the Microsoft Entity-Framework read: [Lerm09], [MSDN09]

6.1.2 The Business Layer

The entity-framework layer makes it is possible to access data in the database. It is also possible to store new data into the database and update existing data. The entity-framework layer does not version data, therefore it is not recommended to directly use this layer. If modules used this layer to load, store, save and change data, each module would have to make sure that data is versioned. Therefore it would be possible to create modules that change data and do not save the old version. It would also be possible to completely delete data from the database. This should not be possible. If changes occur, a new version of the entity should be created that contains a link to the old version. If data needs to be deleted, it should be set to inactive. It should not be possible to completely remove data from the database. This way it is possible to track all changes and to look up older versions of data.

Because of these requirements another layer is needed. This functionality is implemented in the business layer. This layer provides an API that can be used by module developers to work with data in the database. The business layer ensures that all data is being versioned and that no data can be entirely removed from the database. Therefore all modules use the business layer's APIs to work with data in the system. The business layer is an enhancement of the entity-framework layer. Figure 6.7 illustrates this:

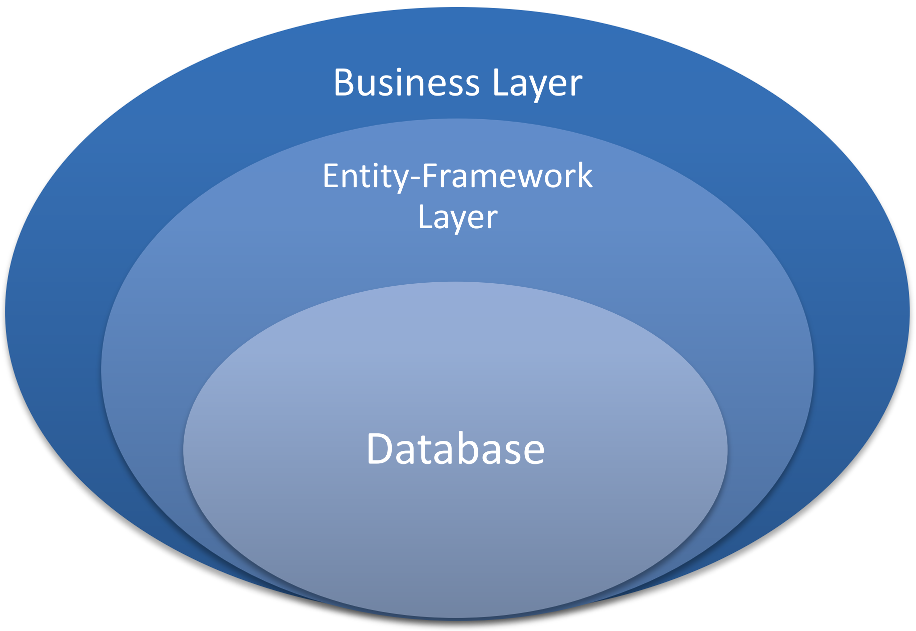


Figure 6.7: Structure of the core system

The database stores all data needed by the system and all its modules. The entity-framework layer creates, updates and deletes data. It is also possible to access data from the database via the entity-framework layer. It provides an object model that can be used to work with data from the database. The business layer uses the object model from the entity-framework layer to work with data. It provides additional functionality that ensures that all changes are versioned and that no data can be entirely removed from the system. Module developers use the API provided by the business layer to work with the data used in the system. They must not use the entity-framework layer or connect directly to the database to access the database or write data to the database, because this would break the versioning functionality.

Figure 6.8 shows the classes of the object model:

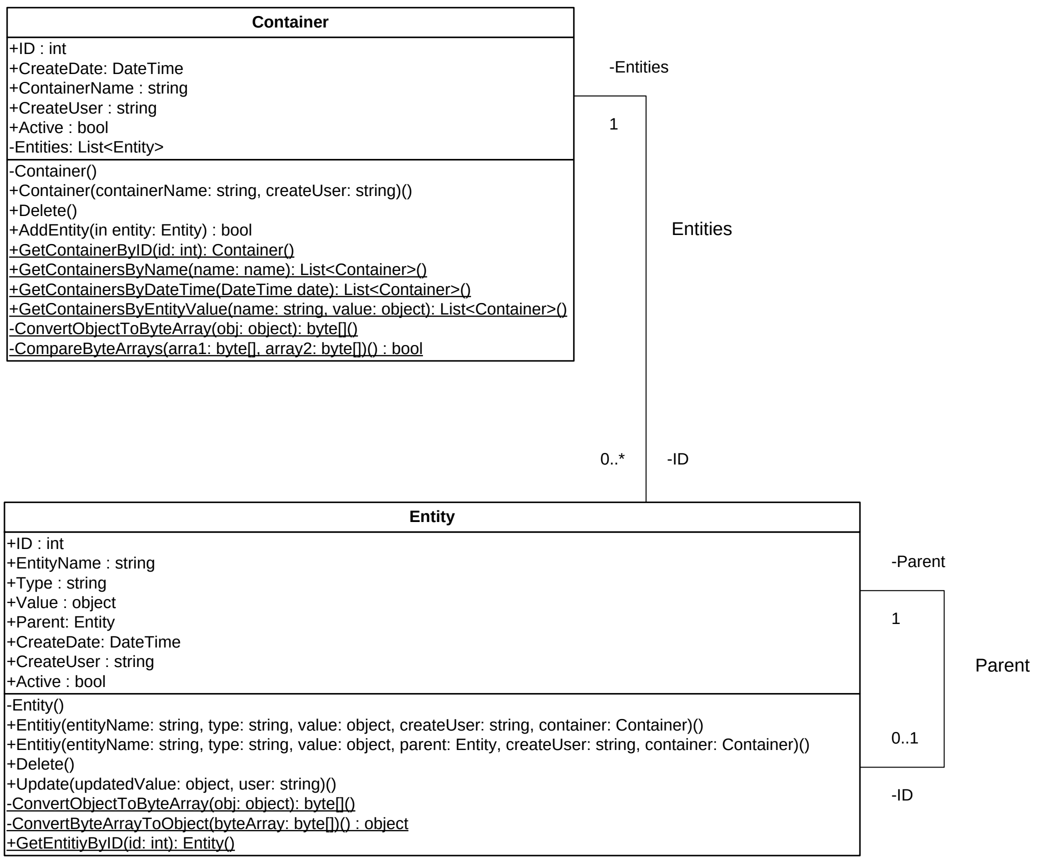

Container
+ID : int
+CreateDate: DateTime
+ContainerName : string
+CreateUser : string
+Active : bool
-Entities: List<Entity>
-Container()
+Container(containerName: string, createUser: string)()
+Delete()
+AddEntity(in entity: Entity) : bool
+GetContainerByID(id: int): Container()
+GetContainersByName(name: name): List<Container>()
+GetContainersByDateTime(DateTime date): List<Container>()
+GetContainersByEntityValue(name: string, value: object): List<Container>()
-ConvertObjectToByteArray(obj: object): byte[]()
-CompareByteArrays(arra1: byte[], array2: byte[])() : bool
-Entities
1
Entities
0..*
-ID
Entity
+ID : int
+EntityName : string
+Type : string
+Value : object
+Parent: Entity
+CreateDate: DateTime
+CreateUser : string
+Active : bool
-Entity()
+Entitiy(entityName: string, type: string, value: object, createUser: string, container: Container)()
+Entitiy(entityName: string, type: string, value: object, parent: Entity, createUser: string, container: Container)()
+Delete()
+Update(updatedValue: object, user: string)()
-ConvertObjectToByteArray(obj: object): byte[]()
-ConvertByteArrayToObject(byteArray: byte[])() : object
+GetEntitiyByID(id: int): Entity()
-Parent
1
Parent
0..1
-ID

Figure 6.8: UML-Diagram of the object model of the core system

Only two classes exist: the Container class and the Entity class. The Entity class stores the data in the Value field. The Value field is of type object and can therefore save any type of data. Every Entity has an ID and an EntityName via which it is possible to access the Entities. The Active field signals whether the Entity is deleted (false) or not (true). If an Entity is updated the Active field is set to false and a new Entity is created. The new Entity contains a link to the old entity in its Parent field. Via this field it is possible to navigate to all previous versions of the Entity. The Entity also saves the DateTime of the creation of the Entity and the username of the user who created the Entity. This creates a full protocol of all changes to an Entity.
The Container class is a bucket that can hold several Entities. The Entities of a Container can be accessed via the Entities field. A Container has a name and saves the DateTime and the username as do the Entities.

The following part describes the methods of the Entity and Container classes and how Containers and Entities can be used to store, update, delete and retrieve data:

Container Data Diagrams

Figure 6.9 shows how the data stored in a Container and its Entities can be visualized.

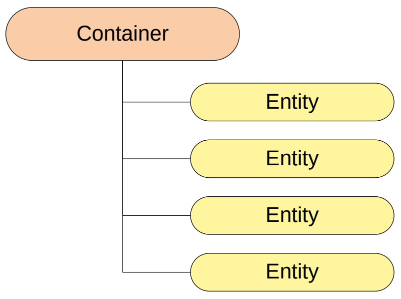


Figure 6.9: Container and Entities

The Entities can be accessed via their Containers.

Because all data needs to be versioned, it is not possible to completely remove or change an Entity. If an Entity is changed, a new Entity is created that contains a link to its old version. Figure 6.10 illustrates this. The green Entity is the old version of the Entity. A new Entity (the yellow one) has been created that contains a link to the green Entity.

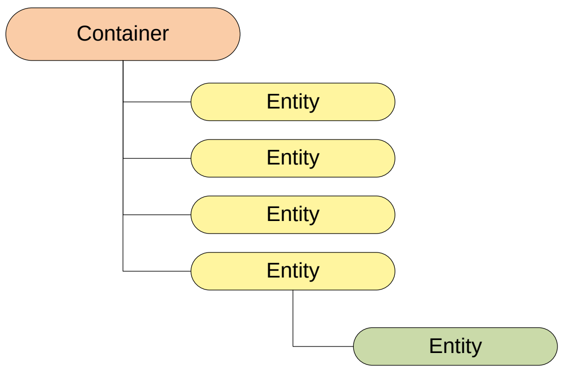


Figure 6.10: A newer version of an `Entity`

It is therefore always possible to access the older versions of an Entity. If the Entity is changed another time, once again a new Entity is created. Figure 6.11 shows the new Container after the Entity has been changed. The yellow Entity is the new Entity. It has a link to the green Entity which in itself has a link to the red Entity.

In code the green Entity can be accessed via the yellow Entity's Parent field.
If a user wants to see the red Entity he needs to use the green Entity's Parent field.

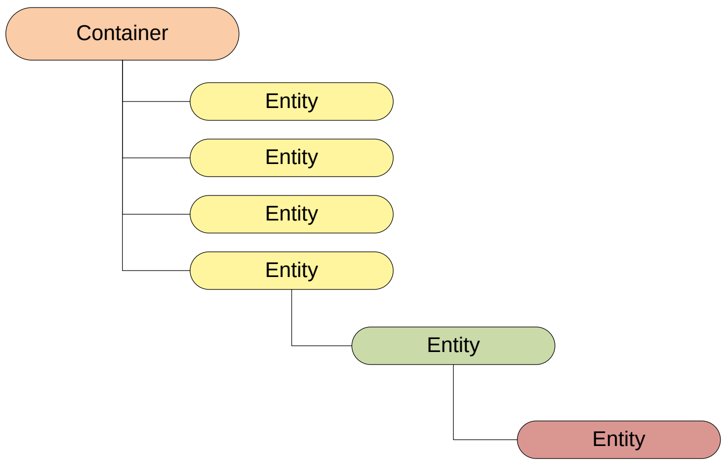


Figure 6.11: Versioning Entities

Figure 6.12 shows an example of a Container that is used in the system (see chapter 6.2.1 for more information about this Container). All of the following examples will use this Container.

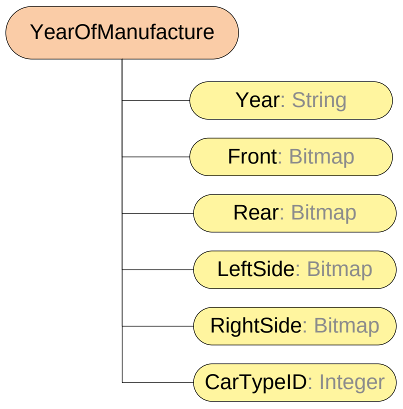


Figure 6.12: Container: YearOfManufacture

Creating Containers and Entities

A new Container can be created by using the constructor of the Container class. The constructor has two parameters: The name of the new Container ("YearOfManufacture") and the username of the user who created the Container. Listing 6.1 shows how a new Container can be created.

```
BackOffice.BusinessLayer.Container container = new
BackOffice.BusinessLayer.Container("YearOfManufacture",
User.Identity.Name);
```

Listing 6.1: Creating a new Container

Creating a new Entity is possible in the same way: The constructor of the Entity class is used. It has four parameters: The name of the Entity ("Year"), the data type ("String"), the value of the Entity (retrieved via a text box) and the container that the Entity will belong too (The YearOfManufacture Container). The Entity will automatically be added to the Container. Later it is possible to retrieve this Entity via this Container.

A second constructor exists that has another parameter: It is used when changing an Entity. The additional Entity is the parent Entity. This constructor is not to be used directly by the developer.

```
BackOffice.BusinessLayer.Entity entity1 = new
BackOffice.BusinessLayer.Entity("Year", "String", TextBox_Year.Text,
User.Identity.Name, container);
```

Lisitng 6.2: Adding an Entity to a Container

The value of an Entity is of type object. Therefore it is possible to store any .NET-type in an Entity. Listing 6.3 shows how the YearOfManufacture Container is created in C# and how the Entities belonging to this Container are created and added to the Container.

```
Bitmap imageFront =
(Bitmap)Bitmap.FromStream(FileUpload_Front.PostedFile.InputStream);

Bitmap imageRear =
(Bitmap)Bitmap.FromStream(FileUpload_Rear.PostedFile.InputStream);

Bitmap imageLeftSide =
(Bitmap)Bitmap.FromStream(FileUpload_LeftSide.PostedFile.InputStream);

Bitmap imageRightSide =
(Bitmap)Bitmap.FromStream(FileUpload_RightSide.PostedFile.InputStream)
;


BackOffice.BusinessLayer.Container container = new
BackOffice.BusinessLayer.Container("YearOfManufacture",
User.Identity.Name);

BackOffice.BusinessLayer.Entity entity1 = new
BackOffice.BusinessLayer.Entity("Year", "String", TextBox_Year.Text,
User.Identity.Name, container);

BackOffice.BusinessLayer.Entity entity2 = new
BackOffice.BusinessLayer.Entity("Front", "Bitmap", imageFront,
User.Identity.Name, container);

BackOffice.BusinessLayer.Entity entity3 = new
BackOffice.BusinessLayer.Entity("Rear", "Bitmap", imageRear,
User.Identity.Name, container);

BackOffice.BusinessLayer.Entity entity4 = new
BackOffice.BusinessLayer.Entity("LeftSide", "Bitmap", imageLeftSide,
User.Identity.Name, container);

BackOffice.BusinessLayer.Entity entity5 = new
BackOffice.BusinessLayer.Entity("RightSide", "Bitmap", imageRightSide,
User.Identity.Name, container);

BackOffice.BusinessLayer.Entity entity6 = new
BackOffice.BusinessLayer.Entity("CarTypeID", "Integer", CarTypeID,
User.Identity.Name, container);
```

Listing 6.3: Code for creating the YearOfManufacture Container and its Entities

As can be seen, the four Entities (Front, Rear, LeftSide and RightSide) store values of type System.Drawing.Bitmap. After this code has been executed the Container, with all its Entities, has been saved into the database.

Retrieving Containers and Entities

Several different methods exist that can be used to retrieve a Container from the database:

- `Container GetContainerByID(int id)`
- `List<Container> GetContainersByName(string name)`
- `List<Container> GetContainersByUserName(string username)`
- `List<Container> GetContainersByDateTime(DateTime date)`
- `List<Container> GetContainersByEntityValue(string name, object value)`

GetContainerByID returns a Container via its ID. All other methods return a generic List of Containers. This is needed because it is possible to retrieve more than one Container by using those parameters. The GetContainersByName method returns all Containers that have a certain name. The GetContainersByUserName method returns all Containers that have been created by a certain user. GetContainersByDateTime retrieves Containers by their date and time. GetContainersByEntityValue can be used to retrieve Containers by the value of one of its Entities.

Listing 6.4 shows how a Container can be retrieved using its id.

```
BackOffice.BusinessLayer.Container container =
BackOffice.BusinessLayer.Container.GetContainerByID(
Convert.ToInt32(ListBox_YearOfManufacture.SelectedValue));
```

Listing 6.4: Retrieving a Container via its id

Listing 6.5 shows how Containers can be retrieved via a value of one of their Entities. It has two parameters: The name of the Entity and the value of the Entity. This method searches through all the Entities of a Container and checks if there is an Entity with this name and if this Entity has got the value that the users is searching for. If this is true, the Container will be returned. This is equivalent to retrieving the data of tables in a database by searching by the values of its columns. In the following example all Containers are retrieved that contain an Entity with the name CarType which has the value selected in a list box.

```
List<BackOffice.BusinessLayer.Container> containers =
BackOffice.BusinessLayer.Container.GetContainersByEntityValue("CarType
ID", Convert.ToInt32(ListBox_CarTypes.SelectedItem.Value));
```

Listing 6.5: Retrieving Containers via an Entity's value

Entities can be retrieved in three ways:

- Entity → `Entity GetEntityByID(int id)`
- Container → `List<Entity> Entities`
- Container → `Entity getEntityByName(string name)`

The Entity class contains a static method (GetEntitiesByID) that returns an Entity by its id. This method is used internally to retrieve Entities. The Container class contains a property (Entities) via which it is possible to access all the Entities of a Container. This is a generic List of type Entity. It is read only. The third way is to use the Container's getEntityByName method. Via this method a certain Entity of a Container can be retrieved by the name of the Entity. E.g. it is possible to retrieve the Entity with the name "Year" (see listing 6.6).

```
BackOffice.BusinessLayer.Entity entity =
container.getEntityByName("Year");
```

Listing 6.6: Retrieving the Entity with the name "Year"

Deleting Containers and Entities

The Container class and the Entity class contain a Delete method:

- Container → `void Delete()`
- Entity →`void Delete()`

If those methods are called, the Container or the Entity is set to inactive. Therefore the data is not entirely deleted and versioning is ensured. If a Container is deleted, the Delete method of all its Entities is called. Listing 6.7 shows how a Container can be deleted.

```
container.Delete();
```

Listing 6.7: Deleting a Container

The developer should always use these methods to delete Containers and Entities. They should never delete them via the entity-framework layer or directly, by deleting the data in the database, because this would break versioning.

Updating Entities

It is possible to change the value of an Entity. This is done by using the Update method:

- `Entity Update(object updatedValue, string user)`

This method has two parameters: updatedValue which contains the new value that is to be stored in the Entity and user which saves the username of the user who made the change.

The method deletes the old Entity (sets it to inactive) and creates a new Entity. It also creates a link to the old Entity. It returns the new Entity. This ensures versioning. All changes can be monitored.

Listing 6.8 shows how the value of an Entity can be changed.

```
entity1.Update(TextBox_ChangeYearOfManufactureYear.Text,
User.Identity.Name);
```

Listing 6.8: Updating an Entity

If the new entity is used later in the program, a new variable should be created. It gets its value via the return value of the Update method. The new Entity can then be used in the program. See listing 6.9 for an example of how to use the new Entity variable.

```
BackOffice.BusinessLayer.Entity newEntity =
entity1.Update(TextBox_ChangeYearOfManufactureYear.Text,
User.Identity.Name);

TextBox_Value.Text = newEntity.Value.ToString();
```

Listing 6.9: Using a new Entity variable

6.2 Administration

The administration module is used by administrators of the central organization to manage car models, users, insurance companies and car repair companies.

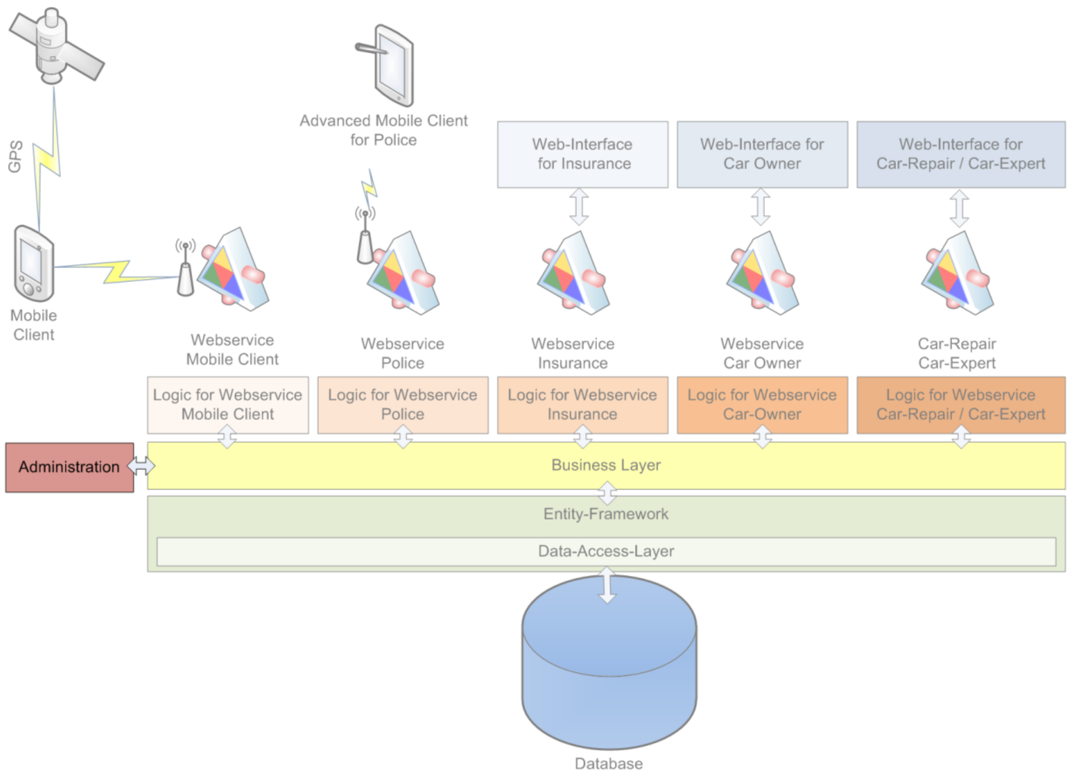

GPS
Mobile Client
Advanced Mobile Client for Police
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.12: The Administration Module

This module can only be accessed by administrators of the central organization that maintain the system. Insurance company employees do not have access to this module.

