Eine Benutzungsoberfläche zur Konfiguration von Testsystemen in der Automobilindustrie:

Konzeption, Design

und prototypische Implementierung

in Form einer XML-basierten Web-Applikation

Masterarbeit

Masterstudiengang - Informationswirtschaft
Fachhochschule Stuttgart - Hochschule der Medien

Roswitha Wettinger

Erstprüfer: Prof. Dr. Wolf-Fritz Riekert
Zweitprüfer: Dr.-Ing. Peter Bach

Stuttgart - 07. Februar 2005

Zusammenfassung

Gegenstand der vorliegenden Arbeit ist die Entwicklung einer Benutzungsoberfläche für eine Anwendung zur Konfiguration von Testsytemen in der Automobilindustrie. Die Anwendung ist als XML-basierte Web-Applikation konzipiert. XML dient als Schnittstelle für den Datenaustausch. Die Darstellung des Konfigurationsergebnisses wird mit XSL und XPath durchgeführt.

Um die Anforderungen an die Benutzungsoberfläche zu erfassen, wird die Personas-Methode eingesetzt. Die Gestaltung und das Design der Benutzungsoberfläche stellt einen weiteren Schwerpunkt dar. Die Unified Modeling Language (UML) dient als Entwicklungswerkzeug.

Die Implementierung erfolgt in zwei Stufen. Die erste Stufe bildet ein statischer Prototyp. Unter Verwendung der serverseitigen Open-Source-Skriptsprache PHP und der clientseitigen Skriptsprache Javascript wird der statische Prototyp zu einem dynamischen Prototyp erweitert.

Der Nutzen der Anwendung, sowie der Ausblick auf Erweiterungsmöglichkeiten und weitere mögliche Einsatzgebiete sind ebenfalls Gegenstand der vorliegenden Arbeit.

Schlagwörter:

Benutzungsoberfläche, Personas-Methode, XML-basierte Web-Applikation, XSL, XPath, PHP, Javascript, UML - Unified Modeling Language

Abstract

The subject of this thesis is the development of a user interface for an application which configures test systems in the automotive industry. The application is designed as an XML-based web application. XML is used as the data exchange interface. The configuration result is displayed using XSL and XPath.

The Personas method is used to document the requirements of the user interface. Another focus is the layout and design of the user interface. The Unified Modeling Language (UML) is the development tool used.

The implementation takes place in two phases. The first phase is a static prototype. The static prototype is transformed into a dynamic prototype using the server-based open source script language, PHP, and the client-based script language, Javascript.

This thesis also focuses on the benefit of the application as well as prospects for possible extension and other possible areas of use.

Keywords:

User Interface, Personas Method, XML-based Web Application, XSL, XPath, PHP, Javascript, Unified Modeling Language (UML)

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ASP	Active Server Pages
CAN	Controller Area Network
CSS	Cascading Style Sheet
DIN	Deutsche Industrie Norm Deutsches Institut für Normung e.V.
DTD	Document Type Definition
ECU	Electronic Control Unit
EN	Europäische Norm
ERD	Entity Relationship Diagram
ETAS	Entwicklungs- und Applikationswerkzeuge für elektronische Systeme GmbH
ETK	Emulatortastkopf
FT-CAN	Fault Tolerant Controller Area Network
HTML	Hypertext Markup Language
I/O	Input / Output
ISO	International Organization of Standardization
LCC	LabCar Configurator
OMG	Object Management Group
PDA	Personal Digital Assistant
PERL	Practical Extraction and Report Language
PHP	PHP Hypertext Preprocessor (rekursives Akronym)

XML	Extended Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation
UML	Unified Modeling Language

1 Überblick

Dieses Kapitel beschreibt Ausgangspunkt und Zielsetzung dieser Arbeit und gibt einen kurzen inhaltlichen Überblick über die Arbeit insgesamt.

1.1 Ausgangspunkt und Zielsetzung

Die vorliegende Masterarbeit entstand im Rahmen meines Masterstudiums der Informationswirtschaft - Master an der Hochschule der Medien - Fachhochschule Stuttgart. Inhalt der Masterarbeit ist ein Projekt, das ich bei der Firma ETAS durchgeführt habe.

Ausgangspunkt war der Entschluss der Firma ETAS, für die Hardware von LabCars (Testsysteme im Automobilbereich) einen Konfigurator im Intranet, beziehungsweise später auch im Internet, den eigenen Mitarbeitern und den Kunden zur Verfügung zu stellen. Für diesen Konfigurator sollte in einem ersten Schritt eine geeignete Benutzungsoberfläche entworfen werden. Die Benutzungsoberfläche sollte dann in Form eines Prototyps implementiert werden. Der Konfigurator selbst sollte als Web-Applikation umgesetzt werden. Dabei war jedoch noch offen, ob die Web-Applikation auf der Basis von PHP[1] oder auf der Basis von Visual Basic[2] implementiert werden sollte. Auch über die Architektur der Anwendung insgesamt gab es noch keine konkrete Vorstellung.

Der LabCar-Konfigurator soll eine breite Benutzerschicht ansprechen. Die Benutzungsoberfläche des LabCar-Konfigurators soll zum einen nicht zu technisch orientiert sein, um auch Zielgruppen ansprechen zu können, die bezüglich der Konfiguration von LabCars noch keine Erfahrung, beziehungsweise Vorkenntnisse haben. Zum anderen soll ein möglichst breites Spektrum an LabCar-Konfigurationen, auch technisch komplexere Konfigurationen, mit dem LabCar-Konfigurator durchgeführt werden können.

---

[1] PHP ist eine serverseitig interpretierte Skriptsprache, die hauptsächlich zur Erstellung dynamischer Web-Seiten verwendet wird. (WIKIPEDIA 2004, Stichwort »PHP«)
PHP wird in Kapitel 3.5 »PHP für die Programmierung des Web-Servers«« näher erläutert.

[2] Visual Basic ist eine Programmiersprache der Firma Microsoft. Visual Basic ist die Standardprogrammiersprache für »Active Server Pages« - ASP. »Active Server Pages« ist eine serverseitige Technologie für dynamisch generierte Webseiten.
(WIKIPEDIA 2004, Stichwort »ASP«)

1.2 Inhalt dieser Arbeit

Zu Beginn wird die Systemumgebung beschrieben und die Anforderungen an die Anwendung definiert. Bei den Anforderungen handelt es sich einerseits um funktionale Anforderungen an die Anwendung selbst. Andererseits gibt es Anforderungen von Seiten der potentiellen und zukünftigen Benutzer[3] des LabCar-Konfigurators. Des Weiteren werden die Anforderungen an die Applikation von Seiten der bereits bestehenden technischen Systeme des Unternehmens analysiert.

Im weiteren Verlauf werden die Konzeption der Anwendung, das Design der Benutzungsoberfläche und die Architektur der Anwendung beschrieben.

Die ausführliche Beschreibung der Implementierung ist ebenfalls Inhalt dieser Arbeit. Das Kapitel »Implementierung« richtet sich vorwiegend an den programmiertechnisch interessierten Leser. Der programmiertechnisch weniger versierte Leser kann dieses Kapitel ohne Weiteres überspringen.

Abschließend werden der Nutzen und Erweiterungsmöglichkeiten der Anwendung beschrieben. In einem Ausblick wird dargestellt, welche Vorteile die Weiterentwicklung und Einführung des LabCar-Konfigurators dem Unternehmen bieten kann.

[3] Die Verwendung der männlichen Form für Benutzer steht hier und im weiteren Verlauf dieser Arbeit unabhängig, sowohl für die männliche als auch weibliche Form, also für Benutzer und Benutzerinnen.

2 Systemumgebung und Anforderungen

In diesem Kapitel werden das Umfeld und die Systemumgebung für den zukünftigen LabCar-Konfigurator beschrieben. Dabei werden inhaltliche und technische Anforderungen an den LabCar-Konfigurator dargestellt. Außerdem wird auch auf die Anforderungen, die sich und von Seiten der Benutzer an den LabCar-Konfigurator ergeben, eingegangen.

2.1 ETAS

Die ETAS GmbH mit Sitz in Stuttgart wurde 1994 als Tochtergesellschaft der Robert Bosch GmbH gegründet.

Als Partner der Automobilindustrie liefert ETAS Lösungen für den gesamten Entwicklungsprozess von so genannten »Eingebetteten Systemen«[4]. ETAS-Systeme bestehen aus Hardware- und Softwarekomponenten, sind modular aufgebaut, skalierbar und sowohl einzeln als auch kombiniert einsetzbar. Der Name ETAS steht für Entwicklungs- und Applikationswerkzeuge für elektronische Systeme. ETAS unterhält weitere Niederlassungen in Frankreich, England, den USA, Japan und Korea.

Inzwischen sind rund 850 Mitarbeiter bei der ETAS Group weltweit beschäftigt.

(vgl. ETAS 2003, Stichwort: »Über ETAS«)

2.2 Automobilelektronik

Durch steigende Kundenansprüche und strenge gesetzliche Vorgaben an

- die Verringerung von Kraftsstoffverbrauch und Schadstoffemissionen, sowie
- die Erhöhung von Fahrsicherheit und Fahrkomfort

kann heute auf die Elektronik in modernen Kraftfahrzeugen nicht mehr verzichtet werden. Das Automobil ist dadurch zum technisch komplexesten Konsumgut geworden.

[4] Der Begriff »Eingebettete Systeme« wird im Kapitel »2.4 Elektronische Systeme des Fahrzeugs« näher erläutert.

Die Anforderungen an die Automobilelektronik unterscheiden sich jedoch wesentlich von anderen Bereichen der Konsumgüterelektronik.

Insbesondere hervorzuheben sind:

- der Einsatz unter oft rauen und wechselnden Umgebungsbedingungen in Bezug auf Temperaturbereich, Feuchtigkeit, Erschütterungen oder hohe Anforderungen an die elektromagnetische Verträglichkeit,
- hohe Anforderungen an die Zuverlässigkeit und Verfügbarkeit,
- hohe Anforderungen an die Sicherheit und
- vergleichsweise sehr lange Produktlebenszyklen.

Diese Anforderungen müssen bei begrenzten Kosten, verkürzter Entwicklungszeit und zunehmender Variantenvielfalt in Produkte umgesetzt werden. Dabei steigt die Produktion auf sehr hohe Stückzahlen. Außerdem gewinnt die Wartung der Fahrzeuge immer mehr an Bedeutung.
(vgl. Schäuffele & Zurawka, 2003)

Hier zeigt sich, wie vielseitig und komplex der Bereich der Automobilelektronik ist. Unter Berücksichtigung der Randbedingungen und der zahlreichen Anforderungen an elektronische Systeme von Fahrzeugen ist die Entwicklung solcher Systeme eine schwierige und komplexe Aufgabe. Diese Komplexität und Vielseitigkeit beeinflusst nachfolgend auch ganz entscheidend die zu entwickelnde Web-Applikation.

2.3 Das System Fahrzeug-Fahrer-Umwelt

Um die Funktionsweise, die Komplexität und die Zusammenhänge von Fahrzeugen und Steuergeräten zu verdeutlichen, werden nachfolgend einige Grundbegriffe aus dem Automobilbereich dargestellt.

Die Elektronik stellt heute eine Schlüsseltechnologie zur Realisierung vieler Innovationen im Fahrzeugbau dar. Fast alle Funktionen des Fahrzeugs werden inzwischen elektronisch gesteuert, geregelt oder überwacht.

Der Aufbau und die Wirkungsweise elektronischer Systeme soll nachfolgend am Beispiel eines elektrohydraulischen Bremssystems veranschaulicht werden. Die elektrohydraulische Bremse vereint die Funktionen des

Bremskraftverstärkers, des Antiblockiersystems (ABS) und der Fahrdynamikregelung.

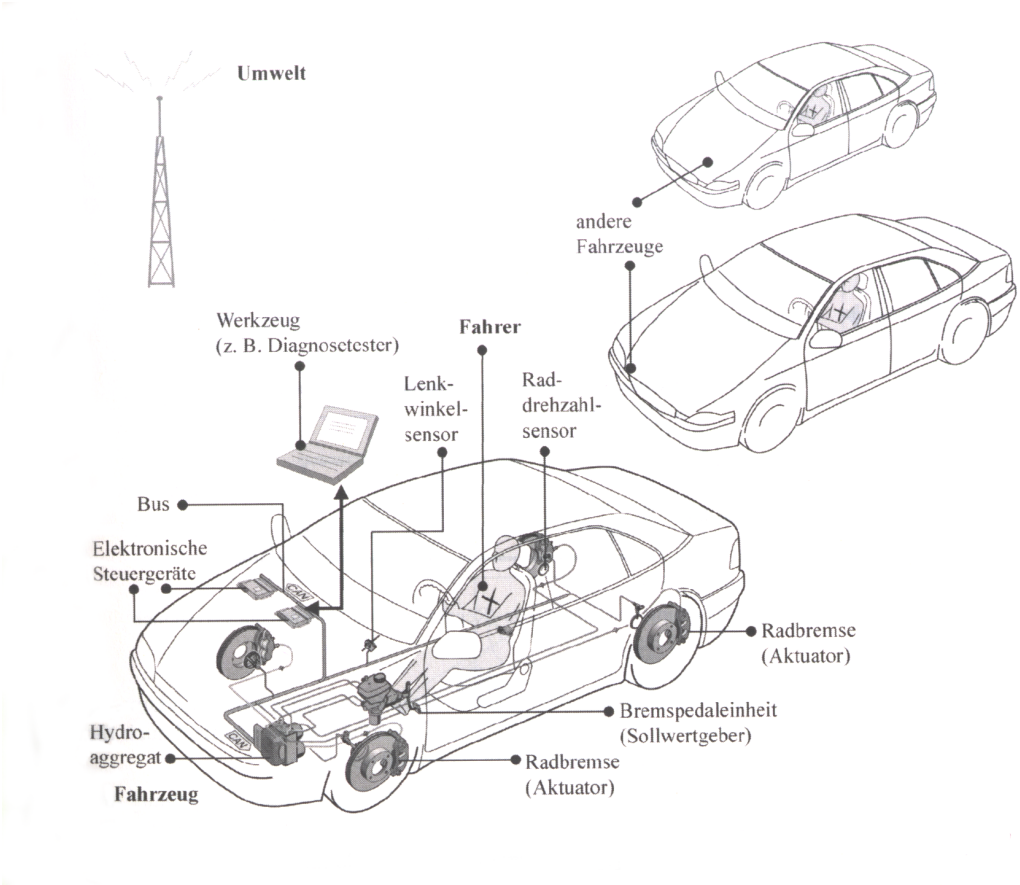

Abbildung 1: Aufbau der elektrohydraulischen Bremse

Die Abbildung ist dem Buch »Automotive Software Engineering« (Schäuffele & Zurawka, 2003, S.3) entnommen.

Betätigt der Fahrer das Bremspedal, wird dies in der Bremspedaleinheit erfasst und elektrisch an das so genannte elektronische Steuergerät übertragen. Im Steuergerät werden unter Verwendung dieses Sollwertes und verschiedener Sensorsignale, wie zum Beispiel dem Lenkwinkelsignal oder den Raddrehzahlsignalen Ausgangsgrößen berechnet.

Diese Ausgangsgrößen wiederum werden zum Hydroaggregat übertragen und dort durch Druckmodulation in Stellgrößen für die Radbremsen umgesetzt. Über die Radbremsen wird das Fahrverhalten des Fahrzeugs, die so genannte Strecke, beeinflusst. Die Radbremsen werden daher als Aktuatoren bezeichnet.

Das Steuergerät kommuniziert über einen Bus[5], zum Beispiel über CAN[6], mit anderen Steuergeräten des Fahrzeugs. Dadurch können weitere Funktionen realisiert werden, die über die bisher genannten Funktionen hinausgehen.

»Der am Beipiel der elektrohydraulischen Bremse dargestellte Systemaufbau ist typisch für alle elektronischen Steuerungs-/Regelungs- und Überwachungssysteme des Fahrzeugs. Im Allgemeinen kann zwischen den folgenden Komponenten des Fahrzeugs unterschieden werden: Sollwertgeber, Sensoren, Aktuatoren, elektronische Steuergeräte und Strecke.«
(Schäuffele & Zurawka, 2003, S. 3)

Sollwertgeber und Sensoren geben die Benutzerwünsche an das Steuergerät weiter. Aus der Sicht des Steuergeräts können Sollwertgeber wie Sensoren behandelt werden. Aktuatoren geben Rückmeldungen von Ereignissen oder Zuständen - beispielsweise durch optische oder akustische Anzeigen - an den Fahrer zurück. Ein Steuergerät ist ein spezialisierter Computer. Beim herkömmlichen interaktiven Computer sind die Ein- und Ausgabegeräte die Tastatur und das Display. Die Ein- und Ausgabegeräte des Steuergerätes sind die Sensoren und die Aktuatoren.

»Die elektronischen Steuergeräte sind miteinander vernetzt und können so Daten austauschen. Fahrer und Umwelt können das Verhalten des Fahrzeugs beeinflussen und sind Komponenten des übergeordneten Systems Fahrzeug-Fahrer-Umwelt.«
(Schäuffele & Zurawka, 2003, S. 3)

2.4 Elektronische Systeme des Fahrzeugs

Ein elektronisches Steurgerät ist für sich gesehen nur ein Mittel zum Zweck. Isoliert stellt es für den Fahrzeugbenutzer keinen Wert dar.

---

[5] Die Bezeichnung Bus ist im Bereich der Datenverarbeitung die umgangssprachliche Abkürzung von Datenbus. Ein Datenbus ist ein Leitungssystem mit zugehörigen Steuerungskomponenten, das zum Austausch von Daten oder Energie zwischen Hardware-Komponenten dient. Bussysteme finden Anwendung insbesondere innerhalb von Computern und zur Verbindung von Computern mit Peripheriegeräten, aber auch in der Ansteuerung von Maschinen (Feldbusse), sowie immer häufiger in Automobilen zur Verbindung der elektronischen Einzelsysteme eines Fahrzeugs.
(WIKIPEDIA 2004, Stichwort: »Bus«)

[6] Controller Area Network (CAN) wird im Kapitel 2.10 »Definition von Signalen« näher erläutert.

»Erst ein vollständiges elektronisches System aus Steuergeräten, Sollwertgebern, Sensoren und Aktuatoren beeinflusst oder überwacht die Strecke und erfüllt so die Benutzererwartungen.«
(Schäuffele & Zurawka, 2003, S. 47)

Das Steuergerät ist für die Fahrzeuginsassen als Komponente des Gesamtsystems Fahrer-Fahrzeug-Umwelt häufig nicht einmal sichtbar. Wird das Steuergerät beispielsweise ausschließlich für Steuerungs- und Regelungsaufgaben eingesetzt, dann hat es meist keine direkten Benutzerschnittstellen. In vielen Fällen haben der Fahrer und die Fahrzeuginsassen nur mittelbaren, oft nur geringen und teilweise auch gar keinen Einfluss auf die Funktionen der Steuergeräte. Systeme mit diesen Merkmalen werden daher auch als eingebettete Systeme (Engl.: Embedded Systems) bezeichnet.

Eingebettete Systeme verfügen über Ein- und Ausgabeeinheiten. Die Ein- und Ausgabeeinheiten ermöglichen es, externe Signale einzulesen und über ausgehende Signale die zu steuernden Größen zu beeinflussen. Jede Ein- und Ausgabeeinheit besitzt einen Anschluss an den internen Bus des Mikrocontrollers und externe Anschlüsse, so genannte Pins[7], an die beispielsweise Sensoren und Aktuatoren angeschlossen werden können.

(vgl. Schäuffele & Zurawka, 2003)

Zur Veranschaulichung ist nachfolgend formal die Schnittstelle eines ABS-Steuergeräts dargestellt.

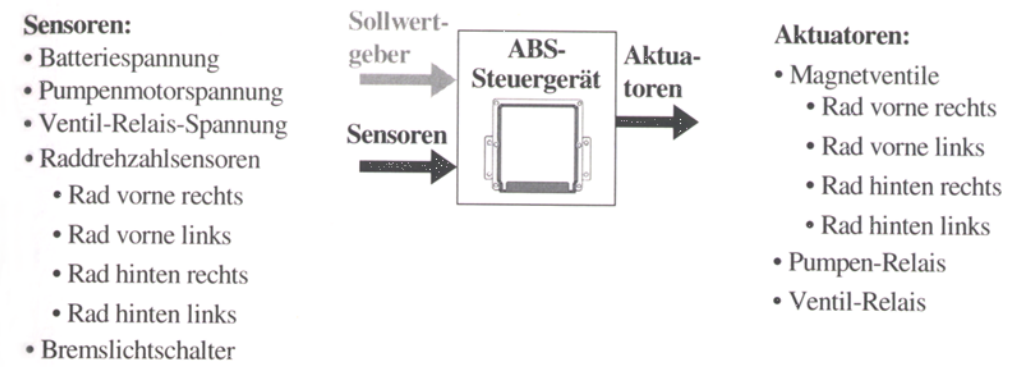

Abbildung 2: Schnittstellen eines ABS-Steuergeräts

[7] Englisch für: Steckanschlüsse

Die Abbildung ist entnommen aus dem Buch »Automotive Software Engineering« (Schäuffele & Zurawka, 2003, S. 11).

Eingebettete Systeme werden nicht nur im Automobilbereich verwendet. Eingebettete Systeme sind heute in Geräten aus den unterschiedlichsten Bereichen integriert, beipielsweise in Waschmaschinen oder in digitalen Kameras.

2.5 LabCar

LabCar, oder zu Deutsch Laborfahrzeug, ist Teil der ETAS-Werkzeugfamilie. LabCar ist ein Echtzeit-Prüfsystem für elektronische Steuergeräte (Engl.: Electronic Control Unit - Abkürzung: ECU), eine virtuelle Fahrzeugumgebung, in die reale oder simulierte Steuergeräte mit allen Ein- und Ausgangsgrößen eingebettet sind. Es passt sich speziellen Bedürfnissen der Kunden an. Mit LabCar lassen sich elektronische Steuergeräte in der Automobilindustrie entwickeln und testen.

Seit Mitte der 90er Jahre verlagern Automobilhersteller und Zulieferer ihre Steuergerätetests zunehmend von der Straße ins Labor. Zunächst entwickelten viele Firmen eigene Testwerkzeuge, was im kleinen Rahmen sehr kostengünstig war. Jedoch waren viele dieser Werkzeuge nicht durchgängig und nahtlos in den gesamten Entwicklungsablauf integriert. Dadurch waren Tests nicht vergleichbar und mussten mehrfach durchgeführt werden. Auch der Support dieser Werkzeuge kostet ab einer bestimmten Anzahl zu viel Zeit und zu viel Geld.

Heute konzentrieren sich Automobilhersteller und Zulieferer zunehmend auf die Definition und Durchführung von Steuergerätetests. Ihre Testsysteme lassen sie sich liefern. Bereits seit 1996 bietet ETAS die ersten LabCar-Testsysteme für Automobil-Seriensteuergeräte am Markt an.

Selbst wenn das neue Fahrzeug erst auf dem Reißbrett existiert, können die realen und geplanten Steuergeräte in Echtzeit und im Verbund mit anderen Steuergeräten getestet werden. Komplexe Funktionen und Wechselwirkungen lassen sich schon im Labor untersuchen. Das macht viel reale Fahrversuche überflüssig, verkürzt die Entwicklungszeit und steigert die Qualität. Mit dem LabCar wird simuliert, was real noch fehlt. Existiert das Fahrzeug erst als Modell und Ansammlung von Daten, steckt der Motor noch in der Entwicklung, sollen Zukunftskonzepte angegangen werden oder fehlen nur

einzelne Komponenten, dann ermöglicht LabCar die Simulation der fehlende Komponenten.

So lassen sich Steuergeräte schon entwickeln und erproben, bevor der neue Prototyp des Motors überhaupt läuft. Oder virtuelle Versuchsfahrten, die noch in ferner Zukunft liegen, können durchgeführt werden. Diese Fahrten sind beliebig reproduzierbar. Mit dem LabCar besteht die Mögichkeit, den Prüfablauf für eine große Anzahl von Funktionen nur einmal definieren zu müssen. Durchführung und Dokumentation (z.B. elektronisch in HTML) erfolgen dann automatisch am LabCar. Die Prüfingenieure gewinnen so Zeit und können beim nächsten Projekt dann auf den bewährten Testumfang aufsetzen.

LabCar-Systeme können modular auf die jeweilige Anwendung und die individuellen Bedürfnisse der Kunden skaliert und dabei jederzeit flexibel erweitert werden - ob nun ein Testsystem für ein einzelnes Steuergerät mit wenigen Pins benötigt wird oder eines für einen komplexen Verbund aus unterschiedlichen Steuergeräten und mehreren hundert Pins.

Mögliche LabCar Einsatzgebiete sind:

- Funktionsentwicklung
- System- und Hardware-Entwicklung
- Software-Entwicklung
- Applikation, Serienfreigabe
- Qualitätssicherung

Die Hard- und Softwareschnittstellen der LabCar-Testsysteme sind offen gestaltet und können individuell, je nach Kundenwunsch und Kundenanforderung, kombiniert werden.
(vgl. ETAS 2003, Stichwort: »Übersicht LabCar«)
(vgl. ETAS - LabCar, 2002)

2.6 LabCar-Hardware

Die LabCar-Hardware setzt sich aus verschiedenen Einschubsystemen zusammen. Diese Einschubsysteme sind modular aufgebaut, können flexibel

erweitert werden und lassen sich beliebig miteinander kombinieren. Durch hochintegrierte, intelligente Einsteckkarten können die Einschubsysteme den individuellen Bedürfnissen der Kunden angepasst werden.
(vgl. ETAS 2003, Stichwort: »Übersicht LabCar«)
(vgl. ETAS - LabCar, 2002)

Es gibt zwei Arten von Einschubsystemen, die Signalboxen und die Lastboxen.

Die Signalbox simuliert die Umgebung des Steuergeräts und interpretiert dabei die Ausgangssignale des Steuergeräts und generiert realitätsnahe Eingangssignale für das Steuergerät.

Eine Lastbox erfüllt zwei Funktionen: Lastsimulation und Fehlersimulation. Unter Lastsimulation versteht man die elektrische Nachbildung von Aktuatoren, beispielsweise über unterschiedliche Widerstände. Diese Ersatzlasten verursachen einen möglichst realistischen Stromfluß am Aktuatorausgang des Steuergerätes. Über die Fehlersimulation können Fehler zwischen Steuergerät und Sensor oder Aktuator nachgebildet werden. Üblicherweise können Kabel- oder Steckverbinderfehler wie Kurzschluß, Unterbrechungen, Widerstandsänderung durch Alterung oder Wackelkontakte simuliert werden.

Nachfolgend sind beispielhaft eine Signalbox und eine Lastbox abgebildet.

Abbildung 3: LabCar - Signalbox ES4100

Quelle: ETAS (2003, Stichwort: LabCar-Hardware)

Abbildung 4: LabCar - Lastbox ES4500

Quelle: ETAS (2003, Stichwort: LabCar-Hardware)

Die Signal- und Lastboxen werden nach den Kundenanforderungen zusammengebaut.

Nachfolgend ist ein leeres Gehäuse (Engl.: Chassis) abgebildet, in das die jeweils benötigten Einsteckkarten eingebaut werden können.

Abbildung 5: LabCar - ES4300 (leeres Gehäuse)

Quelle: ETAS - LabCar-Hardware (2002)

Die folgende Abbildung zeigt beispielhaft eine Einsteckkarte (Engl.: Board).

Abbildung 6: LabCar - Einsteckkarte ES1222

Quelle: ETAS (2003, Stichwort: ES1000-System im Detail)

Auf den Karten können je nach Kundenanforderung Aufsteckmodule angebracht werden. Die Aufsteckmodule werden auch als Piggy-Back (Englisch für: Huckepack) bezeichnet. Die Piggy-Backs beeinflussen die Funktionalität der einzelnen Einsteckkarten.

Die verschiedenen Einsteckkarten verfügen neben den unterschiedlichen Funktionalitäten auch über sehr unterschiedliche Steckanschlüsse (Engl.: Pin). Abhängig vom Umfang der gewünschten Tests, die ein Kunde mit seinem LabCar-System durchführen möchte, werden entsprechend einfachere oder auch komplexere LabCar-Systeme gebaut.

»Eine besondere Herausforderung beim Testen von Steuergeräten ist das Freigeben unterschiedlicher Serienvarianten. Im Motorbereich variieren zum Beispiel die Anzahl der Zylinder, die Art der Einspritzsysteme oder die Aufladung. Zudem erhöht eine zunehmende Vernetzung der Steuergeräte die Anzahl der Funktionsvarianten, die geprüft werden müssen.«
(ETAS - LabCar- Project, 2002)

Hat ein Kunde mehr und vielseitigere Anforderungen oder möchte er komplexere Testszenarien durchführen, benötigt er entprechend komplexere LabCar-Systeme.

Die nachfolgende Abbildung zeigt ein komplexes LabCar. Hier sind in einem Rahmengestell (Engl.: Rack), neben einem Trennadapter und einem Konstanter, mehrere Signal- und Lastboxen integriert.

Abbildung 7: LabCar - Rack

Quelle: ETAS - LabCar-Hardware (2002)

2.7 LabCar-System

Nachfolgende Abbildung zeigt zwei Steuergeräte (Engl.: Electronic Control Unit - Abkürzung: ECU) in Verbindung mit einem LabCar-Testsystem.