6.2.1 Managing Car Brands, Car Types and Years of Manufacture

The system provides car modules that can be used by the mobile client to display a model of a car. Those models are saved as four pre-rendered images of the car.

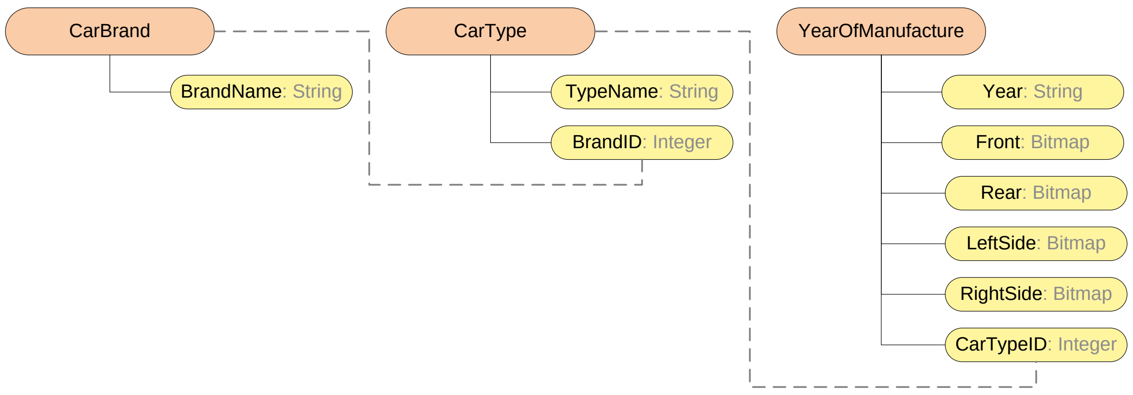


Figure 6.13: Container Data Diagram of the car models

Each car brand will be stored in the database in a CarBrand Container. This Container has one Entity that saves its name. Car brands often manufacture more than one car type (e.g. VW Golf I, VW Golf II, VW Golf III, VW Lupo, VW Fox …). Those car types are saved in the CarType Container. The CarType Container has two Entities: the name of the car type (e.g. VW Fox) and the brand id. The brand id saves the id of the CarBrand Container of the company which created the car type. This id functions as a foreign key, though, the core system does not ensure referential integrity. Since changes can be made to car types, modules are saved as year of manufacture. That means there is a module for each year that the car has been manufactured. This is saved in the YearOfManufacture Container which has six Entities. The Year Entity saves the year of manufacture. The CarTypeID saves the id of the CarType Container of the car type of the model. This functions as a link to the car type. The four remaining Entities save the pre-rendered images of the car module. Figure 6.13 shows the container data diagram of the car models.

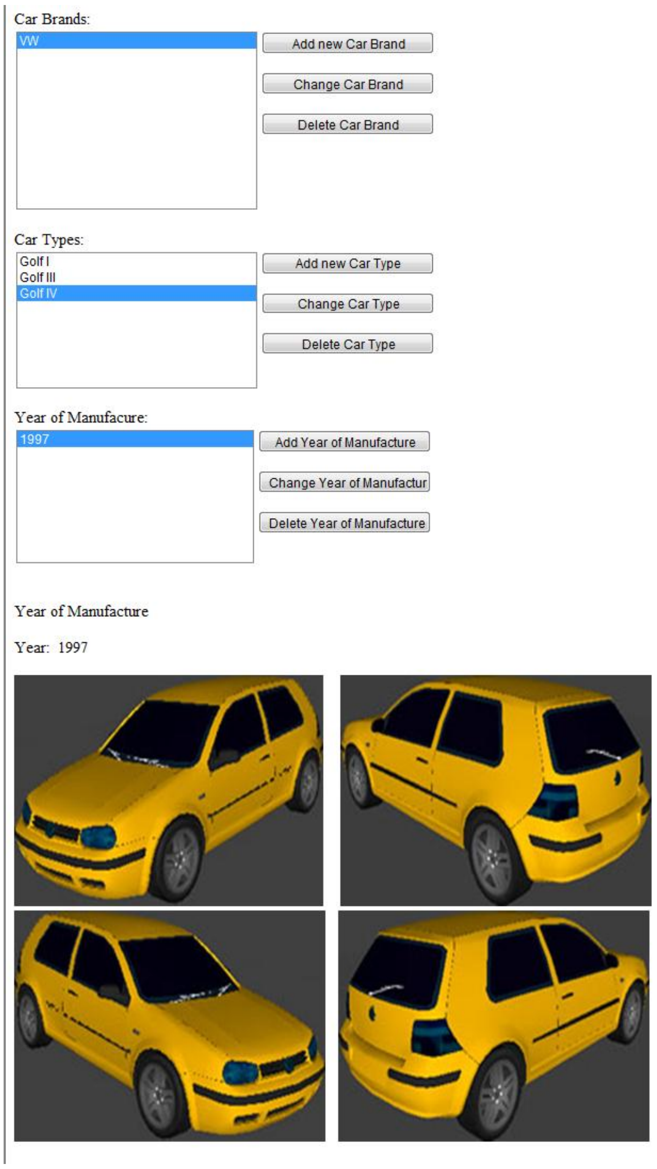
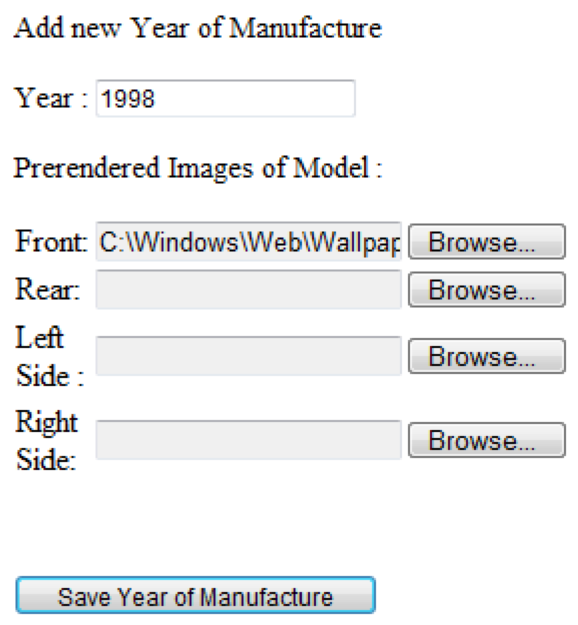


Figure 6.14: Screenshot of managing car models

Figure 6.14 shows the user interface of the car models management module. Car brands, car types and years of manufacture can be added, changed and deleted via this user interface. It consists of three list boxes. If a user clicks on one of them the certain details are displayed, e.g. if a user clicks on a car brand the car types are displayed. If a user clicks on a car type the years of manufacture are displayed. If a user clicks on a year of manufacture the year and the images of the model for this year of manufacture are displayed.

Figure 6.15 shows the user interface that can be used to add a new year of manufacture. The user can upload the images from his computer. The models are then saved to the database.


Add new Year of Manufacture

Year : 1998

Prerendered Images of Model :

Front:	C:\Windows\Web\Wallpar	Browse...
Rear:		Browse...
Left Side :		Browse...
Right Side:		Browse...

Save Year of Manufacture

Figure 6.15: Screenshot of adding a new year of manufacture with the car models

6.2.2 Managing Users

Administrators of the central organization can add, delete and change users and their profile data. Insurance company employees do not need to manage the user data. They will only manage insurance specific data, like for example the insurance number of a customer. The data managed by the administrators of the central organization is basic data needed by the system. It saves the title, first name and last name of a person, as well as their address containing the street, ZIP, city, telephone number and mobile phone number. The identity card number, passport number and driver's license number are saved as well. If some of this data changes (e.g. the user moves to another city), he will have to contact the central organization and ask them to change the data.

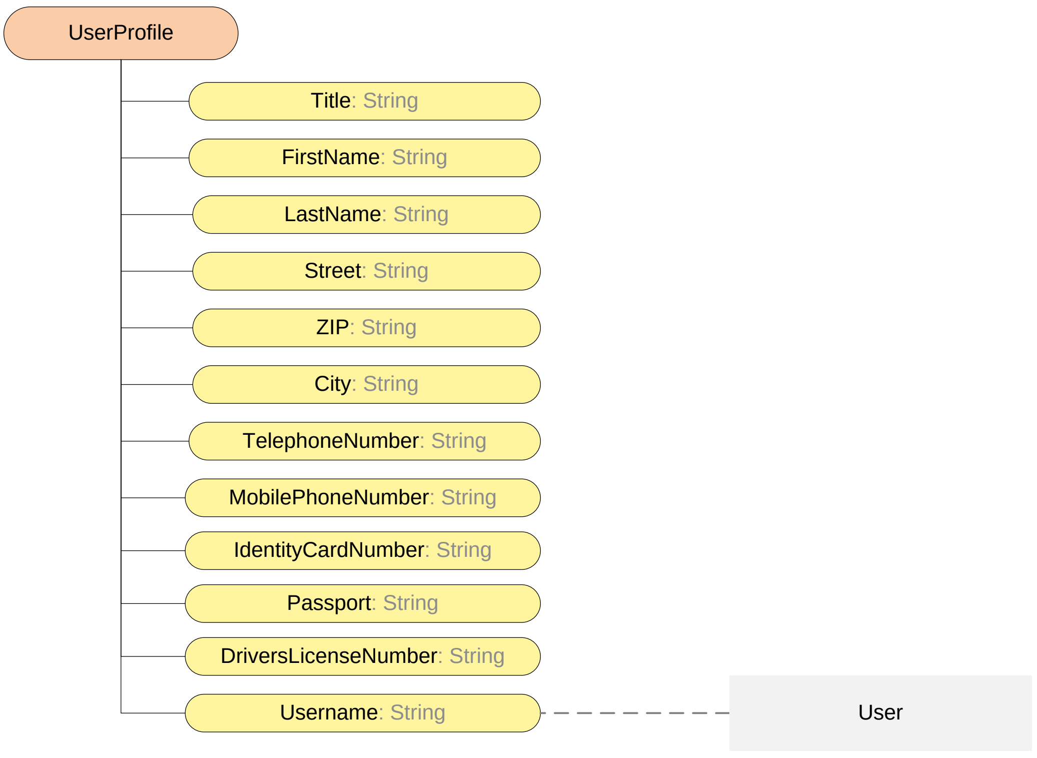
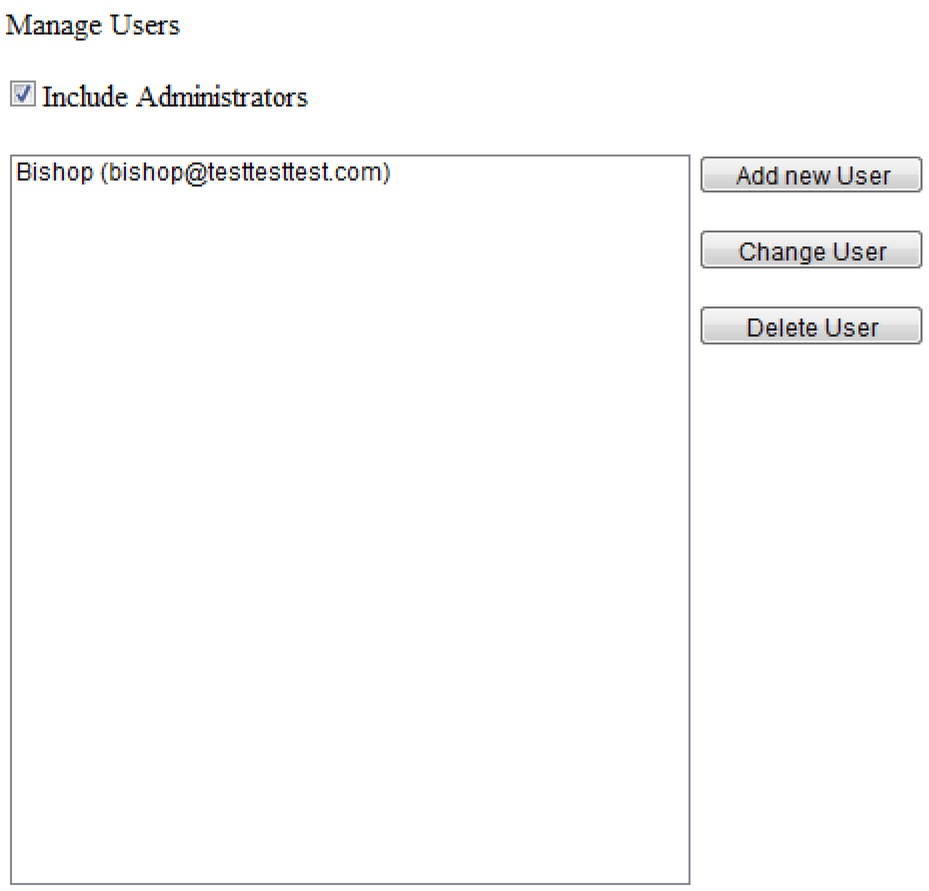


Figure 6.16: Container Data Diagram of User Profile


Manage Users

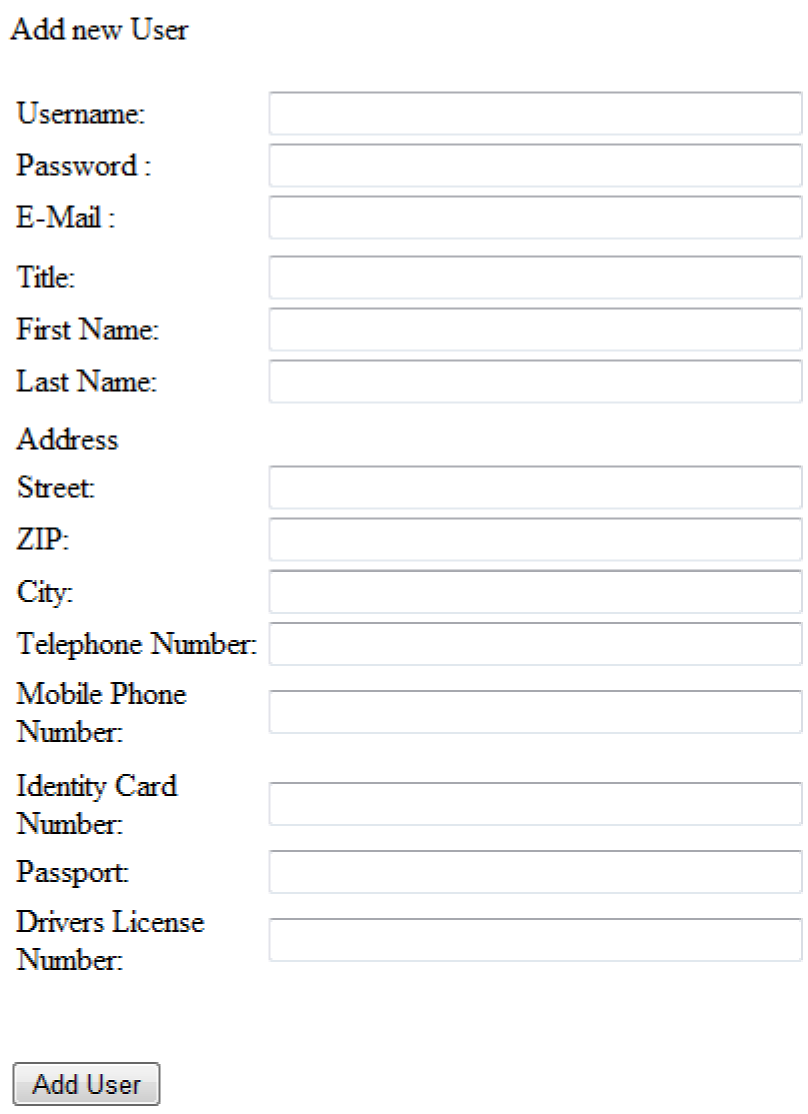
☑ Include Administrators

Bishop (bishop@testtesttest.com)

Figure 6.17: Screenshot of managing users

Figure 6.17 shows the user interface for managing users. New users can be added, existing users changed or deleted. An option exists that prevents displaying administrators in the list.

Figure 6.18 shows how a new user can be added to the system:



Add new User

Username:	
Password :	
E-Mail :	
Title:	
First Name:	
Last Name:	
Address	
Street:	
ZIP:	
City:	
Telephone Number:	
Mobile Phone Number:	
Identity Card Number:	
Passport:	
Drivers License Number:	

Figure 6.18: Screenshot: Adding a new user

6.2.3 Managing Insurance Companies

The central organization will manage all insurance companies. The administrators of this organization can add new insurance companies, change the name of existing insurance companies or delete insurance companies. They can also add and remove users as an employee to and from an insurance company or add and remove users as a customer to and from a company.

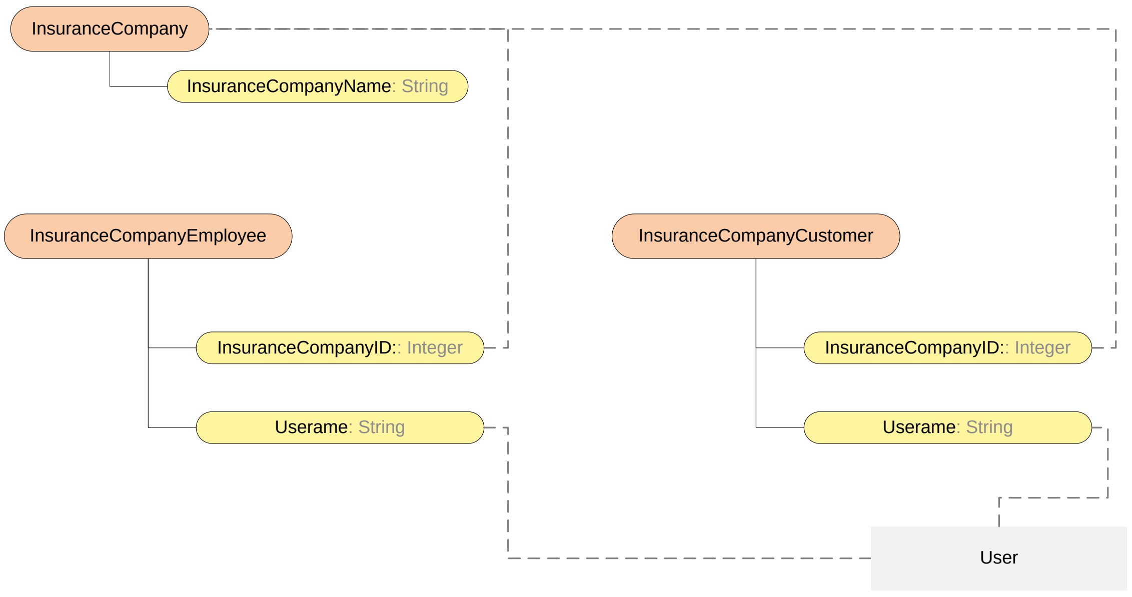
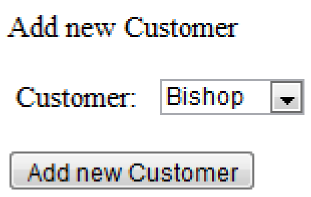


Figure 6.19: Container Data Diagram of Insurance Companies

Each insurance company will be saved in the database in an InsuranceCompany Container. This Container has one Entity (InsuranceCompanyName) which saves the name of the insurance company. There are two more Containers, InsuranceCompanyEmployee, which saves which users are employees of the insurance company and InsuranceCompanyCustomer, which saves the users who are customers of the insurance company. Both save a reference to the InsuranceCompany Container (InsuranceCompanyID) and to the user object (Username).