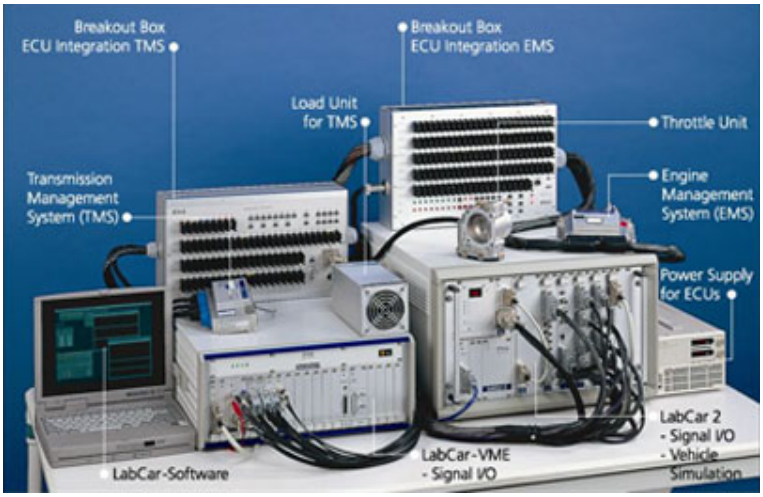

Abbildung 8: LabCar-Testsystem in Verbindung mit Steuergeräten

Quelle: ETAS (2003, Stichwort: Mit LabCar realisierte Projekte)

Getestet werden hier die beiden Steuergeräte TMS (Transmission Management System) und EMS (Engine Management System).

Die Steuerung eines Testszenarios erfolgt über die LabCar-Software. Die LabCar-Software läuft auf einem separaten PC, beziehungsweise in diesem Fall auf einem Laptop. Der Laptop ist mit der LabCar-Hardware verbunden. Die LabCar-Software steuert das LabCar und appliziert die Steuergeräte.

2.8 LabCar-Konfigurator

Um die Konfiguration von LabCar-Systemen zu unterstützen hat sich die Firma ETAS entschlossen ein geeignetes Werkzeug, einen LabCar-Konfigurator (Engl.: LabCar Configurator - Abkürzung: LCC), zu entwickeln. Mit

dem LabCar-Konfigurator sollen virtuelle Konfigurationen für die LabCar-Hardware durchgeführt werden können. Hierfür benötigt der LabCar-Konfigurator Angaben darüber, welche Signale, welche Lasten und welche Fehler mit dem LabCar simuliert werden sollen, beziehungsweise welche Signalein- und Signalausgänge vom Kunden benötigt werden. Der LabCar-Konfigurator soll ein Werkzeug sein, das sowohl Einsteigern als auch Fortgeschrittenen die Konfiguration der LabCar-Hardware erleichtert.

Dabei muss der LabCar-Konfigurator folgendes Problem lösen: Ein Kunde wünscht sich eine bestimmte Menge von Signalen, die durch ein LabCar simuliert werden sollen. Hierfür muss ein entsprechendes LabCar zusammengebaut werden. Für dieses LabCar benötigt man bestimmte Karten und Piggy-Backs. Außerdem wird eine Box benötigt, welche die entsprechenden Karten aufnimmt. Da die Signale aber nicht eins-zu-eins den Karten entsprechen ist das Problem nicht trivial. Mit einer Karte können meist mehrere Signale, unter Umständen sogar gleichzeitig unterschiedliche Signale, realisiert werden. Manche Signaltypen können auch durch unterschiedliche Karten simuliert werden. Dadurch gibt es mehrere Möglichkeiten, von denen die günstigste ermittelt werden muss.

Verkäufern und Interessenten, die noch keine Erfahrung in der Konfiguration von LabCars haben, soll der LabCar-Konfigurator eine Hilfe sein. Dabei steht der Lernaspekt im Vordergrund.

Dem versierteren Verkäufer soll der LabCar-Konfigurator Unterstützung anbieten. Die Wünsche gehen dabei in Richtung einer visualisierten Darstellung der konfigurierten LabCars. Außerdem wird die automatisierte Erstellung einer Bestellliste gewünscht.

Die Konfiguration eines LabCars zusammen mit dem Kunden kann je nach Umfang und Komplexität des LabCars einen längeren Prozess darstellen, der sich über einen längeren Zeitraum erstrecken kann. Hierfür soll der LabCar-Konfigurator den aktuellen Stand der Konfiguration dokumentieren. Auf den aktuellen Stand soll dann im weiteren Verlauf zugegriffen werden können. Ebenso soll der aktuelle Stand jederzeit nach Bedarf erweitert und verändert werden können.

Für die Zukunft ist auch daran gedacht, den LabCar-Konfigurator im Internet Interessenten und Kunden anzubieten. In diesem Fall sind mit dem LabCar-Konfigurator auch Werbe- und Marketing-Interessen verbunden.

Hier spielt die Präsentation der Firma ETAS nach außen hin und die Vermarktung des Produktes LabCar eine wichtige Rolle.

2.9 Notwendige Angaben für die Konfiguration

Um die Konfiguration für ein LabCar durchführen zu können, müssen die benötigten Signalein- und Signalausgänge festgelegt werden. Außerdem muss definiert werden, welche Lasten mit dem LabCar nachgebildet werden sollen und welche Fehler simuliert werden sollen. Diese notwendigen Angaben werden auch als »Requirements« (Englisch für: Anforderungen) bezeichnet.

Über die Signaldefinition werden die benötigten Steckanschlüsse (Pin) festgelegt. Von der Definition der gewünschten Signalein- und Signalausgänge hängt es ab, welche Karten (Boards), Aufsteckmodule (Piggy-Backs) benötigt werden. Abhängig vom Umfang der benötigten LabCar-Komponenten (Boards und Piggy-Backs) wird dann vom LabCar-Konfigurator ein passendes Gehäuse und ein passendes Rahmengestell (Rack) ausgewählt.

2.10 Definition von Signalen

Die Struktur von Signalen wird im Folgenden mit Hilfe eines Entity-Relationship-Diagramms (ERD) beschrieben.

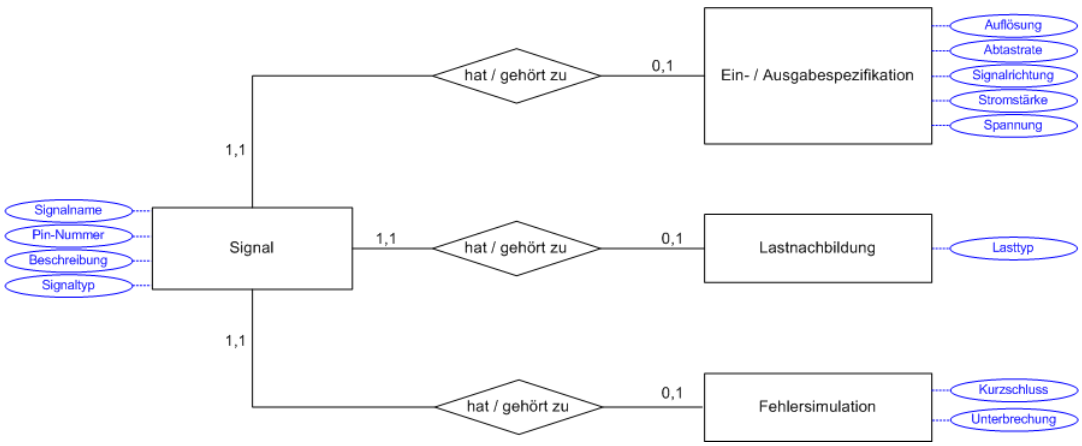

Abbildung 9: ERD - Signal

Dem Entity-Relationship-Diagramm liegt folgende Notation[8] zugrunde:

Die Rechtecke stellen die Objekte, beziehungsweise die Entitäten (Engl.: entities), dar. Die Rauten symbolisieren die Beziehungen (Relationen) zwischen den Entitäten. Die Ellipsen symbolisieren die Attribute einer Entität.

Die Komplexität einer Beziehung wird durch die Angabe von Kardinalitäten bestimmt. Die Kardinalität legt fest, wie viele Entitäten einer Entitätsmenge mit einer Entität einer anderen Entitätsmenge eine Beziehung eingehen kann. Die jeweilige Kardinalität ist von der ausgehenden Entität aus gesehen, direkt bei der Entität, die das Ziel der Beziehung darstellt, notiert. Die Kardinalität gibt an, ob eine Beziehung bestehen muss (1, ...) oder kann (0, ...) und ob eine Beziehung nur einfach (..., 1) oder mehrfach (..., n) (..., m) vorhanden sein kann. Aus der Kardinalität ergibt sich dann, um welche Art von Beziehung es sich handelt, um eine **1:1**-Beziehung, eine **1:n**-Beziehung oder eine **n:m**-Beziehung.

Ein Signal hat einen Signalnamen, eine Pin-Numer, eine Beschreibung und ist einem bestimmten Signaltyp zugeordnet.

Attribut	Beschreibung
Signalname	Der Signalname (Engl.: Signal Name) wird als zusammenhängender String ohne Leerzeichen definiert. Der Signalname muss eindeutig sein. Als Zeichen sind Buchstaben, Zahlen und der Unterstrich erlaubt.
Pin-Nummer	Pin steht für Steckverbindung (Stecker, Steckerstift, Kabelschuh) oder allgemeiner ausgedrückt den Anschluss. Jedem Signal wird eine eindeutige Pin-Nummer (Engl.: Pin Number) zugeordnet. Die Pin-Nummer kann frei gewählt werden. Sie muss eindeutig sein. Als Wert muss eine ganze Zahl (Engl.: Integer) vergeben werden.
Signalbeschreibung	In diesem Bereich kann eine individuelle Signalbeschreibung (Engl.: Description) vorgenommen werden. Die Signalbeschreibung kann in freiem Text formuliert werden.
Signaltyp	Für den Signaltyp (Engl.: Signal Type) stehen folgende Auswahlmöglichkeiten zur Verfügung: I/O, RS422, ETK, CAN und FT-CAN.

[8] vgl. Götz et.al. (1997) und Sauer (2002)

Nachfolgend werden die möglichen Signaltypen beziehungsweise ihre Schnittstellen näher erläutert:

Signaltyp	Beschreibung
I/O	**I/O** (Engl.: Input/Output) steht für Ein- /Ausgabekanal. Wird für ein Signal der Signaltyp I/O definiert, kann über diesen Pin entweder eine Eingabe oder eine Ausgabe erfolgen. Wird für ein Signal der Signaltyp I/O gewählt, muss die Art der Ein-/Ausgabe näher spezifiziert werden. Attribute der Ein-/Ausgabespezifikation sind die Auflösung, die Abtastrate, die Signalrichtung, die Stromstärke und die Spannung.
RS422	Die **RS422**-Schnittstelle bietet eine serielle Datenübertragung über große Entfernungen. (Kolter Electronic 2004, Stichwort »RS422«)
ETK	**ETK** steht für Emulatortastkopf. Der ETK stellt eine Hochleistungsschnittstelle zu eingebetteten Kontrolleinheiten (Engl.: Embedded Control Units) im Fahrzeug dar. (ETAS 2003, Stichwort: »Kompatible Produkte zu LabCar«)
CAN	**CAN** steht für Controller Area Network und wurde von den Firmen Bosch und Intel ursprünglich als Bussystem für Fahrzeuge entwickelt. (ELAU AG, 2004)
FT-CAN	**FT-CAN** steht für »Fault Tolerant CAN«. Fehlertolerant (Engl.: fault tolerant) bedeutet, dass ein System in der Lage ist, auf unerwartete Hardware- oder Softwareausfälle oder -störungen in annehmbarer Weise zu reagieren. Falls eine Komponente des Systems ausfällt, übernimmt eine Sicherungskomponente oder -prozedur die Aufgaben und Funktionen der ausgefallenen Komponente, sodass keine Einbuße im laufenden Betrieb entsteht. Dies bedeutet, dass die Fault-Tolerant-CAN-Schnittstelle gegen raue Umgebungsbedingungen, wie zum Beispiel extreme Kälte und Hitze, so wie sie im Automobilbereich auftreten, geschützt ist. (TechTarget Network 2004, Stichwort: »Fault-Tolerant«) (Motorola, 2000)

Für Signale vom Typ I/O muss die Art der Eingabe und der Ausgabe genauer spezifiziert werden. Dies geschieht über die Ein-/Ausgabespezifikationen.

Die Ein-/Ausgabespezifikationen haben folgende Attribute:

Attribut	Beschreibung
Auflösung	Für die Auflösung (Engl.: Resolution) kann aus folgenden Werten gewählt werden: »digital«, »analog« und »digital (hi res)«. »**digital**« steht für digitale Auflösung, »**analog**« steht für analoge Auflösung und »**digital (hi res)**« steht für eine hohe digitale Auflösung (Engl.: digital high resolution).
Abtastrate	Die gewünschte Abtastrate (Engl.: Sampling Rate) muss in Hertz (1/sec.) definiert werden.
Signalrichtung	Die Signalrichtung (Engl.: Direction) kann folgende Werte annehmen: »input«, »output« oder »bi-directional«. Die Blickrichtung geht dabei vom Steuergerät (Engl.: Engineering Control Unit - ECU) aus. »**input**« bedeutet dabei in eingehender Richtung, »**output**« in ausgehender Richtung und »**bi-directional**« bi-direktional, also in beide Richtungen.
Stromstärke	Die gewünschte Stromstärke (Engl.: Current) muss in Ampere angegeben werden. Dabei können für die Stromstärke zwei Werte definiert werden: die Dauerstromstärke (Engl.: **Current Permanent**) und ein Stromstärkenspitzenwert (Engl.: **Current Peak**).
Spannung	Die für das entsprechende Signal gewünschte elektrische Spannung (Engl.: Voltage) muss in Volt angegeben werden.

Ein Signal kann über eine Lastnachbildung verfügen. Folgende Werte stehen für das Attribut Lasttyp zur Auswahl:

Lasttyp	Beschreibung
No Load	»**No Load**« bedeutet, dass für dieses Signal keine Lastnachbildung gewünscht wird.
Original Load	»**Original Load**« bedeutet eine Originallast-Nachbildung wird gewünscht.
Diesel [Common Rail]	»**Diesel [Common Rail]**« heißt, dass eine Lastnachbildung für Dieselkraftstoff-Einspritzung gewünscht wird.

Lasttyp	Beschreibung
Gasoline Direct Injection	»**Gasoline Direct Injection**« bedeutet, eine Lastnachbildung für Benzin-Direkteinspritzung wird gewünscht.
Piezo Injection	»**Piezo Injection**« bedeutet, es wird eine Lastnachbildung für Piezo-Einspritzung gewünscht. Die Piezo-Einspritzung ist eine spezielle Einspritztechnik mit hohen Spannungen bis zu 400 Volt.

Außerdem kann ein Signal über eine Fehlersimulation verfügen. Als Fehler kann ein Kurzschluss und / oder eine Unterbrechung (im Sinne von Leiterbahnunterbrechung beziehungsweise Kabelbruch) simuliert werden.

Die Art des Kurzschlusses beziehungsweise der Unterbrechung muss dabei näher spezifiziert werden. Folgende Möglichkeiten stehen für die Simulation eines Kurzschlusses beziehungweise einer Unterbrechung zur Auswahl:

Attribut	Beschreibung
Ground	»**Ground**« bedeutet Kurzschluss beziehungsweise Unterbrechung gegen Erde / Masse.
UBATT	»**UBATT**« bedeutet Kurzschluss beziehungsweise Unterbrechung gegen Batteriespannung.
Pin To Pin	»**Pin To Pin**« bedeutet Kurzschluss beziehungsweise Unterbrechung Pin gegen einen anderen Pin.
Inline Resistor	»**Inline Resistor**« bedeutet Kurzschluss beziehungsweise Unterbrechung über einen Serienwiderstand.

2.11 Benutzergruppen

»Je genauer und konkreter Sie Ihre Zielgruppe - oder Zielgruppen - im Blick haben und sich in sie hineinversetzen, umso erfolgreicher werden Sie sie [Anm. d. Verf.: die Zielgruppe] erreichen. Deshalb empfiehlt es sich auch, Mitglieder der Zielgruppe recht früh in die Planung einzubeziehen und erste Prototypen mit ihnen zu testen.

Es lohnt sich, sich intensiv um die Zielgruppenfrage zu kümmern und Zeit und Energie darauf zu verwenden, die Adressaten genau zu fokussieren.«
(Thissen, 2003, S.32)

Das Spektrum der Benutzergruppen des LabCar-Konfigurators ist umfangreich. Zur Zielgruppe gehören zunächst Mitarbeiter der Firma ETAS selbst. Die Gruppe der ETAS-Mitarbeiter konzentriert sich dabei vor allem auf die Verkäufer von LabCars. Der LabCar-Konfigurator soll sowohl neuen und unerfahrenen Verkäufern als auch versierten Verkäufern als Hilfsmittel und Werkzeug dienen. Kunden und Interessenten, die sich ein LabCar kaufen möchten oder sich einfach nur informieren möchten, stellen eine weitere Benutzergruppe dar.

2.12 Definition von Personas

Um die Benutzergruppen genauer einzugrenzen wurde die so genannte Personas-Methode verwendet. Das Personas-Konzept beziehungsweise die Personas-Methode wurde von Alan Cooper entwickelt. Um eine Software so zu gestalten, dass sie den Benutzeranforderungen entgegenkommt, empfiehlt Cooper die Definition von so genannten Personas:

»However, our goal is to design software that will bend and stretch and adapt to the user's needs ... In our design process, we never refer to the 'user'. Instead, we refer to a very special individual: a persona.«
(Cooper, 1999)

Thissen (2003, S.32) schreibt über diese Methode, dass sie äußerst effektiv ist, und dass es durch die Definition von Personas möglich wird, ein benutzerzentriertes Screen-Design[9] für eine Anwendung zu entwickeln.

Ausgangspunkt der Methode ist eine präzise Beschreibung der Nutzer und ihrer Ziele. Cooper nennt dies 'Goal Directed Design'. Dabei werden so genannte Personas definiert. Eine Persona ist kein realer Mensch, sondern der Archetyp eines Nutzers beziehungsweise die Pauschalierung einer bestimmten Nutzergruppe.

»Personas werden durch das, was sie erreichen wollen (ihre Ziele), definiert. Sie werden in der Definition sehr plastisch und anschaulich und sind dem

[9] Das englische Wort »Screen« steht für Bildschirm.

Entwickler eine wunderbare Hilfe, sich in den potentiellen Nutzer hineinzuversetzen und über ihn im Entwicklungsteam zu kommunizieren ... Damit die Persona allen am Projekt Beteiligten plastisch vor Augen steht, ist es wichtig, ihr einen Namen und ein 'Gesicht' zu geben. Beschreiben Sie ihre konkreten Ziele und typische Szenarios, in denen die Persona Ihr Angebot [Anm. d. Verf.: Ihre Anwendung] nutzen wird. Auf diese Weise ist es dann möglich, sehr effektiv ein Angebot zu gestalten, das die Bedürfnisse dieser Persona - und damit der realen Nutzer - sehr genau trifft.«
(Thissen, 2003, S.34)

Die Definition von Personas für den LabCar-Konfigurator erfolgte auf der Basis von Interviews und Gesprächen mit Mitarbeitern aus dem Bereich »Testsysteme« der Firma ETAS. Nach der Analyse der Befragungsergebnisse wurden folgende Personas charakterisiert:

Verkäufer A

Verkäufer A ist sehr versiert in der Konfiguration von LabCar-Systemen. Er benötigt keine Hilfestellung für Konfiguration eines LabCar. Für ihn ist die Konfiguration eines LabCar einfach. Auch die Konfiguration komplexerer LabCar-Systeme bereitet ihm keine Schwierigkeit. Er wünscht sich den LabCar-Konfigurator lediglich als komfortable Visualisierungshilfe.

Bei der Ausgabe wünscht er sich eine grafische Darstellung des konfigurierten LabCar. Die grafische Darstellung möchte er auch in das Angebot, das für den Kunden erstellt wird, einfügen.

Außerdem wünscht er sich folgende Ausgaben:

- Bestellinformationen
- Gesamtgewicht
- Ausgabe der Leistungsaufnahme (Stromstärke).

Verkäufer B

Verkäufer B ist ebenfalls sehr versiert in der Konfiguration von LabCar-Systemen. Dennoch begrüßt er die Umsetzung eines LabCar-Konfigurators. Für ihn ist der LabCar-Konfigurator in erster Linie ein Werkzeug, das ihn bei Kundengesprächen und im Kundenberatungsprozess unterstützen soll.

An den LabCar-Konfigurator hat er folgende Wünsche:

- Der LabCar-Konfigurator soll ihm als Werkzeug dienen, das er beim Kunden vor Ort einsetzen kann. Mit Hilfe des LabCar-Konfigurators möchte Verkäufer B mit dem Kunden zusammen ein LabCar-System konfigurieren können. Dabei möchte er technische Details abfragen können.
- Die Eingabe der für die Konfiguration notwendigen Angaben soll in Form einer Tabelle möglich sein. Hierfür werden die Signalein- und Signalausgänge in eine Tabelle geschrieben. Die Tabelle soll dann über den LabCar-Konfigurator auf den zentralen Server hochgeladen werden können. Anhand der Daten aus dieser Tabelle soll dann vom LabCar-Konfigurator eine Konfiguration durchgeführt werden. Außerdem möchte er auch eine vorbereitete leere Tabelle herunterladen können.
- Die Ausgabe des LabCar-Konfigurators soll eine grafische Darstellung des konfigurierten LabCar anbieten und zum anderen eine Auflistung der technischen Details anbieten. Die direkte Erstellung eines Angebots soll ebenfalls möglich sein.
- Der LabCar-Konfigurator soll außerdem auch dem Kunden als Instrument dienen. Damit soll der Kunde selbst via Internet ein LabCar-System konfigurieren können, technische Details einsehen können und ein Preisangebot erhalten. Ist das LabCar-System, das der Kunde konfigurieren möchte, zu komplex, soll der Kunde direkt mit einem Verkäufer, beziehungsweise einem Fachmann der Firma ETAS, Kontakt aufnehmen können.

Verkäufer C

Verkäufer C ist neu im Unternehmen. Er ist Japaner und ist in der japanischen Niederlassung von ETAS tätig. Seine Muttersprache ist Japanisch. Außerdem spricht er Englisch.

Verkäufer C hat keine Kenntnisse über die Konfiguration eines LabCar. Er weiß nicht welche Angaben für die Konfiguration eines LabCars notwendig sind. Er hat Schwierigkeiten mit der Definition von Signalein- und

Signalausgängen. Auch bei der Definition der Lastnachbildung braucht er Hilfestellung.

Mit dem LabCar-Konfigurator möchte sich Verkäufer C Kenntnisse über die Konfiguration eines LabCars aneignen. Der LabCar-Konfigurator soll ihm als Lerninstrument und Trainingswerkzeug dienen. Auf den LabCar-Konfigurator möchte er über das Firmen-Intranet zugreifen.

Kunde X

Kunde X ist Internet-Benutzer und möchte sich via Internet informieren. Wie Verkäufer C hat er keine Kenntnisse über die Konfiguration eines LabCar und benötigt Hilfestellung. Für ihn steht die Information im Vordergrund. Er möchte mehr über LabCars erfahren. Außerdem interessieren ihn die Preise. Kunde X könnte man als Amateur bezeichnen.

Kunde Y

Kunde Y kennt das Produkt LabCar schon. Er hat Erfahrung mit der Definition von Signalein- und Signalausgängen. In der Vergangenheit hat er bereits mit Hilfe von Tabellen Signalein- und Signalausgänge definiert.

Kunde Y möchte sich ebenfalls via Internet informieren. Interessant ist für ihn, welchen Preis er für welche Systemkomponenten bezahlen muss. Außerdem interessiert ihn auch, ob es neue LabCar-Systemkomponenten gibt, die für ihn interessant sein könnten.

Kunde Y möchte jeweils eine Darstellung des von ihm konfigurierten LabCar-Systems. Außerdem möchte er wissen, wo (an welchen LabCar-Hardwarekomponenten) die von ihm gewünschten Signalein- und Signalausgänge sitzen. Kunde Y könnte man als Experten bezeichnen.

2.13 Benutzeranforderungen

Aufgrund von Interviews und Gesprächen mit Mitarbeitern der ETAS GmbH und der Definition von Personas ergaben sich folgende Anfordungen von Seiten der zukünftigen Benutzer an den LabCar-Konfigurator (Engl.: LabCar Configurator):

- Der LabCar-Konfigurator soll den Verkäufer von LabCar im Verkaufsprozess und bei der Verhandlung mit dem Kunden unterstützen.

- Der LabCar-Konfigurator soll den aktuellen Status des Verhandlungsbeziehungsweise des Verkaufsprozesses mit dem Kunden festhalten und dokumentieren.
- Der LabCar-Konfigurator soll eine Visualisierung von virtuell konfigurierten LabCars anbieten.
- Der LabCar-Konfigurator soll Nachwuchsverkäufern als Lerninstrument beim Einlernprozess dienen. Dabei soll der LabCar-Konfigurator auch von Nachwuchsverkäufern in anderen Ländern als Lerninstrument genutzt werden können.
- Der LabCar-Konfigurator soll sowohl dem Kunden, als auch dem unerfahrenen Verkäufer, Hilfestellung bei der Definition von Signaleinund Signalausgängen bieten.
- Der LabCar-Konfigurator soll Interessenten und Kunden Informationen (neue Systemkopmonenten, Preise) zum Produkt LabCar anbieten und Hilfestellung bei der Festlegung ihrer eigenen Anforderungen an ein LabCar geben.
- Der LabCar-Konfigurator soll auch Listen in Tabellenform verarbeiten können.
- Der LabCar-Konfigurator soll in englischer Sprache umgesetzt werden, da die Zielgruppe international ist.

2.14 Anwendungsfälle

Anhand der Benutzeranforderungen wurden Anwendungsfälle (Engl.: use cases) entworfen und dargestellt. Hierfür wurde die Unified Modeling Language (UML) und die darin definierten Anwendungsfalldiagramme, beziehungsweise Use-Case-Diagramme, verwendet.

Die »Unified Modeling Language« (UML) ist ein Standard, der die Notation und Semantik zur Visualisierung, Konstruktion und Dokumentation von Modellen für die Geschäftsprozessmodellierung und für die objektorientierte Softwareentwicklung definiert. Die UML bietet den Entwicklern die Möglichkeit, den Entwurf und die Entwicklung von Softwaremodellen

auf einheitlicher Basis zu diskutieren. Die UML lag und liegt bei der Object-Management-Group (OMG)[10] zur Standardisierung vor.

Entwickelt wurde die UML anfangs von Grady Booch, Ivar Jacobsen und Jim Rumbaugh. Sie kombinierten die besten Ideen objektorientierter Entwicklungsmethoden und entwickelten daraus die UML. Die weitere Entwicklung der UML wurde im Wesentlichen von der Firma Rational Software vorangetrieben. Viele führende Unternehmen der Computerbranche (Microsoft, Oracle, Hewlett-Packard . . .) wirkten aktiv an der Entwicklung mit und unterstützen die UML.

Die UML hat es sich zur Aufgabe gemacht, möglichst den gesamten Software-Entwicklungsprozess mit grafischen Elementen zu unterstützen. Dies erfolgt durch verschiedene Diagramme, die den unterschiedlichen Kontexten und Aufgaben im Software-Entwicklungsprozess angepasst sind. Eine Einführung in die UML und Beschreibungen der einzelnen UML-Diagramme finden sich bei Dumke (2004).

Die UML definiert unter anderem so genannte Anwendungsfalldiagramme. Mit Anwendungsfalldiagrammen kann man die Anforderungen an ein Softwaresystem darstellen.

Ein Anwendungsfalldiagramm besteht aus einer Menge von Anwendungsfällen und stellt die Beziehungen zwischen Akteuren und Anwendungsfällen dar. Es zeigt das äußerlich erkennbare Systemverhalten aus der Sicht eines Anwenders. Ein Akteur ist eine Person, Organisation oder wiederum ein System der mit dem Anwendungsfallsystem kommuniziert und es beeinflußt. Der Akteur befindet sich außerhalb der Systemgrenze, das heißt, er ist nicht Teil des Anwendungsfallsystems.

Anwendungsfälle können hierarchisch geschachtelt werden. Dies bedeutet, ein Anwendungsfall kann durch weitere Anwendungsfalldiagramme in weitere genauere Anwendungsfälle untergliedert werden. Anwendungsfälle tragen zur Identifikation einen eindeutigen Namen oder eine Nummer.

Anwendungsfalldiagramme sind vorwiegend ein Hilfsmittel zur Anforderungsermittlung und dienen weniger der Verhaltensbeschreibung und dem Systemdesign. Ein Anwendungsfalldiagramm ist keine Ablaufbeschreibung, es wird keine Ablaufreihenfolge definiert. Dafür gibt es andere UML-Verhaltensdiagramme, zum Beispiel Aktivitäts- und Zustandsdiagramme.

[10] www.omg.org

Nachfolgend ist der Anwendungsfall (Engl.: Use Case) »LabCar-Konfiguration« dargestellt. Dieser Anwendungsfall stellt in der Hierarchie der Anwendungsfälle die oberste Ebene dar.

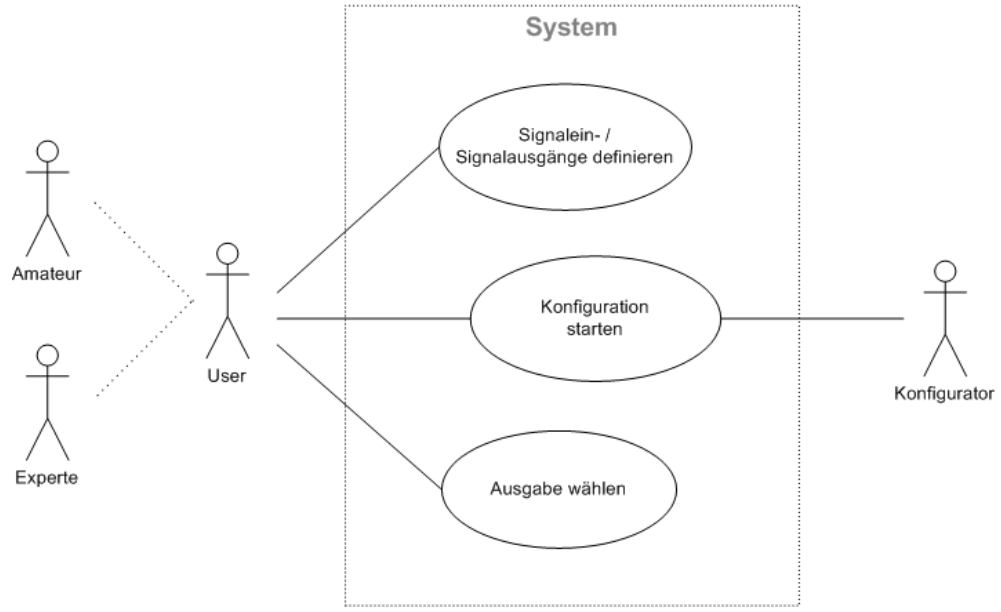

Abbildung 10: Use Case: »LabCar-Konfiguration«

Der Benutzer (Engl.: User) kann entweder Amateur oder auch Experte sein. Der Benutzer kann die Signalein- und Signalausgänge, die er haben möchte, festlegen. Des Weiteren kann der Benutzer die Konfiguration starten. Bei der Ausgabe kann der Benutzer zwischen verschiedenen Ausgabemöglichkeiten wählen.

Nachfolgend wird der Anwendungsfall (Engl.: Use Case) »Signalein- / Signalausgänge definieren« näher spezifiziert.

Abbildung 11: Use Case: »Signalein- / Signalausgänge definieren«

Bei der Definition von Signalein- und Signalausgängen gibt es zwei verschiedene Möglichkeiten. Der Amateur definiert seine Signale direkt im System. Er kann Signale hinzufügen, ändern oder löschen. Außerdem kann

er die aktuelle Signalliste lokal auf seinem PC speichern oder eine früher erstellte Signalliste in das System laden. Der Experte kann eine Tabelle benutzen, in welcher er die gewünschten Signale definiert. Diese Tabelle kann er dann ins System laden.

Der Anwendungsfall »Ausgabe wählen« beinhaltet die folgenden beiden Anwendungsfälle:

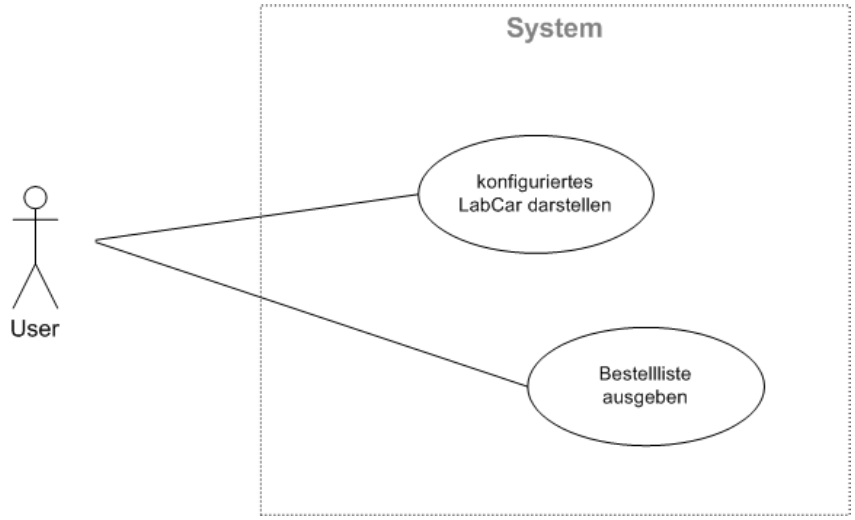

Abbildung 12: Use Case: »Ausgabe wählen«

Bei er Wahl der Ausgabe des Konfigurationsergebnisses hat der Benutzer zwei Möglichkeiten. Er kann sich das konfigurierte LabCar in Form einer Baumstruktur anzeigen lassen oder sich eine Bestellliste ausgeben lassen.

2.15 Funktionale Anforderungen

An den LabCar-Konfigurator gibt es folgende funktionale Anforderungen:

- Der LabCar-Konfigurator muss die eingegebenen Daten auf Validität prüfen, damit eine Konfiguration durchgeführt werden kann.
- Bei unplausiblen Eingaben von Seiten des Benutzers muss der LabCar-Konfigurator sinnvolle Fehlermeldungen ausgeben. Der LabCar-Konfigurator muss es dem Benuzter ermöglichen, seine Fehler zu korrigieren oder er muss systemseitig eine Korrektur anbieten. Bei sys-

temseitigen Korrekturen muss der Benutzer über die vorgenommenen Korrekturen informiert werden.

- Der LabCar-Konfigurator muss es erlauben, die Definition von Signalen zu unterbrechen und zu einem späteren Zeitpunkt fortzusetzen. Hierfür ist es notwendig, dass der Zwischenstand gespeichert werden kann und zu einem beliebigen späteren Zeitpunkt wieder darauf zugegriffen werden kann.
- Die Einführung neuer LabCar-Systemkomponenten müssen vom LabCar-Konfigurator aktuell berücksichtigt werden.
- Preisänderungen bei den LabCar-Systemkomponenten müssen ebenfalls aktuell berücksichtigt werden.
- Der LabCar-Konfigurator muss Tabellen verarbeiten können.

2.16 Systemanforderungen

Der LabCar-Konfigurator soll im Intranet und später auch im Internet zur Verfügung stehen. Vor diesem Hintergrund ergeben sich an den LabCar-Konfigurator folgende Systemanforderungen:

- Der LabCar-Konfigurator muss als **Web-Applikation** umgesetzt werden.

  Für das Intranet steht bereits ein Web-Server zur Verfügung, auf dem ein PHP-Interpreter installiert ist, der die serverseitige Programmierung von Web-Applikationen erlaubt (siehe hierzu auch Kapitel 3.5 »PHP für die Programmierung des Web-Servers«). Viele bereits bestehende Intranetanwendungen innerhalb der ETAS GmbH basieren auf PHP. Aus diesem Grund bietet sich für die Umsetzung des LabCar-Konfigurators ebenfalls die Verwendung dieser Programmiersprache an. Da der LabCar-Konfigurator später auch im Internet angeboten werden soll, ist die Umsetzung des Lab- Car-Konfigurators in dieser Form ideal. Für den Zugang zu einer Web-Applikation ist clientseitig ein Web-Browser notwendig. Sowohl im Intranet als auch im Internet gehört ein Webbrowser inzwischen zur üblichen Grundausstattung der Clients.

- Der LabCar-Konfigurator muss über eine **Schnittstelle auf bestehende Datenhaltungssysteme der ETAS GmbH** zugreifen können.

  Die Daten zu den LabCar-Komponenten befinden sich derzeit auf verschiedenen Systemen. Die Preise kommen beispielsweise aus einem SAP-System[11]. Weitere Daten kommen aus Oracle-Datenbanken und E-Matrix[12]. Bei diesen Daten handelt es sich um höchst sensible Unternehmensdaten. Aus Sicherheitsgründen ist daher eine direkte Anbindung an die internen Datenbank-Systeme der ETAS GmbH nicht möglich. Deshalb musste eine Schnittstelle geschaffen werden, über die die Daten kontrolliert ausgetauscht werden können. Eine solche heterogene Systemlandschaft im Bereich des Datenmanagements liegt in Unternehmen häufig vor. Ein Datenaustausch über eine systemunabhängige XML-Schnittstelle bietet sich hier an.

[11] SAP steht für »**S**ysteme, **A**nwendungen, **P**rodukte in der Datenverarbeitung«. Die SAP AG ist drittgrößter Softwarelieferant der Welt und bietet maßgeschneiderte Software für Unternehmen an. (SAP, 2004)

[12] E-Matrix Global - EMG ist führender Anbieter von Softwarelösungen für das Management von Unternehmen. Mit E-Matrix können Themenbereiche wie Datenvisualisierung, Data Mining, Data Warehousing, Business Intelligence und Customer Relationship - CRM verarbeitet und bewältigt werden. (E-Matrix Global , 2003)

3 Konzeption der Anwendung

Die Konzeption, die dem LabCar-Konfigurator zugrunde liegt, umfasst zum einen die Festlegung der genutzten Technologien, zum anderen aber auch inhaltliche Gesichtspunkte, wie die Thematik des Wissensmanagements und die Steuerung von Geschäftsprozessen. Auf diese Aspekte wird im Folgenden eingegangen.

3.1 Struktur des LabCar-Konfigurators

Anhand der in Kapitel 2 »Systemumgebung und Anforderungen« analysierten Systemumgebung, Anforderungen und Anwendungsfälle wurde für den LabCar-Konfigurator folgende Struktur entworfen:

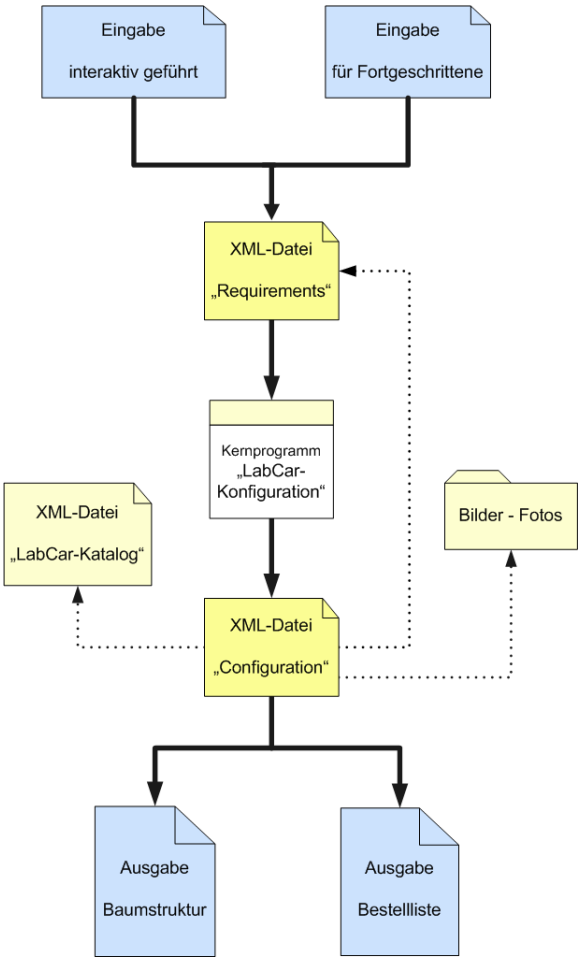

Abbildung 13: Struktur des LabCar-Konfigurators

Wie in Kapitel 2.16 »Systemanforderungen« gefordert, sollte der LabCar-Konfigurator als Web-Applikation umgesetzt werden. Bestandteile einer Web-Applikation sind unter anderem ein Web-Server und Web-Clients. In der oben dargestellten Struktur des LabCar-Konfigurators sind die blauen Komponenten dem Client zugeordnet und die gelben Komponenten dem Server.

Für den Benutzer gibt es zwei verschiedene Einstiegsmöglichkeiten. Die eine besteht in einer interaktiv geführten Dateneingabe. Diese Einstiegsmöglichkeit ist für den noch ungeübten Benutzer gedacht. Für den fortgeschrittenen Benutzer besteht eine weitere Eingabemöglichkeit. Die Dateneingabe erfolgt hier über eine Tabelle. Der Benutzer trägt die gewünschten Signale selbständig in eine Tabelle ein. Anschließend wird die Tabelle an den Server geschickt und serverseitig verarbeitet.

Die vom Benutzer eingegeben Daten werden serverseitig in eine XML-Datei »Requirements« (Englisch für: Anforderungen) geschrieben. Diese XML-Datei enthält alle notwendigen Angaben, die benötigt werden, um eine Konfiguration durchführen zu können.

Das Kernstück des LabCar-Konfigurators bildet das Konfigurationsprogramm »LabCar-Konfiguration«. Dieses Kernprogramm stellt sozusagen die Intelligenz des LabCar-Konfigurators dar. Das Kernprogramm ist in der Lage anhand der vom Benutzer definierten Angaben eine virtuelle Konfiguration durchzuführen. Dabei werden die entsprechenden LabCar-Hardware-Komponenten ausgewählt. Das Ergebnis dieser Konfiguration wird wiederum in eine XML-Datei »Configuration« (Englisch für: Konfiguration oder Zusammensetzung) geschrieben. Das Kernprogramm »LabCar-Konfiguration« ist nicht Gegenstand der vorliegenden Arbeit. Diese Komponente ist deshalb in der obigen Abbildung in der Farbe Weiß dargestellt. Gegenstand der vorliegenden Arbeit ist die Benutzungsoberfläche, sowie die Schnittstellen zum Kernprogramm »LabCar-Konfiguration«.

Bei der Ausgabe kann der Benutzer zwischen verschiedenen Möglichkeiten wählen. Er kann sich das konfigurierte LabCar in einer baumstrukturartigen Darstellung ausgeben lassen. Die Darstellung entspricht dabei dem physikalischen Aufbau des LabCars. Als weitere Möglichkeit kann der Benutzer eine Ausgabe in Form einer Bestellliste auswählen.

Die Schnittstellen für den Datenaustausch beim LabCar-Konfigurator beruhen auf XML. Bei der Ausgabe wird auf einen XML-LabCar-Katalog zu-

gegriffen, der Informationen und technische Details für die einzelnen LabCar-Komponenten enthält. Der LabCar-Katalog existiert derzeit nicht in dieser Form. Die Daten kommen aus ganz unterschiedlichen Dokumenten (White Papers und Printdokumente) und Datenbanken (SAP und E-Matrix). Mit dem XML-LabCar-Katalog wurde eine Schnittstelle geschaffen, über die die Daten ausgetauscht werden können. Zusätzlich wird auf ein Bilderarchiv zugegriffen, in dem sich Bilder einzelner LabCar-Komponenten befinden. Die Formate für die Bilder sind die bei Web-Systemen häufig verwendeten und hierfür optimierten Formate JPEG[13] und GIF[14].

Die XML-Dateien »Requirements« und »Konfiguration« stellen die Verbindung zur Benutzungsoberfläche her. Die XML-Datei »Requirements« enthält die vom Benutzer definierten Anforderungen, die er an das LabCar hat und ist gleichsam das Ergebnis der Benutzereingaben. Die Ausgabe des Konfigurationsergebnisses wird über die XML-Datei »Configuration« ermöglicht, wobei die XML-Datei »Configuration« über XPath auf die XML-Dateien »Requirements« und »LabCar-Katalog« zugreift.

[13] JPEG steht für »Joint Photographic Experts Group« und bezeichnet auch das von diesem Gremium entwickelte Kompressions-Verfahren für digitale Bilder. Die Dateinamen dieses Formats enden meist auf `.jpg` oder `.jpeg`.

[14] GIF steht für »Graphics Interchange Format« und ist ein digitales Bildformat mit guter verlustfreier Komprimierung für Bilder mit geringer Farbtiefe (2 bis 256 Farben). Nach heutigen Maßstäben ist das GIF-Format für die Speicherung digitaler Fotos aufgrund der geringen Farbtiefe allerdings ungeeignet. Die Dateinamen dieses Formats enden in der Regel auf `.gif`.

3.2 Web-Applikation

Der LabCar-Konfigurator ist als Web-Applikation konzipiert. Web-Applikationen haben ganz spezifische Eigenschaften. Die Architektur einer Web-Applikation lässt sich folgendermaßen darstellen:

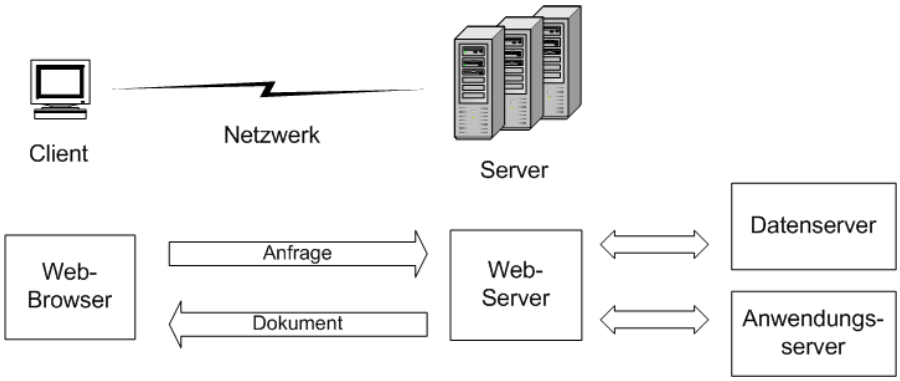

Abbildung 14: Web-Architektur

Balzert (2000, S. 716) beschreibt die Web-Architektur als »Verteilung einer Anwendung auf ein Netzwerk (Internet, Intranet), das aus vielen Web-Clients (Clients, auf denen ein Web-Browser läuft), mindestens mit einem Web-Server, sowie Anwendungs- und Datenserver besteht. Die Verbindung zwischen Web-Client und Web-Server erfolgt über das HTTP-Protokoll, das pro Anforderung jeweils eine neue Verbindung aufbaut.«

Durch die inzwischen weite Verbreitung des Internet wird die Technik des Internet zunehmend auch für firmeninterne Netze (Intranet) verwendet.

»Auf dem Server eines Unternehmens muss, um den Zugriff über Web-Browser zu ermöglichen, ein Web-Server laufen. Dieser stellt für alle Benutzer, Mitarbeiter und Außenstehende, den Eintrittspunkt in die Unternehmenslösung dar. Der Web-Server muss den Zugriff auf, beziehungsweise die Veränderung von Unternehmensdaten unterstützen. Zu diesem Zweck haben sich eine Reihe von serverseitigen Web-Konzepten etabliert, die vielfach auf Skripten basieren.« (Balzert, 2000, S. 946)

Serverseitige Skripte

Statt einer einfachen HTML-Seite ruft der Web-Browser ein Skript auf dem Server auf und übergibt gleichzeitig Parameter an das Skript. Daten kön-

nen vom Benutzer auch über ein Formular eingegeben werden. Die eingegebenen Daten werden dann zum Web-Server geschickt. Der Web-Server übergibt die eingegeben Daten als Parameter an ein Skript. Das Skript kann die Daten beispielsweise prüfen und anschließend in eine Datenbank oder Datei schreiben.

Balzert (2000, S. 946) beschreibt ein serverseitiges Skript als Prozedur, die auf dem Web-Server als Reaktion auf die Anfrage gestartet wird und die vom Browser übermittelte Parameter erhält. Das Skript führt eine Reihe von Aktionen durch und produziert eine Ausgabe. Diese Ausgabe, meist eine HTML-Seite, wird zum Browser geschickt und dem Benutzer angezeigt.

Neuere Browser sind inzwischen in der Lage, außer HTML-Dateien auch XML-Dateien anzuzeigen. Diese Möglichkeit wird beim LabCar-Konfigurator genutzt.

Clientseitige Skripte

Eine andere Kategorie stellen clientseitige Skripte dar. Es handelt sich um kleine Programme, die in eine HTML-Seite eingebettet sind und vom Browser ausgeführt werden.

Balzert (2000, S. 946) schreibt: »Bei der Eingabe von Daten in HTML-Formulare ist es häufig sinnvoll, die Eingaben zu prüfen, bevor sie an den Web-Server geschickt werden. Mit einem clientseitigen Skript lässt sich oft eine bessere Reaktionszeit der Anwendung erzielen und außerdem die Netzbelastung senken, als wenn fehlerhafte Daten erst zum Server geschickt und Fehlermeldungen wieder zum Browser übertragen werden müssen.«

Beim LabCar-Konfigurator werden sowohl serverseitige Skripte als auch clientseitige Skripte eingesetzt. Dadurch wird die Benutzerfreundlichkeit und der Komfort der Anwendung erhöht.

Die Entscheidung, den LabCar-Konfigurator als Web-Applikation umzusetzen bietet weitere Vorteile. Balzert (2000, S.946) fasst die Vorteile einer Web-Architektur wie folgt zusammen:

- Viele Mitarbeiter kennen inzwischen Web-Browser. Dadurch reduziert sich der Trainingsaufwand.

- Die Wartung vereinfacht sich. Auf den Client-Computersystemen ist neben dem Betriebssystem nur noch der Web-Browser zu installieren. Alle anderen Komponenten der Unternehmenslösung liegen auf einem oder wenigen Servern.
- Der Zugang von außen (z. B. für Kunden) zu den firmeneigenen Anwendungssystemen vereinfacht sich. Kunden können z.B. über ihren privaten Internet-Zugang mit Anwendungen des Unternehmens arbeiten. Es müssen keine zusätzlichen Infrastrukturen geschaffen werden.
- Die Hardware-Anforderungen an die Client-Computersysteme werden reduziert, da sie nur noch den Web-Browser ausführen müssen.

3.3 Darstellungs- und Serviceorientierte Web-Applikation

Für Bodoff (2002) ist eine Web-Applikation eine dynamische Erweiterung eines Web-Servers. Dabei definiert sie zwei Arten von Web-Applikationen.

1. Darstellungsorientierte Web-Applikationen
2. Serviceorientierte Web-Applikationen

Die darstellungsorientierten Web-Applikationen erzeugen dynamische Web-Seiten. Abhängig von den Eingaben des Benutzers werden unterschiedliche Varianten von Web-Seiten-Code (HTML) erzeugt.

Eine serviceorientierte Web-Applikation dagegen realisiert den Endpunkt eines Webdienstes. Serviceorientierte Web-Applikationen werden dabei oft über darstellungsorientierte Web-Applikationen aufgerufen.

Der LabCar-Konfigurator ersetzt eine Dienstleistung, beziehungsweise einen Service, der seither durch ETAS-Mitarbeiter angeboten wird. Der Service einer virtuellen LabCar-Konfiguration wird über das Kernprogramm LabCar-Konfiguration realisiert und dem Benutzer zur Verfügung gestellt. Über die Benutzungsoberfläche hat der Benutzer Zugriff auf diesen Service. Der LabCar-Konfigurator als Web-Applikation insgesamt ist damit einer serviceorientierten Web-Applikation zuzuordnen.

Das Kernprogramm Lab- Car-Konfiguration wird über die Benutzungsoberfläche aufgerufen. Die Benutzungsoberfläche des LabCar-Konfigurators für

sich gesehen, stellt nach Bodoff (2002) eine darstellungsorientierte Web-Applikation dar. Der Benutzer erhält beispielsweise, abhängig von den Signalen, die er definiert hat, dynamisch generierte Web-Seiten, die den aktuellen Status seiner persönlichen Signalliste anzeigen.

3.4 Web-Technologien

Der ursprüngliche Mechanismus, Benutzereingaben auf einem Web-System zu verarbeiten, ist der Mechanismus des Common Gateway Interface (CGI). Der CGI ist ein Standard, der es dem Web-Benutzer erlaubt, Anwendungen auf einem Server auszuführen. Diese Möglichkeit bringt aber auch Sicherheitsprobleme mit sich. CGI-Module können in jeder beliebigen Programmiersprache geschrieben werden, auch in einer Skriptsprache.

Dabei gibt es heute zwei verschiedene Vorgehensweisen: die eine basiert auf kompilierten Modulen und die andere auf interpretierten Skripten.

Lösungen mit kompilierten Modulen verwenden bereits kompilierte Binärdateien, die geladen werden können, und vom Web-Server ausgeführt werden. Diese Module greifen auf standardisierte Schnittstellen, so genannte APIs[15] zu. APIs stellen die Informationen, die bei der Benutzeranfrage übertragen werden, zur Verfügung. Die Werte und Namen aller Informationsfelder werden dabei über die URL übertragen. Diese Module erzeugen dann eine HTML-Ausgabedatei, die an den anfragenden Webbrowser geschickt wird. Einige bekannte Implementierungen dieses Ansatzes sind die Microsoft Internet Server API, die Netscape Server API und Java Servlets.

Kompilierte Module sind insbesondere für große Serienanwendungen geeignet und dort sehr effizient. Die größten Nachteile liegen dagegen in der Entwicklung und Wartung der Anwendung. Kompilierte Module verbinden in der Regel die Geschäftslogik mit der Erstellung der HTML-Seiten. Die Module enthalten oft viele Zeilen mit HTML-Code. Dies ist dann für den Programmierer oft verwirrend und schwer lesbar. Außerdem müssen die Module bei Änderungen und Updates stets neu kompiliert und neu geladen werden. Ein kompiliertes Modul kann dann typischerweise ein Geschäftsanwendungsprogramm sein, das HTML-Code ausgibt.

[15] »API ist die Abkürzung für Application Programming Interface (Englisch für Schnittstelle zur Applikationsprogrammierung).«
(WIKIPEDIA 2004, Stichwort: »Application Programming Interface«)

Interpretierte Skripte dagegen sind typischerweise HTML-Seiten, die zusätzlich Geschäftslogik verarbeiten. Eine Skriptseite liegt im Dateisystem eines Web-Servers. Die Skriptseite enthält Skripte, die vom Web-Server interpretiert werden. Die Skripte interagieren mit Geschäftsobjekten auf einem Anwendungsserver und geben schließlich HTML-Seiten aus. Die Namenserweiterungen der Dateien (Engl.: file extension) zeigen dem Web-Server an, mit welchem Filter er die Seite interpretieren muss. Einige bekannte Anbieter in dieser Kategorie sind Java Server Pages (JSP), Microsoft Active Server Pages (ASP) und PHP. Die entsprechenden typischen Namenserweiterungen lauten dabei `.aspx`, `.jsp` und `.php`.

Die folgende Abbildung zeigt die Beziehungen zwischen den einzelnen Web-Komponenten dieser Technologie und dem Web-Server. Der Datenserver in der Abbildung steht dabei für jede beliebige serverseitige Quelle. Diese schließt auch externe Systeme und andere Anwendungen mit ein.

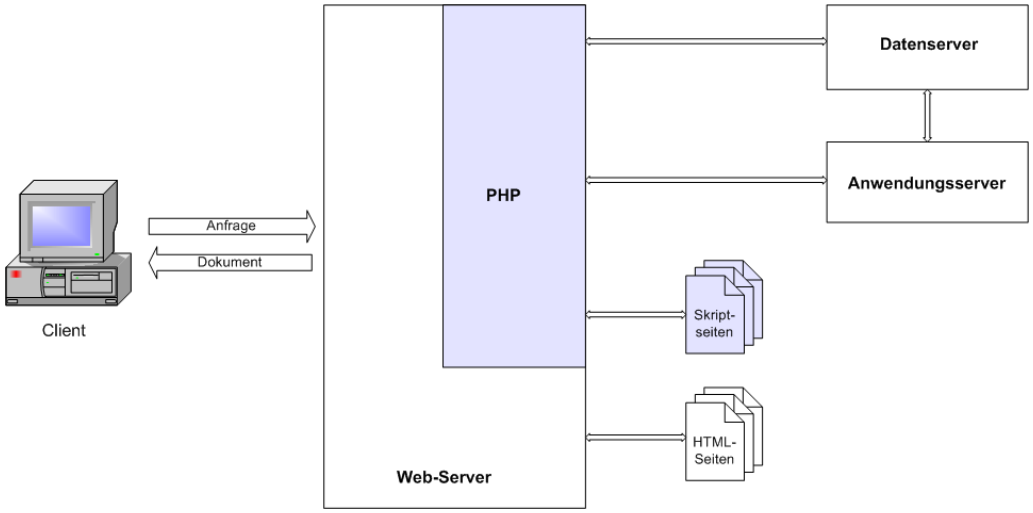

Abbildung 15: Skript-Technologie

Die Skript-Technologie wird vom Web-Server nur dann aufgerufen, nachdem er festgestellt hat, ob die vom Client aufgerufene Seite tatsächlich Skripte enthält, die interpretiert werden müssen. Dies geschieht über die Namensendung der Dateien.

Für die serverseitige Programmierung des LabCar-Konfigurators wurde PHP gewählt. Wenn der Web-Server eine Anforderung mit einer entspre-

chenden Namensendung erhält, lokalisiert er diese Datei im entsprechenden Verzeichnis und übergibt diese Seite an den entsprechenden Anwendungsserver, beziehungsweise den entsprechenden Filter. Die Namensendung beim LabCar-Konfigurator ist `.php`. Der Anwendungsserver bereitet die Seite auf, indem er alle serverseitigen Skripte auf der Seite auswertet (interpretiert) und mit allen erforderlichen serverseitigen Ressourcen kommuniziert. Das Ergebnis ist eine richtig formatierte HTML-Seite, die an den anfragenden Web-Browser zurückgeschickt wird.

Der Vorteil von Skript-Seiten liegt darin, dass sie leicht und schnell entwickelt und eingesetzt werden können. Skript-Seiten enthalten typischerweise so gut wie keine Geschäftslogik. Skript-Seiten werden meist als Bindeglied benutzt, um die HTML-Benutzungsoberfläche des Systems mit der Geschäftslogik zu verbinden.

Die Wahl der Technologie, die bei einer Web-Applikation verwendet wird, ist abhängig von der Art der Anwendung, vom Unternehmen und auch vom Entwicklerteam. Auf einem Server können viele verschiedene Technologien gleichzeitig eingesetzt werden.

Beim Entwurf einer Web-Applikation ist es daher notwendig, ein Modell des gesamten Systems zu erstellen, das alle wesentlichen Komponenten der Web-Applikation enthält. Server, Browser, Web-Seiten und die verwendeten Technologien sind wichtige architektonische Elemente und gehören zum Modell.

3.5 PHP für die Programmierung des Web-Servers

PHP wurde 1995 von Rasmus Lerdorf entwickelt. PHP stand für 'Personal Home Page Tools'. Seit PHP 3 steht PHP als Akronym für 'PHP: Hypertext Preprocessor'. PHP ist eine serverseitig interpretierte Skriptsprache. Die PHP-Syntax ist weitgehend an die Syntax von C[16], beziehungsweise von PERL[17], angelehnt. (vgl. The PHP Group 2004, Stichwort: »PHP Handbuch - Die Geschichte von PHP«)

Krause (2000, S.5) schreibt über PHP: » [...] von allem nur das Beste [...], die perfekte Sprache für's Web [...] mit der bekannten Syntax von C, der Einfachheit von ASP und der Leistungsfähigkeit von PERL«.