Figure 6.20 shows how a new customer can be added to an insurance company:


Add new Customer

Customer: Bishop

Figure 6.20: Adding a customer to an insurance company

Figure 6.21 shows how an insurance company can be managed:

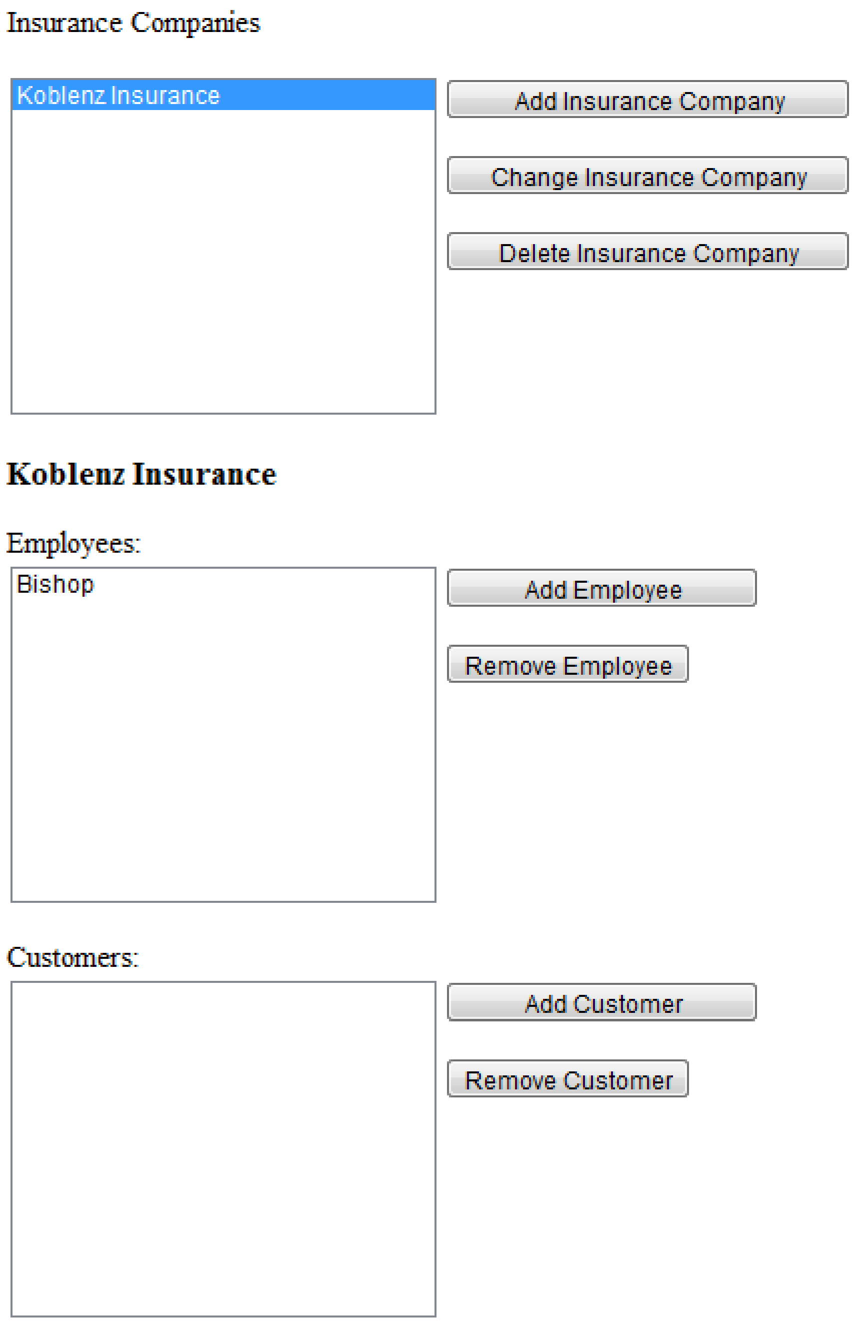


Figure 6.21: Managing insurance companies

Employees of an insurance company can later access the website for insurance companies and process their customers' accident cases.

6.2.4 Managing Car Repair Companies

The central organization will manage all car repair companies. The administrators of this organization can add new car repair companies, change the name of existing car repair companies or delete car repair companies. They can also add or remove users as an employee to and from a car repair company.

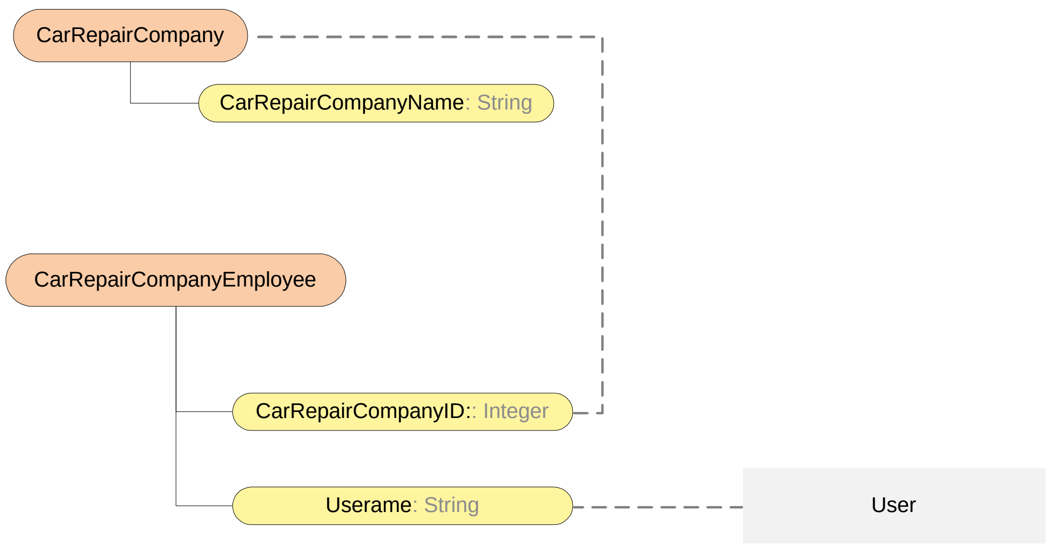


Figure 6.22: Container Data Diagram of Car Repair Companies

The CarRepairCompany Container has one Entity (CarRepairCompanyName) which saves the name of the car repair company. This Container represents a car repair company. The CarRepairCompanyEmployee Container saves all employees that work at the car repair company. It has two Entities: CarRepairCompanyID saves the id of the car repair company's Container. This id functions as a link to the CarRepairCompany Container. Username saves the usersname of the employee.

Figure 6.23 shows how a new employee can be added to a car repair company:

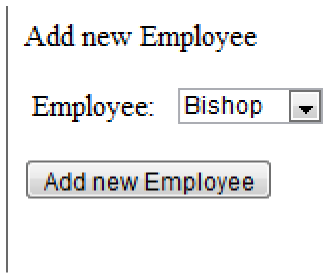
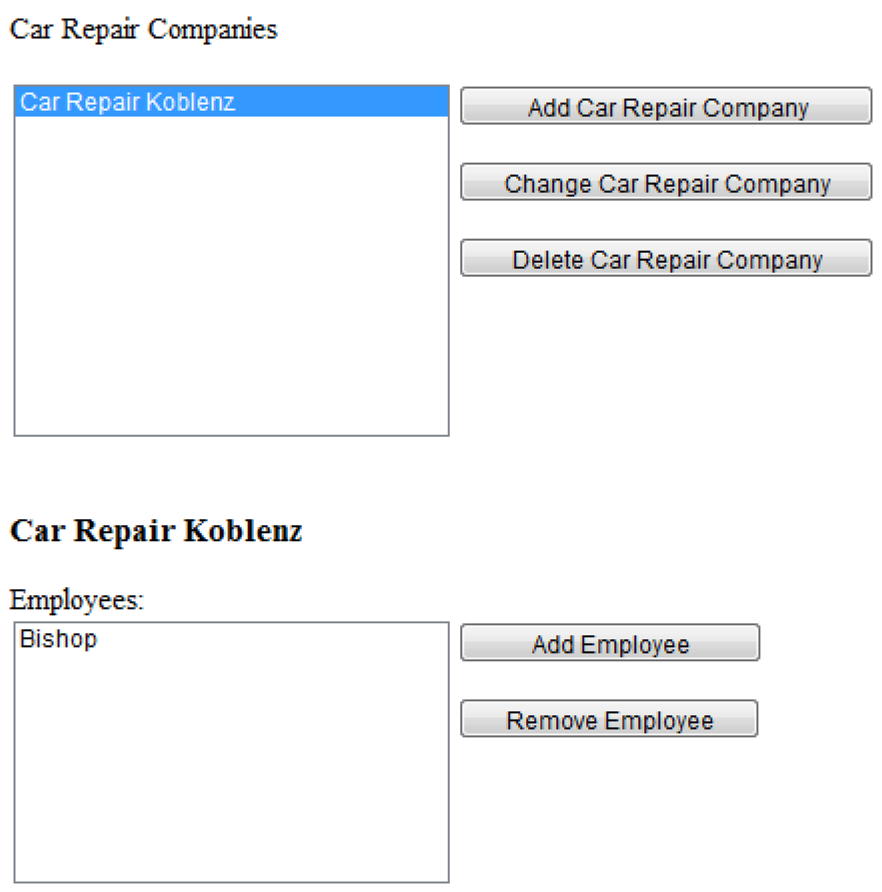


Figure 6.23: Adding an employee to a car repair company

Figure 6.24 shows how car repair companies can be managed:


Car Repair Companies

Car Repair Koblenz

Car Repair Koblenz

Employees:

Bishop

Figure 6.24: Managing car repair companies

6.3 Web Service for the Mobile Client

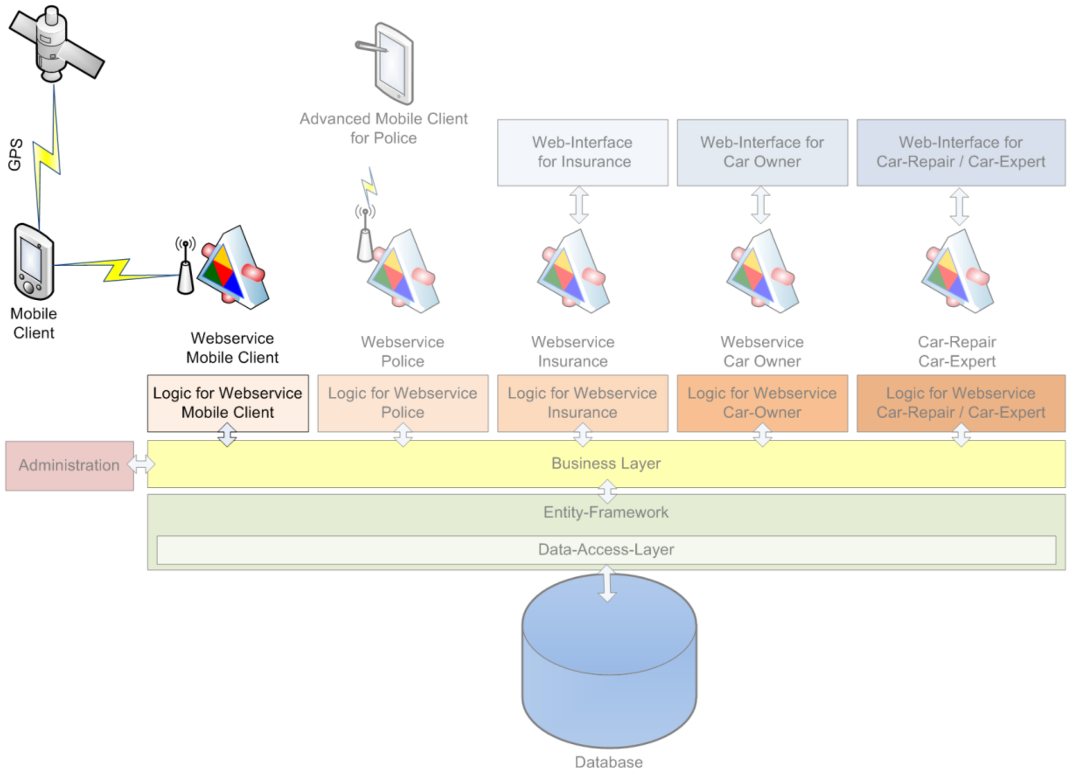

GPS
Mobile Client
Webservice Mobile Client
Advanced Mobile Client for Police
Webservice Police
Web-Interface for Insurance
Webservice Insurance
Web-Interface for Car Owner
Webservice Car Owner
Web-Interface for Car-Repair / Car-Expert
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.25: The Mobile Client Module

The mobile client module provides a web service which contains three methods that can be used by the mobile client to retrieve and send data.

Method Name	Parameters	Return Type
GetPersonalData	TelephoneNumber: string password: string	PersonalData
GetModels	brand: string type: string yearOfManufacture: string	Bitmap[]
SendAccidentData	accidentData: string	bool

Table 6.1: Web Service for the Mobile Client

The first method, GetPersonalData, has two parameters. TelephoneNumber is the car owner's telephone number. The second parameter is a password.

The method returns an object of type PersonalData. This object contains the Title, FirstName, LastName, Street, ZIP, TelephoneNumber and MobilePhoneNumber of the car owner. It also contains an array of Car objects. These represent the car owner's cars. A car object consists of the LicensePlateNumber, Brand, Type, YearOfManufacture and Models (pre rendered bitmap images of the car model). If an error occurs or no car owner can be found, an empty PersonalData object is returned with the Valid field set to false.

The second method, GetModels, can be used in order to retrieve images of pre rendered models of a certain car. The brand, car type and year of manufacture are required by this method. It returns an array of type bitmap.

The third method, SendAccidentData, can be used in order to send a report about an accident case from the mobile client to the central system. It has one parameter that contains a string representation of the xml-document that contains the accident data. The method returns a boolean value. This value is set to true if the report has been successfully saved in the central system. It is set to false if an error occurred.

Proxy Web Service

It was not possible to access the WCF web service via a Windows Mobile client. Therefore a proxy web service has been created that is located between the web service for the mobile client and the mobile client itself. Figure 6.26 illustrates this. The proxy web service is a classic asmx-web service.

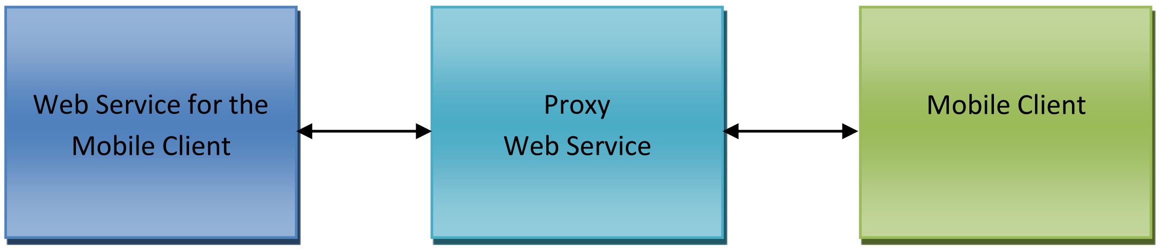


Figure 6.26: Proxy Web Service

There were also some problems transmitting the bitmap images via the asmx-web service. That is why the images are transferred as byte arrays. Listing 6.10 shows a method via which it is possible to convert a byte array to a bitmap.

```
private Bitmap ConvertByteArrayToBitmap(byte[] byteArray)
{
      System.IO.MemoryStream stream = new
      System.IO.MemoryStream(byteArray);
      System.Drawing.Bitmap bitmap = new Bitmap(stream);
      return bitmap;
}
```

Listing 6.10: Converting a byte array to a bitmap

Method Name	Parameters	Return Type
GetPersonalData	TelephoneNumber: string password: string	PersonalData
GetModels	brand: string type: string yearOfManufacture: string	byte[][]
SendAccidentData	accidentData: string	bool

Table 6.2: Proxy Web Service for the Mobile Client

Figure 6.27 shows how the data, that has been submitted by the mobile client, is saved in the back-office:

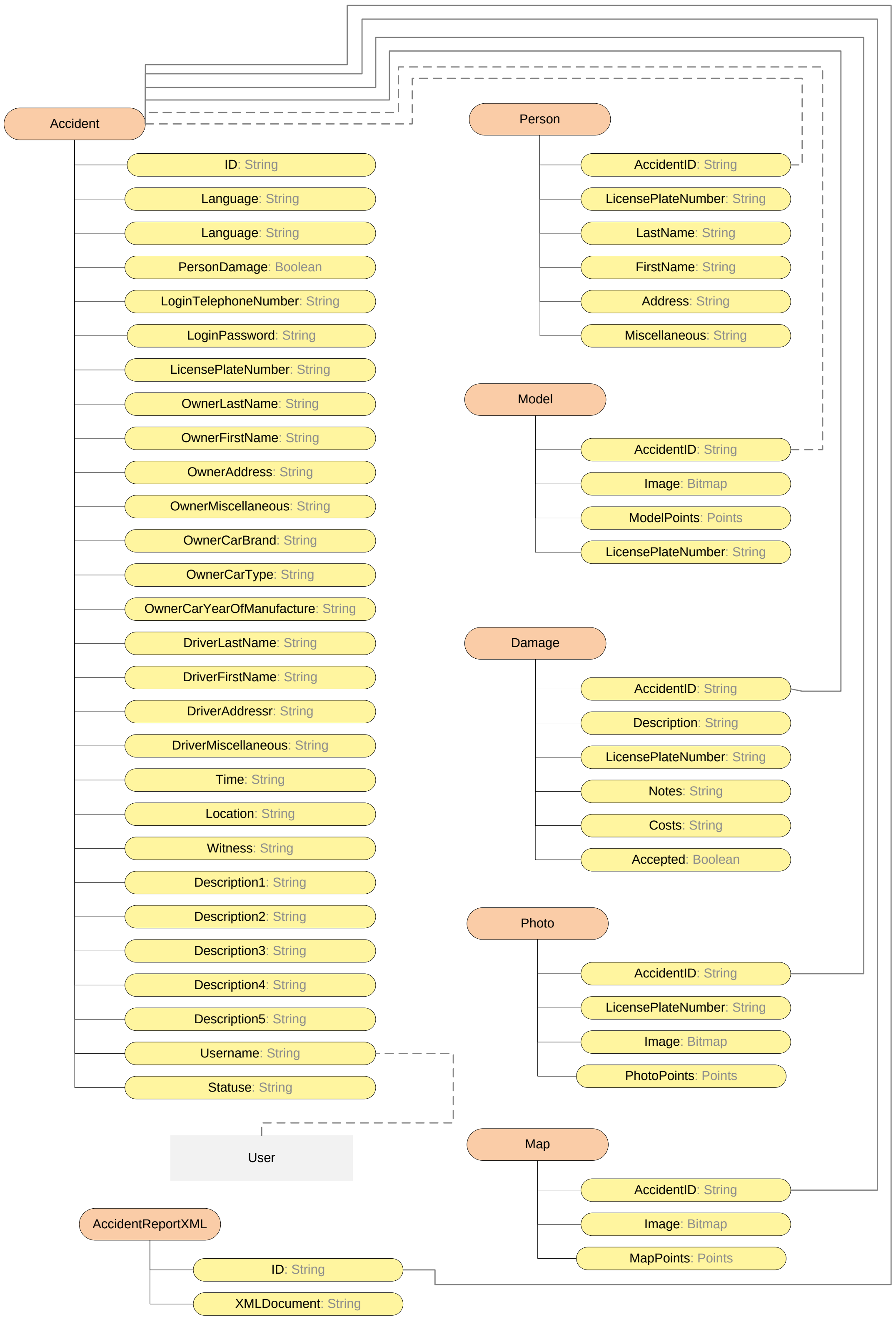
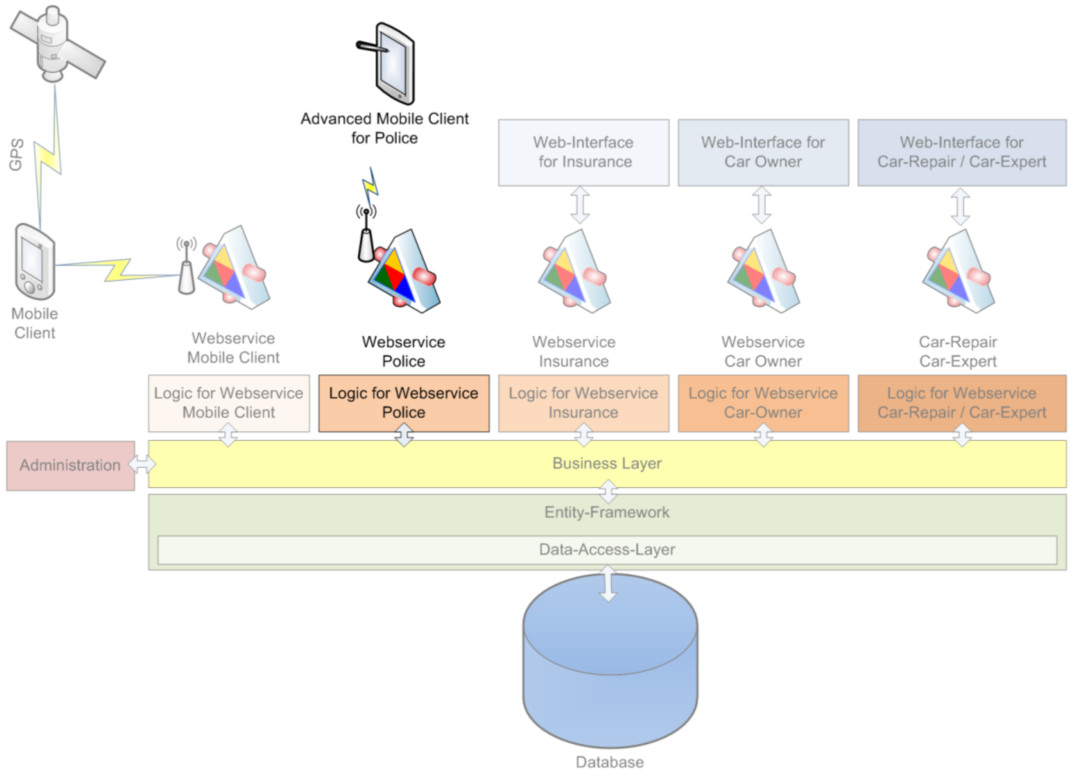


Figure 6.27: Container Data Diagram of Accident

The data is saved in seven containers: Accident, Person, Model, Damage, Photo , Map and AccidentReportXML, which saves the original xml file that has been submitted by the client.