PHP wird hauptsächlich zur Erstellung dynamischer Web-Sites[18] verwendet. PHP ist ein Open-Source-Projekt[19] und frei erhältlich. PHP zeichnet sich besonders aus durch die leichte Erlernbarkeit. Da PHP frei erhältlich und leicht erlernbar ist, und ein PHP-Interpreter bereits auf dem Intranet-Server von ETAS installiert war, wurde für die Umsetzung des LabCar-Konfigurators PHP gewählt.

PHP kann direkt in eine HTML-Seite eingebettet werden oder auch für sich alleine stehen. Der PHP-Quelltext wird nicht direkt an den Browser übermittelt, sondern zuerst vom Interpreter auf dem Web-Server ausgeführt. Die Ausgabe des Skriptes wird dann an den Browser geschickt. In den meisten Fällen ist die Ausgabe eine HTML-Seite oder HTML-Code. Daher rührt auch der Name »Hypertext Preprocessor«. »Hypertext Preprocessor« bedeutet: vor (Engl.: pre-) der Ausgabe des Hypertextes wird noch etwas verarbeitet, beziehungsweise aufbereitet (Engl.: to process).

PHP 5 ist die derzeit aktuellste freigegebene Version von PHP. Das Entwicklerteam von PHP umfasst dutzende Entwickler, sowie dutzende andere, welche an PHP verwandten Projekten oder dem Dokumentationsprojekt

---

[16] C ist eine Programmiersprache, die in den frühen 70er Jahren entwickelt wurde. C ist sowohl im kommerziellen als auch im Open-Source-Bereich weit verbreitet und sehr beliebt. (WIKIPEDIA 2004, Stichwort: C (Programmiersprache))

[17] PERL ist eine Programmiersprache, genauer eine Skriptsprache, und steht für »Practical Extraction and Report Language«. (WIKIPEDIA 2004, Stichwort: PERL)

[18] Eine Website ist die Gesamtheit aller HTML-Dokumente eines Angebots oder einer Selbstdarstellung im WWW, zum Beispiel eines Unternehmens, einer Organisation oder einer Privatperson. Die Start- beziehungsweise Leitseite einer Website ist die Homepage. (Wissen.de 2004, Stichwort: Web-Site)

[19] Open-Source bedeutet: für jedermann frei zugänglich.

arbeiten. (vgl. The PHP Group 2004, Stichwort: »PHP Handbuch - Die Geschichte von PHP«)

»Heute wird PHP von (schätzungsweise) hunderttausenden Entwicklern verwendet, und es wird von mehreren Millionen Sites berichtet, auf welchen PHP installiert ist, was mit über 20% der Domains im Internet zu Buche schlägt.«
(Kronsbein, 2004)

3.6 Session-Management

Da die Kommunikation zwischen Client und Server auf dem verbindungslosen HTTP-Protokoll beruht, ist es für den Server keine einfache Aufgabe, jede Anfrage des Clients im Auge zu behalten und sie gleichzeitig mit vorangegangenen Anfragen in Verbindung zu bringen. Dies ist deshalb so schwierig, weil mit jeder neuen Anfrage eine komplett neue Verbindung aufgebaut wird.

Für den LabCar-Konfigurator ist es notwendig, den Client-Status serverseitig festzuhalten und zu steuern. Einerseits müssen die jeweiligen Benutzereingaben zum aktuellen Signal festgehalten werden. Andererseits muss auch die jeweilige individuelle Signalliste für jeden Benutzer mitgeführt werden. Um diese Aufgaben lösen zu können, ist ein so genanntes Session-Management notwendig.

Eine Session (Englisch für: Sitzung) repräsentiert eine einzelne zusammenhängende Nutzung eines Systems durch einen Benutzer. Eine Session beinhaltet normalerweise viele ausführbare Web-Seiten und viele Interaktionen mit der Geschäftslogik auf einem Anwendungsserver. Um das Ziel eines Geschäftsvorfalles (Engl.: use case) zu erreichen, müssen oft viele ausführbare Web-Seiten erfolgreich aufgerufen werden. Dabei ist es häufig notwendig, die Übersicht über den Ablauf einer Session zu behalten. Hierfür gibt es verschiedene Möglichkeiten.

Eine Möglichkeit, eine Session bei einer Web-Appliaktion aufrechtzuerhalten, besteht darin, einen eindeutigen Schlüssel zu vergeben. Dieser eindeutige Schlüssel wird jedesmal in der URL an das System übergeben und identifiziert so eine Session eindeutig. Der eindeutige Schlüssel wird serverseitig über ein eigenes Verzeichnis oder in einer serverseitigen Übersicht

verwaltet. Bei diesem Mechanismus wird jede einzelne Session-URL dynamisch erzeugt. Damit sind dann alle Parameter eingeschlossen, die entweder den ganzen Sessionstatus beinhalten oder nur den Schlüssel, der sich in einem serverseitigen Verzeichnis befindet.

Eine weitere Möglichkeit besteht darin, den Sessionstatus in Cookies[20] zu speichern. Dabei können die Werte des Sessionstatus selbst oder auch nur ein eindeutiger Schlüssel abgespeichert werden, wobei der Schlüssel dann serverseitig in einem Verzeichnis verwaltet wird. Hat der Benutzer im Web-Browser jedoch Cookies deaktiviert, kann er die Web-Applikation nicht in ihrer vollen Funktionalität nutzen.

Die dritte Möglichkeit besteht darin, jedesmal die Werte selbst in der URL zu übergeben. Bei vielen Werten wird dies jedoch unübersichtlich und schränkt die Performance (Verarbeitungsgeschwindigkeit) der Applikation ein. Des Weiteren ergibt sich bei sensibleren Daten hier das Problem, dass alle Daten immer und für jeden offen eingesehen werden können, was unter Umständen nicht erwünscht ist.

Aufgrund der Nachteile der beiden letzten Möglichkeiten, wurde für den LabCar-Konfigurator die erste Möglichkeit, die Vergabe eines eindeutigen Schlüssels, gewählt. Im nachfolgenden Flussdiagramm ist die Vergabe eines Benutzerschlüssels für das Session-Management beim LabCar-Konfigurator schematisch dargestellt.

---

[20] »Ein Cookie (amerikan. Englisch: Plätzchen, Keks) bezeichnet Informationen, die ein Webserver zu einem Browser sendet, um dem zustandslosen HTTP-Protokoll die Möglichkeit zu geben, Information zwischen Aufrufen zu speichern. Cookies werden in den Header-Teilen von HTTP-Anfragen und HTTP-Antworten übertragen. Man kann zwischen persistenten Cookies und Session-Cookies unterscheiden. Erstere werden dauerhaft gespeichert (beispielsweise auf der Festplatte), während letztere nur für die Länge einer Sitzung gespeichert werden. Wenn ein Webserver Cookies zu einem Webbrowser sendet, werden sie lokal auf dem Endgerät gespeichert (auf Computern üblicherweise in einer Textdatei). Die Cookies können dann bei jedem Aufruf der entsprechenden Website vom Server abgefragt werden. Damit ist eine beständige Verbindung zwischen dem Browser und Server gewährleistet. Cookies können beliebige Informationen enthalten, die zwischen Browser und Server ausgetauscht wurden. Dieses Konzept wurde von der Firma Netscape entwickelt und ... spezifiziert.«
(WIKIPEDIA 2004, Stichwort: »Cookie«)

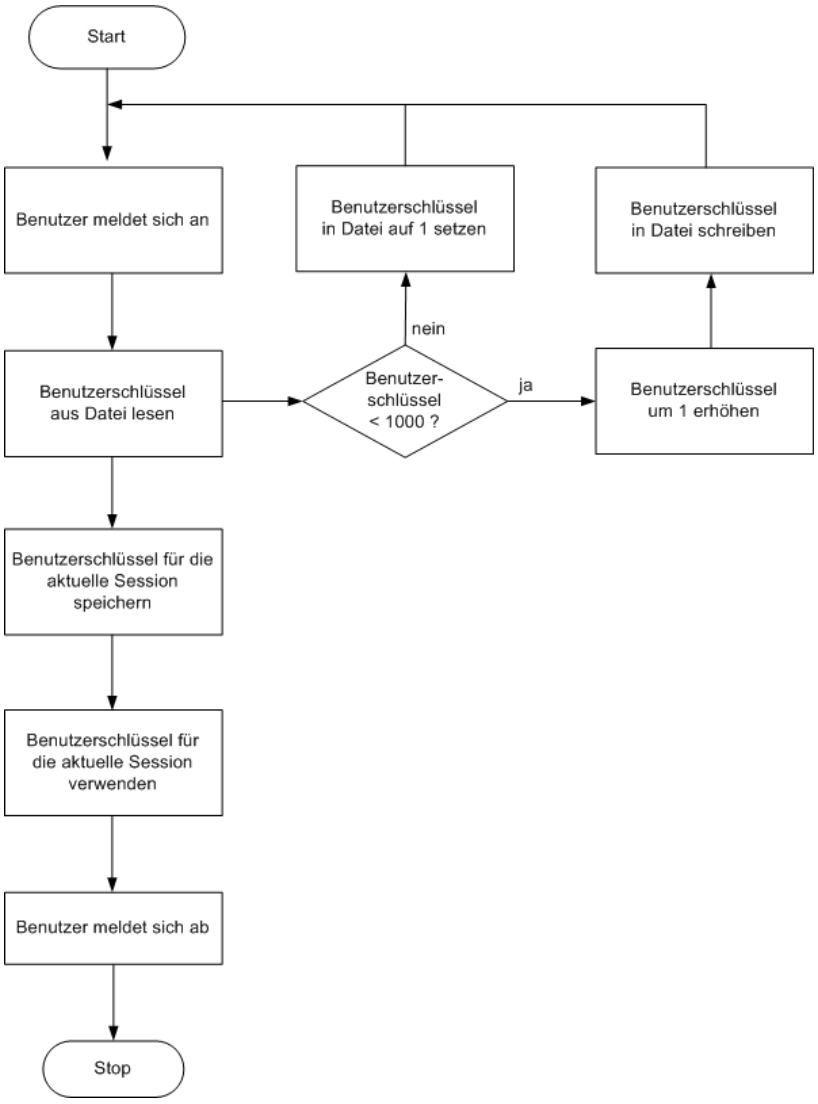

Abbildung 16: Flussdiagramm: »Vergabe eines Benutzerschlüssels«

Für den Benutzerschlüssel wird eine fortlaufende Nummer vergeben. Da der Zugriff auf den LabCar-Konfigurator im Intranet in einem begrenzten Umfang stattfindet, wird der Schlüssel bis 999 weitergezählt. Danach wird als Benutzerschlüssel wieder die 1 vergeben.

Der LabCar-Konfigurator wird vorerst nur im Intranet angeboten. Deshalb wird für den Benutzerschlüssel keine hohe Sicherheitsstufe benötigt. Wenn der LabCar-Konfigurator später auch im Internet zur Verfügung gestellt wird und eine höhere Sicherheitstufe gewünscht wird, könnte dies durch ein zusätzliches Zufallselement erreicht werden. Beispielsweise könnte die laufende Nummer mit einem zusätzlichen längeren und zufallsgenerierten

String (Englisch für: Zeichenkette) verkettet werden. Dies hätte den Vorteil, dass man den Schlüssel nicht ohne Weiteres erraten kann und sich damit nicht in eine laufende Kommunikation einklinken kann. Für den LabCar-Konfigurator im Intranet und den hier zu entwicklenden Prototyp ist eine so hohe Sicherheitsstufe allerdings nicht notwendig. Im Vordergrund steht derzeit nicht der Sicherheitsaspekt, sondern die Notwendigkeit eines Session-Managements.

3.7 Steuerung von Geschäftsprozessen

Nach Conallen (2003) erlauben es Web-Applikationen dem Benutzer, Einfluss auf Geschäftsobjekte und Geschäftsprozesse auf dem Server zu nehmen. Dies ist auch beim LabCar-Konfigurator der Fall. Eine virtuelle LabCar-Konfiguration stellt ein Geschäftsobjekt, beziehungsweise einen Geschäftsprozess, dar. Im Gegensatz zu Web-Sites kann der Benutzer bei Web-Applikationen Geschäftsprozesse auf dem Server beeinflussen. Wenn auf dem Server kein Geschäftsprozess abläuft, liegt auch keine Web-Applikation vor. Systeme, bei denen der Web-Server - oder ein Anwendungsserver, der einen Web-Server verwendet, um damit Benutzereingaben zu verarbeiten - es dem Benutzer erlaubt, über den Web-Browser Geschäftsprozesse zu beeinflussen, werden als Web-Applikationen bezeichnet.

Die Architektur einer Web-Site ist statisch. Die Architektur einer Web-Applikation enthält dieselben prinzipiellen Komponenten, wie eine Web-Site: einen Web-Server, eine Netzwerkverbindung und einen Web-Client, beziehungsweise einen Webbrowser. Web-Applikationen beinhalten zusätzlich einen Applikationsserver. Dies unterscheidet eine Web-Applikation von einer Web-Site. Der Applikationsserver ermöglicht es der Web-Applikation, Geschäftsprozesse und -objekte zu steuern.

Auch für die einfachsten Web-Applikationen gilt: Der Benutzer muss Daten übermitteln und nicht nur zu einer neuen Web-Seite navigieren. Typischerweise kann der Benutzer bei einer Web-Applikation sehr verschiedenartige Eingaben übermitteln. Er kann einfachen Text eingeben, aus Listen auswählen, Ankreuzfelder (Engl.: checkboxes) ankreuzen, oder auch Binärdaten oder ganze Dateien übertragen.

Für die interaktiv gesteuerte Dateneingabe werden beim LabCar-Konfigurator Formulare und Frames eingesetzt. Conallen (2003) zählt Formulare

und Frames zu den möglichen Bestandteilen einer Web-Applikation.

Eine Web-Applikation beschreibt Conallen (2003) als ein »Web-System, das eine Geschäftslogik nach außen hin zur Verfügung stellt ... Web-Applikationen erlauben es dem Benutzer, Einfluss auf Geschäftsobjekte, beziehungsweise Geschäftsprozesse, zu nehmen ... Web-Applikationen beinhalten außerdem einen Applikationsserver [beziehungsweise eine Applikationsserveranwendung, d.Verf.]. Der Applikationsserver ermöglicht es der Web-Applikation, Geschäftsprozesse und -objekte zu steuern.«

Das Kernstück, das Konfigurationsprogramm »LabCar-Konfiguration«, beinhaltet die eigentliche Geschäftslogik der Web-Applikation. Das Konfigurationsprogramm »LabCar-Konfiguration« ist eine selbständige Einheit, eine eigenständige Applikation, die unabhängig von der Benutzungsoberfläche in einer beliebigen Sprache und Technologie umgesetzt werden kann und als eigenständige Anwendung auf dem Server, beziehungsweise auf dem Anwendungsserver, läuft.

Die erzeugten XML-Dateien »Requirements« und »Configuration« stellen Geschäftsobjekte bzw. Geschäftsprozesse dar. Diese Geschäftobjekte sind sowohl Teil, als auch Ergebnis von Geschäftsprozessen.

3.8 Wissensmanagement

Der LabCar-Konfigurator ist außerdem ein Instrument für das Wissensmanagement. Wie im vorigen Kapitel beschrieben, stellen die erzeugten XML-Dateien »Requirements« und »Konfiguration« Geschäftsobjekte, beziehungsweise Geschäftsprozesse dar. In der herkömmlichen Vorgehensweise werden diese Geschäftsobjekte gemeinsam vom Kunden, vom ETAS-Verkäufer und den LabCar-Hardwarefachleuten erzeugt und teilweise in expliziter Form niedergeschrieben, existieren manchmal aber auch nur als implizites Wissen im Kopf dieser Personen. Beim LabCar-Konfigurator nehmen nun diese, bisher zum Teil nur als implizites Wissen existierenden Geschäftsobjekte in den XML-Dateien »Requirements« und »Konfiguration« explizit Gestalt an.

Nonaka und Takeuchi (1997) beschreiben, wie sich Wissen epistemologisch (die Erkenntnis betreffend) und ontologisch (das Wesen und die Eigenschaften betreffend) entwickelt. Mittels der »Wissensspirale« stellen sie dabei die Umwandlungsformen zwischen explizitem und implizitem Wissen dar.

Eine Form der Umwandlung ist die Externalisierung. Bei der Externalisierung wird implizites Wissen artikuliert und in explizite Konzepte umgewandelt. Externalisierung ist der Schlüsselprozess bei der Wissensumwandlung, da aus bestehendem impliziten Wissen neue explizite Konzepte geschaffen werden. Weitere Ausführungen zum Thema Wissensumwandlung und Wissensmanagement finden sich bei Probst et al. (1999) und bei Schütt (2000).

Mit Hilfe des LabCar-Konfigurators wird implizites Wissen von ETAS-Mitarbeitern in explizite Formen umgewandelt. Damit ist der LabCar-Konfigurator ein Instrument für das Wissensmanagement. Implizites Wissen von ETAS-Verkäufern wird externalisiert und damit Kunden und anderen ETAS-Mitarbeitern verfügbar gemacht. Darüber hinaus kann der LabCar-Konfigurator neue Verkäufer im Einlernprozess unterstützen. Die Externalisierung von Wissen bildet die Basis für Wissensumwandlung, Wissensverteilung und, nicht zu vergessen, zuletzt auch die Basis für die Schaffung von neuem Wissen. Die Schaffung neuen Wissens bildet die Grundlage für Innovation.

3.9 XML

Web-Applikationen müssen mit vielfältigen Problemen fertig werden. Daten müssen von verschiedenartigsten Systemen, Datenbanken, Verzeichnissen und Anwendungen übertragen werden. Web-Applikationen müssen in der Lage sein, mit anderen verschiedenartigsten Systemen und Technologien kommunizieren zu können. Beim LabCar-Konfigurator mussten diese Probleme ebenfalls gelöst werden. XML bietet für die Lösung dieser Problematik hervorragende Möglichkeiten.

XML steht für Extensible Markup Language (dt.: erweiterbare Auszeichnungssprache). Eine Auszeichnungssprache dient der Beschreibung von Daten oder des Verfahrens, beziehungsweise der Schritte, die für die Darstellung der Daten notwendig sind. XML ist ein Standard zur Erstellung strukturierter, maschinen- oder menschenlesbaren Dateien. Die Dokumenttyp-Definition (Engl.: Document Type Definition - Abkürzung: DTD) definiert dabei den grundsätzlichen Aufbau von XML-Dateien. Die Namen der einzelnen Strukturelemente können frei gewählt werden. Programme die XML-

Daten verarbeiten heißen XML-Parser[21]. (vgl. WIKIPEDIA 2004, Stichwort: »XML«)

Mit XML und den zugehörigen Dokumenttyp-Definitionen kann leicht eine Schnittstelle für die Datenübertragung zu und von anderen Systemen, Datenbanken und Applikationen geschaffen werden. Außerdem kann die Darstellung von XML-Dokumenten über XSLT (Extensible Stylesheet Language Transformation) gesteuert werden. Der bekannteste Nutzen von XML liegt in der Trennung von Inhalt (Engl.: content) und Darstellung (Engl.: presentation). Das XML-Dokument selbst enthält den Inhalt und XSLT sorgt für eine geeignete Darstellung auf dem Client.

Beim LabCar-Konfigurator wird die XML-Technologie sowohl für den Datenaustausch und die Datenintegration, als auch für die Datenpräsentation genutzt.

3.10 Datenaustausch und Datenintegration mit XML

Nehrbass (2004) bemerkt, dass XML auf Grund seiner universellen und flexiblen Struktureigenschaften viel versprechende Möglichkeiten als Hilfsmittel für Datenaustausch und Datenintegration eröffnet. Die Stärke von XML liegt für Nehrbass darin, dass XML zur strukturierten Darstellung nahezu beliebiger Daten universell einsetzbar ist.

»Als zusätzliche Abstraktionsebene kann XML helfen, unterliegende Strukturen von Datenquellen zu verbergen, und kann Anwendern und Applikationen eine einheitliche Schnittstelle zur Verfügung stellen. Entwickler können von quellenspezifischen Datenformaten abstrahieren und bei der Lösung von Datenintegrationsproblemen diese direkt auf inhaltlicher Ebene anstatt auf der Ebene verschiedener Datenformate adressieren. [...] XML kann bei der Integration von Applikationsdaten als Zwischenschicht und XSLT als universeller Übersetzer eingesetzt werden.«
(Nehrbass, 2004)

Mit der XML-Abfragesprache XQuery können integrierte Sichten über beliebige Kollektionen von XML-Datenquellen erzeugt werden . XPath ist Teilmenge von XQuery.

[21] Engl.: to parse - übersetzen

Ziel der Datenintegration ist für Nehrbass (2004), dass mehrere unabhängige Datenquellen über ein einziges globales Schema (DTD oder XML-Schema) abgefragt werden könnnen. Die Hauptanwendung für XML liegt für ihn im Datenaustausch und in der Datenintegration.

Middendorf (2002) nennt XML ein system- und herstellerunabhängiges Datenaustauschformat zwischen heterogenen Applikationen. XML ermöglicht es über die DTD die Daten inhaltlich zu beschreiben. Die Erhaltung der Bedeutung der Daten ist beim Datenaustausch zwischen Applikationen der kritische Punkt. Wenn nun die beteiligten Applikationen über Import-/ Export-Module verfügen, die nicht nur einfach den XML-Text importieren, sondern auch die Semantik der Daten erkennen und beibehalten können, spricht man von semantischer Interoperabilität der Daten.
(vgl. Middendorf, 2002)

Es werden nicht nur einfach die Daten übernommen, sondern auch ihre inhaltliche Bedeutung. Beim LabCar-Konfigurator bedeutet dies beispielsweise, das die Beschreibung einer LabCar-Hardwarekomponenten nicht als einfacher Text übernommen wird, sondern dass auch die Bedeutung der einzelnen Beschreibungsbestandteile einer LabCar-Hardwarekomponenten erhalten bleibt. Mit XML kann dies sehr gut gelöst werden, indem über die DTD die Datenaustausch-Schnittstelle festgelegt wird. Alle beteiligten Applikationen können so unter der Kenntnis der Semantik der Elemente der DTD einen konsistenten Import/Export der Daten realisieren.

Beim LabCar-Konfigurator sind zudem auch die Schnittstellen zwischen der Benutzungsoberfläche und dem Kernprogramm »LabCar-Konfiguration« über XML realisiert. Die Benutzeranforderungen werden in einer XML-Datei »Requirements« festgehalten und an das Kernprogramm »LabCar-Konfiguration« weitergegeben. Das Ergebnis des Konfigurationsprogrammes ist wiederum eine XML-Datei. Die XML-Datei »Konfiguration« bildet dabei die Schnittstelle zwischen Kernprogramm »LabCar-Konfiguration« und Benutzungsoberfläche. Die Datenausgabe und die Datenpräsentation der XML-Datei »Konfiguration« wurde mit Hilfe von XSLT durchgeführt.

3.11 Darstellung des Konfigurationsergebnisses mit XSL

Die Darstellung des Konfigurationsergebnisses wurde mit Hilfe von XSL realisiert. In diesem Kapitel geht es um die prinzipiellen Möglichkeiten,

die XSL für die Darstellung von XML-Dateien bietet.

Wie die Ausgabe konkret aussieht, ist im nächsten Kapitel »Design der Benutzungsoberfläche« und dort im Unterkapitel 4.11 »Ausgabe« dargestellt.

Die Sprache XSL (Extended Stylesheet Language) besteht aus zwei Komponenten:

1. aus einer Komponente zur Formatierung von XML-Daten, die auch XSL-FO (XSL Formatting Objects) genannt wird und
2. aus einer Komponente zur Transformation von XML-Daten in andere XML-Daten. Für die Transformationskomponente hat sich die Abkürzung XSLT (Extended Stylesheet Language Transformation) etabliert.

Mit XSLT kann ein XML-Dokument in ein anderes XML-Dokument, beziehungsweise in ein HTML-Dokument, transformiert werden. XPath ist eine Sprache, die es emöglicht, Teile in einer XML-Datei anzusprechen. XPath kann innerhalb von XSLT verwendet werden und bildet gleichsam eine Subsprache innerhalb von XSL.

Der LabCar-Konfigurator verwendet bei der Ausgabe und Darstellung des Konfigurationsergebnisses zwei verschiedene Arten von XSL-Stylesheets. Für die Ausgabe der Baumstruktur wurde ein Push-Stylesheet verwendet. Für die Ausgabe der Bestellliste wurde ein Pull-Stylesheet implementiert.

Beim Push-Stylesheet (Engl.: to push - schieben) werden die Elemente in der Struktur, wie sie im XML-Quelldokument vorliegen, ausgegeben. Beim Pull-Stylesheet dagegen, wird für die Ausgabe eine eigene Struktur aufgebaut. Der Inhalt wird aus dem, beziehungsweise auch aus mehreren XML-Dokumenten so herausgezogen (Engl.: to pull), wie er benötig wird.

Den Unterschied zwischen Push- und Pull-Stylesheets erklärt Williams (2002) in seinem Artikel »XMLfor Data: XSL style sheets: push or pull?« sehr anschaulich. Die Art des XSL-Stylesheets hat einen wesentlichen Einfluss auf die Komplexität des Quellcodes und auf die Wartung der Stylesheets.

In welcher Weise die XSL-Stylesheets für den LabCar-Konfigurator implementiert wurden, wird in den Kapiteln 6.2.7 »Tree-View-Stylesheet« und 6.2.8 »Component-List-Stylesheet« beschrieben.

4 Design der Benutzungsoberfläche

Dieses Kapitel befasst sich mit dem Design der Benutzungsoberfläche des LabCar-Konfigurators. Zum einen geht es um die Klärung des Begriffs Design und die Beschreibung der Elemente, die für das Design von Bedeutung sind. Zum anderen wird aufgezeigt, wie die Benutzungsoberfläche des LabCar-Konfigurators realisiert wurde und welche Faktoren und Grundlagen für das Design berücksichtigt wurden.

4.1 Der Designbegriff

»Das Design ist die Domäne, in der die Interaktion zwischen Benutzer und Produkt strukturiert wird, um effektive Handlungen zu ermöglichen.« (Bonsiepe G., 1996)

Der Design-Theoretiker Gui Bonsiepe hat den modernen Designbegriff durch sein ontologisches Designdiagramm beschrieben. Das ontologische Designdiagramm setzt sich aus drei Elementen zusammen:

- dem Benutzer,
- der zu bewältigenden Aufgabe und
- dem Werkzeug, das zum Lösen der Aufgabe benötigt wird.

Diese drei Bereiche werden durch das Interface[22] miteinander verbunden.

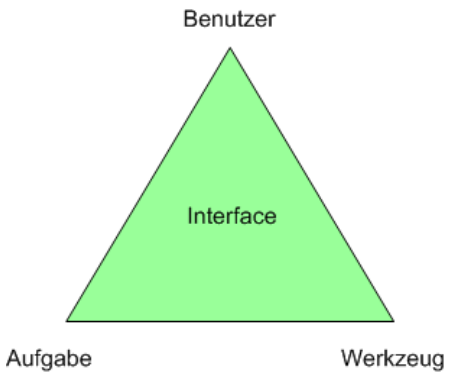

Abbildung 17: Ontologisches Designdiagramm

[22] Englisch für: Verbindung, Berührungsfläche, Schnittstelle

Bonsiepe (1996, S.20) beschreibt das Interface als zentralen Bereich, auf den der Designer seine Aufmerksamkeit richtet. Weiter merkt Bonsiepe an: »Durch das Design des Interface wird der Handlungsraum des Nutzers von Produkten gegliedert«.

Beim LabCar-Konfigurator besteht die Aufgabe des Benutzers darin, ein LabCar zu konfigurieren. Als Werkzeug verwendet der Benutzer den LabCar-Konfigurator. Die Benutzungsoberfläche des LabCar-Konfigurators ist das Interface. Über die Benutzungsoberfläche kann der Benutzer die Aufgabe der LabCar-Konfiguration durchführen. Das Interface stellt außerdem auch die Verbindung zwischen dem Benutzer und dem Werkzeug, dem LabCar-Konfigurator, her.

4.2 Analyse von Benutzungsoberflächen anderer Konfiguratoren

Um Anregungen für die Gestaltung der neu zu entwerfenden Benutzungsoberfläche zu bekommen, wurden andere, bereits vorhandene Web-Applikationen analysiert.

Dabei wurden verschiedene Konfiguratoren aus dem Automobilbereich herangezogen, die über das Internet frei zugänglich waren. Analysiert wurden unter anderem Konfiguratoren der Firmen VW, Audi, Opel und Peugeot.

Die Analyse ergab folgende Ergebnisse:

- Eine feste Seitengrundstruktur zog sich durch die gesamte Anwendung.
- Die Seitenbreite bewegte sich zwischen 760 und 830 Pixel.
- Die Anordnung der Elemente blieb im Großen und Ganzen über die gesamte Anwendung hinweg konsistent dieselbe.
- Es gibt Bereiche für Navigationselemente, es gibt Bereiche für Interaktionselemente und es gibt Bereiche für die Anzeige des aktuellen Status.
- Ein Logo oder ein Markenkennzeichen wurde die gesamte Anwendung hinweg angezeigt.

- Die ausgewählten Farben waren an die jeweiligen Unternehmensfarben angepasst.

Die Ergebnisse dieser Analyse waren im weiteren Verlauf mit ausschlaggebend für die Gestaltung der Benutzungsoberfläche des LabCar-Konfigurators.