6.4 Web Service for the Police



GPS
Mobile Client
Advanced Mobile Client for Police
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.28: The Police Module

The web service for the police consists of only one method via which it is possible to retrieve data about a person who is registered in the system.

Method Name	Parameters	Return Type
GetPersonalData	number: string type: Type	PersonalData

Table 6.3: Web Service for the Police

The data can be requested in several ways: The identity card number, passport number, driver's license number or license plate number can be used in order to get a person's data.

The first parameter contains the number. The second parameter is an enumeration of type Type (which can have the following values: Type.IdentityCardNumber, Type.Passport, Type.DriversLicense, Type.LicensePlateNumber), which indicates which kind of number is used. The Method returns an object of type PersonalData. This object contains the Title, FirstName, LastName, Street, ZIP, TelephoneNumber and MobilePhoneNumber of the car owner.
If an error occurs or no car owner can be found, an empty PersonalData object is returned with the Valid field set to false.

The police client uses this service in order to pre fill forms. This makes it easier and faster for the police to fill out the form.

6.5 Insurance Company

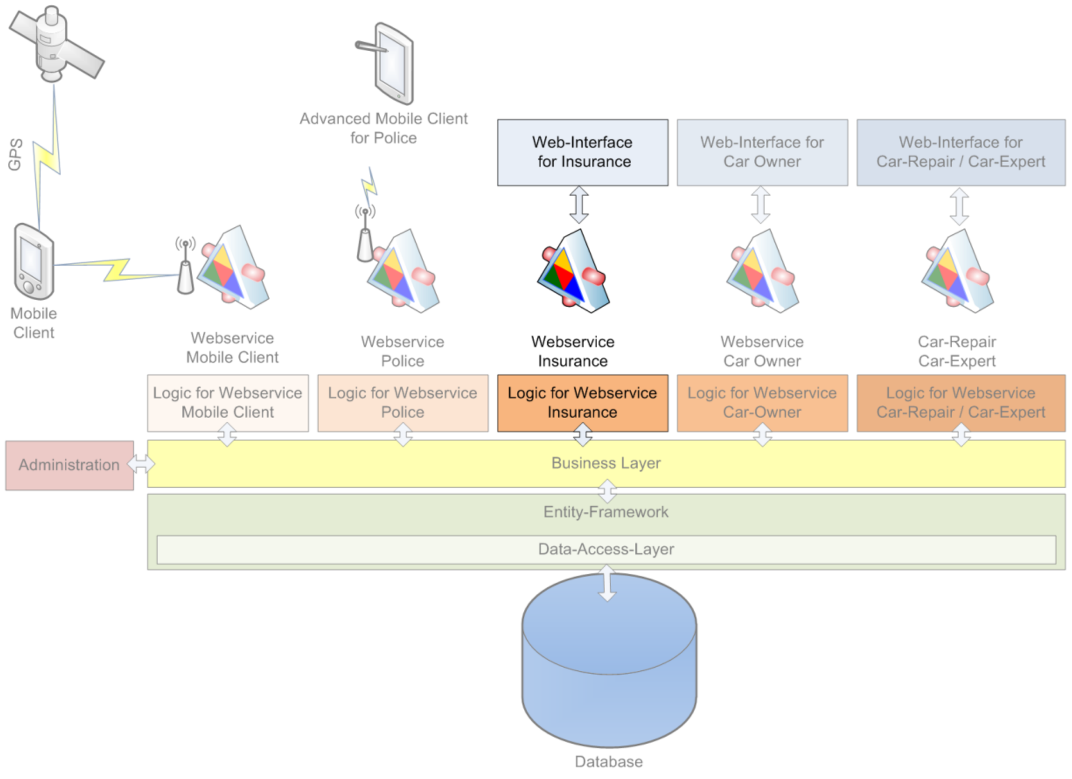

GPS
Mobile Client
Advanced Mobile Client for Police
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.29: The Insurance Module

The module for the insurance company provides a web service which can be used in order to integrate the system into the software infrastructure of the insurance company.

6.5.1 Web Service

The web service for the insurance company consists of several methods via which an insurance company can manage customers and manage accidents. Table 6.4 lists all the methods of the web service for the insurance company:

Method Name	Parameters	Return Type
GetCustomers	insuranceCompanyID: int employeeUsername: string password: string	Customer[]
GetCarRepairCompaniesForAUser	insuranceCompanyID: int employeeUsername: string password: string	CarRepairCompany[]
GetCarRepairCompanies		CarRepairCompany[]
AddCarRepairCompanyToUser	carRepairCompany: CarRepairCompany username: string employeeUsername: string password: string	
RemoveCarRepairCompany FromUser	carRepairCompany: CarRepairCompany username: string employeeUsername: string	

	password: string	
SaveCustomerChanges	customer: Customer employeeUsername: string password: string	
GetCarBrands		CarBrand[]
GetCarTypes	carBrand: CarBrand	CarType[]
GetYearsOfManufacture	carType: CarType	YearOfManufacture[]
AddCar	car: Car username: string employeeUsername: string insuranceCompanyID: int password: string	
GetCarsForUser	username: string employeeUsername: string password: string	Car[]
GetCarsForUserFor InsuranceCompany (GetCarForUser)	username: string insuranceCompanyID: int employeeUsername: string password: string	Car[]
DeleteCar	car: Car employeeUsername: string password: string	
GetCarbyLicensePlateNumber	licensePlateNumber: string employeeUsername: string password: string	Car
GetAccidentsForUser	username: string employeeUsername: string password: string	Accident[]
GetAccidentsForUserFor InsuranceCompany (GetAccidentsForUser)	username: string insuranceCompanyID: int employeeUsername: string password: string	Accident[]
GetAccident	id: string employeeUsername: string password: string	Accident
GetOtherCar	id: int employeeUsername: string password: string	Accident.OtherCar
GetDamage	id: int employeeUsername: string password: string	Accident.Damage
AddDamage	damage: Accident.Damage accident: Accident employeeUsername: string password: string	
ChangeDamage	damage: Accident.Damage accident: Accident employeeUsername: string password: string	
DeleteDamage	damage: Accident.Damage accident: Accident	

	employeeUsername: string password: string	
ChangeOtherCar	otherCar: Accident.OtherCar accident: Accident employeeUsername: string password: string	
ChangeAccident	accident: Accident employeeUsername: string password: string	
GetAccidentReportXML	accidentID: string employeeUsername: string password: string	XmlDocument
GetModel	id: int employeeUsername: string password: string	Accident.Model
GetPhoto	id: int employeeUsername: string password: string	Accident.Photo
GetMap	id: int employeeUsername: string password: string	Accident.Map

Table 6.4: Web Service for the Insurance Company

The web service uses several classes in order to save and transfer data. Those classes are shown in figure 6.30. The Customer class saves data about a customer of the insurance company. The Accident class, which contains the OtherCar, Damage, Model, Photo and Map class, saves all data about an accident case. The Car class represents a car.

Methods in the web service get the username and the password of the insurance company employee who is calling a method and validates if they are correct. The methods also check whether a user is allowed to access the data or do the action he is about to do.

The web service methods call methods of the insurance company model which retrieve data, save data, change data or delete data. This helps to keep the web service methods clean. They only contain the authentication and authorization code as well as the method calls to the module methods. The module (InsuranceCompany.cs) contains all the functionality needed.

Table 6.5 lists all the methods of the insurance company module.

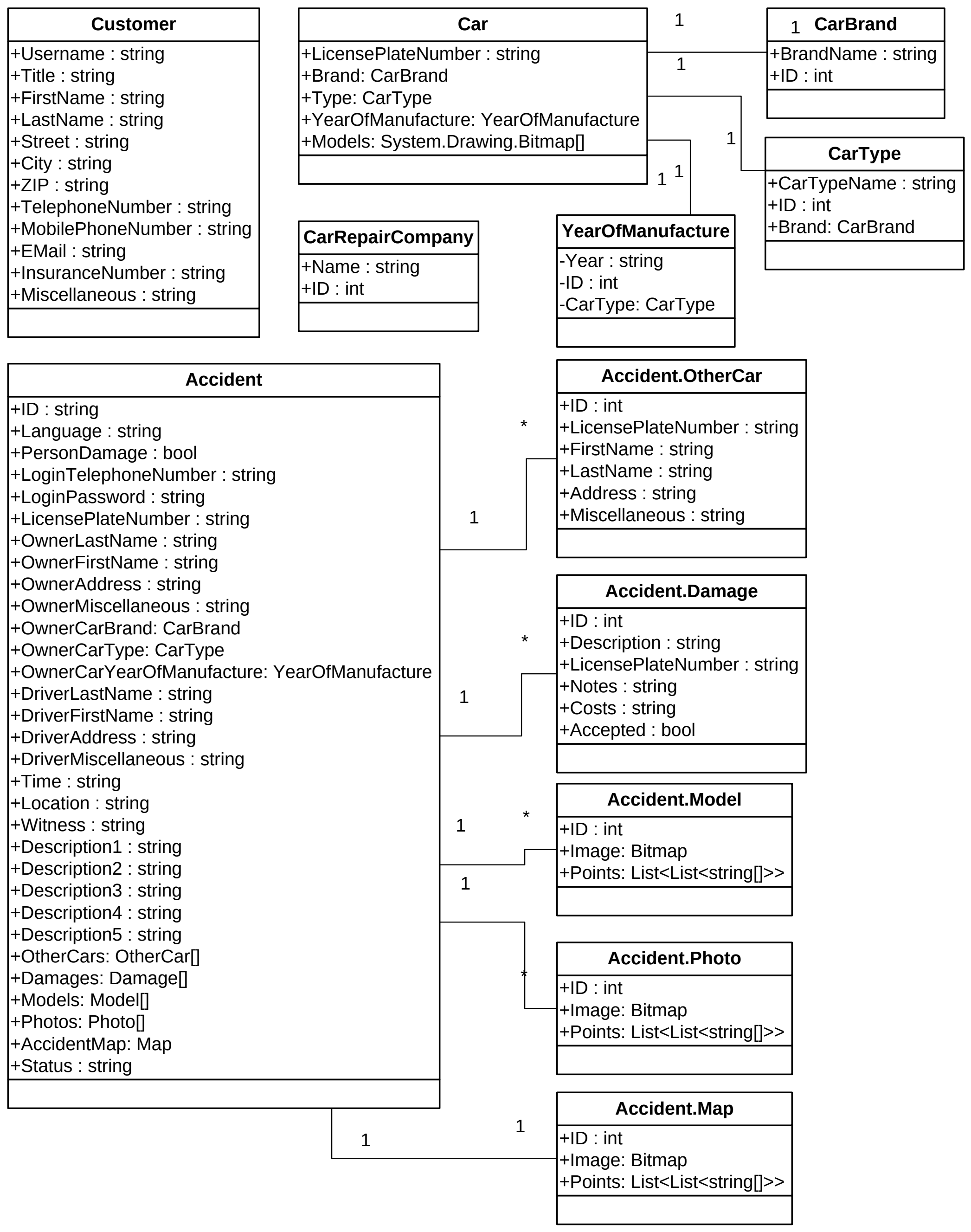


Figure 6.30: UML Diagram of Classes used by the Web Service for the Insurance Company

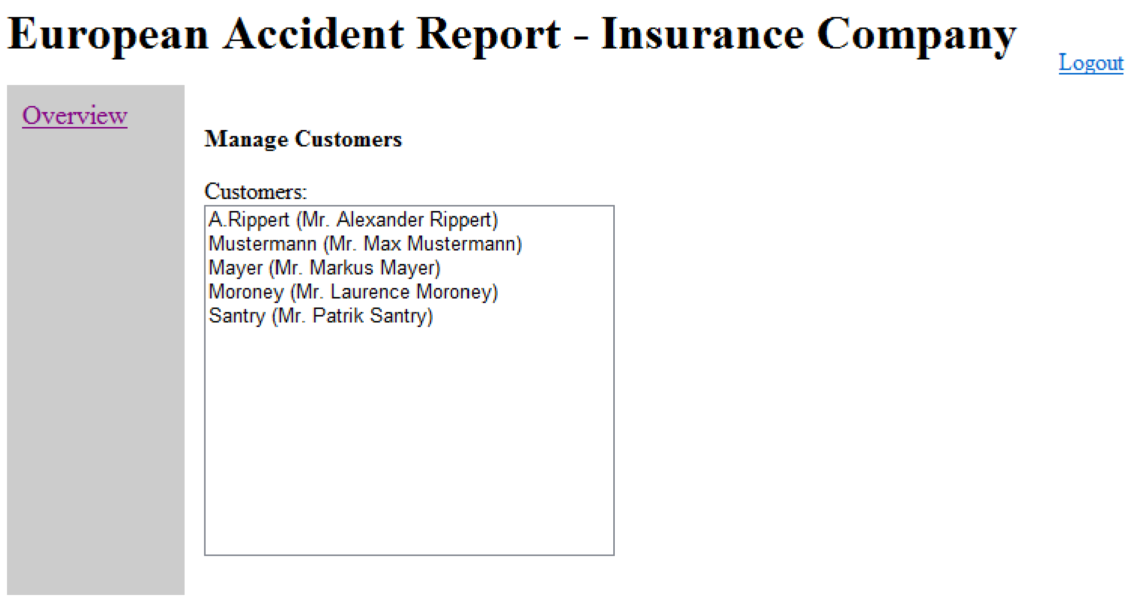
Method Name	Parameters	Return Type
FillCustomer	customerUsername: string	Customer
FillCustomers	insuranceCompanyID: int	List<Customer>
FillCarRepairCompanies		List<CarRepairCompany>
FillCarRepairCompaniesForUser	username: string	List<CarRepairCompany>
SaveCarRepairCompany	carRepairCompany: CarRepairCompany username: string employeeUsername: string	
RemoveCarRepairCompany	carRepairCompany: CarRepairCompany username: string employeeUsername: string	
SaveUserChanges	customer: Customer employeeUsername: string	
FillCarBrands		List<CarBrand>
FillCarTypes	carBrand: CarBrand	List<CarType>
FillYearOfManufacture	carType: CarType	List<YearOfManufacture>
SaveCar	car: Car username: string employeeUsername: string insuranceCompanyID: int	
FillCarsForUser	username: string	List<Car>
FillCarsForUser	username: string insuranceCompanyID: int	List<Car)
RemoveCar	car: Car	
FillCarByLicensePlateNumber	licensePlateNumber: string	Car
FillAccidentsForUser	username: string	List<Accident>
FillAccidentsForUser	username: string insuranceCompanyID: int	List<Accident>
FillAccident	id: int	Accident
FillOtherCar	id: int	Accident.OtherCar
FillDamage	id: int	Accident.Damage
SaveDamage	damage: Accident.Damage accident: Accident employeeUsername: string	
SaveDamageChanges	damage: Accident.Damage accident: Accident employeeUsername: string	
DeleteDamage	damage: Accident.Damage employeeUsername: string	
SaveOtherCarChanges	otherCar: Accident.OtherCar accident: Accident employeeUsername: string	
SaveAccidentChanges	accident: Accident employeeUsername: string	
FillAccidentReportXML	accidentID: string	XmlDocument

FillModel	id: int	Accident.Model
FillPhoto	id: int	Accident.Photo
FillMap	id: int	Accident.Map
CheckIfEmployeeBelongs ToInsuranceCompany	username: string insuranceCompanyID: int	bool
CheckIfInsuranceCompany EmployeeIsAllowedToSee DataOfUser	customerUsername: string employeeUsername: string	bool
FillUsernameOfCarOwner	car: Car	string
FillUsernameOfOtherCar Accident	car: Accident.OtherCar	string
FillUsernameOfDamage	damage: Accident.Damage	string
FillAccidentOfAccidentModel	model: Accident.Model	Accident
FillAccidentOfAccidentPhoto	photo: Accident.Photo	Accident
FillAccidentOfAccidentMap	map: Accident.Map	Accident

Table 6.5: Insurance Company Module Methods

6.5.2 Web Application

Insurance company employees can use the web application in order to manage their customers and the accident cases and damages of those customers. Figure 6.31 shows the starting page of the web application. It contains a list with all customers of the insurance company to which the insurance company employee belongs.



European Accident Report - Insurance Company

Logout

Overview

Manage Customers

Customers:

A.Rippert (Mr. Alexander Rippert)
Mustermann (Mr. Max Mustermann)
Mayer (Mr. Markus Mayer)
Moroney (Mr. Laurence Moroney)
Santry (Mr. Patrik Santry)

Figure 6.31: Insurance Company Customers

Insurance company specific data about a customer is saved in the InsuranceCompanyCustomerData Container. This container consists of three entities. The Username saves the username of the customer. Miscellaneous can contain random data that does not fit into other fields. InsuranceNumber contains the insurance number of the customer.

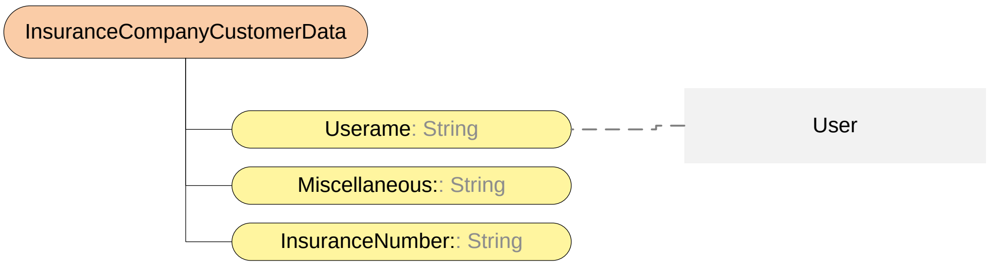
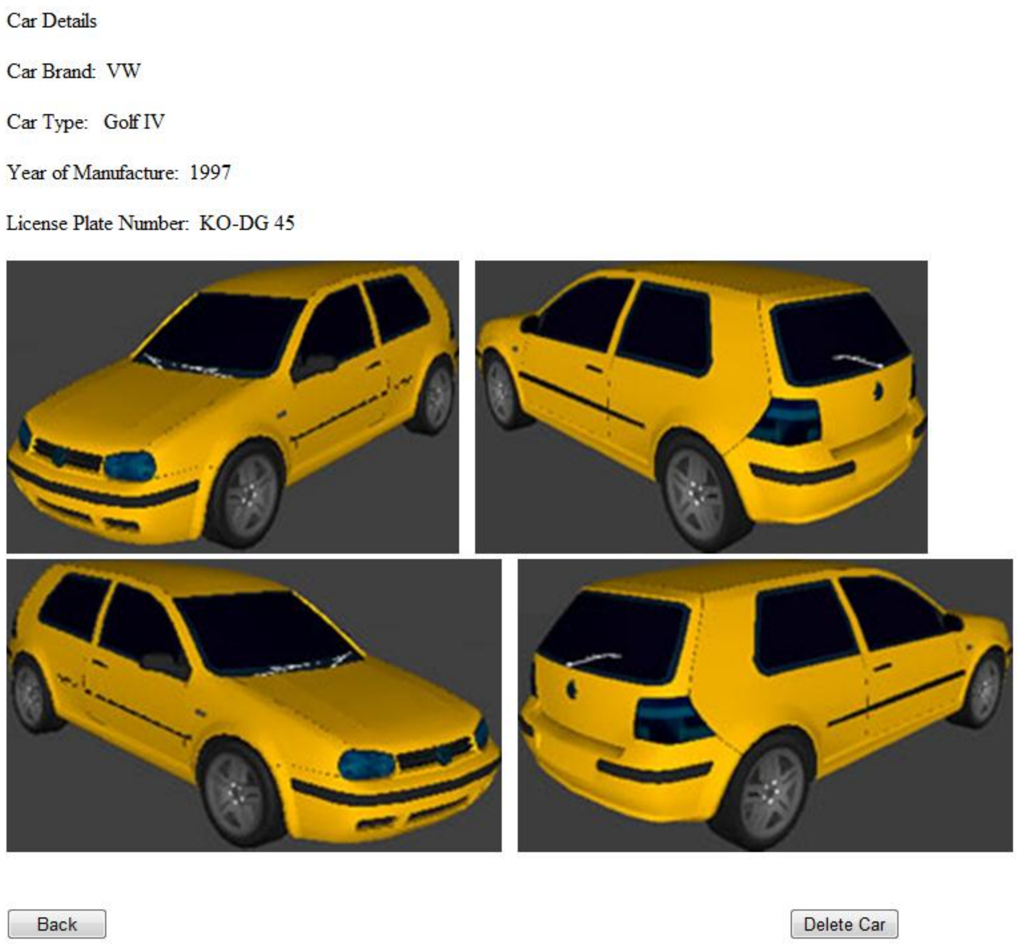


Figure 6.32: Container Data Diagram of Insurance Company Customer Data

Clicking on a customer in the above list (shown in figure 6.31) opens a detailed overview about the customer (see figure 6.34). It contains the name and address of the customer, his insurance number, which can also be changed via this form, a text field for inserting miscellaneous data, as well as a list with the customer's cars, another list with the customer's car repair companies, and finally a list with the accidents.

It is possible to add and delete cars to and from the customer (see figure 6.35). The car brand, type and year of manufacture as well as the license plate number are saved for each car. Clicking on a car in the list displays its details including pre-rendered images of the car model that belongs to the car (see figure 6.33).



Car Details

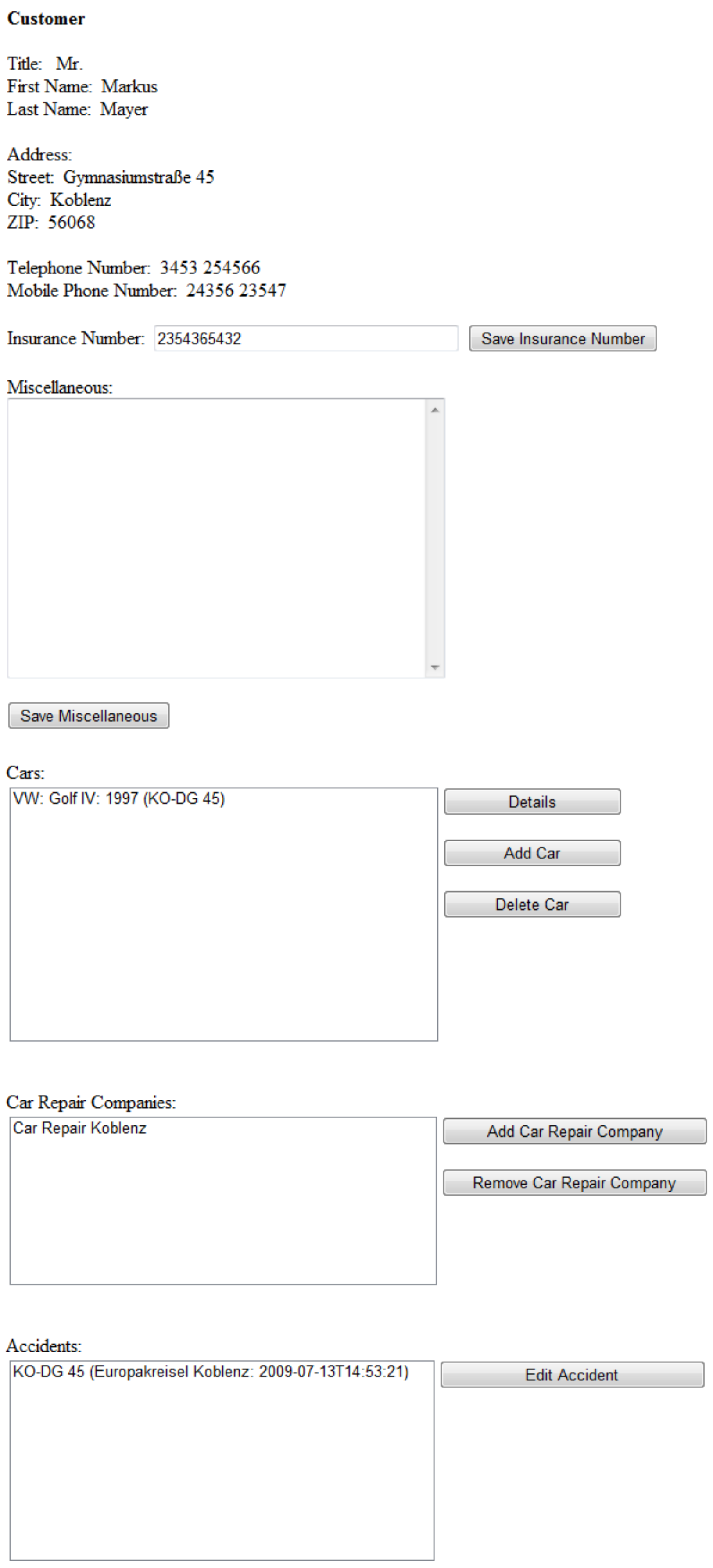
Car Brand: VW

Car Type: Golf IV

Year of Manufacture: 1997

License Plate Number: KO-DG 45

Figure 6.33: Car Details


Customer

Title: Mr.
First Name: Markus
Last Name: Mayer

Address:
Street: Gymnasiumstraße 45
City: Koblenz
ZIP: 56068

Telephone Number: 3453 254566
Mobile Phone Number: 24356 23547

Insurance Number: 2354365432

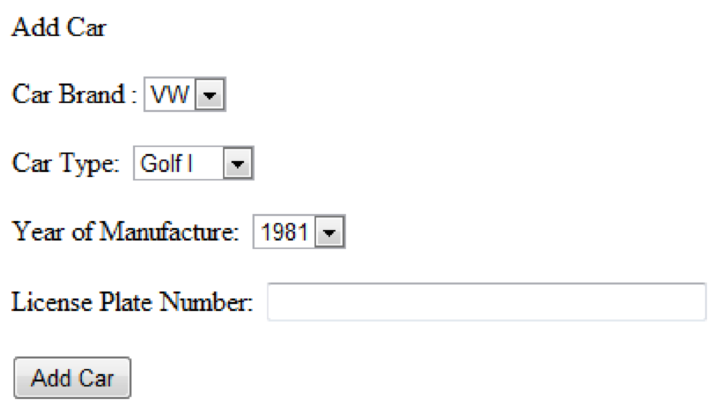
Miscellaneous:

Cars:
VW: Golf IV: 1997 (KO-DG 45)

Car Repair Companies:
Car Repair Koblenz

Accidents:
KO-DG 45 (Europakreisel Koblenz: 2009-07-13T14:53:21)

Figure 6.34: Manage Insurance Company Customer


Add Car

Car Brand : VW

Car Type: Golf I

Year of Manufacture: 1981

License Plate Number:

Add Car

Figure 6.35: Add Car

An insurance company employee can add a car repair company to a user. After this, the car repair company can access the user's data and create a damage evaluation for the user's cars. The Container shown in figure 6.36 saves the link between a user and a car repair company. It consists of two Entities: CarRepairCompanyID saves a link to the company and Username saves a link to the user.

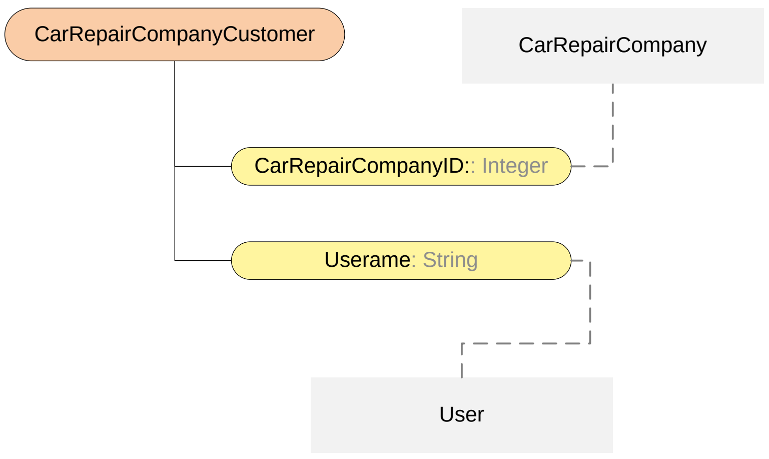
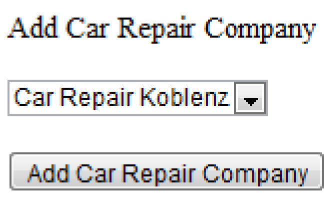


Figure 6.36: Container Data Diagram of Car Repair Company Customers

Figure 6.37 shows how a car-repair company can be added to a user:

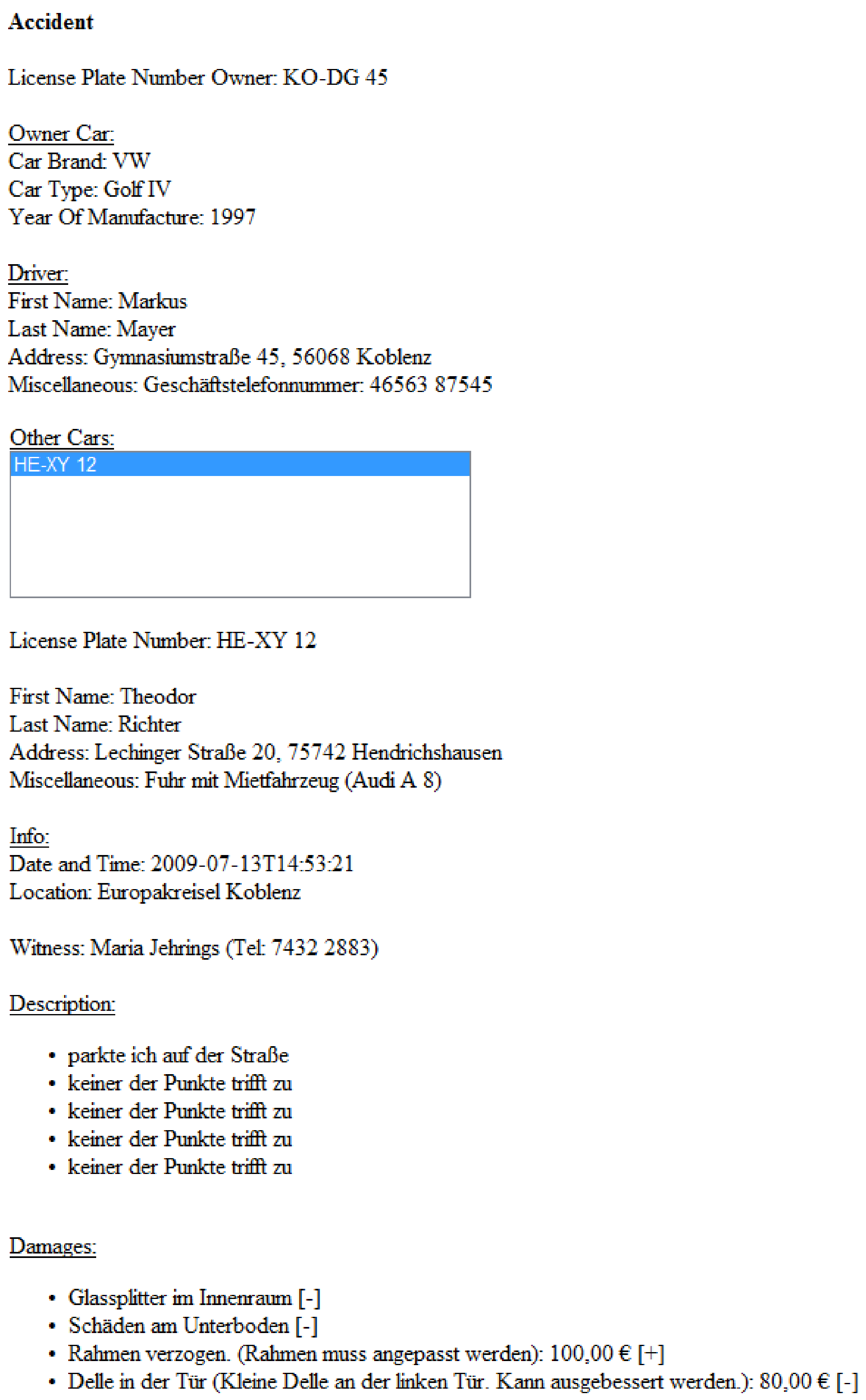

Add Car Repair Company

Car Repair Koblenz

Add Car Repair Company

Figure 6.37: Adding a car repair company to a user

Selecting an accident in the accidents list (figure 6.34) opens its details (see figure 6.38 and 6.39).



Accident

License Plate Number Owner: KO-DG 45

Owner Car:
Car Brand: VW
Car Type: Golf IV
Year Of Manufacture: 1997

Driver:
First Name: Markus
Last Name: Mayer
Address: Gymnasiumstraße 45, 56068 Koblenz
Miscellaneous: Geschäftstelefonnummer: 46563 87545

Other Cars:

HE-XY 12

License Plate Number: HE-XY 12

First Name: Theodor
Last Name: Richter
Address: Lechinger Straße 20, 75742 Hendrichshausen
Miscellaneous: Fuhr mit Mietfahrzeug (Audi A 8)

Info:
Date and Time: 2009-07-13T14:53:21
Location: Europakreisel Koblenz

Witness: Maria Jehrings (Tel: 7432 2883)

Description:

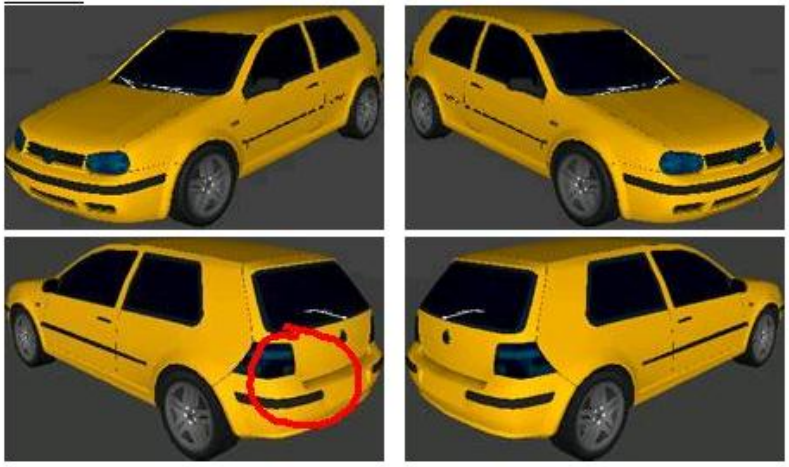
- parkte ich auf der Straße
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu

Damages:

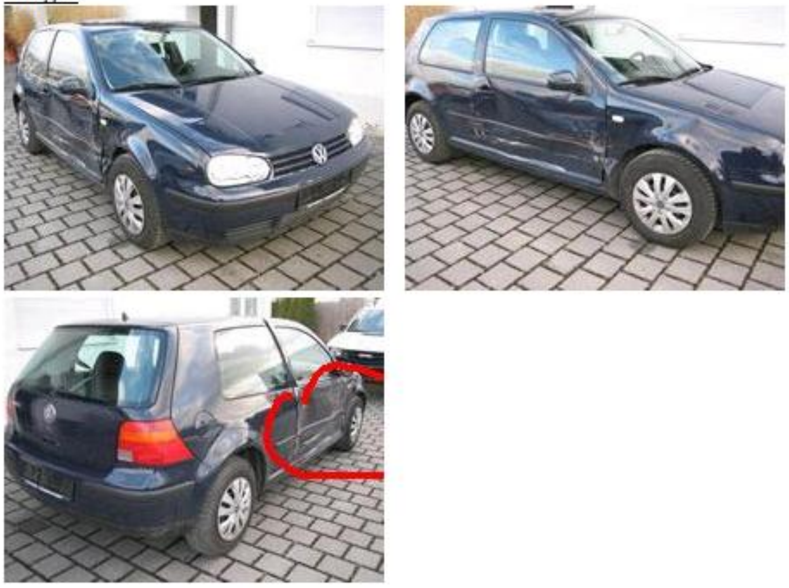
- Glassplitter im Innenraum [-]
- Schäden am Unterboden [-]
- Rahmen verzogen. (Rahmen muss angepasst werden): 100,00 € [+]
- Delle in der Tür (Kleine Delle an der linken Tür. Kann ausgebessert werden.): 80,00 € [-]

Figure 6.38: Accident Overview Part 1

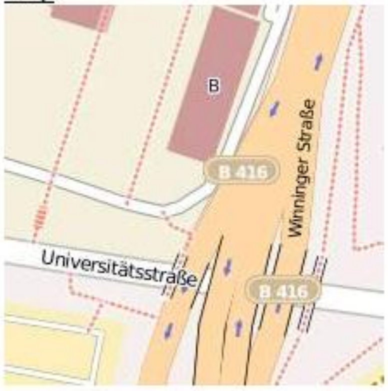
Models



Images



Map



Status: In Process

Show all Versions

Show Original XML Document

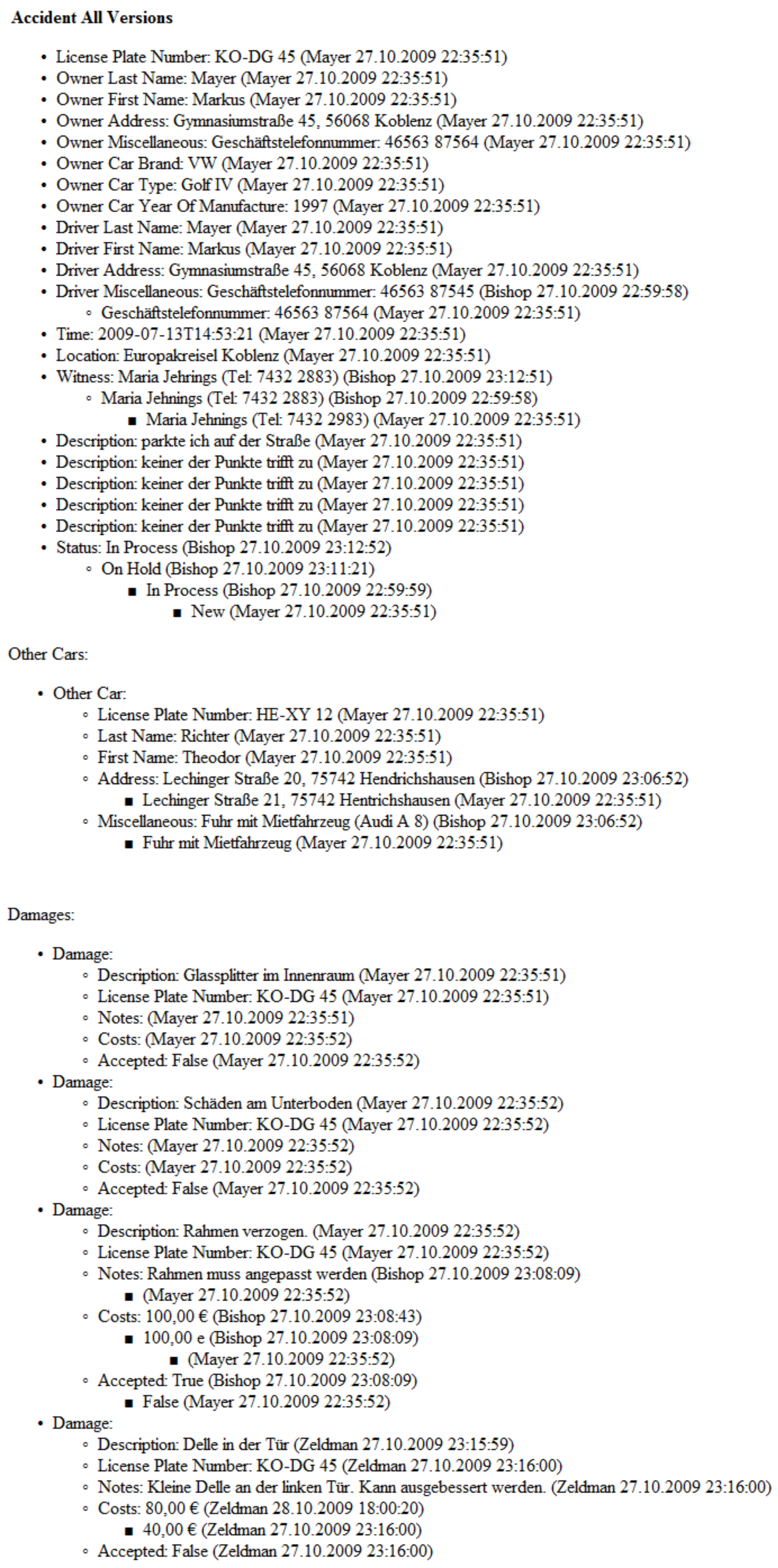
Figure 6.39: Accident Overview Part 2

The details about the accident contain the name and address of the car owner, as well as the car driver. A list exists containing all other cars that had been involved in the accident. Clicking on a car opens its details that contain the name and address as well as the license plate number of the car. In order to add random data, there is a miscellaneous field. The next section states the location and the time of the accident. Next a list of witnesses who saw the accident, followed by a list of descriptions of the accident is displayed. After this, the damages are listed. Each damage contains a short description. It can also contain the costs of repairing the damage and detailed notes about the damage. The minus or plus symbol indicates whether the insurance company has accepted the damage (plus) or not (minus). If a damage is accepted the insurance company will pay for it. After this, pre-rendered images of the model are displayed, including marks of the damages that had been made on the mobile client while creating the accident report. Below this, a list of photos can be found that can contain marks. The photos were also made on the mobile client and can help evaluate a damage or help recreating the accident. Then, a map of the location where the accident happened is displayed. This map can contain marks and cars that show where the cars involved in the accident were located. Last, the status of the accident case is displayed. This status can be changed by the insurance company and indicates if the case is *new*, *in process*, *on hold* or *finished*.

After this, two links can be found:

Show all Versions opens a page that lists all data about an accident case, including all versions of the data (see figure 6.40). That means that it is possible to access old versions of data that have been changed. Each version contains the value of the element, the creation date and the user who has created the element or who has changed the data. This way an insurance company employee can see which data has been changes at which time and who changed the data.

Show Original XML Document opens the xml document that has been sent by the mobile client (see figure 6.41). This can be handy if there are legal problems because of changes made to the report by insurance company employees. In such a case the original xml document can be used to see what has been send from the mobile client (in addition to the page that shows all versions of the data of an accident case). This can also be used to validate that no errors have been made while parsing the xml file and saving the data into the system.