4.3 Seitenstruktur des LabCar-Konfigurators

Für den LabCar-Konfigurator wurde folgende Seitenstruktur entworfen:

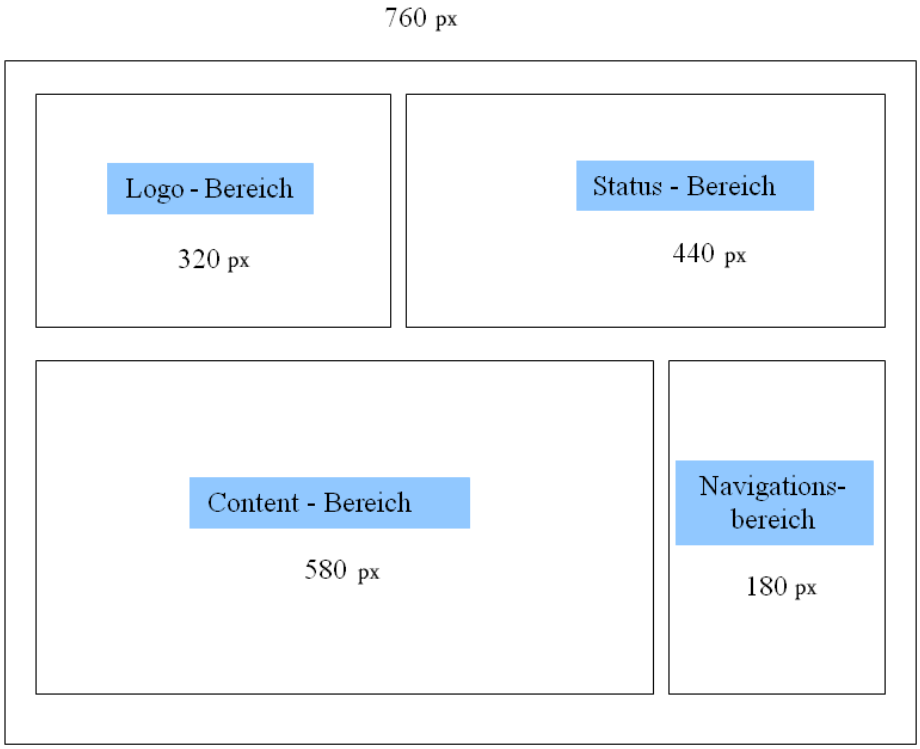

Abbildung 18: Seitenstruktur des LabCar-Konfigurators

Als Gesamtbreite wurde für den LabCar-Konfigurator 760 Pixel gewählt. Die Zahlenangaben in dieser Abbildung bedeuten Pixelangaben. Diese Seitenstruktur bleibt über die gesamte Anwendung hinweg dieselbe. Die Gesamtbreite mit 760 Pixel wurde relativ eng gewählt. Grund hierfür war, dass die Applikation später auch auf kleineren Geräten, wie beispielsweise

auf einem PDA[23], auf einem Handheld-Computer[24] oder auf einem kleineren Notebook anzeigbar und bedienbar bleiben soll.

»Eine implizite Struktur des Bildschirminhalts setzt die einzelnen Elemente in Beziehung zueinander und lässt sie so zu einer Gesamtheit werden ... Für den Ersteller multimedialer Produkte ist es empfehlenswert, sich für den Aufbau der Bildschirmseiten grundsätzliche Muster zu entwerfen.« (Thissen, 2003, S.148)

Im Logo-Bereich wird das ETAS-Logo, das LabCar-Konfigurator-Logo und ein LabCar-Bild über die gesamte Anwendung hinweg angezeigt.

Im Status-Bereich wird dem Benutzer angezeigt, welche Signalein- und Signalausgänge er bereits festgelegt hat. Hier werden die definierten Signalein- und Signalausgänge in einer Signalliste angezeigt. Die Liste kann bearbeitet werden. Bei jedem Signal befindet sich ein Auswahlkästchen, in Form eines »Radio-Button« (Englisch für: Schaltknopf bei älteren Radiogeräten), über den das jeweilige Signal ausgewählt werden kann. Unterhalb der Signalliste gibt es weitere Schaltflächen, mit denen das jeweils ausgewählte Signal geändert oder gelöscht werden kann. Außerdem kann eine neue Signallliste in den Status-Bereich geladen werden und die aktuell vorhandene Signalliste kann vom Benutzer lokal gespeichert und gesichert werden.

Im Content-Bereich findet die Bearbeitung des aktuellen Arbeitsschritts statt. Dies kann beispielsweise die Bearbeitung eines aktuellen Signals sein. Außerdem gibt es in diesem Bereich Rückmeldungen an den Benutzer, beziehungsweise Informationen zum aktuellen Arbeitsschritt.

Im Navigationsbereich sind Schaltflächen für die Navigation angeordnet. Die Schaltflächen im Navigationsbereich dienen außerdem auch der Orientierung. Befindet sich der Benutzer in einem bestimmten Bereich, ist die dazugehörige Schaltfläche farblich invers dargestellt.

Prinzip der direkten Manipulation

Die Signalliste im Status-Bereich, beziehungsweise das jeweils aktuelle Signal im Content-Bereich, sind Objekte, die direkt durch den Benutzer ver-

[23] PDA steht für »Personal Digital Assistant«.

[24] Englisch für: Computer im Taschenformat

ändert werden können. Das Prinzip, das hier zugrunde liegt, ist die so genannte direkte Manipulation.

Shneiderman (2002, S. 249) konstruiert ein integriertes Portrait der direkten Manipulation mit drei Prinzipen:

1. fortlaufende Darstellung der interessanten Objekte und Aktionen mit bedeutungsvollen visuellen Metaphern,
2. physische Aktionen oder Drücken von gekennzeichneten Schaltflächen statt komplexer Syntax und
3. schnelle, inkrementelle und umkehrbare Operationen, deren Effekte auf dem Zielobjekt sofort sichtbar sind.

Die Signalliste im Status-Bereich stellt ein Zielobjekt beim LabCar-Konfigurator dar. Die Signalliste wird über die ganze Eingabe hinweg als Liste dargestellt und lässt sich direkt durch den Benutzer manipulieren und verändern. Das jeweils aktuelle Signal im Content-Bereich stellt ein weiteres Zielobjekt dar. Die Definition eines Signals kann inkrementell vorgenommen werden. Eingaben können jederzeit rückgängig gemacht werden. Änderungen sind sofort sichtbar. Sämtliche Aktionen können über Schaltflächen vorgenommen werden.

Die Vorteile der direkten Manipulation fasst Shneiderman (2002) folgendermaßen zusammen: »Das Interessensobjekt wird angezeigt, so dass die Interface-Aktionen dicht beim übergeordneten Aufgabenbereich liegen. Es gibt wenig Bedarf an mentaler Zergliederung der Aufgaben. Jede Aktion produziert im Aufgabenbereich ein verständliches Ergebnis, das im Interface sofort sichtbar wird. Die Nähe vom Aufgabenbereich zum Interfacebereich reduziert für den Benutzer Stress und Belastung durch Problemlösung.«

Weiter bemerkt Shneiderman (2002, S. 250): »Direkt manipulative Systeme können vom Anwender erfordern, wesentliche Kenntnisse über die Aufgabe zu erlernen. Jedoch müssen Anwender nur eine bescheidene Menge von Kenntnissen über das Interface und syntaktische Details erwerben.«

Gestaltgesetze

Beim Entwurf der Seitenstruktur und der Komposition der Seitenelemente wurden auch die so genannten »Gestaltgesetze« berücksichtigt.

Zu Beginn des 20. Jahrhunderts haben einige Psychologen (Koffka[25], Köhler[26] Wertheimer[27], Arnheim[28] u.a.) Gesetzmäßigkeiten der menschlichen Wahrnehmung in den »Gestaltgesetzen« beschrieben. Die Beachtung der Gestaltgesetze hilft, die Elemente des Bildschirms so zu komponieren, dass sie der menschlichen Wahrnehmung entgegenkommen.

Zu den Gestaltgesetzen gehört unter anderem das Gesetz der Nähe. Dieses Gesetz besagt, dass Elemente, die räumlich nah beieinander liegen, als zusammengehörend wahrgenommen werden. Zusammengehörende Elemente sollten daher räumlich nahe beeinander gruppiert werden. Die Dateneingabefelder im Content-Bereich und die Navigationschaltflächen im Navigationsbereich gehören jeweils zusammen und sind beim LabCar-Konfigurator deshalb nahe beianderliegend gruppiert und zusammengefasst.

Ein weiteres Gestaltgesetz ist das Gesetz der Ähnlichkeit. Ähnlich aussehende Elemente werden von der menschlichen Kognition als zusammengehörend wahrgenommen. Zusammengehörende Elemente sollten aus diesem Grund optisch ähnlich markiert werden. Die Dateneingabefelder und die Navigationsschaltflächen des LabCar-Konfigurators haben jeweils dieselbe Farbe und Größe und sind gleichartig beschriftet.

Das Gesetz der guten Fortsetzung gehört ebenfalls zu den Gestaltgesetzen. Optische Elemente, die in einer gewissen Kontinuität angeordnet sind (z.B. entlang einer Linie), werden als zusammengehörend wahrgenommen. Es ist deshalb von Vorteil, zusammengehörende Elemente entlang einer Linie anzuordnen. Beim LabCar-Konfigurator sind beispielsweise zusammengehörende Dateneingabefelder und Navigationsschaltflächen jeweils an einer waagrechten oder senkrechten Linie angeordnet.

---

[25] Koffka (1967)

[26] Köhler (1992)

[27] Wertheimer (1925)

[28] Arnheim (1983) und (1985)

4.4 Goldener Schnitt

Die Seiteneinteilung des LabCar-Konfigurators wurde in Anlehnung an das Prinzip des Goldenen Schnitts vorgenommen.

Der »Goldene Schnitt« (Lateinisch: sectio aurea) ist ein bestimmtes Verhätlnis zweier Zahlen, meist der Längen von Strecken, das in der Kunst und Architekur oft als ideale Proportion und als Inbegriff von Ästhetik und Harmonie angesehen wird.

In Architektur und Kunst wurde in der Vergangenheit vielfach darauf geachtet, bei Einteilungen die Teilverhältnisse des Goldenen Schnittes zu wahren, da die Ergebnisse dann als besonders harmonisch empfunden werden.

In der Natur findet sich der Goldene Schnitt in guter Näherung beispielsweise im Verhältnis der Längen der Finger- und Armknochen des Menschen zueinander, zum Beispiel das Verhältnis des Unterarmes zum Oberarm.

In der Mathematik stellt sich der Goldene Schnitt als ein irrationales Zahlenverhältnis dar und zeichnet sich durch eine Fülle interessanter mathematischer Eigenschaften aus. Seit der Antike gilt der Goldene Schnitt als ideale Proportion für Geometrie, Architektur und Kunst, beispielsweise für die ästhetische Unterteilung eines Körpers oder einer Buchseite. Der kleinere Teil verhält sich zum größeren wie der größere Teil zum Ganzen.
(vgl. Net-Lexikon 2004, Stichwort: »Goldener Schnitt«)

Beim LabCar-Konfigurator wurden für die einzelnen Bereiche unterschiedliche Breiten gewählt. Der schmalere Logo-Bereich steht neben dem breiteren Status-Bereich. Darunter steht der Content-Bereich und der im Verhältnis zum Content-Bereich schmalere Navigationsbereich. Hiermit soll ein harmonisches Gesamtbild entstehen. Durch den Wechsel der Verhältnisse der einzelnen Bereiche soll ein harmonisches Gleichgewicht erzielt werden. Ähnliche Strukturen finden sich auch in den analysierten Konfiguratoren anderer Unternehmen.

4.5 Seitengestaltung des LabCar-Konfigurators

Beim Starten des LabCar-Konfigurators erscheint die folgende Startseite im Browser-Fenster. Die Abbildung ist als Screenshot [29] dargestellt.

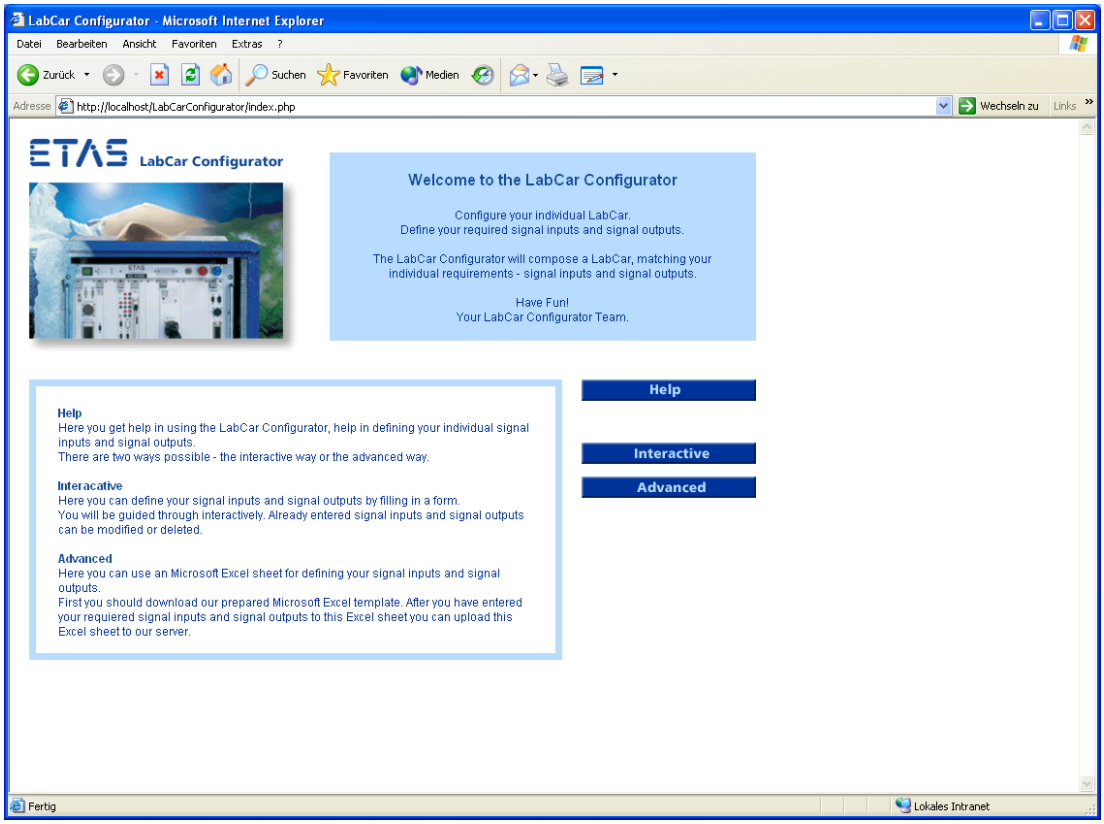

Abbildung 19: Screenshot: LabCar-Konfigurator - Startseite

Entsprechend der oben in Kapitel 4.3 beschriebenen Seitenstruktur des LabCar-Konfigurators finden sich in den einzelnen Bereichen verschiedene Screen-Design-Elemente. Zu Screen-Design-Elementen zählen beispielsweise Orientierungselemente, Navigationselemente, Emotionselemente und Inhaltselemente. (vgl. Thissen, 2003, S. 20)

Im Logo-Bereich ist das ETAS-Firmenlogo und ein LabCar-Bild aus dem Printbereich integriert. Der Logo-Bereich bleibt über die gesamte Anwendung hinweg gleich. Dadurch soll dem Benutzer Kontinuität vermittelt werden, zum einen in Bezug auf die Wiedererkennung der Coorporate Identity

[29] Das englische Wort »Screenshot« steht für Bildschirmabzug.

und zum anderen als Orientierung für den Benutzer. Der Logo-Bereich hat die Funktion der Wiedererkennung, ähnlich wie im Bereich des Marketing die Wiedererkennung von Marken.

Im Status-Bereich befindet sich eine Begrüßung, im Content-Bereich eine kurze Einführung zum Einstieg und die möglichen Wege »Interactive« und »Advanced«. Bei der Eingabemöglichkeit »Interactive« wird der weniger erfahrene Benutzer interaktiv durch die Anwendung geführt. Die Eingabemöglichkeit »Advanced« richtet sich an den erfahrenen fortgeschrittenen Benutzer. Der fortgeschrittene Benutzer hat hier die Möglichkeit, die von ihm gewünschten Anforderungen clientseitig, über ein Tabellenkalkulationsprogramm, in einer Tabelle zu definieren und direkt an das System zu übergeben. Die beiden Wege werden weiter unten in den Kapiteln 4.10 »Eingabe - Interactive« und 4.12 »Eingabe - Advanced« ausführlich beschrieben.

Im Navigationsbereich findet der Benutzer die entsprechenden Schaltflächen (Engl.: button) für die Navigation.

4.6 Für den LabCar-Konfigurator verwendete Farben

Thissen (2003, S.160) schreibt: »Der Einsatz von Farben bietet die Möglichkeit, Informationen eine weitere Dimension hinzuzufügen.« Weiter schreibt Thissen: »Farben suggerieren unterschwellig eine eigene Botschaft, lösen Gefühle aus und können Aussagen unterstützen ... Sie können als Orientierungshilfe dienen, Informationen strukturieren und Unterschiede verdeutlichen. Sie können aber auch den Zugriff auf Informationen erleichtern. Farben werden häufig unbewusst wahrgenommen und lösen immer Emotionen aus.«

Bei der Gestaltung der Benutzungsoberfläche des LabCar-Konfigurators wurden die Farben Blau und Weiß und in kleinerem Umfang die Farbe Grau verwendet. Die Farbauswahl beim LabCar-Konfigurator ist an die Firmenfarbe Blau der ETAS GmbH angepasst.

Blau

Thissen (2003, S.185) schreibt über die Farbe Blau: »Es wird behauptet, dass die meisten Menschen Blau als Lieblingsfarbe haben. Auf jeden Fall

ist die Farbe äußerst beliebt. Sie hat etwas Nobles, Stabiles, Freundschaftliches, Sympathisches ... Blau steht ... für Nüchternheit, für Logik und scharfes Denken, sowie für Technologie.«

Weiß

Der Farbe Weiß werden die Eigenschaften von Sachlichkeit und Klarheit zugeschrieben.

Grau

Die Farbe Grau symbolisiert die Eigenschaften von Eleganz und Sachlichkeit. Grau symbolisiert auch die Technologie. Über die Farbe Grau schreibt Thissen (2003, S.205): »Grau ist die Farbe zwischen Schwarz und Weiß - das Mittelmaß, ohne Charakter. Grau ist die Farbe der Neutralität und des Unscheinbaren (eine 'graue Maus' ist eine wenig auffallende Frau).«

Der LabCar-Konfigurator ist eine komplexe und technikorientierte Anwendung. Die Farben Weiß und Grau sollen Sachlichkeit und Klarheit vermitteln. Durch ein sachliches, logisches und klares Design des LabCar-Konfigurators soll dem Benutzer geholfen werden, die komplexen Inhalte und Signaldefinitionen in den Griff zu bekommen. Durch Hilfestellungen von Seiten des Systems und ein entsprechend passendes Farbdesign soll dem Benutzer das Gefühl von Sicherheit vermittelt werden. Die Farbe Blau steht für Technologie und Logik und soll einladend auf den Benutzer wirken.

Jede Farbe für sich verfügt über ein breites Spektrum an Farbintensität. Für den LabCar-Konfigurator wurden insbesondere helle und zarte Farben eingesetzt.

Zarte Farben vermitteln den Eindruck von Empfindlichkeit. Sie eignen sich gut als Hintergrundfarben oder können zarte, empfindliche Sachverhalte unterstreichen. (vgl. Krüger 2004, Stichwort: »Zarte Farben«)

Helle Farben wirken prinzipiell leicht und freundlich. Sie vermitteln einen Eindruck von Weite, von viel Raum. Sie wirken stimmungsaufhellend und belebend. Andererseits treten helle Farben in den Hintergrund. Deshalb eignen sie sich auch besonders gut als Hintergrundfarben für Texte und Bilder. (vgl. Krüger 2004, Stichwort: »Helle Farben«)

4.7 Blicksteuerung durch optische Signale

»Gute Bildschirmseiten sind nicht eintönig, sondern weisen eine gewisse Spannung und Dynamik auf. Ruhige Bereiche sollten den Bereichen gegenüberstehen, in denen etwas geschieht, die sich in den Vordergrund drängen. Erst aus dieser Kombination heraus lassen sich spannende Seiten gestalten. Wichtiges sollte dabei auch optisch von Unwichtigem unterschieden werden.«
(Thissen 2003, S. 158)

Realisieren lässt sich dies dadurch, dass die Seitenaufteilung zwischen ruhigen Stellen und belebten Stellen unterscheidet. Eine weitere Möglichkeit bieten Kontraste innerhalb einer Seite.

Durch die Aufteilung der Gesamtseite des LabCar-Konfigurators in die Bereiche Logo-Bereich, Status-Bereich, Content-Bereich und Navigationsbereich wird die Bildung von Kontrasten ermöglicht. Der Logo-Bereich steht als ruhiger Bereich im Kontrast zum Status- und Content-Bereich. Der Navigationsbereich gehört ebenfalls eher zum ruhigeren Bereich.

Hat der Benutzer alle gewünschten Signale definiert und eingegeben, kann er die Konfiguration starten. Die Schaltfläche »Start Configuration« ist als blinkende Schaltfläche dargestellt. Dadurch wird ebenfalls Spannung und Dynamik erzeugt. Die besondere Aufmerksamkeit des Benutzers wird damit auf diese Schaltfläche und die damit verbundene Funktion gelenkt.

4.8 ISO-Norm 9241-10: »Gestaltung von Dialogen«

Die Benutzungsoberfläche des LabCar-Konfigurators stellt einen Dialog zwischen dem Benutzer und der Anwendung dar. Für die Gestaltung von Dialogen gibt es eine eigene ISO-Norm[30], die ISO 9241-10. Die ISO 9241-10 wurde beim Entwurf der Benutzungsoberfläche des Lab-Car-Konfigurators und insbesondere beim Aufbau der Navigationsstruktur berücksichtigt.

Nach ISO 9241-10 sollte bei der Gestaltung von Dialogen auf folgende Merkmale geachtet werden:

- **Aufgabenangemessenheit**
  Aufgabenangemessenheit bedeutet, dass der Dialog der Aufgabe, die der Benutzer zu erledigen hat, angemessen sein soll. Der Benutzer soll nicht unnötig durch den Dialog belastet werden, sondern bei der Erledigung seiner Aufgabe optimal unterstützt werden.

- **Selbstbeschreibungsfähigkeit**
  Der Dialog soll unmittelbar klar und verständlich sein. Er soll an das Vorwissen des Benutzers und an dessen Bedürfnisse angepasst sein.

- **Steuerbarkeit**
  Der Benutzer soll die Geschwindigkeit und den Ablauf des Dialoges selbst steuern können. Hierzu gehört auch, dass Eingaben rückgängig gemacht werden können.

- **Erwartungskonformität**
  Der Dialog soll den Erwartungen des Benuzters entsprechen. Hierzu gehört, dass die Dialoge konsistent sind. So kann der Benutzer sich im Umgang mit dem System, aufgrund seiner Erfahrungen mit dem System, auf das System einstellen.

- **Fehlertoleranz**
  Bei fehlerhaften Eingaben des Benutzers soll der Dialog auf den Fehler hinweisen oder dem Benutzer Korrekturmöglichkeiten anbieten.

---

[30] ISO steht als Abkürzung für International Organization for Standardization, eine Institution, welche die Normung international koordiniert. Der Zweck der ISO ist die Förderung der Normung in der Welt, um den Austausch von Gütern und Dienstleistungen zu unterstützen und die gegenseitige Zusammenarbeit in verschiedenen technischen Bereichen zu entwickeln. Die ISO-Normen dienen außerdem der Schaffung von Qualitätsstandards in den unterschiedlichsten Bereichen.
(vgl. QM-Lexikon 2004, Stichwort: »ISO«)

4.9 Navigationsstruktur

Den Einstieg in den LabCar-Konfigurator bildet die Startseite, die in Kapitel 4.5 »Seitengestaltung des LabCar-Konfigurators« abgebildet ist.

Danach hat der Benutzer die Wahl. Er kann zwischen der Eingabemöglichkeit »Interactive« und der Eingabemöglichkeit »Advanced« wählen. Die beiden Wege für die Eingabe und die Ausgabe werden in den nachfolgenden Kapiteln anhand von Screenshots[31] dargestellt und erläutert.

Hat der Benutzer alle gewünschten Signalein- und Signalausgänge definiert kann er die Konfiguration starten und sich das Konfigurationsergebnis ausgeben lassen. Das Konfigurationsergebnis kann er sich auf zwei verschiedene Arten darstellen lassen.

Zusätzlich kann der Benutzer auch über das Browser-Programm auf zuvor aufgerufene Seiten zurückspringen, ohne dass seine Eingaben verloren gehen.

In den nachfolgenden Kapiteln werden die verschiedenen Eingabe- und Ausgabemöglichkeiten des LabCar-Konfigurators dargestellt und erläutert.

4.10 Eingabe »Interactive«

Im Bereich »Interactive« hat der Benutzer die Möglichkeit, die von ihm gewünschten Signalein- und Signalausgänge zu definieren. Dabei wird er vom System geführt und bekommt Hilfestellung. Fehlerhafte Angaben des Benutzers werden korrigiert und die Anwendung informiert den Benutzer über die Korrekturen, die von Seiten des Systems vorgenommen werden.

[31] Das englische Wort »Screenshot« steht für Bildschirmabzug.

Die interaktiv geführte Eingabe »Interactive« startet mit der folgenden Bildschirmseite:

Abbildung 20: Screenshot: Interactive - Signal (1. Ebene)

Wie im Entity-Relationship-Diagramm »Signal« für die Definition von Signalen (Abbildung 9) dargestellt, besteht ein Signal aus drei verschiedenen Bereichen: den allgemeinen Signaldefinitionen, den Ein- / Ausgabespezifikationen und der Last- und Fehlersimulation. Diese drei Bereiche werden in der Benutzungsoberfläche durch die Bereiche »Signal«, »I/O Specifications« und »Load & Error Simulation« in Form eines dreiteiligen Formulars abgebildet. Über die Navigationsschaltflächen »Signal«, »I/O Specifications« und »Load & Error Simulation« kann der Benutzer beliebig zwischen den drei Teilformularen wechseln und nach Belieben navigieren.

Die Dreiteilung des Formulars war notwendig, weil ein einziges Formular, welches sich über mehrer Bildschirmseiten erstreckt hätte, nicht benutzerfreundlich gewesen wäre. Der Benutzer hätte dann häufig scrollen müssen und die Übersicht wäre verloren gegangen. Durch die Dreiteilung passt nun

jedes einzelne Formular komplett auf eine Bildschirmseite und die aktuellen Informationen bleiben dem Benutzer so vollständig vor Augen.

Im ersten Teilformular, das dem Bereich »Signal« zugeordnet ist, kann der Benutzer einen Signalnamen (Engl.: Signal Name) definieren. Er kann außerdem eine Pin-Nummer (Engl.: Pin Number) festlegen, eine Signalbeschreibung (Engl.: Signal Description) vornehmen und den Signaltyp (Engl.: Signal Type) festlegen. Für den Signaltyp kann der Benutzer einen Wert aus einer Liste auswählen. Mögliche Werte sind »IO«, »RS422«, »ETK«, »CAN« oder »FT-CAN«. Wurde für den Signaltyp »IO« ausgewählt, muss die Art der Ein-/Ausgabe näher spezifiziert werden. Die Spezifikation der Ein-/Ausgabe kann im zweiten Teilformular vorgenommen werden.

Abbildung 21: Screenshot: Interactive - I/O Specifications (1. Ebene)

Der zweite Formularteil beinhaltet die »I/O Specifications«. Es gibt Datenfelder, in die der Benutzer die gewünschten Werte frei eintragen kann und es gibt Datenfelder mit vorgegebenen Werten. Hier kann der Benutzer aus einer Liste den jeweils gewünschten Wert auswählen.

Für die Auflösung (Engl.: Resolution) kann ein Wert ausgewählt werden. Wie in Kapitel 2.10 »Definition von Signalen« beschrieben, stehen folgende Werte zur Auswahl: »analog«, »digital« oder »analog-hi-res«. Bei Signalrichtung (Engl.: Direction) kann der Benutzer zwischen »bi-directional«, »input« oder »output« wählen. Für die Spannung - »Voltage« gibt es ebenfalls eine vorgegebene Liste mit Wertebereichen, die zur Auswahl stehen. Für die Abtastrate (Engl.: Sampling Rate), die Dauerstromstärke (Engl.: Permanent Current) und den Stromstärkenspitzenwert(Engl.: Peak Current) können die Werte frei eingegeben werden.

Hat der Benutzer im ersten Teilbereich »Signal« ein Signal definiert, werden die Daten zur Identifikation des aktuellen Signals, der Signalname, die Pin-Nummer und die Signalbeschreibung, im zweiten und dritten Teilformular im Content-Bereich in der obersten Zeile dargestellt. So weiß der Benutzer immer, welches Signal er aktuell bearbeitet.

Im dritten Teilformular kann der Benutzer die gewünschte Last- und Fehlersimulation (Engl.: Load & Error Simulation) für das aktuelle Signal festlegen.