Accident All Versions

- License Plate Number: KO-DG 45 (Mayer 27.10.2009 22:35:51)
- Owner Last Name: Mayer (Mayer 27.10.2009 22:35:51)
- Owner First Name: Markus (Mayer 27.10.2009 22:35:51)
- Owner Address: Gymnasiumstraße 45, 56068 Koblenz (Mayer 27.10.2009 22:35:51)
- Owner Miscellaneous: Geschäftstelefonnummer: 46563 87564 (Mayer 27.10.2009 22:35:51)
- Owner Car Brand: VW (Mayer 27.10.2009 22:35:51)
- Owner Car Type: Golf IV (Mayer 27.10.2009 22:35:51)
- Owner Car Year Of Manufacture: 1997 (Mayer 27.10.2009 22:35:51)
- Driver Last Name: Mayer (Mayer 27.10.2009 22:35:51)
- Driver First Name: Markus (Mayer 27.10.2009 22:35:51)
- Driver Address: Gymnasiumstraße 45, 56068 Koblenz (Mayer 27.10.2009 22:35:51)
- Driver Miscellaneous: Geschäftstelefonnummer: 46563 87545 (Bishop 27.10.2009 22:59:58)
    - Geschäftstelefonnummer: 46563 87564 (Mayer 27.10.2009 22:35:51)
- Time: 2009-07-13T14:53:21 (Mayer 27.10.2009 22:35:51)
- Location: Europakreisel Koblenz (Mayer 27.10.2009 22:35:51)
- Witness: Maria Jehrings (Tel: 7432 2883) (Bishop 27.10.2009 23:12:51)
    - Maria Jehnings (Tel: 7432 2883) (Bishop 27.10.2009 22:59:58)
        - Maria Jehnings (Tel: 7432 2983) (Mayer 27.10.2009 22:35:51)
- Description: parkte ich auf der Straße (Mayer 27.10.2009 22:35:51)
- Description: keiner der Punkte trifft zu (Mayer 27.10.2009 22:35:51)
- Description: keiner der Punkte trifft zu (Mayer 27.10.2009 22:35:51)
- Description: keiner der Punkte trifft zu (Mayer 27.10.2009 22:35:51)
- Description: keiner der Punkte trifft zu (Mayer 27.10.2009 22:35:51)
- Status: In Process (Bishop 27.10.2009 23:12:52)
    - On Hold (Bishop 27.10.2009 23:11:21)
        - In Process (Bishop 27.10.2009 22:59:59)
            - New (Mayer 27.10.2009 22:35:51)

Other Cars:

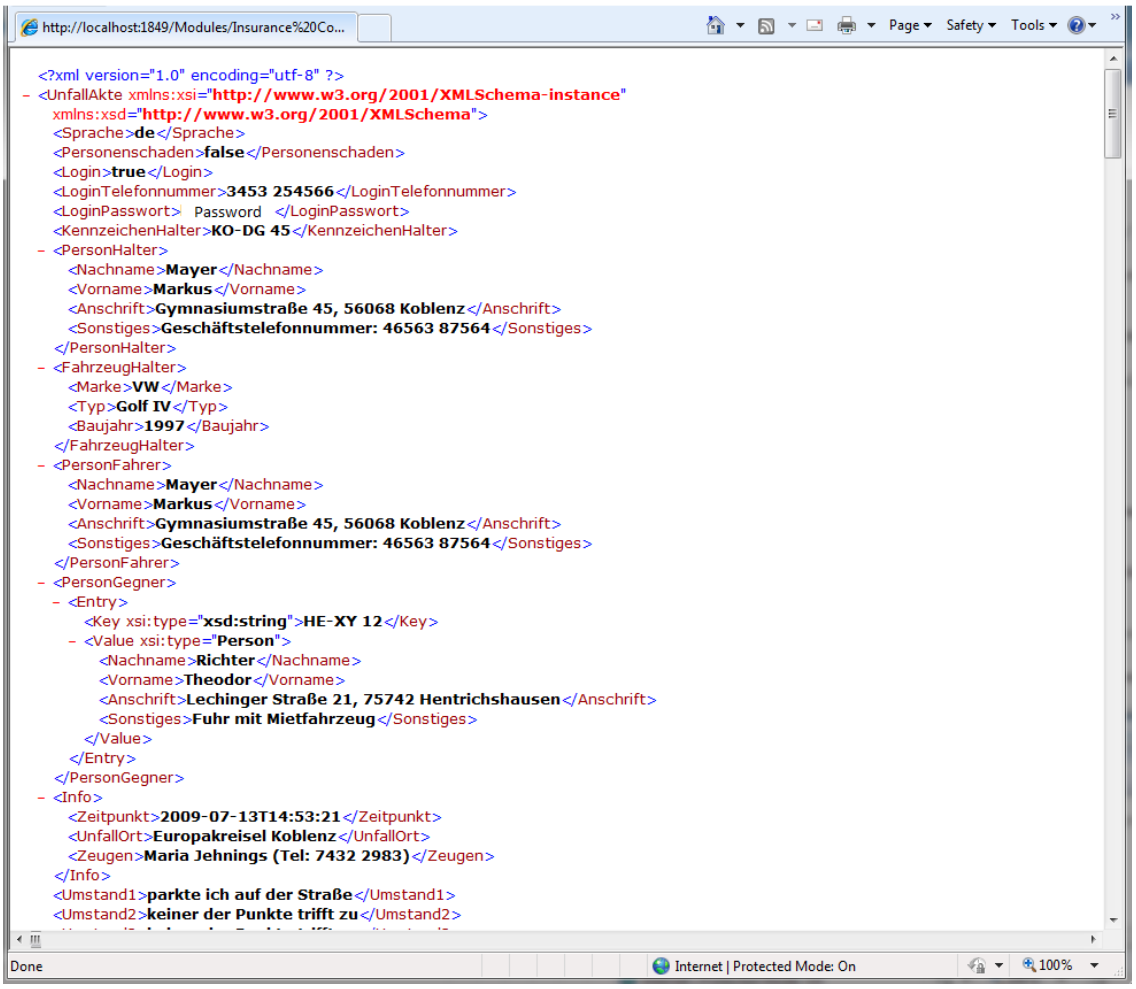
- Other Car:
    - License Plate Number: HE-XY 12 (Mayer 27.10.2009 22:35:51)
    - Last Name: Richter (Mayer 27.10.2009 22:35:51)
    - First Name: Theodor (Mayer 27.10.2009 22:35:51)
    - Address: Lechinger Straße 20, 75742 Hendrichshausen (Bishop 27.10.2009 23:06:52)
        - Lechinger Straße 21, 75742 Hentrichshausen (Mayer 27.10.2009 22:35:51)
    - Miscellaneous: Fuhr mit Mietfahrzeug (Audi A 8) (Bishop 27.10.2009 23:06:52)
        - Fuhr mit Mietfahrzeug (Mayer 27.10.2009 22:35:51)

Damages:

- Damage:
    - Description: Glassplitter im Innenraum (Mayer 27.10.2009 22:35:51)
    - License Plate Number: KO-DG 45 (Mayer 27.10.2009 22:35:51)
    - Notes: (Mayer 27.10.2009 22:35:51)
    - Costs: (Mayer 27.10.2009 22:35:52)
    - Accepted: False (Mayer 27.10.2009 22:35:52)
- Damage:
    - Description: Schäden am Unterboden (Mayer 27.10.2009 22:35:52)
    - License Plate Number: KO-DG 45 (Mayer 27.10.2009 22:35:52)
    - Notes: (Mayer 27.10.2009 22:35:52)
    - Costs: (Mayer 27.10.2009 22:35:52)
    - Accepted: False (Mayer 27.10.2009 22:35:52)
- Damage:
    - Description: Rahmen verzogen. (Mayer 27.10.2009 22:35:52)
    - License Plate Number: KO-DG 45 (Mayer 27.10.2009 22:35:52)
    - Notes: Rahmen muss angepasst werden (Bishop 27.10.2009 23:08:09)
        - (Mayer 27.10.2009 22:35:52)
    - Costs: 100,00 € (Bishop 27.10.2009 23:08:43)
        - 100,00 e (Bishop 27.10.2009 23:08:09)
            - (Mayer 27.10.2009 22:35:52)
    - Accepted: True (Bishop 27.10.2009 23:08:09)
        - False (Mayer 27.10.2009 22:35:52)
- Damage:
    - Description: Delle in der Tür (Zeldman 27.10.2009 23:15:59)
    - License Plate Number: KO-DG 45 (Zeldman 27.10.2009 23:16:00)
    - Notes: Kleine Delle an der linken Tür. Kann ausgebessert werden. (Zeldman 27.10.2009 23:16:00)
    - Costs: 80,00 € (Zeldman 28.10.2009 18:00:20)
        - 40,00 € (Zeldman 27.10.2009 23:16:00)
    - Accepted: False (Zeldman 27.10.2009 23:16:00)

Figure 6.40: Overview: Display all Versions of Data

Accident All Versions

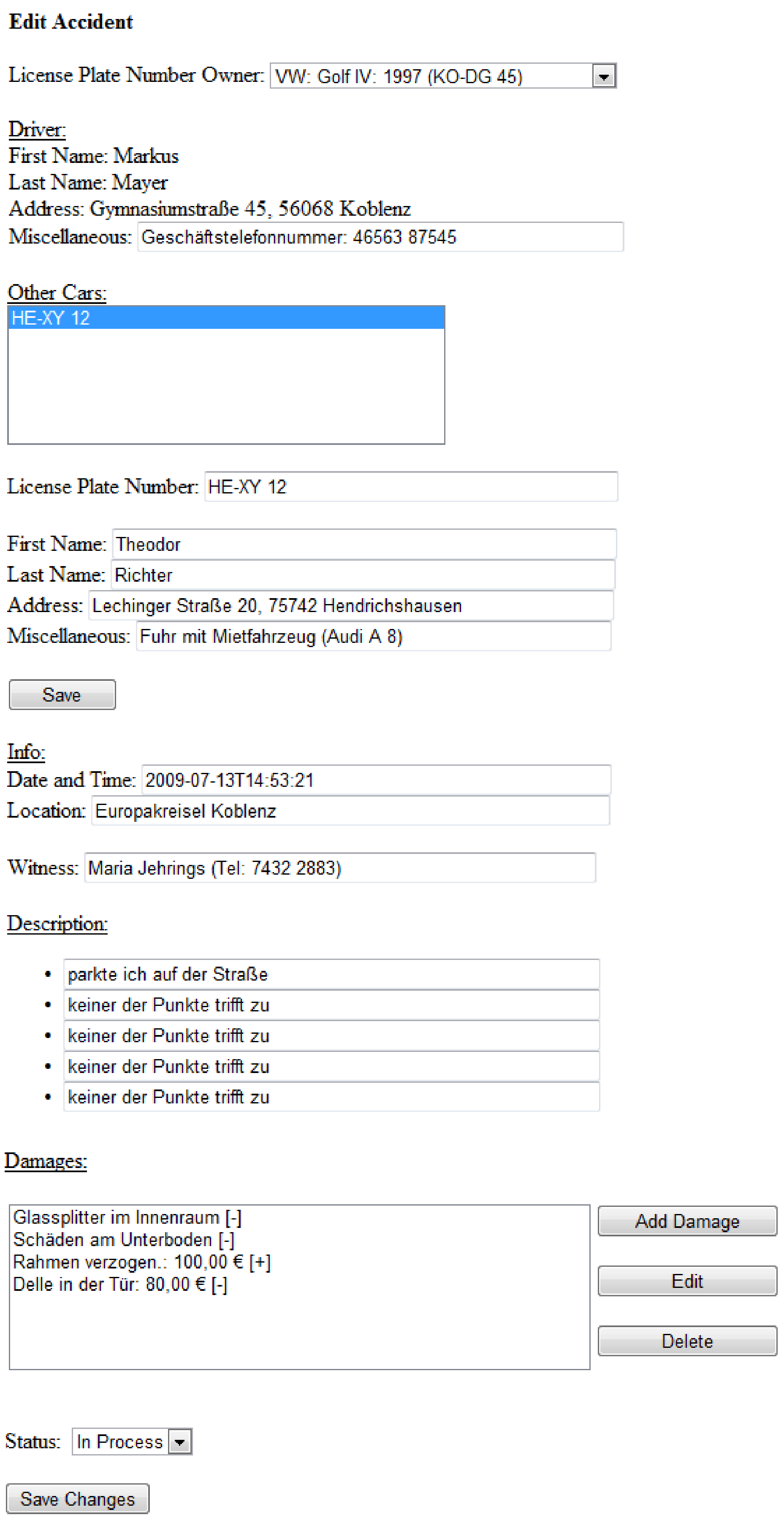

http://localhost:1849/Modules/Insurance%20Co...
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

```
  <?xml version="1.0" encoding="utf-8" ?>
- <UnfallAkte xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Sprache>de</Sprache>
    <Personenschaden>false</Personenschaden>
    <Login>true</Login>
    <LoginTelefonnummer>3453 254566</LoginTelefonnummer>
    <LoginPasswort> Password </LoginPasswort>
    <KennzeichenHalter>KO-DG 45</KennzeichenHalter>
  - <PersonHalter>
      <Nachname>Mayer</Nachname>
      <Vorname>Markus</Vorname>
      <Anschrift>Gymnasiumstraße 45, 56068 Koblenz</Anschrift>
      <Sonstiges>Geschäftstelefonnummer: 46563 87564</Sonstiges>
    </PersonHalter>
  - <FahrzeugHalter>
      <Marke>VW</Marke>
      <Typ>Golf IV</Typ>
      <Baujahr>1997</Baujahr>
    </FahrzeugHalter>
  - <PersonFahrer>
      <Nachname>Mayer</Nachname>
      <Vorname>Markus</Vorname>
      <Anschrift>Gymnasiumstraße 45, 56068 Koblenz</Anschrift>
      <Sonstiges>Geschäftstelefonnummer: 46563 87564</Sonstiges>
    </PersonFahrer>
  - <PersonGegner>
    - <Entry>
        <Key xsi:type="xsd:string">HE-XY 12</Key>
      - <Value xsi:type="Person">
          <Nachname>Richter</Nachname>
          <Vorname>Theodor</Vorname>
          <Anschrift>Lechinger Straße 21, 75742 Hentrichshausen</Anschrift>
          <Sonstiges>Fuhr mit Mietfahrzeug</Sonstiges>
        </Value>
      </Entry>
    </PersonGegner>
  - <Info>
      <Zeitpunkt>2009-07-13T14:53:21</Zeitpunkt>
      <UnfallOrt>Europakreisel Koblenz</UnfallOrt>
      <Zeugen>Maria Jehnings (Tel: 7432 2983)</Zeugen>
    </Info>
    <Umstand1>parkte ich auf der Straße</Umstand1>
    <Umstand2>keiner der Punkte trifft zu</Umstand2>
```

Figure 6.41: Show Original XML Document

By clicking the edit button at the right side of the accidents list (see figure 6.34) the insurance company employee can change the data of an accident case (see figure 6.42). The name and address of the driver and car owner cannot be changed, because they are managed by the central organization that is responsible for administrating the back-office of the electronic version of the European Accident Report.

Damages can be added, deleted and changed. The insurance company employee can write a short description, a longer note and add the costs of the damage. He can also decide whether the insurance company will pay for the damage repair by checking the accepted check box (see figure 6.43).



Edit Accident

License Plate Number Owner: VW: Golf IV: 1997 (KO-DG 45)

Driver:
First Name: Markus
Last Name: Mayer
Address: Gymnasiumstraße 45, 56068 Koblenz
Miscellaneous: Geschäftstelefonnummer: 46563 87545

Other Cars:
HE-XY 12

License Plate Number: HE-XY 12

First Name: Theodor
Last Name: Richter
Address: Lechinger Straße 20, 75742 Hendrichshausen
Miscellaneous: Fuhr mit Mietfahrzeug (Audi A 8)

Info:
Date and Time: 2009-07-13T14:53:21
Location: Europakreisel Koblenz

Witness: Maria Jehrings (Tel: 7432 2883)

Description:

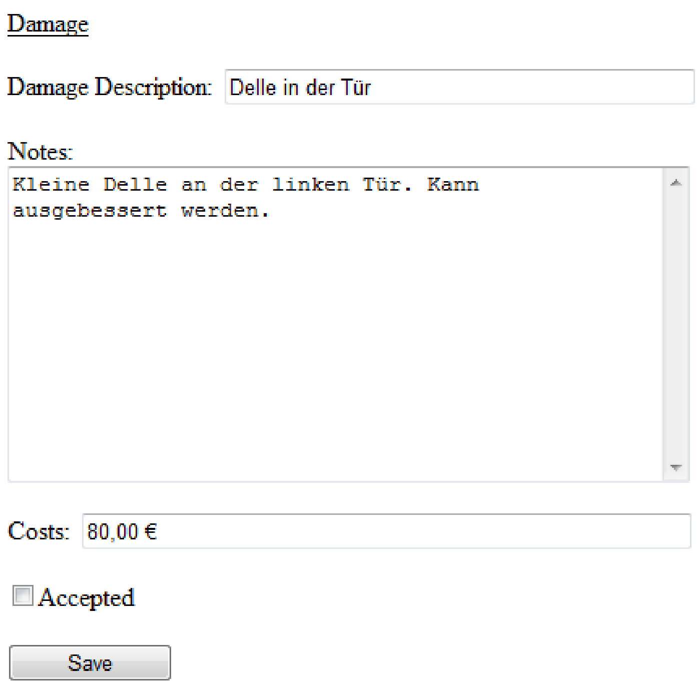
- parkte ich auf der Straße
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu

Damages:

Glassplitter im Innenraum [-]
Schäden am Unterboden [-]
Rahmen verzogen.: 100,00 € [+]
Delle in der Tür: 80,00 € [-]

Status: In Process

Figure 6.42: Edit Accident


Damage

Damage Description: Delle in der Tür

Notes:

Kleine Delle an der linken Tür. Kann ausgebessert werden.

Costs: 80,00 €

☐ Accepted

Figure 6.43: Edit Damage

6.6 Car Owner

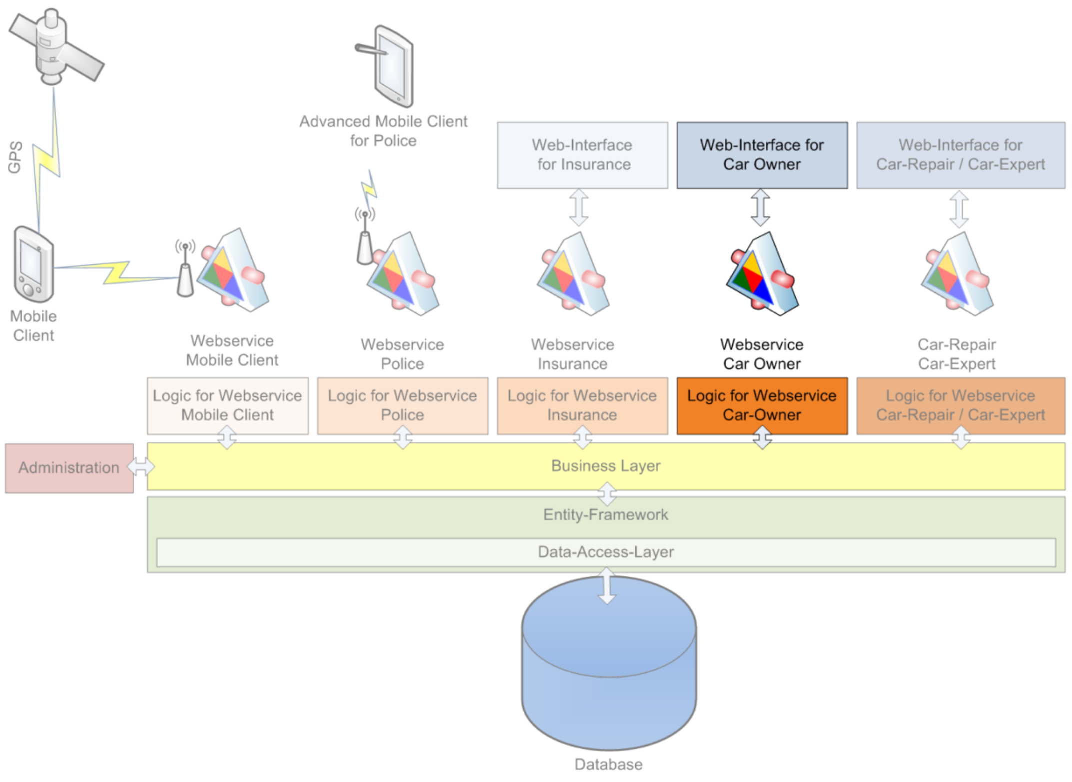

GPS
Advanced Mobile Client for Police
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Mobile Client
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.44: The Car Owner Module

The module for the car owner provides a web service which can be used to access data about accidents by the car owner.