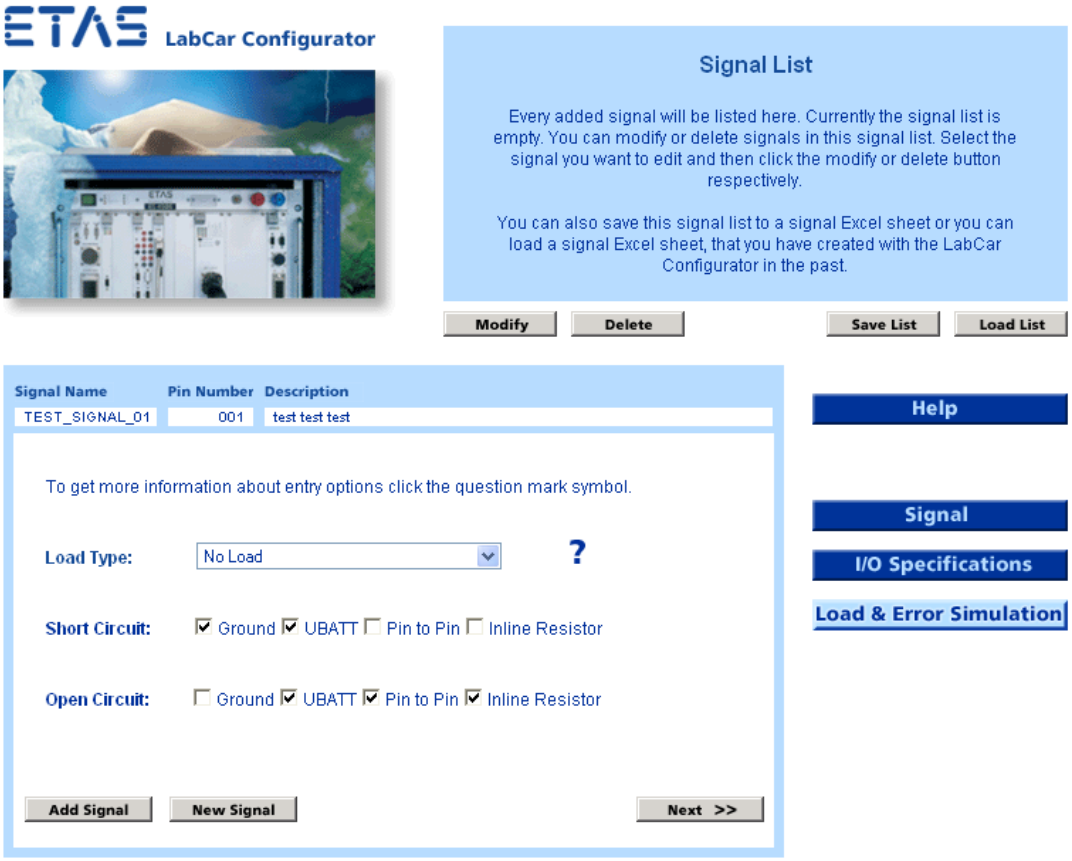

Abbildung 22: Screenshot: Interactive - Load & Error Simulation (1. Ebene)

Für die Lastnachbildung (Engl.: Load Type) stehen die Werte »No Load«, »Original Load«, »Common Rail - Diesel«, »Gasoline (Direct Injection)« oder »Piezo Injection« zur Auswahl. Die Merkmale Kurzschluss (Engl.: Short Circuit) und Unterbrechung (Engl.: Open Circuit) können vom Benutzer über Ankreuzfelder näher spezifiziert werden.

Im Status-Bereich wird die aktuelle Signalliste angezeigt. Solange noch kein Signal vorhanden ist, existiert auch noch keine Signalliste. In diesem Fall bleibt der Status-Bereich leer, beziehungsweise es erscheint ein allgemeiner Informationstext im Status-Bereich.

Die Navigationsschaltflächen dienen außerdem auch der Orientierung. Um anzuzeigen in welchem Formularteil sich der Benutzer gerade befindet, wird die entsprechende Schaltfläche invers dargestellt. Der Schaltflächenhintergrund ist hellblau eingefärbt, im Gegensatz zu den übrigen dunkelblau hinterlegten Schaltflächen im Navigationsbereich. Die Schrift wechselt dabei von hellblau auf dunkelblau.

Die Help-Schaltfläche im Navigationsbereich bietet dem Benutzer eine allgemeine Hilfestellung zur Bedienung des LabCar-Konfigurators.

Der noch unerfahrene Benutzer kann über die Next-Schaltfläche im Content-Bereich navigieren und wird automatisch richtig geführt. Hat der Benutzer einen anderen Signaltyp als »IO« ausgewählt, werden die I/O-Spezifikationen automatisch übersprungen und es geht sofort mit dem dritten Teil des Formulars, der Last- und Fehlernachbildung, weiter.

Über die Add-Signal-Schaltfläche kann der Benutzer das Signal, welches er aktuell im Content-Bereich bearbeitet, der Signalliste hinzufügen. Ist das erste Signal hinzugefügt, gibt es eine Signalliste, die dann im Status-Bereich angezeigt wird. Je mehr Signale der Benutzer definiert hat, um so umfangreicher wird auch die Signalliste. Mit der Add-Signal-Schaltfläche kann der Benutzer auch Signale zur Signalliste hinzufügen, ohne dass er alle notwendigen Werte definert hat. Fehlende Werte werden automatisch vom System mit vorgegebenen Werten (Defaultwerten) ersetzt. Das System informiert den Benutzer darüber, dass es automatisch Defaultwerte eingesetzt hat.

Zusätzlich zu den blau eingefärbten Schaltflächen im Navigationsbereich kommen weitere Navigationselemente oder so genannte Funktionsschaltflächen im Status-Bereich und im Content-Bereich hinzu. Diese sind grau eingefärbt. Wird der Mauszeiger darüberbewegt, verändern diese Schaltflächen ihr Aussehen. Der darauf befindliche Text wird leicht aufgehellt.

Mit der New-Signal-Schaltfläche kann der Benutzer ein neues Signal definieren. Die aktuellen Daten werden dabei gelöscht. Vor dem Löschen der Daten muss der Benutzer noch einmal bestätigen, dass er mit der Löschung einverstanden ist. Möchte der Benutzer eine Löschung rückgängig machen, kann er dies auch in begrenztem Umfang durch Zurücknavigieren im Browser-Programm erreichen.

Im weiteren Verlauf kann der Benutzer beliebig über die Schaltflächen im Navigationsbereich, »Signal«, »I/O Specifications« oder »Load & Error Simulation« navigieren. So kann er die Reihenfolge seiner Eingabe selbst bestimmen und hat die Möglichkeit Änderungen oder Korrekturen am aktuellen Signal beliebig vorzunehmen.

Klickt der Benutzer auf das Fragezeichen neben den Eingabemöglichkeiten, bekommt er Hilfestellung zum betreffenden Dateneingabefeld. Beispielhaft ist dies in der folgenden Abbildung für das Merkmal »Signal Name« dargestellt.

Abbildung 23: Screenshot: Interactive - Signal Help (1. Ebene)

Folgende Bildschirmseite erscheint, wenn der Benutzer lediglich Daten im Bereich »Signal« definiert und für den Signaltyp den Wert »IO« ausgewählt hat. Das System informiert den Benutzer, dass für die noch fehlenden Werte vom System automatisch vorgegebene Werte eingesetzt werden.

Abbildung 24: Screenshot: Interactive - Warnmeldungen (2. Ebene)

Das System verfügt über zahlreiche Hinweise für den Benutzer. Die Meldungen an den Benutzer beginnen mit der Überschrift »Please Notice«. Über diese Rückmeldungen an den Benutzer wird die Dateneingabe gesteuert und gleichzeitig wird der Benutzer über Aktivitäten des Systems informiert. Beispielsweise ersetzt das System in bestimmten Fällen fehlende Angaben mit vorgegebenen Werten (Defaultwerten). Der Benutzer hat jedoch jederzeit die Möglichkeit einzugreifen und Änderungen vorzunehmen, sofern er dies wünscht.

Klickt der Benutzer auf die Add-Signal-Schaltfläche, wird das Signal der Signalliste - »Signal List« - im Status-Bereich rechts oben hinzugefügt. Sobald eine Signalliste existiert, wird diese rechts oben im Status-Bereich angezeigt.

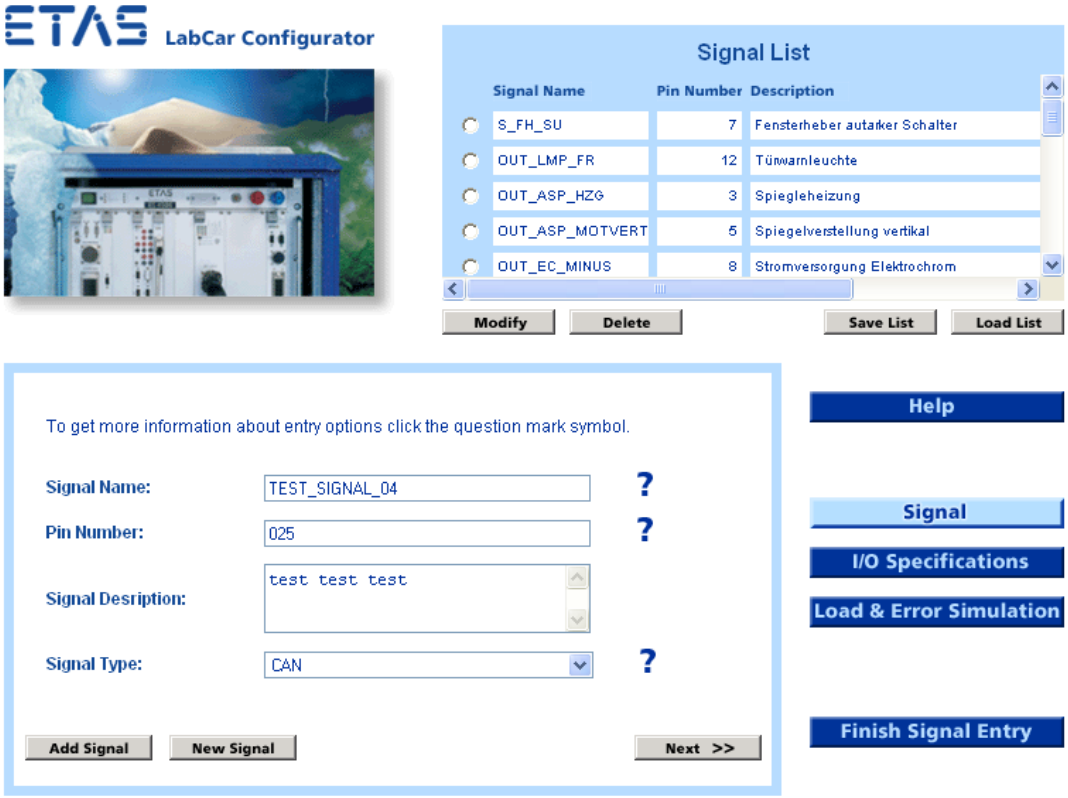

Abbildung 25: Screenshot: Interactive - Signal (2. Ebene)

In der 2. Ebene existiert jetzt eine Signalliste im Status-Bereich rechts oben. Auch hier findet der Benutzer wieder die bereits aus der 1. Ebene bekannten drei Teilformulare, die er über die Schaltflächen »Signal«, »I/O Specifications« oder »Load & Error Simulation« anspringen kann.

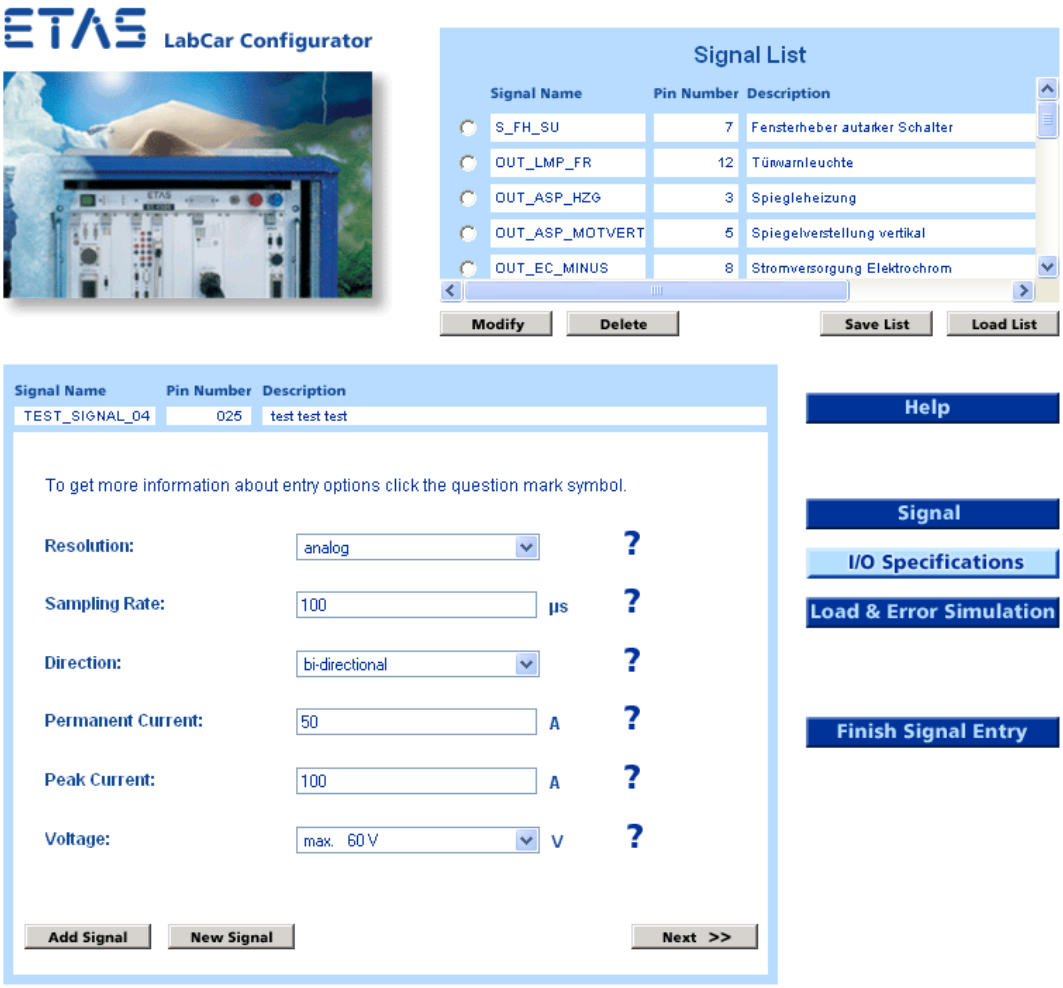

Abbildung 26: Screenshot: Interactive - I/O Specifications (2. Ebene)

Wurde für das aktuelle Signal der Signaltyp »IO« ausgewählt, können im Bereich »I/O Speifications« wieder die Ein-/Ausgabespezifikationen näher bestimmt werden.

Die Angaben zur Last- und Fehlersimulation können wiederum im dritten Teilformular »Load & Error Simulation« eingetragen werden.

Abbildung 27: Screenshot: Interactive - Load & Error Simulation (2. Ebene)

In der Signalliste - »Signal List« rechts oben sind alle Signale eingetragen, die der Benutzer bis jetzt definiert hat. Über die Benutzung der Scroll-Balken kann sich der Benutzer weitere Bereiche der Signalliste anzeigen lassen. Unterhalb der Signalliste - »Signal List« befinden sich die Schaltflächen »Modify«, »Delete«, »Save List« und »Load List«. Über diese Schaltflächen hat der Benutzer direkten Zugriff auf die Signalliste. Er kann einzelne Signale löschen oder ändern. Die Auswahl eines Signals erfolgt über die Auswahlfelder (Radio Button), die sich in der Signalliste vor jedem einzelnen Signal befinden.

Wählt der Benutzer aus der Signalliste im Status-Bereich ein Signal aus und klickt auf die Modify-Schaltfläche, werden alle Werte des ausgewählten Signals in den Content-Bereich geladen. Damit hat der Benutzer dann die Möglichkeit, das ausgewählte Signal beliebig zu bearbeiten. Die Navigations- und Eingabemöglichkeiten bleiben dieselben. Damit wird an das, was der Benutzer bereits kennt, angeknüpft. Hat der Benutzer alle gewünsch-

ten Änderungen vorgenommen, muss er das Signal über die Add-Signal-Schaltfläche erneut zur Signallliste hinzufügen.

Weiter hat der Benutzer die Möglichkeit ein Signal aus der Signalliste auszuwählen und zu löschen. Nach Anklicken der Delete-Schaltfläche verlangt das System nochmals eine Löschbestätigung, damit der Benutzer nicht ungewollt oder aus Versehen eine Löschaktion in Gang setzt. Damit hat der Benutzer auch die Möglichkeit die gestartete Löschaktion rückgängig zu machen (siehe hierzu auch: Kapitel 4.8 »ISO-Norm 9241-10: Gestaltung von Dialogen«).

Über die Schaltfläche »Save List« kann der Benutzer die aktuelle Signalliste lokal auf seinem Computer speichern. Damit hat der Benutzer die Möglichkeit, zu einem beliebigen späteren Zeitpunkt an der Signalliste weiterzuarbeiten.

Über die Schaltfläche »Load List« kann der Benutzer eine Signalliste, die er zu einem früheren Zeitpunkt erstellt und gespeichert hat, in das System laden. Es ist dabei gleichgültig, ob die Liste über den Weg »Interactive« oder »Advanced« erstellt wurde.

Das Format der hochgeladenen Signalliste muss dem CSV-Format[32] entsprechen. Entspricht die hochgeladene Signalliste nicht dem erforderlichen Format, erscheint eine Fehlermeldung.

Hat der Benutzer alle Signale definiert, die er benötigt, kann er über die Schaltfläche »Finish Signal Entry« die Signaleingabe beenden. Über die Schaltfläche »Start Configuration« kann der Benutzer dann die Konfiguration starten.

Die Schaltfläche »Start Configuration« ist als animierte Schaltfläche gestaltet. Er blinkt invers, das bedeutet, die Farben wechseln zwischen Dunkelblau und Hellblau. Die Blinkrate beträgt etwa eine halbe Sekunde. Damit wird die Aufmerksamkeit des Benutzers auf diese Schaltfläche fokussiert. Damit verbunden ist die Botschaft: Nun ist es geschafft. Die Informationen, die der Benutzer an das System übermittelt hat, sind so weit vollständig. Das System hat jetzt alle Informationen, die es benötigt, um eine Konfiguration durchführen zu können. Während die Konfiguration auf dem Server läuft blinkt die Schaltfläche weiter, um dem Benutzer anzuzeigen, dass er

[32] CSV steht für »Comma Separated Value«.
Das CSV-Format wird in Kapitel »6.3.1 Excel-Tabelle im CSV-Format« beschrieben.

weiterhin mit dem System verbunden ist und die Konfiguration durchgeführt wird.

Hat das System die Konfiguration abgeschlossen, kann der Benutzer auswählen, in welcher Weise das Konfigurationsergebnis ausgegeben werden soll. Zur Auswahl stehen die Darstellungen »Tree View« (Englisch für: Baumansicht) und »Component List« (Englisch für: Bestellliste).

4.11 Ausgabe

Wählt der Benuzter die Darstellung »Tree View« erscheint eine Bildschirmseite, die etwa folgendes Aussehen hat. Die Darstellung ändert sich jeweils in Abhängigkeit vom Konfigurationsergebnis.

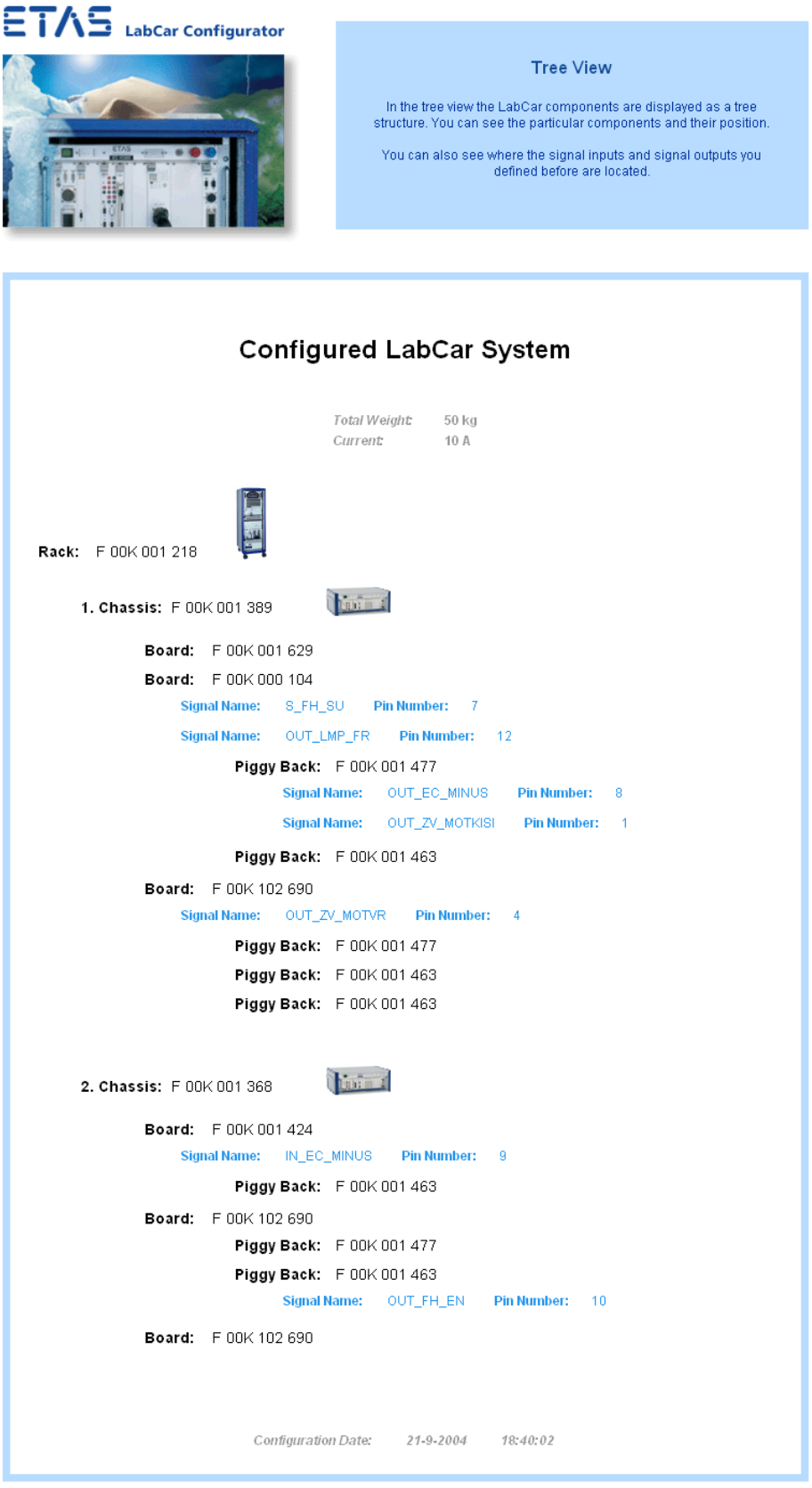

Abbildung 28: Screenshot: Ausgabe - Tree View

Der Logo- und Status-Bereich sind weiterhin vohanden. Damit wird eine optische Verbindung zum bisherigen Seitenlayout hergestellt. Der Content- und Navigationsbereich werden nun zu einem Bereich zusammengefasst. Navigation ist nicht mehr nötig, da das Ziel der Anwendung, die Konfiguration eines LabCars, erreicht ist. Durch die Zusammenfassung des Content- und Navigationsbereichs entsteht mehr Platz für die Darstellung des Konfigurationsergebnisses.

Die Darstellung »Tree View« entspricht dabei dem physikalischen Aufbau des LabCars. Außerdem sind die vom Benutzer definierten Signale dargestellt. Die Signale sind in Hellblau dargestellt. Sie befinden sich jeweils unterhalb der LabCar-Hardwarekomponente, an der sich der entsprechende Signalausgang befindet. Zu Beginn werden außerdem das Gesamtgewicht und die Leistungsaufnahme als Stromstärke (Engl.: Current) ausgegeben. Das Datum und die Uhrzeit der Konfiguration stehen am Schluss der Ausgabe.

Bei der Darstellung »Tree View« werden so genannte Thumbnails verwendet.

»Als Thumbnail (Englisch für: Daumennagel) werden kleine digitale Grafiken beziehungsweise Bilder bezeichnet, die als Vorschau für eine größere Version dienen. Der Einsatz von Thumbnails kann unterschiedliche Gründe haben. Zum einen haben kleine Bilder natürlich eine kürzere Ladezeit als große, was besonders im Internet zum Tragen kommt. ... Der Benutzer kann anhand der Thumbnails entscheiden, welches Bild er in voller Größe betrachten möchte. ... Ein weiterer Grund Thumbnails einzusetzen ist die Platzersparnis. Viele Bilder können so auf engstem Raum untergebracht werden. Der Besucher kann sich schnell einen Überblick verschaffen ... Somit kann der beschränkte Platz einer Seite effektiv genutzt, und die Attraktivität durch Interaktivität weiter gesteigert, werden.«
(WIKIPEDIA 2004, Stichwort: »Thumbnail«)

Wählt der Benuzter die Darstellung »Component List« erscheint eine Bildschirmseite, die etwa folgendes Aussehen hat.

ETAS LabCar Configurator

Component List

The component list displays all components in the form of an order list.
Order numbers, component descriptions, the price and some technical data are listed here.

LabCar - Component List

Configuration Date: 21-9-2004 18:40:02

Rack

F 00K 001 218	ES4500 Component Rack	---- text for sales and distribution ----	950.00.--	EUR

Chassis

F 00K 001 368	ES4100.1 Chassis VME64x	---- text for sales and distribution ----	480.00.--	EUR
F 00K 001 389	ES4105.1 Chassis VME64x	---- text for sales and distribution ----	500.00.--	EUR

Boards

F 00K 000 104	ES1650.1 Piggyback Carrier Board	---- text for sales and distribution ----	360.00.--	EUR
F 00K 001 424	ES4571 Prototyping Board	---- text for sales and distribution ----	230.00.--	EUR
F 00K 001 629	ES1130.2 Simulation Controller Board	---- text for sales and distribution ----	240.00.--	EUR
F 00K 102 690	ES4540 High Current Relay Board	---- text for sales and distribution ----	280.00.--	EUR
F 00K 102 690	ES4540 High Current Relay Board	---- text for sales and distribution ----	280.00.--	EUR
F 00K 102 690	ES4540 High Current Relay Board	---- text for sales and distribution ----	280.00.--	EUR

Piggy Backs

*5 **

F 00K 001 463	PB1650DIO1.1 Digital I/O Piggyback (8/8-CH)	---- text for sales and distribution ----	180.00.--	EUR
			900.00.--	*EUR*

*3 **

F 00K 001 477	PB1650ADC1.1 A/D Piggyback (8-CH)	---- text for sales and distribution ----	210.00.--	EUR
			630.00.--	*EUR*

	5130.00 .--	*EUR*
sales tax	*820.80 .--*	*EUR*
total	*5950.80 .--*	*EUR*

Abbildung 29: Screenshot: Ausgabe - Component List

Auch die Darstellung »Component List« ist abhängig vom Konfigurationsergebnis und ändert sich jeweils entsprechend. Die Darstellung entspricht einer Bestellliste. Zu Beginn wird das Datum und die Uhrzeit der Konfiguration ausgegeben. Die einzelnen LabCar-Hardwarekomponenten werden zusammengefasst, gruppiert und sortiert. Außerdem werden Preise und Komponentenbeschreibungen ausgegeben. Zum Schluss wird die Mehrwertsteuer und die Gesamtsumme berechnet.

Beim vorliegenden Prototyp des LabCar-Konfigurators wird vorerst nur bei den Piggy-Backs die Anzahl bestimmt. Ausgegeben wird dann die Anzahl, die Komponentenbeschreibung des entsprechenden Piggy-Backs und der Einzelpreis. Entsprechend der Anzahl wird dann die Zwischensumme berechnet.

In dieser Form sollen später dann alle übrigen Komponenten zusammengefasst und ausgegeben werden.

4.12 Eingabe »Advanced«

Der Bereich »Advanced« ist für fortgeschrittene Benutzer gedacht, die ihre Signale unter Verwendung einer Excel-Tabelle definieren möchten. Dieser Bereich bietet dem Benutzer den Download einer Excel-Tabellenvorlage an.

Neben der Möglichkeit eine Excel-Tabellenvorlage herunterzuladen, kann der Benutzer eine Excel-Tabelle ins System beziehungsweise auf den Server laden.

Die nachfolgende Bildschirmseite zeigt sich, wenn der Benutzer im Bereich »Advanced« die Schaltfläche »Upload« angeklickt hat.

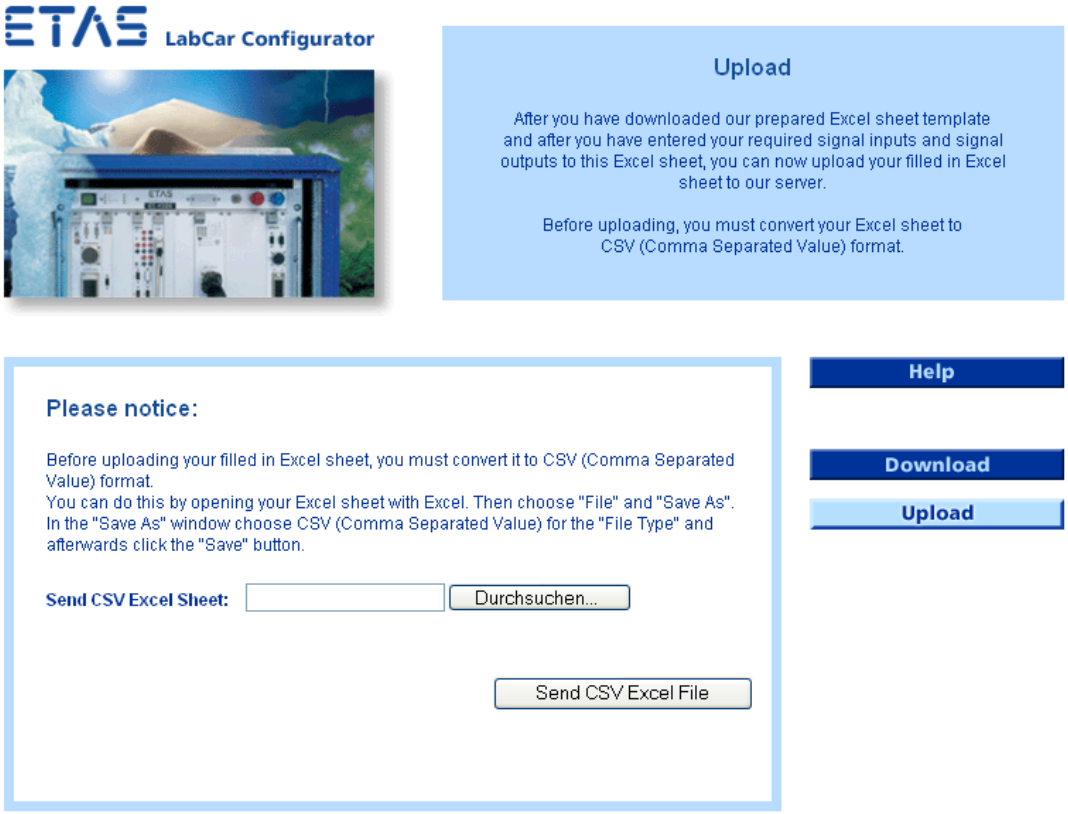

Abbildung 30: Screenshot: Advanced - Upload

Hier kann der Benutzer die Excel-Tabelle, in die er die gewünschten Signalein- und Signalausgänge eingetragen hat, auswählen und anschließend an den Server senden. Damit die Excel-Tabelle serverseitig verarbeitet werden kann, muss der Benutzer die Excel-Tabelle vor dem Hochladen in eine Tabelle im CSV-Format[33] konvertieren. Die Konvertierung kann direkt in Excel sehr einfach durchgeführt werden. Hierfür wird in Excel »Speichern unter« gewählt und dann das Dateiformat CSV eingestellt.

Entspricht die hochgeladene Datei nicht dem erforderlichen Format, erscheint eine Fehlermeldung. Wurde die Datei erfolgreich hochgeladen, wird dem Benutzer im Status-Bereich die Meldung angezeigt, dass die Excel-Tabelle erfolgreich zum Server hochgeladen wurde und dass die Signaldaten nun vollständig auf dem Server vorliegen.

Wie bei der Eingabe »Interactive« erscheint dann auch bei der Eingabe

[33] CSV steht für »Comma Separated Value«.
Das CSV-Format wird in Kapitel »6.3.1 Excel-Tabelle im CSV-Format« beschrieben.

»Advanced« die invers blinkende Start-Configuration-Schaltfläche. Die blinkende Schaltfläche signalisiert dem Benutzer, dass er seinen Part erfolgreich abgeschlossen hat und dass er nun bei der Kernfunktion des LabCar-Konfigurators angekommen ist. Während die Konfiguration durchgeführt wird, blinkt die Schaltfläche weiter, um dem Benutzer zu signalisieren, dass er weiterhin mit dem System verbunden ist und die Konfiguration durchgeführt wird.

Nachdem das System die Konfiguration durchgeführt hat erscheint im Status-Bereich die Meldung, dass die Konfiguration abgeschlossen ist. Im Navigationsbereich kann der Benutzer über die entsprechende Schaltfläche auswählen, in welcher Darstellung er die Ausgabe des Konfigurationsergebnisses wünscht. Der Benutzer kann auch hier wieder zwischen den Darstellungen »Tree View« und »Component List« auswählen.

4.13 Corporate Identity

Neben Werbung (Engl.: Advertising) und Verkaufsförderung (Engl.: Sales Promotions) sind die »Public Relations« (Englisch für: Beziehungen nach außen - zur Öffentlichkeit hin) die dritte Säule der Kommunikationspolitik eines Unternehmens. Hierbei unterscheiden sich die einzelnen Säulen jedoch durch den Gegenstand der Kommunikation: Im Mittelpunkt steht das Unternehmen, nicht die einzelne Ware.

In jüngerer Zeit erfolgt eine Integration von Marketing und Öffentlichkeitsarbeit unter dem Dach der Unternehmenspersönlichkeit (Engl.: Corporate Identity). Der Öffentlichkeitsarbeit fällt unter anderem die Aufgabe der Unternehmenskommunikation (Engl.: Corporate Communications) zu. Wesentlich an diesem Gedanken ist die Integration aller Maßnahmen (integrierte Kommunikation), um »immer in die gleiche Kerbe hauen« zu können. (vgl. Net-Lexikon 2004, Stichwort: »Öffentlichkeitsarbeit - Einordnung in die Unternehmensfunktionen«)

Ein wichtiger Punkt der »Coorporate Identity« ist somit die Präsentation des Unternehmens, sowohl innerhalb des Unternehmens, als auch nach außen hin. Diese Präsentation muss nach dem Grundsatz, des »immer in die gleiche Kerbe Hauens«, möglichst eine kontinuierlich gleichbleibende sein.

Für den LabCar-Konfigurator bedeutet dies, dass er bereits Bestehendes integriert und darauf aufbaut. Zum einen wurde das ETAS-Logo in der ge-

wohnten ETAS-Farbe blau eingebaut. Zum anderen wurde ein Bild, das bereits im Printbereich auf Werbepostern und in Flyern verwendet wird, in die Benutzungsoberfläche des LabCar-Konfigurators integriert.

Damit soll eine Verbindung, zwischen bereits Bekanntem und der neuen Applikation, dem LabCar-Konfigurator, geschaffen werden. Das Bild des LabCars in bildet damit einen optischen Anreiz. Für den Nutzer innerhalb des Unternehmens, der dieses Bild oder ähnliche Bilder schon kennt, soll durch dieses Bild ein Erinnerungseffekt angestoßen werden. Durch den optischen Anreiz soll beim jeweiligen Nutzer sein ganz spezifisches Wissen im Bereich LabCar ins Bewusstsein gerückt werden. Der Wissensumfang wird dabei sehr unterschiedlich sein.

Für den Nutzer außerhalb, beispielsweise den Kunden, wird eher die emotionale Komponente des Bildes im Vordergrund stehen. Ein LabCar, umgeben von Extrembedingungen, wie extreme Kälte, Hitze, Frost und Feuchtigkeit. Damit lautet die Botschaft, dass es sich bei LabCars um robuste Produkte handelt, die auch extremen Bedingungen standhalten. Mit einer Firma, die solche Produkte herstellt und verkauft, verbindet der Kunde auch ein Unternehmensbild. Außerdem wird mit diesem Bild eine Verbindung zu den Testszenarien, die mit den LabCars durchgeführt werden sollen, hergestellt. Entsprechend den in Kapitel 2.3 »Das System Fahrzeug-Fahrer-Umwelt« beschriebenen Anforderungen an die Autombilelektronik, müssen auch die Testsysteme rauen und wechselnden Umgebungsbedingungen in Bezug auf Temperaturbereich, Feuchtigkeit, Erschütterungen oder hohen Anforderungen an die elektromagnetische Verträglichkeit, standhalten.

Dabei liegt es im Interesse eines Unternehmens, dass das Unternehmensbild des Kunden mit dem Bild, das das Unternehmen von sich selbst hat, also der eigenen »Coorporate Identity«, so weit als möglich im Einklang steht.

4.14 Texte

Das Lesen am Monitor ist mühsam.

Der amerikanische Web-Forscher Jakob Nielsen schreibt:

»How Users Read on the Web?
They don't. People rarely read Web pages word by word; instead, they scan the page, picking out individual words and sentences.«
(Nielsen, 1997)

Nielsen fordert in der Konsequenz, Texte zu verfassen, die der Leser scannen kann. Nielsen empfieht deshalb die Anwendung folgender Mittel:

- hervorgehobene Schlüsselworte,
- Unterüberschriften,
- gegliederte Listen,
- ein Gedanke je Absatz und
- nur die halbe Wortzahl wie bei der konventionellen Art zu schreiben.

Bei der Verfassung der Texte für den LabCar-Konfigurator wurden aus diesem Grund die Texte auf die notwendigsten Informationen beschränkt. Für jede inhaltliche Information wurde nach Möglichkeit ein eigener Absatz gesetzt. Außerdem wurden Unterüberschriften verwendet. Rückmeldungen an den Benutzer werden mit einem fettgedrucktem hervorgehobenen »Please Notice« überschrieben. Für die Fehlermeldungen bei der interaktiven Signaldefinition wurde für jeden Fehler ein eigener Absatz gewählt.

Als Schrift wurde die serifenlose Schrift Arial gewählt. Für die Darstellung am Bildschirm sind serifenlose Schriften besser geeignet und lassen sich am Bildschirm besser lesen als Schriften mit Serifen.
(vgl. Thissen, 2003, S. 94. ff.)

Die Texte des LabCar-Konfigurators sind auf Englisch. Der Grund hierfür liegt darin, dass sich der LabCar-Konfigurator an eine internationale Benutzerzielgruppe richtet.[34]

[34] siehe hierzu auch: Kapitel 2.13 »Benutzeranforderungen«

5 Architektur der Anwendung

Mit Hilfe von UML-Diagrammen und Entity-Relationship-Diagrammen werden die Anforderungen an die Architektur des LabCar-Konfigurators beschrieben. Die Architektur des LabCar-Konfigurators wird dann in Anlehnung an das Modell einer Drei-Schichten-Architektur dargestellt und zusammengefasst.

5.1 Softwarearchitektur

"Eine Softwarearchitektur beschreibt die Struktur des Softwaresystems durch Systemkomponenten und ihre Beziehungen untereinander."
(Balzert, 2000, S. 696)

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationship among them."
(Bass et al., 1998)

Die Softwarearchitektur repräsentiert die früheste Entscheidung über das Softwaredesign. Mit ihr werden alle Parameter wie Modifizierbarkeit, Wartbarkeit, Sicherheit und Performance in gewissen Grenzen festgelegt und sind in späteren Entwicklungsphasen nur noch mit erheblichen Kosten- und Zeitaufwendungen abänderbar. Die Entscheidung über das Design einer Softwarearchitektur ist somit eine der kritischsten und wichtigsten Punkte während des Softwareentwicklungsprozesses.
(vgl. WIKIPEDIA 2004, Stichwort: »Softwarearchitektur«)

Beim Entwurf der Softwarachitektur wurde die bereits weiter oben beschriebene Unified Modeling Language (UML) eingesetzt. In den nachfolgenden Kapiteln werden die Anforderungen an die Architektur des LabCar-Konfigurators mit Hilfe von UML-Diagrammen dargestellt. Es wurden Aktivitätsdiagramme (Engl.: Activity Diagram), ein Zustandsdiagramm (Engl.: State Diagram) und ein Einsatzdiagramm (Engl.: Deployment Diagram) entworfen.

5.2 Modellierung von Arbeitsabläufen

Um Arbeitsabläufe bei der LabCar-Konfiguration zu modellieren wurden Aktivitätsdiagramme (Engl.: Activity Diagramm) entwickelt. Aktivitätsdiagramme sind Bestandteil der UML. Mit einem Aktivitätsdiagramm können auch parallele Prozesse modelliert werden. Mit Aktivitätsdiagrammen kann das dynamische Verhalten eines Systems dargestellt werden.

Aktivitätsdiagramme enthalten folgende Elemente:

- Aktivitäten
- Transitionen
- Synchronisationslinien
- Swimlanes (Englisch für: Schwimmbahnen) - optional

In Aktivitätsdiagrammen werden die Objekte eines Vorganges mittels der Aktivitäten beschrieben, die sie während des Ablaufes ausführen. Eine Aktivität ist ein einzelner Schritt innerhalb eines Programmablaufes. Eine Transition ist der Übergang zweier Aktivitäten. Beim Erreichen der Transition ist die vorhergehende Aktivität abgeschlossen. Bei Synchronisationslinien wird auf alle eingehenden Transitionen gewartet und alle ausgehenden Transitionen werden gleichzeitig gestartet. Die optionalen Swimlanes begrenzen normalerweise Klassen, das bedeutet, alles innerhalb einer Schwimmbahn gehört zu einer Klasse. Swimlanes können optional verwendet werden.

Im nachfolgenden Aktivitätsdiagramm ist der Vorgang »Signalliste erstellen« dargestellt. Der Benutzer (Engl.: User) ist dabei der Amateur, der vom System interaktiv unterstützt wird.

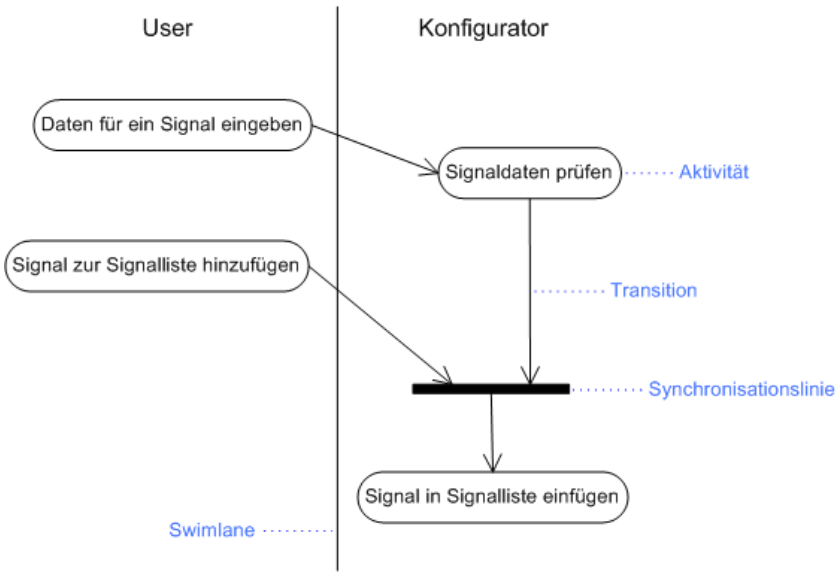

Abbildung 31: Aktivitätsdiagramm: »Signalliste erstellen«

Der Ablauf für die Durchführung einer LabCar-Konfiguration ist im nachfolgenden Aktivitätsdiagramm dargestellt.

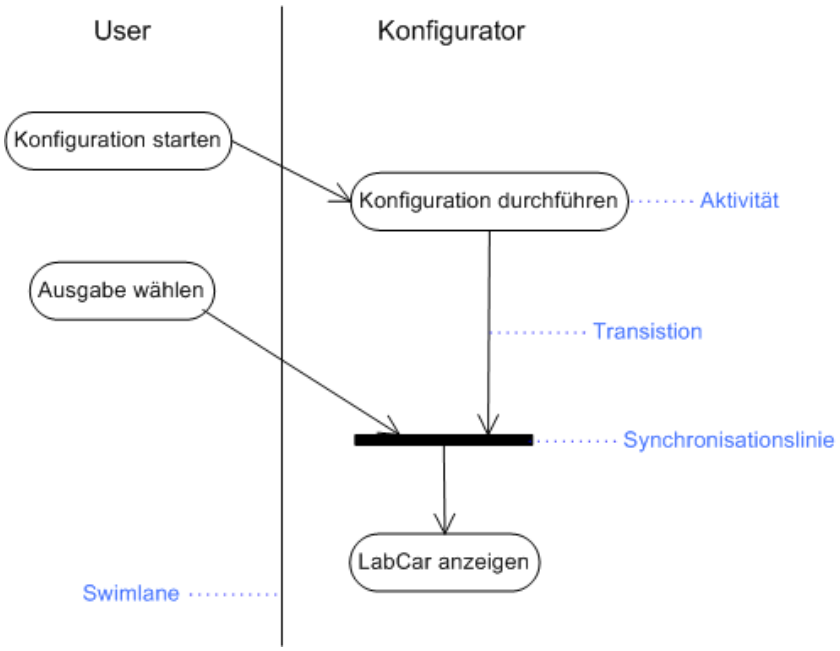

Abbildung 32: Aktivitätsdiagramm: »Konfiguration durchführen«

Der Vorgang der Konfiguration mit Hilfe einer Excel-Tabelle ist im folgenden Aktivitätsdiagramm zusammengefasst.

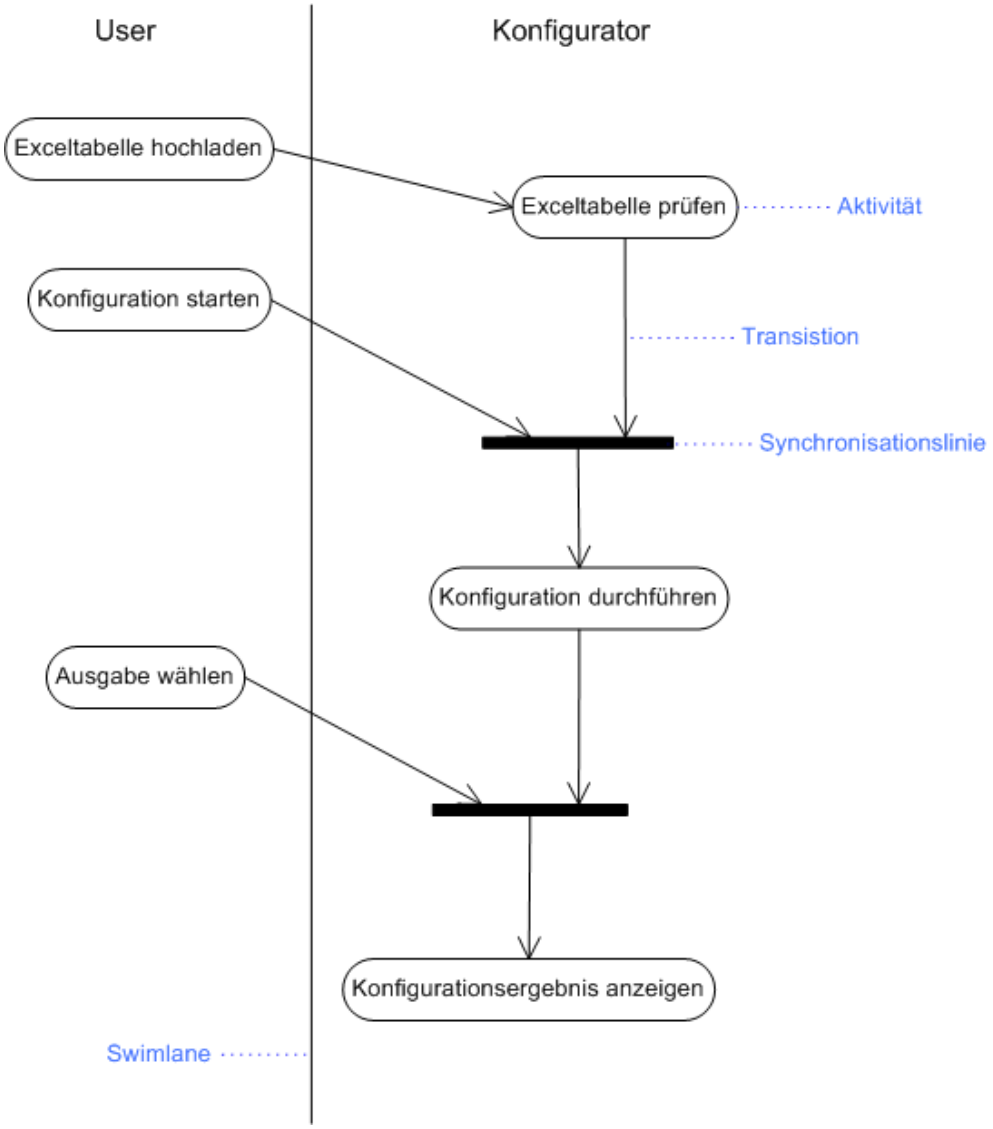

Abbildung 33: Aktivitätsdiagramm: »Konfiguration mit einer Excel-Tabelle«

Der Benutzer ist hier der Experte, der in der Lage ist, seine Signale selbständig in einer Excel-Tabelle zu definieren.

5.3 Datenmodellierung

Wie bereits beschrieben konnte auf die vorhandenen Daten nicht direkt zugegriffen werden. Es war notwendig eigene Schnittstellen für den Datenaustausch zu schaffen. Hierfür mussten geeignete Datenstrukturen definiert und modelliert werden. Die Datenmodellierung wurde mit Hilfe des Entity-Relationship-Modells vorgenommen. Die Schnittstellen selbst wurden dann in XML implementiert. Die Implementierung wird in Kapitel 6 »Implementierung« beschrieben.

Ziel der Datenmodellierung im Entity-Relationship-Modell ist es einen Ausschnitt der realen Welt abzubilden, beziehungsweise Teile der Umgebung zu beschreiben. In diesem Ausschnitt existieren verschiedenartige Objekte, beziehungsweise Einheiten (Engl.: Entity), die in unterschiedlichen Beziehungen (Engl.: Relationship) zueinander stehen.

Das Entity-Relationship-Modell ist ein semantisches Datenmodell und ist von der späteren Implementierung völlig unabhängig. (vgl. Herde, 2004)

Nachfolgend werden die entworfenen Entity-Relationship-Diagramme dargestellt und näher erläutert.

5.4 Datenmodell für den LabCar-Konfigurator

Das Datenmodell, das einer LabCar-Konfiguration zugrunde liegt, wird nachfolgend in Entity-Relationship-Diagrammen (Abkürzung: ERD) dargestellt.

Beim nachfolgenden ERD wird von einem Katalog ausgegangen, in dem LabCar-Komponententypen aufgelistet sind und näher spezifiziert werden.

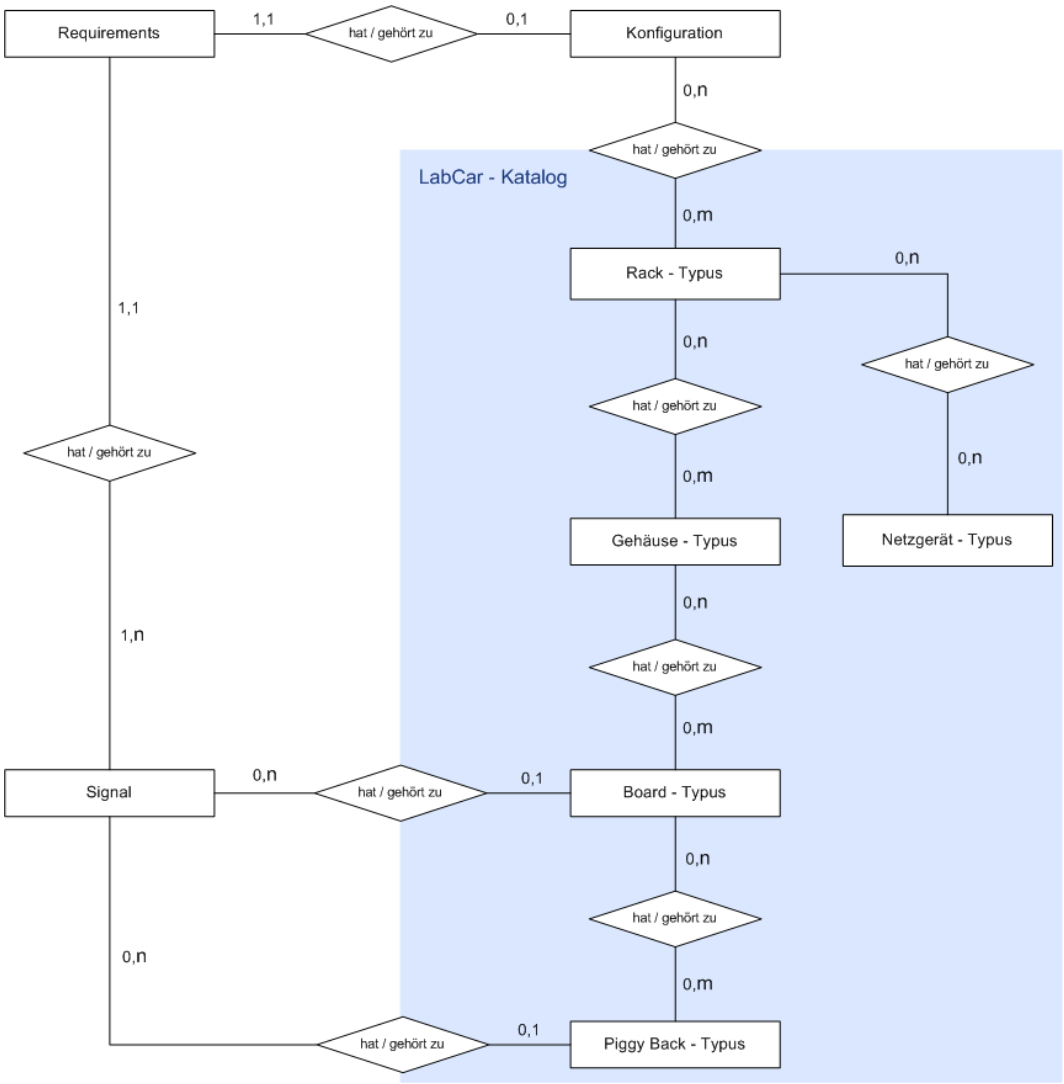

Abbildung 34: ERD - LabCar - Komponententypen

Es liegen hier zwischen den einzelnen Komponententypen komplex-komplexe Beziehungen beziehungsweise so genannte n:m-Beziehungen vor. Ein bestimmter Rack-Typus kann beispielsweise kein Mal, einmal oder auch mehrmals zu verschiedenen Konfigurationsobjekten gehören. Ein Piggy-Back-Typus kann kein Mal, einmal oder auch mehrmals auf einen bestimmten Karten-Typ (Board-Typus) aufgesteckt werden. Ein definiertes Signal kann einem bestimmten Karten-Typ einmal oder auch kein Mal zugeordnet werden. Ein Karten-Typ kann keinen, einen oder auch mehrere definierte Signalein- / Signalausgänge bedienen. Wie solche n:m-Beziehungen in XML umgesetzt werden können, wird in Kapitel 6 »Implementierung« näher beschrieben.

Betrachtet man eine konkrete Konfiguration in der Realität, ändert sich das ERD. Dem nachfolgenden ERD liegt ein konkretes und real konfiguriertes LabCar mit real vorhanden LabCar-Komponenten zugrunde.

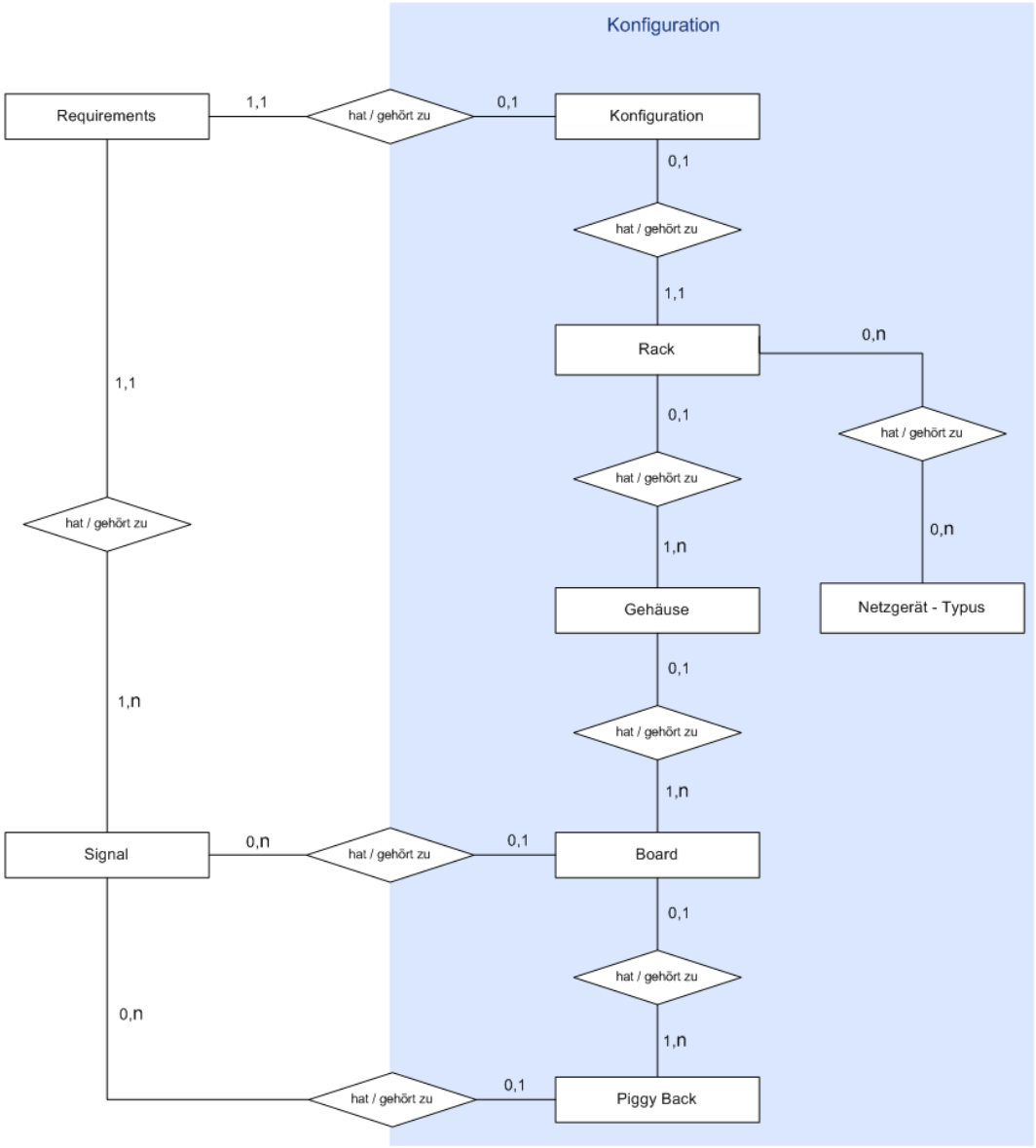

Abbildung 35: ERD - LabCar - Konfiguration

Bei diesem zweiten ERD wird davon ausgegangen, dass es sich um LabCar-Komponenten handelt, die in der Realität als konkrete Objekte existieren. Von einem Komponententypus kann es in der Realität aber mehrere Exemplare geben. Jedes einzelne Exemplar ist dabei eindeutig identifizierbar.