6.6.1 Web Service

The web service for the car owner consists of several methods via which a car owner can get an overview of his accidents and manage his car repair companies. Table 6.6 lists all the methods of the web service for the car owner:

Method Name	Parameters	Return Type
GetCarRepairCompaniesForUser	username: string password: string	CarRepairCompany[]
GetCarRepairCompanies		CarRepairCompany[]
AddCarRepairCompanyToUser	carRepairCompany: CarRepairCompany username: string password: string	
RemoveCarRepairCompanyFromUser	carRepairCompany: CarRepairCompany	
GetOwnerData	username: string password: string	Owner
GetAccidentsForUser	username: string password: string	Accident[]
GetAccident	id: int username: string password: string	Accident
GetOtherCar	id: int username: string password: string	Accident.OtherCar
GetModel	id: int username: string password: string	Accident.Model
GetPhoto	id: int username: string password: string	Accident.Photo
GetMap	id: int username: string password: string	Accident.Map

Table 6.6: Web Service for the Car Owner

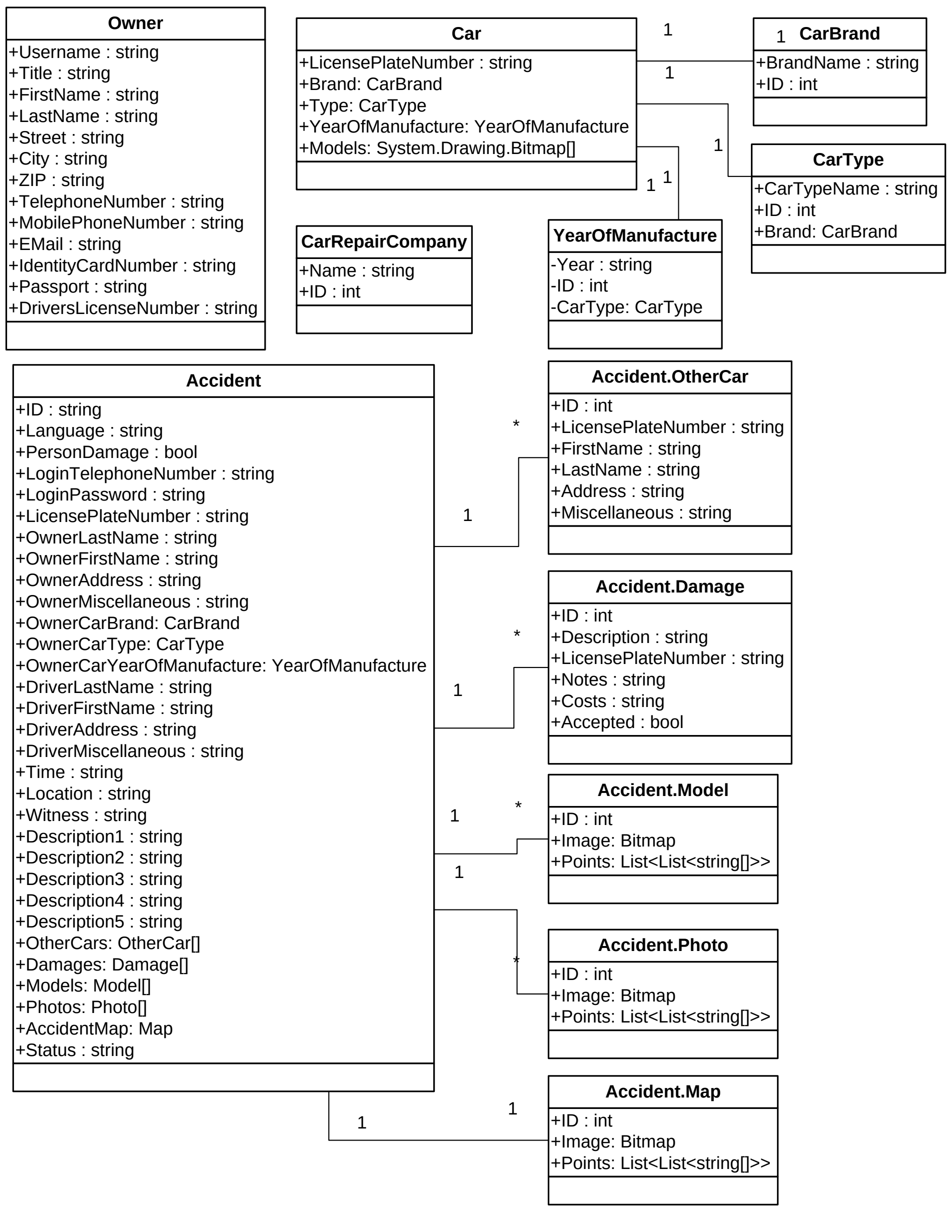


Figure 6.45: UML Diagram of Classes used by the Web Service for the Car Owner

The web service uses several classes in order to save and transfer data. Those classes are shown in figure 6.45. The Owner class saves data about the car owner. The Accident class, which contains the OtherCar, Damage, Model, Photo and Map class, saves all data about an accident case. The Car class represents a car.

Methods in the web service get the username and password of the car owner who is calling a method and validates if they are correct.

The web service methods call methods of the car owner model which retrieve data, save data, change data or delete data. This helps to keep the web service methods clean. They only contain the authentication and authorization code as well as the method calls to the module methods. The module (CarOwner.cs) contains all the functionality needed.

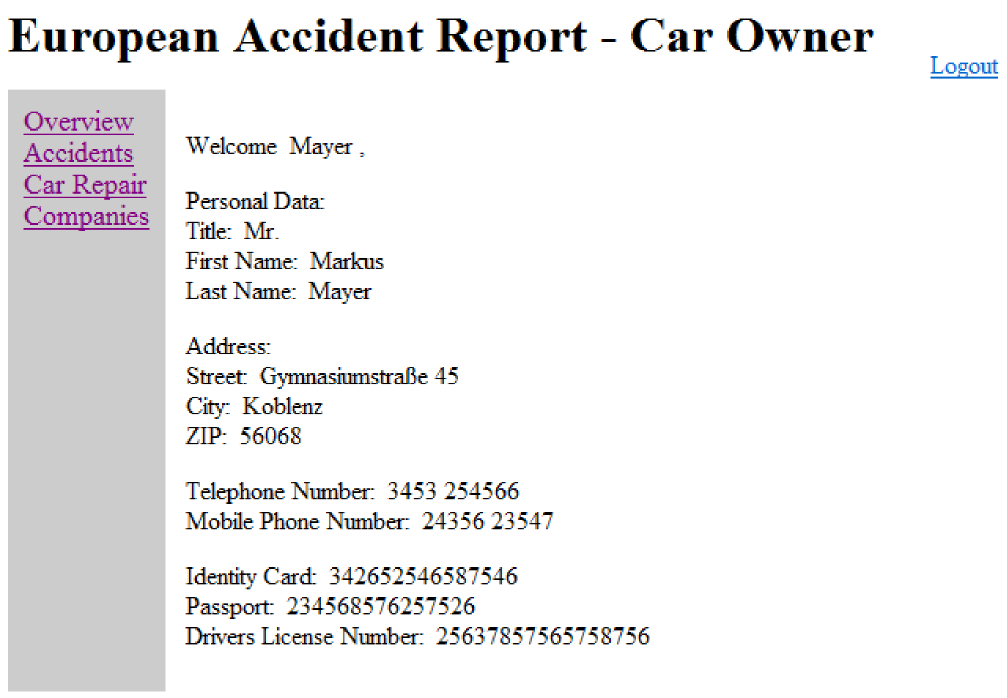
Table 6.7 lists all the methods of the car owner module.

Method Name	Parameters	Return Type
FillCarRepairCompanies		List<CarRepairCompany>
FillCarRepairCompanies ForUser	username: string	List<CarRepairCompany>
SaveCarRepairCompany	carRepairCompany: CarRepairCompany username: string	
RemoveCarRepairCompany	carRepaircompany: CarRepairCompany username: string	
FillCarBrands		List<CarBrand>
FillCarTypes	carBrand: CarBrand	List<CarType>
FillYearsOfManufacture	carType: CarType	List<YearOfManufacture>
FillOwnerData	username: string	Owner
FillAccidentsForUser	username: string	List<Accident>
FillAccident	id: string	Accident
FillOtherCar	id: int	Accident.OtherCar
FillModel	id: int	Accident.Model
FillPhoto	id: int	Accident.Photo
FillMap	id: int	Acciden.Map

Table 6.7: Car Owner Module Methods

6.6.2 Web Application

The car owner can see his personal information in the web application. This includes his name, address, contact informaiton, and identity car number, passport number and driver's license number (see figure 6.46).



Welcome Mayer ,

Personal Data:
Title: Mr.
First Name: Markus
Last Name: Mayer

Address:
Street: Gymnasiumstraße 45
City: Koblenz
ZIP: 56068

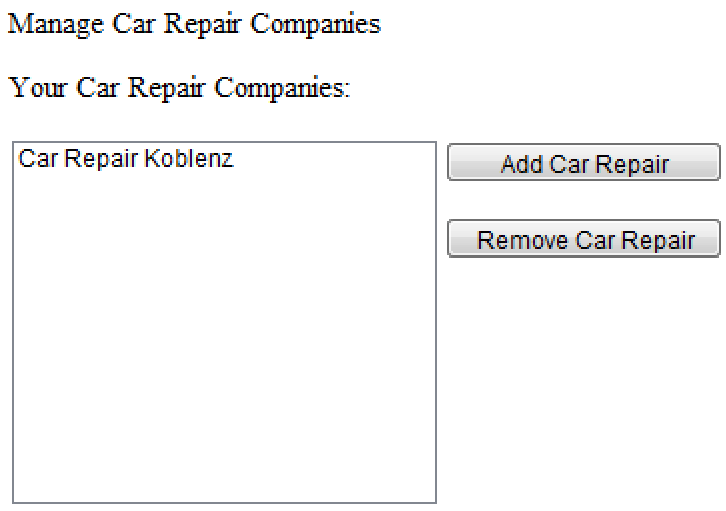
Telephone Number: 3453 254566
Mobile Phone Number: 24356 23547

Identity Card: 342652546587546
Passport: 234568576257526
Drivers License Number: 2563785765758756

Figure 6.46: Car Owner Overview

A car owner can add and remove car repair companies. If a company is added, it can access the user's data and can evaluate damage caused by an accident. Both car owners and insurance companies can choose which car repair companies can access the data. See chapter 6.5.2 for more information about how the link between car repair companies and users is saved in a Container.

Figure 6.47 shows how car repair companies can be managed. It contains a button for adding and a button for removing a car repair company. It displays all car repair companies in a list that are allowed to access the user's data.

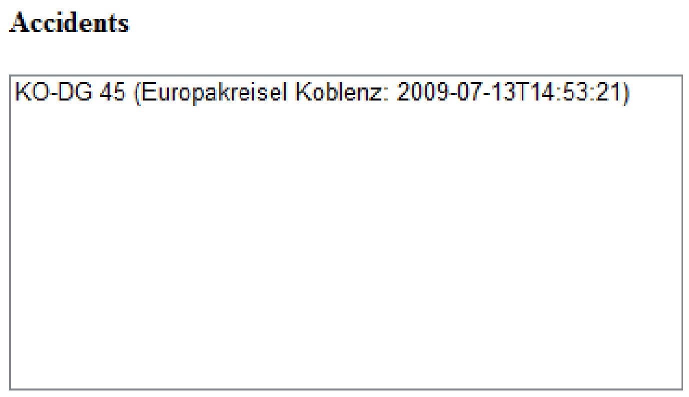

Manage Car Repair Companies

Your Car Repair Companies:

Car Repair Koblenz

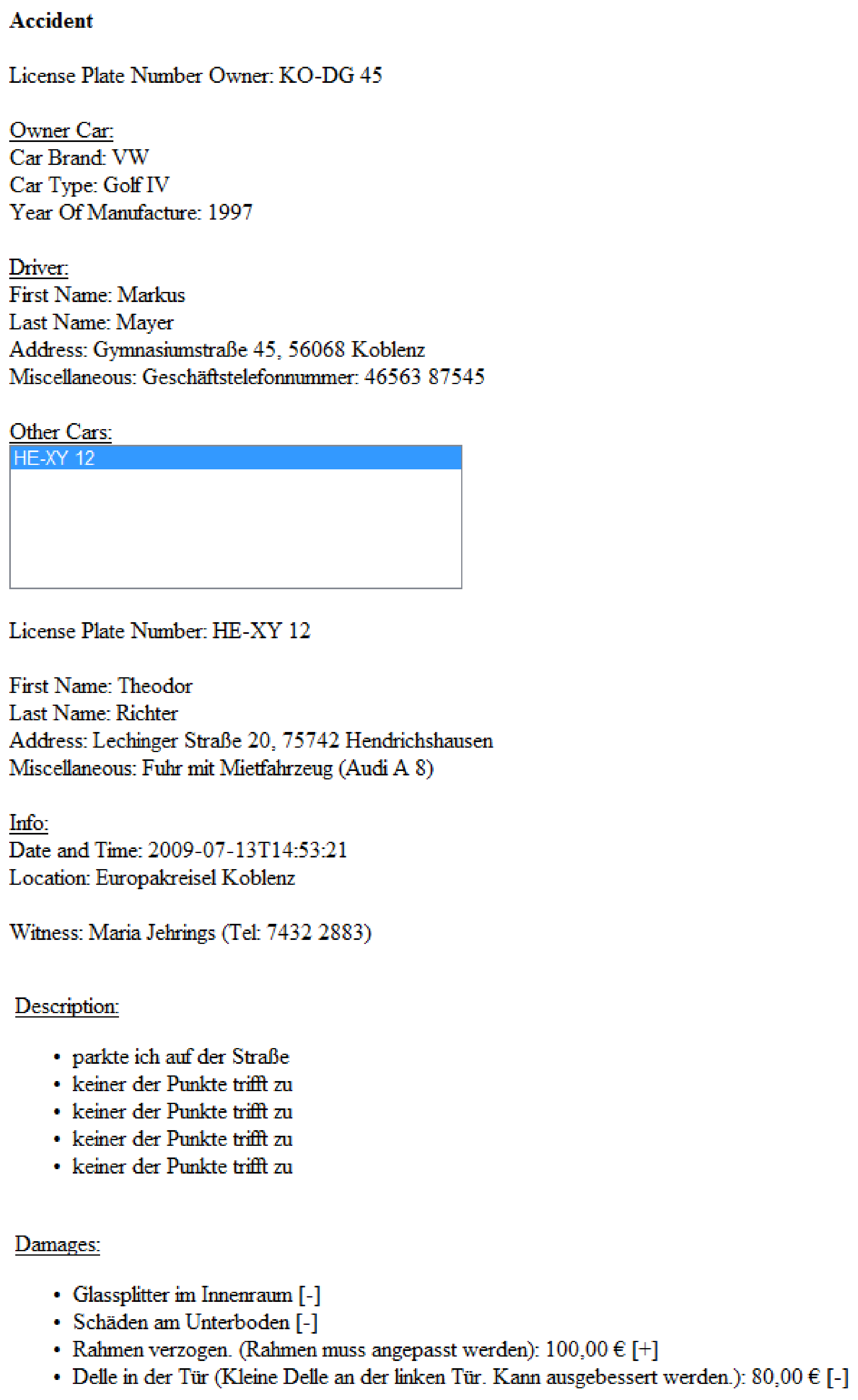
Figure 6.47: Managing Car Repair Companies

The car owner can also view his accident cases (see figure 6.48 and 6.49/6.50). The same information is displayed here as in the web application for the insurance company employee (see chapter 6.5.2 for a description of the different fields of the overview of the accident case). The car owner can also see the status of the accident case and therefore knows what his insurance company is doing.


Accidents

KO-DG 45 (Europakreisel Koblenz: 2009-07-13T14:53:21)

Figure 6.48: Overview Accidents



Accident

License Plate Number Owner: KO-DG 45

Owner Car:
Car Brand: VW
Car Type: Golf IV
Year Of Manufacture: 1997

Driver:
First Name: Markus
Last Name: Mayer
Address: Gymnasiumstraße 45, 56068 Koblenz
Miscellaneous: Geschäftstelefonnummer: 46563 87545

Other Cars:

HE-XY 12

License Plate Number: HE-XY 12

First Name: Theodor
Last Name: Richter
Address: Lechinger Straße 20, 75742 Hendrichshausen
Miscellaneous: Fuhr mit Mietfahrzeug (Audi A 8)

Info:
Date and Time: 2009-07-13T14:53:21
Location: Europakreisel Koblenz

Witness: Maria Jehrings (Tel: 7432 2883)

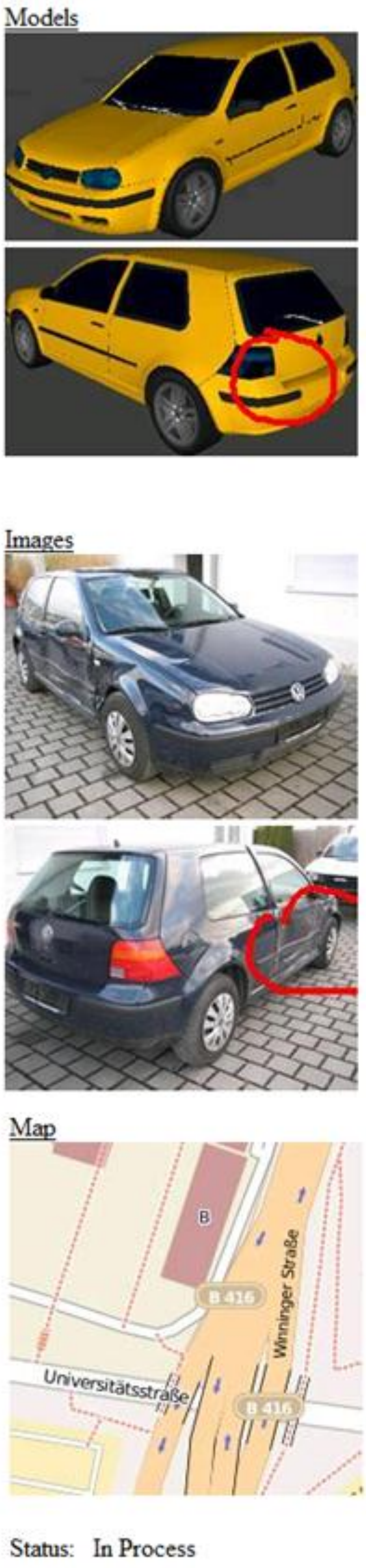
Description:

- parkte ich auf der Straße
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu
- keiner der Punkte trifft zu

Damages:

- Glassplitter im Innenraum [-]
- Schäden am Unterboden [-]
- Rahmen verzogen. (Rahmen muss angepasst werden): 100,00 € [+]
- Delle in der Tür (Kleine Delle an der linken Tür. Kann ausgebessert werden.): 80,00 € [-]

Figure 6.49: Overview of Accident for Car Owner Part 1



Status: In Process

Figure 6.50: Overview of Accident for Car Owner Part 2

6.7 Car Repair

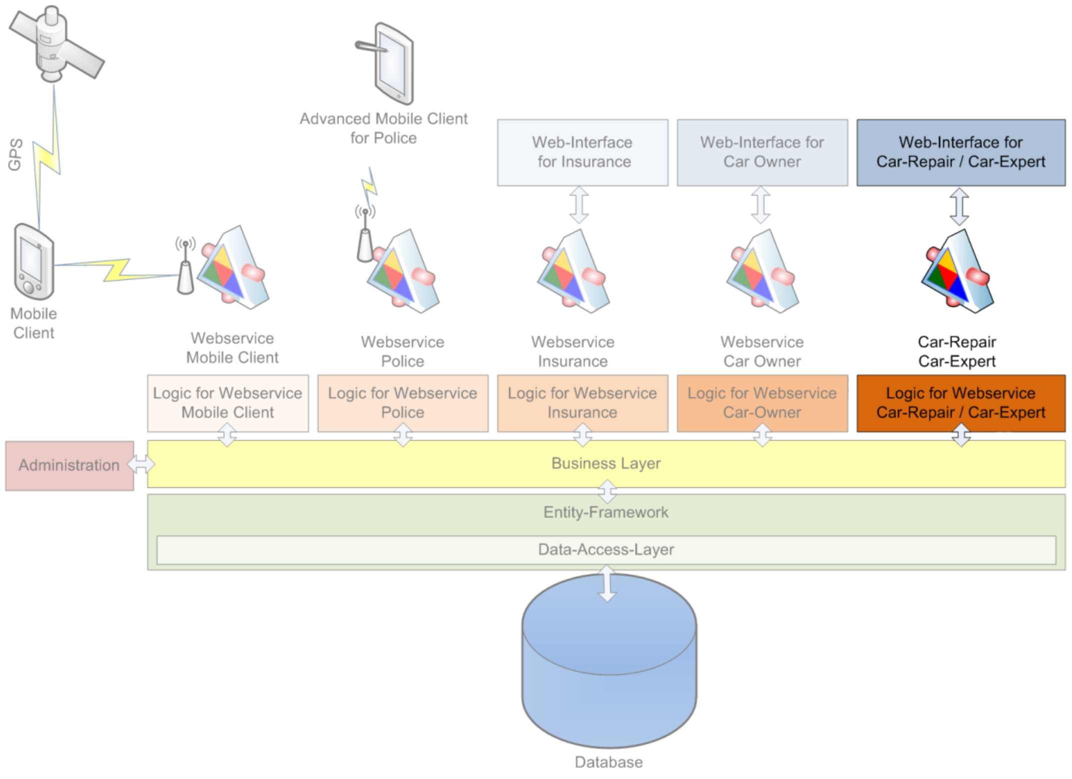

GPS
Mobile Client
Advanced Mobile Client for Police
Web-Interface for Insurance
Web-Interface for Car Owner
Web-Interface for Car-Repair / Car-Expert
Webservice Mobile Client
Webservice Police
Webservice Insurance
Webservice Car Owner
Car-Repair Car-Expert
Logic for Webservice Mobile Client
Logic for Webservice Police
Logic for Webservice Insurance
Logic for Webservice Car-Owner
Logic for Webservice Car-Repair / Car-Expert
Administration
Business Layer
Entity-Framework
Data-Access-Layer
Database

Figure 6.51: The Car Repair Module

The module for the car repair company provides a web service which can be used in order to access and manage damages of customers.

6.7.1 Web Service

The web service for the car repair company consists of several methods via which a car repair company can manage customers and manage damages. Table 6.8 lists all the methods of the web service for the car repair company:

Method Name	Parameters	Return Type
GetCustomers	carRepairCompanyID: int	Customer[]
GetCarsForUser	username: string employeeUsername: string password: string	Car[]
GetAccidentsByCar	licensePlateNumber: string employeeUsername: string password: string	Accident[]
GetDamagesByAccident	accidentID: string employeeUsername: string password: string	Damage[]
GetDamage	id: int employeeUsername: string password: string	Damage
DeleteDamage	damage: Damage	

	employeeUsername: string password: string
AddDamage	damage: Damage employeeUsername: string password: string
ChangeDamage	damage: Damage employeeUsername: string password: string

Table 6.8: Web Service for the Car Repair Company

The web service uses several classes in order to save and transfer data. Those classes are shown in figure 6.52. The Customer class saves data about a customer of the car repair company. The Damage class, which belongs to the Accident class, saves all data about a damage.
The Car class represents a car.

Methods in the web service get the username and the password of the car repair company employee who is calling a method and validates if they are correct. The methods also check whether a user is allowed to access the data or do the action he is about to do.

The web service methods call methods of the car repair company model which retrieve data, save data, change data or delete data. This helps to keep the web service methods clean. They only contain the authentication and authorization code as well as the method calls to the module methods. The module (CarRepairCompany.cs) contains all the functionality needed.

Table 6.9 lists all the methods of the car repair company module.

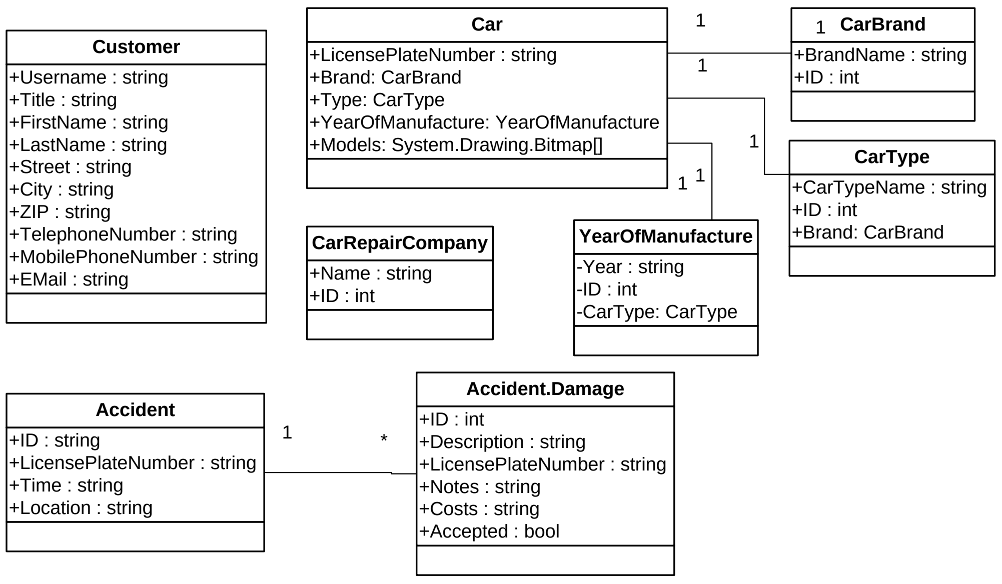


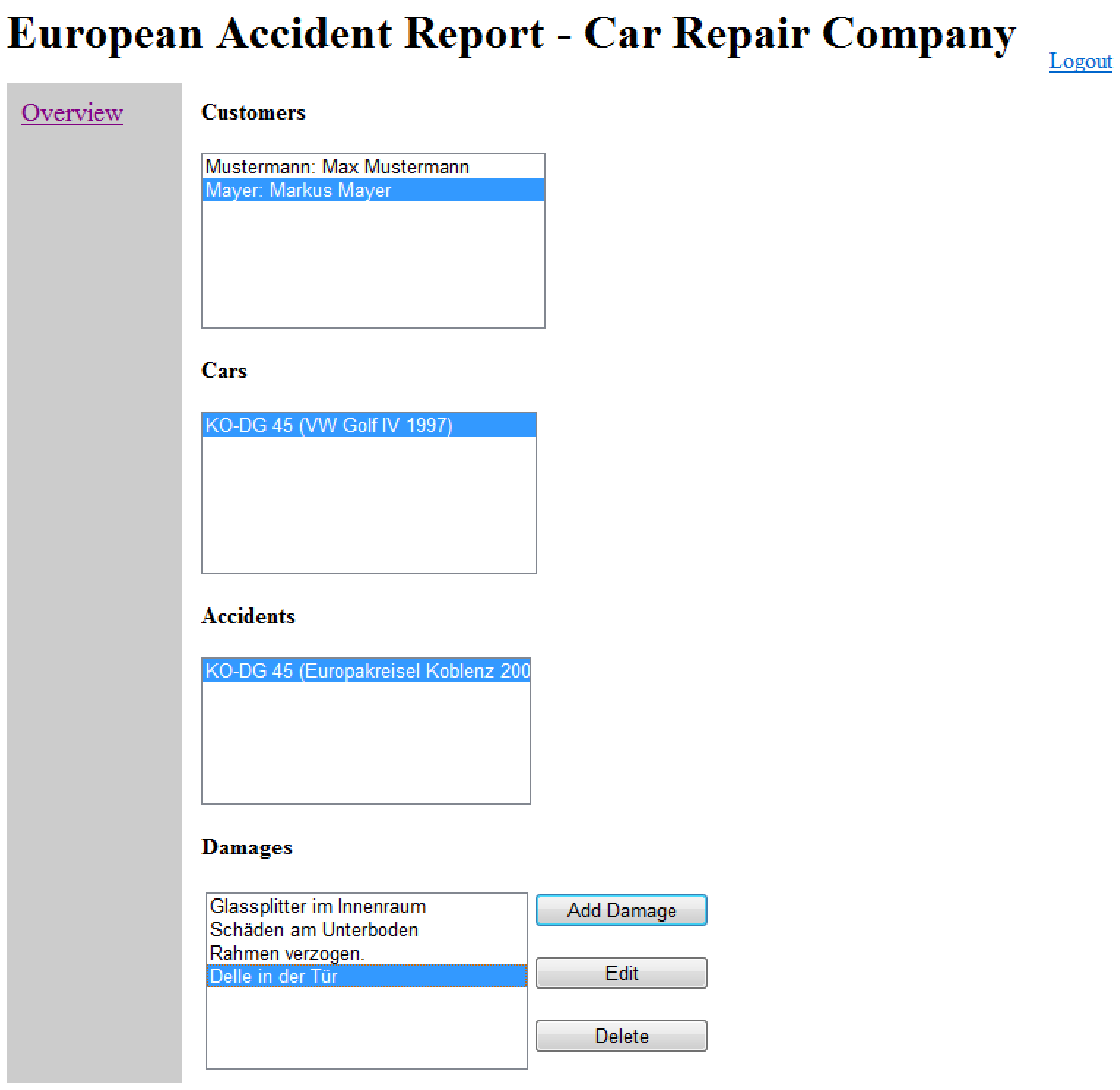
Figure 6.52: UML Diagram of Classes used by the Web Service for the Car Repair Company

Method Name	Parameters	Return Type
FillCustomer	customerUsername	Customer
FillCustomers	insuranceCompanyID: int	List<Customer>
FillCarsForUser	username: string	List<Car>
FillAccidentsByCar	licensePlateNumber: string	List<Accident>
FillDamagesbyAccident	accidentID: string	List<Damage>
FillDamage	id: int	Damage
DeleteDamage	damage: Damage employeeUsername: string	
SaveDamage	damage: Damage employeeUsername: string	
SaveDamageChanges	damage: Damage employeeUsername: string	
CheckIfEmployeeBelongs ToCarRepairCOmpany	employeeUsername: string carRepairCompanyID: int	bool
CheckIfCarRepairCompany EmployeeIsAllowedTo SeeDataOfUser	customerUsername: string employeeUsername: string	bool
FillUsernameByLicensePlate Number	licensePlateNumber: string	string
FillUsernameOfDamage	damage: Damage	string

Table 6.9: Car Repair Company Module Methods

6.7.2 Web Application

The web application for the car repair company enables a car repair company employee to add, remove and edit damages (see figure 6.53). First a list with all customers of the car repair company is displayed. Selecting one of them shows a list with the customer's cars. When a car is selected all accidents that this car has been involved in are displayed. Selecting an accident shows a list of all damages that occurred in this accident. The car repair company employee can now add new damages, delete damages or edit damages.


European Accident Report - Car Repair Company

Logout

Overview

Customers

Mustermann: Max Mustermann
Mayer: Markus Mayer

Cars

KO-DG 45 (VW Golf IV 1997)

Accidents

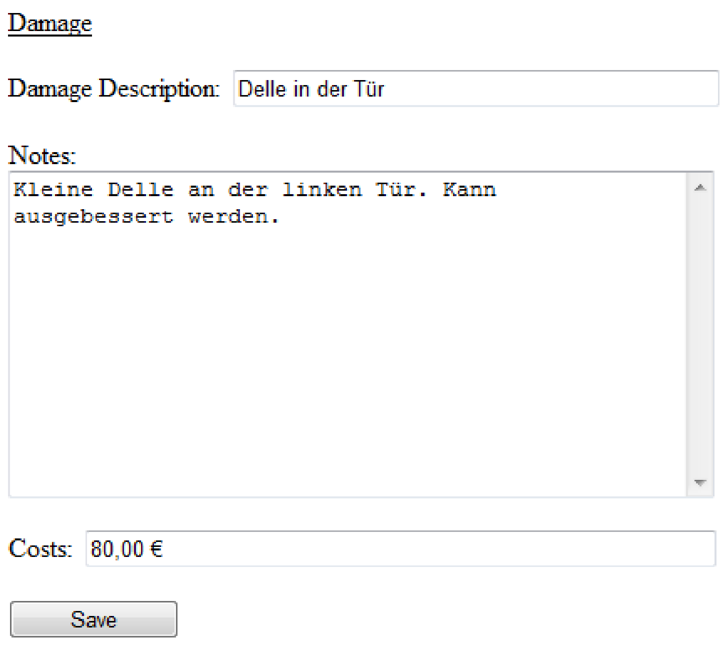
KO-DG 45 (Europakreisel Koblenz 200

Damages

Glassplitter im Innenraum
Schäden am Unterboden
Rahmen verzogen.
Delle in der Tür

Figure 6.53: Car Repair Company Overview

When adding or editing a damage, the car repair company employee can add a short description, a longer note and the costs of repairing the damage (see figure 6.54).


Damage

Damage Description: Delle in der Tür

Notes:

Kleine Delle an der linken Tür. Kann ausgebessert werden.

Costs: 80,00 €

Figure 6.54: Edit Damage

7 Conclusion

This chapter provides a summary of this bachelor thesis and describes what functionality can be added to the system in the future. It also contains improvement ideas.

7.1 Summary and Results

The electronic version of the European Accident Report, which can be used by filling a form on a mobile phone, makes it easier to document an accident case, because the user is guided through the form on a step by step basis and can therefore not forget to include data that is needed by the insurance company to process the accident. The task of this bachelor thesis was to create a back-office in which all data needed by an insurance company is saved. This data should be versioned, that means it must be possible to see when changes have been made and who made those changes.

A generic framework has been created as a result of this bachelor thesis that can be used to save data related to accident cases, access, update and query this data, as well as manage users. New functionality can be added to this system by adding modules. Those can save all data needed by this module in the database, without needing to change the schema information of the database. This is done by using the container-entity-model of the core system. This makes it easy to add new modules to the system, because the module developer does not need to know how the internals of the framework work; developers only need to learn the API of the framework. The data, which can be saved into the database, can be of any type, which makes it possible to save various kinds of data, like images, videos or even 3D-models into the database.

Five modules have been created that can be used to process accident cases. They can also be seen as examples of how modules can be developed.

The mobile client module provides a web service via which the mobile phone can get data about a person and his cars and send the accident report to the system.

The police client module offers a web service via which the police client can access the data it needs.

The insurance company module provides a web service that enables an insurance company to integrate the system into their software infrastructure. A web application has also been implemented that offers basic functionality for processing accident cases.

The car repair company module consists of a web service as well as a web application that can be used by car repair companies to add and evaluate damages.

The car owner module, provides both a web service as well as a web application that can be used to see the status of an accident case of the car owner. The car owner can also choose car repair companies via this web application.

Another web application has been created that can be used by administrators of the central system to manage users, car repair companies, insurance companies and car models.

7.2 Outlook

In this bachelor thesis a generic framework has been created that makes it easy to create modules that use the functionality of this core system in order to save, query and update data from and to the database via the use of an abstract model (container-entity-model). This system is fully working, but performance improvements could be made to make the system faster and more scalable. This could be achieved by implementing lazy loading to the querying mechanism. Today Containers are loaded that contain all the entities including old versions of entities. This makes it easy to use the Containers, but it creates overhead. By adding lazy loading to the system, this could be avoided.

An advanced querying mechanism could be added to the system. Today it is possible to query data by the Container name, Container id, Entity id and by the value of an Entity. It is possible to load every container needed from the database by using those methods, but this is not always easy and convenient. A method that could return all Containers with a certain name that include an Entity with a certain value would be useful (today the developer must manually check if the Container, which is returned, is of the type he wants). It would also be useful to be able to query Containers and Entities by using an SQL-like syntax. This way it would be possible to return Containers that include several Entities that match certain conditions. This would make it easier to query data.

The modules have been developed as examples of how modules can be developed. They only have a basic user interface. This could be improved. More functionality could be added to the system, e.g. it would be helpful for the car owner to be able to interact with the insurance company. Today the car owner can only see the accident data. If he could edit this data and add missing data, he would not have to contact the insurance company if he wants to correct mistakes he made on the mobile client or add additional data.

Today the insurance company, as well as the car owner can only see an image of a map including two cars and arrows of the accident location. This map and the cars cannot be changed. It would be useful to have a detailed, editable map that shows how an accident occurred (see Figure 2.1 for an example of such a map). This would make it easier to detect insurance frauds, because the insurance company employee can better see how the accident occurred and which damages have been caused by the accident and which damages cannot be caused by the accident. This visualization could also be used on the police client to create a sketch of the accident.

The current system uses pre-rendered images of 3D-models that can be used to mark damages. Using real 3D-models would make it easier to see exactly where the damages are on the car. The user could turn the 3D-model into every direction he needs in order to add a damage mark (e.g. at the bottom of the car).

Another problem is the security of the system. E.g. today the mobile client uses a telephone number and a password to verify a user. This is problematic because the user might forget his password. A better solution to this problem should be implemented.

Bibliography

[ADAC09] *ADAC Statistik*. (2009, February 1). Retrieved February 1, 2009, from ADAC Statistik: http://www.adac.de/images/Wichtige%20EckdatenStatstik_081128_tcm8-948.pdf

[Arlt09a] Arlt, S. (2009) EU Accident Report: Police Client (Presentation) (2009, October 5). Retrieved October 5, 2009, from http://userpages.uni-koblenz.de/~arlt/EUAccidentReport.pptx

[Arlt09b] Arlt, S. (2009) EU Accident Report: Mobile Client(Movie) (2009, October 5). Retrieved October 5, 2009, from http://userpages.uni-koblenz.de/~arlt/MobileClient.mov

[ASPN09] *The Official Microsoft ASP.NET Site*. (2009, January 1). Retrieved January 1, 2009, from ASP.NET: http://www.asp.net/

[BenG09] Ben-Gan, I., (2009). *Microsoft SQL Server 2008 - T-SQL Fundamentals.* Redmond: Microsoft Press.

[BKSK06] Ben-Gan, I., Kollar, L., Sarka, D., Kass, S., & Campbell, D. (2006). *Inside Microsoft SQL Server 2005: T-SQL Querying.* Redmond: Microsoft Press.

[BSWK06] Ben-Gan, I., Sarka, D., Wolter, R., Kass, S., & Kollar, L. (2006). *Inside Microsoft SQL Server 2005: T-SQL Programming.* Redmond: Microsoft Press.

[CONR07] Conard, James (2007) *Lap Around Visual Studio 2008 and the .NET-Framework 3.5* (Presentation). Redmond: Microsoft

[Date09] *Datenschutz.de*. (2009, August 4). Retrieved August 4, 2009, from Datenschutz.de: http://www.datenschutz.de/

[DoKo03] Doberenz, W., & Kowalski, T. (2003). *Datenbank-Programmierung mit Visual Studio C# .NET.* Unterschleißheim: Microsoft Press Deutschland.

[DrRa06] Dröge, R., & Raatz, M. (2006). *Microsoft SQL Server 2005 - Konfiguration, Administration, Programmierung (2. Auflage).* Unterschleißheim: Microsoft Press Deutschland.

[Espo08] Esposito, D. (2008). *Programming Microsoft ASP.NET 3.5.* Redmond: Microsoft Press.

[Gail04] Gailey, J. H. (2004). *Understanding Web Services Specifications and the WSE.* Redmond: Microsoft Press.

[Gill08] Gille, N. (2008). *Elektronische Form des EU-Unfallberichts für Mobiltelefone (.Net Client) (Diplomarbeit).* Koblenz: Universität Koblenz Landau.

[Hund06] Hundhausen, R. (2006). *Working with Microsoft Visual Studio 2005 Team System.* Redmond: Microsoft Press.

[KlKn07] Klass, J., & Knopp, T. (2007). *Der European Accident Report – Eine automatische Erfassung des europäischen Unfallberichts (Studienarbeit).* Koblenz: Universität Koblenz Landau.

[Lemp08] Lempa, T. (2008). *Schadensvisualisierung (Diplomarbeit).* Koblenz: Universität Koblenz Landau.

[Lerm09] Lerman, J. (2009). *Programming Entity Framework.* Sebastopol: O'Reilly Media.

[LoMü03] Lorenz, P. A., & Müller, C. A. (2003). *Programmieren lernen in ASP.NET mit C#.* München Wien: Carl Hanser Verlag.

[LoSt02] Louis, D., & Strasser, S. (2002). *C# in 21 Tagen.* München: Markt+Technik Verlag.

[Micr09a] *SQL Server 2008 Overview, data platform, store data*. (2009, January 1). Retrieved January 1, 2009, from Microsoft.com SQL Server 2008: http://www.microsoft.com/sqlserver/2008/en/us/default.aspx

[Micr09b] *Visual Studio Development System*. (2008, January 1). Retrieved January 1, 2008, from Microsoft Visual Studio 2008: http://msdn.microsoft.com/en-us/vstudio/products/default.aspx

[Micr09c] Microsoft Expression Web. (2009, June 5). Retrieved June 5, 2009, from Microsoft Expression Web: http://www.microsoft.com/expression /products/Web_Overview.aspx

[MSDN09] *ADO.NET Entity Framework*. (2009, January 2). Retrieved January 2, 2009, from MSDN: http://msdn.microsoft.com/en-us/data/aa937723.aspx

[Nort09] Northrup, T. (2009). *Microsoft .NET Framework - Application Development Foundation (Second Edition) (MCTS EXAM 70-536).* Redmond: Microsoft Press.

[Plat04] Platt, D. S. (2004). *The Microsoft Platform AHEAD.* Redmond: Microsoft Press.

[Prei05] Preishuber, H. (2005). *ASP.NET 2.0 Crashkurs - Schnelleinstieg in neue Technologien und Tools.* Unterschleißheim: Microsoft Press Deutschland.

[Rich06] Richter, J. (2006). *CLR via C# (Second Edition).* Redmond: Microsoft Press.

[Schu09] *Schufa*. (2009, February 1). Retrieved February 1, 2009, from Schufa: http://www.schufa.de

[Schw05] Schwichtenberg, H. (2005). *Microsoft .net 2.0 Crashkurs - Schnelleinstieg in neue Technologien und Tools.* Unterschleißheim: Microsoft Press Deutschland.

[Smit07] Smith, J. (2007). *Inside Windows Communication Foundation.* Redmond: Microsoft Press.

[SSVG05] *ASP.MAG - Das Magazin für Web Development mit ASP.NET Vol. 1. 2005 (Sonderausgabe des dot.net magazin).* Unterhaching: Software & Support Verlag GmbH.

[Stan09] Stanek, W. R. (2009). *Microsoft SQL Server 2008 - Administrator's Pocket Consultant.* Redmond: Microsoft Press.

[Trai09] *Framework*. (2009, February 2). Retrieved February 2, 2009, from trainingon.net: http://trainingon.net/Articles/ArticleImages/136_Framework.png

[Unfa09a] *Unfallskizze*. (2009, July 3). Retrieved July 3, 2009, from unfallskizze.de: http://unfallskizze.de

[Unfa09b] *European Accident Report*. (2009, July 2). Retrieved July 2, 2009, from www.unfallskizze.de: http://www.unfallskizze.de/ressourcen/Europa-Unfallbericht_EN.php