Dadurch verändern sich Komplexitäten. Aus den n:m-Beziehungen im ersten ERD »LabCar-Komponententypen« werden 1:m-Beziehungen im zweiten ERD »LabCar-Konfiguration«.

5.5 Zustandsbeschreibung einer Signalliste

Ein Objekt kann in seinem Leben verschiedenartige Zustände annehmen. Mit Hilfe des Zustandsdiagrammes (Engl.: State Diagram) visualisiert man diese, sowie Funktionen, die zu Zustandsänderungen des Objektes führen. Zustandsdiagramme sind in der UML definiert.

Ein Zustandsdiagramm beschreibt eine hypothetische Maschine, die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet. Sie besteht aus:

- einem Anfangszustand
- einer endlichen Menge von Zuständen
- einer endlichen Menge von Ereignissen
- einer endlichen Anzahl von Transitionen, die den Übergang des Objektes von einem zum nächsten Zustand beschreiben
- einem oder mehrerer Endzustände

Im Zustandsdiagramm werden die Zustände als abgerundete Rechtecke verbunden durch Pfeile, auf denen die Transitionen stehen, visualisiert. Eine Transition ist ein Zustandsübergang eines Objektes, der durch ein wesentliches Vorkommnis - ein Ereignis ausgelöst wird. Startzustand ist ein gefüllter Kreis, die Endzustände sind leere Kreise mit einem kleineren gefüllten in der Mitte. (vgl. Dumke, 2004)

Nachfolgend ist das Zustandsdiagramm einer Signalliste für den LabCar-Konfigurator dargestellt:

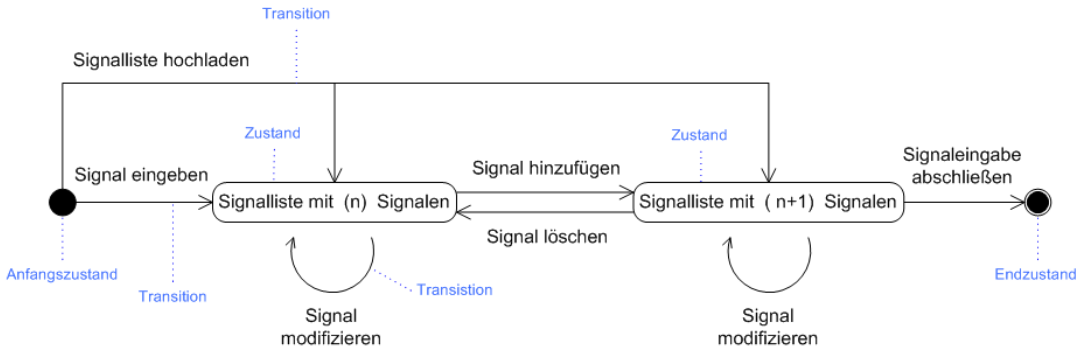

Abbildung 36: Zustandsdiagramm - Signalliste

(n) steht dabei für die Anzahl der Signale. (n) kann den Wert 0 annehmen und kann bis zu einer endlichen Anzahl von Signalen ansteigen. Beim LabCar-Konfigurator liegt die Grenze für komplexe LabCar-Systeme bei mehreren hundert Signalen.

5.6 Visualisierung der Hardware-Verbindungen

Um Hardware und deren Verbindungen zu visualisieren, können Einsatzdiagramme (Engl.: Deployment Diagram) verwendet werden. Einsatzdiagramme sind ebenfalls in der UML definiert. Bei Einsatzdiagrammen ist zu beachten, dass Individuen beziehungsweise Exemplare Verwendung finden und keine Klassen.

Die Knoten stellen Hardware- oder Verarbeitungseinheiten dar. Zwischen Knoten existieren Verbindungen, die physikalische Kommunikationspfade repräsentieren. Ein Einsatzdiagramm beschreibt, welche Komponenten (Objekte) auf welchen Knoten ablaufen und welche Abhängigkeiten bestehen. Knoten werden im Einsatzdiagramm als Quader visualisiert. Knoten die miteinander kommunizieren werden durch Linien miteinander verbunden.

Für den LabCar-Konfigurator wurde folgendes Einsatzdiagramm entworfen:

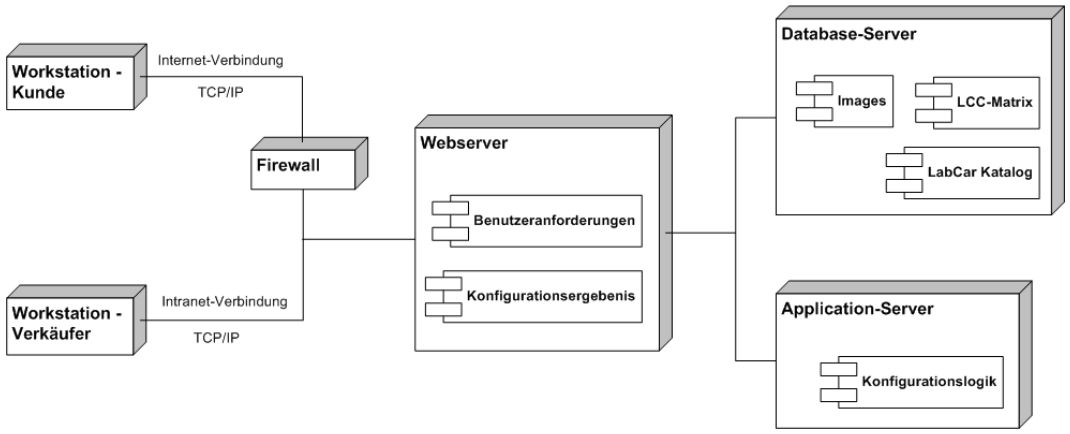

Abbildung 37: Einsatzdiagramm

Für den Zugriff auf die Anwendung gibt es zwei Möglichkeiten, zum einen den internen Zugriff über das Intranet und zum anderen den Zugriff von extern über das Internet.

Auf dem Web-Server werden die Anforderungen abgelegt, die der jeweilige Benutzer definiert hat. Sobald eine Konfiguration durchgeführt wurde, existiert für den entsprechenden Benutzer dann auch ein Konfigurationsergebnis. Sowohl die Benutzeranforderungen, als auch das Konfigurationsergebnis werden als XML-Dateien auf dem Web-Server gespeichert.

Auf einem Datenbankserver (Engl.: Database Server) liegen ein LabCar-Katalog, ebenfalls in XML-Format, Bilder (Thumbnails) für die einzelnen LabCar-Komponenten und eine Matrix zur Konfiguration von LabCars. Die Matrix zur Konfiguration von LabCars stellt eine mehrdimensionale Tabelle dar. Anhand dieser Tabelle können die LabCar-Komponenten bestimmt werden, die für die jeweiligen Benutzeranforderungen, beziehungsweise für die jeweiligen Signalein- und Signalausgänge, benötigt werden.

Auf einem Anwendungsserver (Engl.: Application Server) liegt die Konfigurationslogik. Die Konfigurationslogik ist das Programm, das anhand der Benutzeranforderungen das Konfigurationsergebnis erzeugt, also die Konfiguration selbst durchführt. Dabei greift die Konfigurationslogik auf eine LabCar-Konfigurationsmatrix[35] zu.

[35] Eine Matrix ist ein Schema oder eine mehrdimensionale Tabelle, in welcher zusammenhängende Faktoren in ihrer Beziehung zueinander dargestellt werden. Bei der LCC-Matrix bedeutet dies beispielsweise die Darstellung der Beziehungen zwischen den LabCar-Hardwarekomponenten und den gewünschten Anforderungen.

Dabei können die hier dargestellten einzelnen Komponenten, zum Beispiel Firewall und Web-Server oder Web-Server und Anwendungsserver physikalisch auch auf demselben Server laufen. Die logischen Einheiten bleiben dabei jedoch in der oben dargestellten Weise getrennt.

5.7 Drei-Schichten-Architektur

Balzert (2000) beschreibt eine Schichtenarchitektur als Gliederung einer Softwarearchitektur in hierarchische Schichten. Zwischen den Schichten kann eine lineare, strikte oder baumartige Ordnung bestehen. Jede Schicht besteht aus Systemkomponenten. Anwendungen werden oft nach einer Drei-Schichten-Architektur aufgebaut. Die drei Schichten gliedern die folgenden Bereiche:

- Benutzungsoberfläche
- eigentliche Anwendung
- Datenhaltung

5.8 Web-Architektur

Bei einer Web-Architektur ergeben sich bei den Schichten einige Veränderungen. Der Client enthält einen Web-Browser und wird deshalb als Web-Client bezeichnet. Mit dem Web-Server kommt eine weitere Schicht hinzu, die eine spezielle Funktionalität besitzt.

Bei der Web-Architektur lassen sich nach Balzert (2000) vier Schichten unterscheiden:

- Der Web-Client präsentiert die Benutzungsoberfläche.
- Der Web-Server übernimmt die Verteilung von HTML-Dokumenten und Multimedia-Ojekten, die über das HTTP-Protokoll vom Web-Client angefragt werden. Außerdem stellt er die Kommunikation des Web-Client mit der Anwendungslogik sicher.
- Der Anwendungsserver ist für die Steuerung der Geschäftsprozesse und die Fachkonzept-Objekte zuständig.

- Der Daten-Server ist für die Verwaltung, Bereitstellung und Änderung der persistenten (dauerhaften) Daten zuständig.

5.9 Architektur des LabCar-Konfigurators

Die Architektur des LabCar-Konfigurators lässt sich mit folgendem Modell beschreiben:

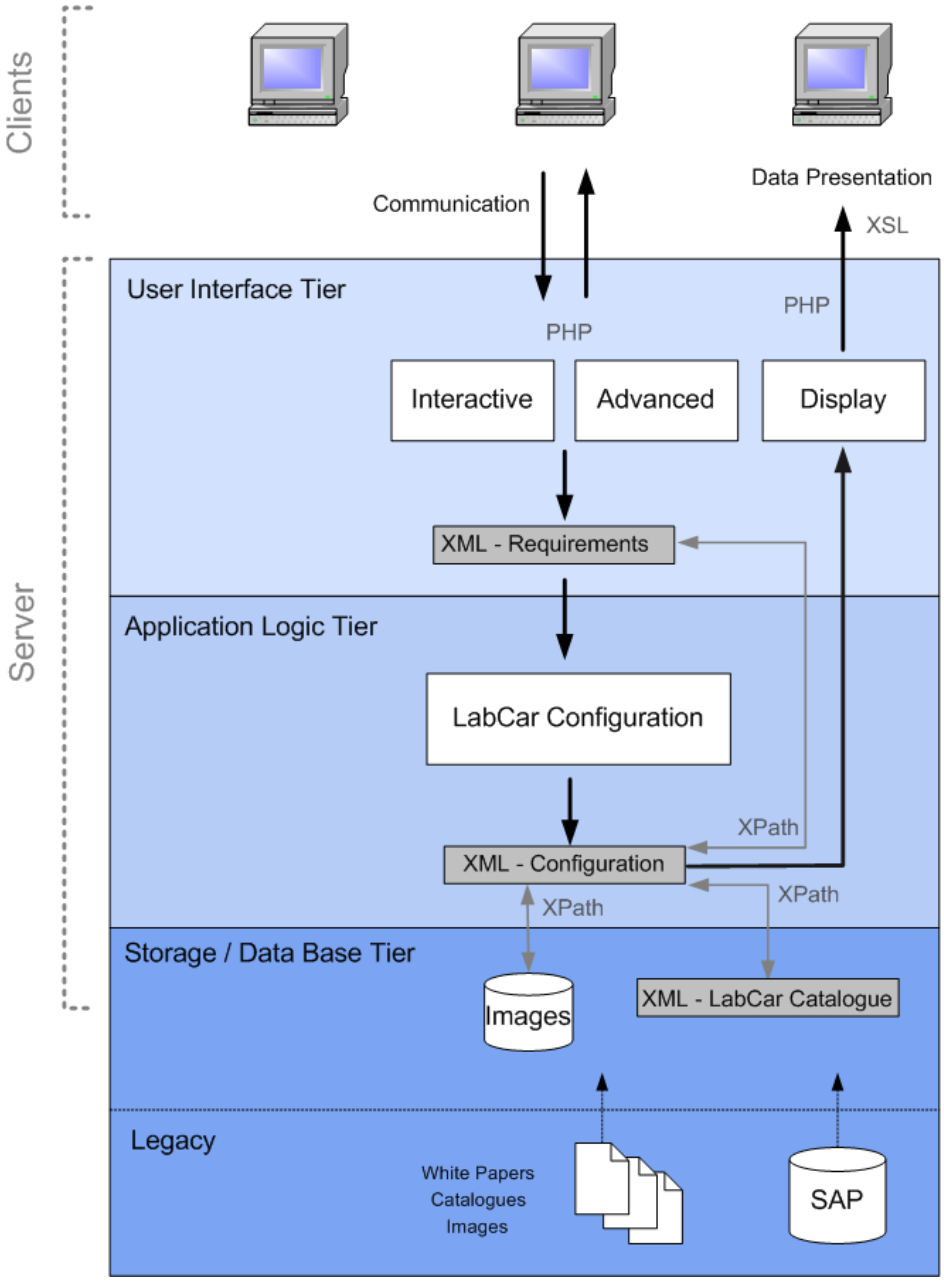

Abbildung 38: Architektur des LabCar-Konfigurators

Die Benutzungsoberflächen-Schicht (Engl.: User Interface Tier) wird vom Web-Server übernommen. Der Web-Server stellt die Verbindung zur Anwendungsschicht (Engl.: Application Logic Tier) her. Die Anwendungslogik kann physikalisch auf einem separaten Anwendungsserver liegen. Auf der Ebene der Anwendungsschicht wird der eigentliche Geschäftsprozess über das Kernprogramm »LabCar-Konfiguration« gesteuert. In der Datenhaltungsschicht (Engl.: Storage/Data Base Tier) werden die benötigten Daten bereitgestellt. Hier wird auf Daten und Dokumente bereits vorhandener Systeme (Engl.: Legacy) zugegriffen.

Die XML-Objekte »XML-Requirements«, »XML-Configuration« und »XML LabCar Catalogue« (Englisch für: XML-LabCar-Katalog) stellen dabei die Kommunikationsschnittstellen zwischen den einzelnen Schichten - »User Interface Tier«, »Application Logic Tier« und »Storage/Data Base Tier«) dar und stellen die Verbindung und zum Web-Client her.

Die Benutzungsoberfläche auf dem Web-Server wird über PHP gesteuert. Für die Darstellung des Konfigurationsergebnisses auf dem Web-Client kommt XSL und X-Path zum Einsatz.

6 Implementierung

Das Kapitel »Implementierung« befasst sich mit der technischen Umsetzung des LabCar-Konfigurators. Die Implementierung wurde unter Verwendung von XML, XSL, PHP und Javascript durchgeführt. Die Umsetzung wird dabei auszugsweise und an ausgewählten Beispielen dargestellt.

Dieses Kapitel richtet sich vorwiegend an den programmiertechnisch interessierten Leser. Der programmiertechnisch weniger versierte Leser kann dieses Kapitel ohne Weiteres überspringen.

6.1 Stufen der Implementierung

Der LabCar-Konfigurator wurde in zwei Stufen implementiert. In der ersten Stufe wurde ein statischer Prototyp umgesetzt. Der statische Prototyp reagiert nicht auf Benutzereingaben. In einer zweiten Stufe wurde der statische Prototyp durch dynamische Komponenten erweitert. Der dynamische Prototyp ist in der Lage Benutzereingaben dynamisch zu verarbeiten.

Nachfolgend werden die einzelnen Stufen beschrieben.

6.2 Statischer Prototyp

Beim statischen Prototyp sind die vorhandenen HTML-Seiten über die einzelnen Funktionsschaltflächen der Benutzungsoberfläche in einer fest vorgegebenen, statischen Netzstruktur miteinander verbunden (Engl.: to link). Die Eingaben des Benutzers werden nicht berücksichtigt. Es ist lediglich eine fest vorgegebene Navigation für den Benutzer möglich. Auch die Ausgabe ist unabhängig von den Benutzereingaben immer dieselbe.

Ein solcher statischer Prototyp ist für die Entwicklung eines Systems ein wichtiger Baustein. Er dient als Diskussionsgrundlage mit dem oder den Auftraggebern und den zukünftigen Benutzern. Der statische Prototyp zeigt die Funktionen der Benutzungsoberfläche auf. In diesem statischen Zustand kann dann noch leicht und ohne großen Aufwand weitere Funktionalität hinzugefügt werden oder die Reihenfolge der Eingabe- und Wahlmöglichkeiten geändert werden. Bei der Entwicklung des LabCar-Konfigurators wurden der statische Prototyp beispielsweise noch um die Funktionen »Load List« und »Save List« im Status-Bereich erweitert.

6.2.1 HTML-Seitengrundstruktur

In HTML wird das Tabellen-Element häufig für das Web-Seiten-Layout verwendet. Über Tabellen kann in HTML eine Seitengrundstruktur erzeugt werden. Seitenelemente können über HTML-Tabellen positioniert werden. Die HTML-Tabellen können hierbei auch verschachtelt werden. Da die Gitternetzlinien für den Betrachter nicht sichtbar sind, spricht man auch von der Technik der »Blinden Tabellen«.

Bei der Umsetzung des LabCar-Konfigurators wurde diese Technik ebenfalls verwendet. Die einzelnen Bereiche, wurden entsprechend der für den LabCar-Konfigurator entworfenen Seitenstruktur definiert und voneinander abgegrenzt. Es wurden Ränder, Rahmen, Abstände, Anordnungen der einzelnen Bereiche und unterschiedliche Einfärbungen des Hintergrunds realisiert. Dabei wurden die in `styles.css` definierten Klassen verwendet. Für die HTML-Tabellen wurden die HTML-Tags `<table>`, `<colgroup>`, `<col>`, `<tr>` und `<td>` eingesetzt.

Die Signalliste im Status-Bereich wird über ein HTML-Inline-Frame eingebunden. Ein Inline-Frame erlaubt es innerhalb eines Webdokuments einen Bereich festzulegen, in welchem wiederum ein eigenes Webdokument angezeigt wird. (vgl. Münz 2001, Stichwort: Inline-Frame)

6.2.2 Cascading Stylesheets

Cascading Stylesheets (CSS) ist eine HTML-Ergänzungssprache. Mit CSS können HTML-Elemente zentral formatiert werden. Unter anderem können Schriftarten und Hintergrundfarben definiert werden. Dabei werden Klassen definiert, auf die dann später im HTML-Code Bezug genommen wird. Auf diese Weise kann für große Projekte ein einheitliches Layout entworfen werden. Mit wenigen kleinen Änderungen an der zentralen Stylesheet-Datei kann für viele HTML-Dateien gleichzeitig ein anderes Aussehen bewirkt werden.

CSS Stylesheets unterstützen zum einen die professionelle Gestaltung beim Webdesign und helfen ein Corporate Design für große Projekte oder unternehmensspezifische Layouts umzusetzen.

Für den LabCar-Konfigurator wurde ebenfalls ein CSS-Stylesheet verwendet. Nachfolgend ein Auszug aus der Stylesheet-Datei `styles.css` des LabCar-Konfigurators.

```
.....
.lightblue_background {
background-color : #B7DCFF;
}
.....
.darkblue_text {
font-size : 12px;
color : #00309C;
}
.....
.filled_in_text {
font-size : 12px;
color : #00309C;
}
.....
.signal_line {
background-color : white;
font-size : 10px;
color : #00309C;
}
.signal_line_caption {
background-color : #B7DCFF;
font-size : 10px;
font-weight: bold;
}
.....
```

6.2.3 Excel-Tabelle für die Signaldefinitionen

Für die Definition der Signalein- und Signalausgängen wurde eine eigene Exceltabelle entworfen. Bei den Daten in der nachfolgend abgebildeten Exceltabelle handelt es sich um fiktive Werte. Die Exceltabelle sieht folgendermaßen aus:

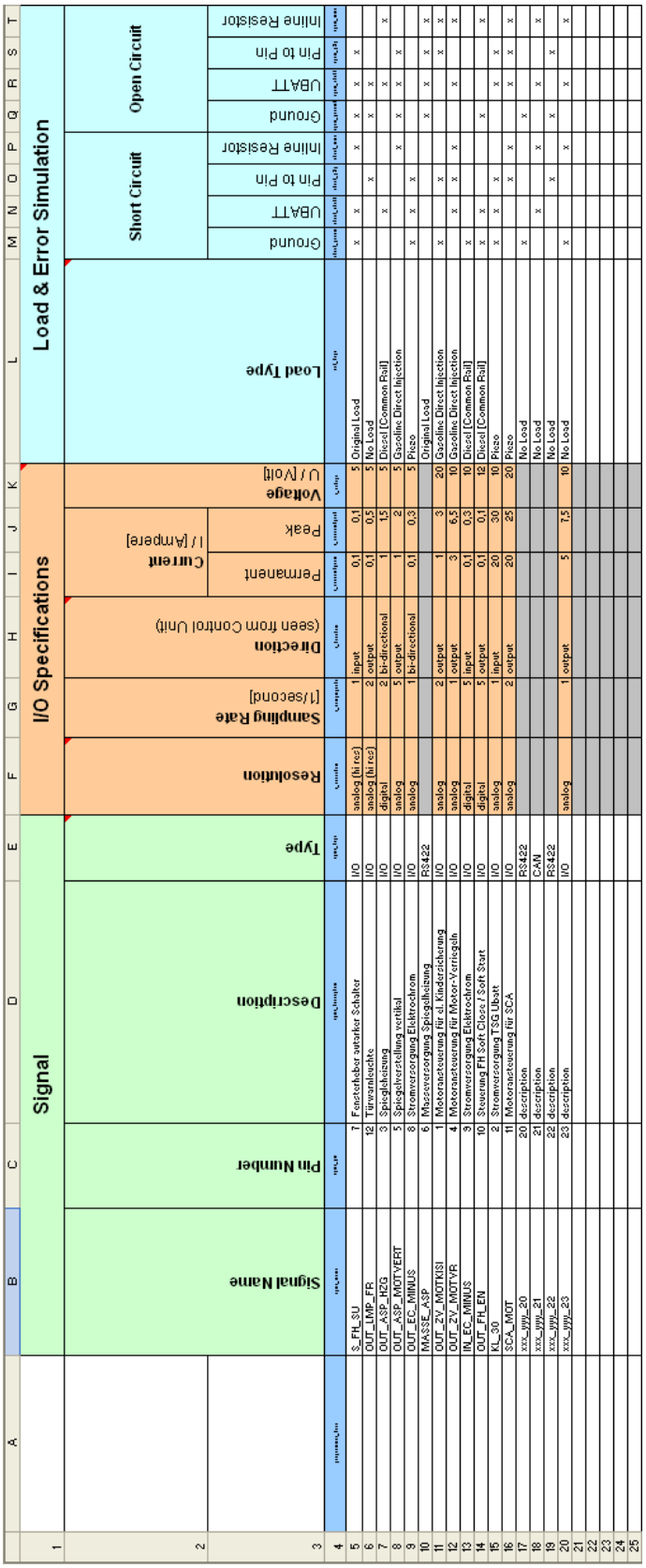

Abbildung 39: Excel-Tabelle zur Definition von Signalen

Die Exceltabelle ist in drei Bereiche gegliedert. Im ersten, hellgrünen Bereich »Signal« werden grundlegende Daten zum jeweiligen Signal definiert. Im zweiten orangefarbig unterlegten Bereich »I/O Specifications« werden Signale vom Signaltyp »I/O« (Input/Output) näher spezifiziert. Im dritten hellblauen Bereich »Load & Error Simulation« wird festgelegt, ob und über welche Lastnachbildung das betreffende Signal verfügen soll, und ob eine beziehungsweise welche Fehlersimulation für das jeweilige Signal möglich sein soll.

Jede Spalte in der Exceltabelle entspricht einem Signalmerkmal. Die einzelnen Signalmerkmale sind in Kapitel 2.10 »Definition von Signalen« ausführlich beschrieben.

Programmiertechnische Attribute

In Zeile vier (mittelblaue Zeile) werden Schlüsselwerte beziehungsweise Attribute für die programmiertechnische Umsetzung, so genannte »Programmer Keys« (Engl.: Schlüssel für den Programmierer), definiert. Diese Zeile ist ausschließlich für die programmiertechnische Umsetzung von Bedeutung. Für den Benutzer des LabCar-Konfigurators hat diese Zeile keine Bedeutung. In der Ecxel-Tabellenvorla- ge, die sich der Benutzer über den Eingabeweg »Advanced« herunterladen kann ist diese Zeile ausgeblendet. Dies bedeutet, dass sie zwar vorhanden ist, für den Benutzer aber nicht sichtbar.

Die Tabellenzeile »Programmer Keys« beinhaltet folgende Werte:
»programmer_keys«, »signal_name«, »signal_pin«, »signal_description«, »signal_type«, »io_resolution«, »io_samplingrate«, »io_direction«, »io_currentperm«, »io_currentpeak«, »io_voltage«, »load_type«, »short_ground«, »short_ubatt«, »short_p2p«, »short_inline«, »open_ground«, »open_ubatt«, »open_p2p« und »open_inline«.

Jeder Spalte, beziehungsweise jedem Signalmerkmal, wird damit ein eigener eindeutiger Name oder Bezeichner zugewiesen. Diese Bezeichner oder Schlüssel für den Programmierer (Engl.: programmer key) werden nachfolgend bei der Implementierung und Programmierung des LabCar-Konfigurators verwendet. Die Attribute der XML-Dateien beispielsweise sind nach denselben Namen benannt.

6.2.4 XML-Dateien

XML steht für Extended Markup Language (zu deutsch: Erweiterbare Auszeichungssprache). Das Auszeichungssprachenkonzept von XML lässt sich sehr vielseitig einsetzen. Mit diesem Konzept lassen sich Daten strukturieren und beschreiben. Elemente, die durch so genannte Tags (Englisch für: Kennzeichung, Anhängeschildchen) markiert werden, können frei definiert werden. Verschachtelungsregeln, Attribute und erlaubte Wertzuweisungen sind weitere Bestandteile des XML-Konzepts.

Eine XML-Datei `Requirements.xml` kann zum Beispiel folgendermaßen aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Requirements SYSTEM "Requirements.dtd">
<Requirements IdNr="210" Date="23-9-2004" Time="18:02:00">
    <Signal signal_name="S_FH_SU" signal_pin="7"
            signal_type="IO">
        <SignalDescription>
            Fensterheber autarker Schalter
        </SignalDescription>
        <IOSpecifications
            io_resolution="analog-hi-res" io_samplingrate="1"
            io_direction="input" io_currentperm="..."
            io_currentpeak="..." io_voltage="...">
        </IOSpecifications>
        <LESimulation>
            <LoadType load_type="Original_Load"/>
            <ShortCircuit short_ground="ja" short_inline="ja"/>
            <OpenCircuit open_ground="ja" open_ubatt="ja"
                         open_p2p="ja"/>
        </LESimulation>
    </Signal>
    <Signal signal_name="OUT_LMP_FR" signal_pin="12"
            signal_type="IO">
        <SignalDescription> ..... </SignalDescription>
        <IOSpecifications ..... ></IOSpecifications>
        <LESimulation> ..... </LESimulation>
    </Signal>
    .....
```

```
    .....
    <Signal signal_name="MASSE_ASP" signal_pin="6"
            signal_type="RS422">
        <SignalDescription> ..... </SignalDescription>
        <LESimulation> ..... </LESimulation>
    </Signal>
    .....
    <Signal ..... > ..... </Signal>
    .....
</Requirements>
```

Das Element `Signal` beispielsweise wird mit den Tags `<Signal> ..... </Signal>` ausgezeichnet. Außerdem verfügt das Element `Signal` über die Attribute `signal_name`, `signal_pin` und `signal_type`. Die erlaubten Wertzuweisungen sind in der zugehörigen Datei `Requirements.dtd` festgelegt, ebenso die Verschachtelungsregeln. Die möglichen Verschachtelungen kann man sehr schön auch direkt im Code der XML-Datei selbst sehen. Der Aufbau und die Verschachtelungen der Elemente in einer XML-Datei entspricht einer Baumstruktur.

Nachfolgend ist beispielhaft eine XML-Datei `Configuration.xml` abgebildet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Configuration SYSTEM "Configuration.dtd">
<?xml-stylesheet type="text/xsl" href="Configuration.xsl"?>
<Configuration IdNr="175" Date="3-2-2004" Time="..... >
    <Rack BestellNr="F 00K 001 218" Preis="950">

        <!-- Signalbox-->

        <Gehaeuse BestellNr="F 00K 001 389" Preis="500">
            <Karte BestellNr="F 00K 001 629" Preis="240"/>
            <Karte BestellNr="F 00K 000 104" Preis="360">
                <SignalKey KurzBez="testsignal_01"/>
                <SignalKey KurzBez="testsignal_02"/>
                .....
```

```
                .....
                <PiggyBack BestellNr="F 00K 001 477"
                           Preis="210">
                    <SignalKey KurzBez="testsignal_03"/>
                    <SignalKey KurzBez="testsignal_04"/>
                </PiggyBack>
                <PiggyBack BestellNr="F 00K 001 463"
                           Preis="180"/>
            </Karte>
            <Karte BestellNr="F 00K 102 690" Preis="280">
                <SignalKey ..... />
                <PiggyBack ..... />
                <PiggyBack ..... />
                .....
            </Karte>
        </Gehaeuse>

        <!-- Lastbox-->

        <Gehaeuse ..... >
            <Karte ..... >
                <PiggyBack ..... />
                .....
            </Karte>
            .....
        </Gehaeuse>

    </Rack>
    <Kommentar> ..... </Kommentar>
    <XMLRequirementsRessource SourceDatei=" ..... "/>
    <ImagesRessource SourceFolder=" ..... "/>
    <LabCarComponentRessource SourceDatei=" ..... "/>
</Configuration>
```

Das Element `SignalKey` beispielsweise, kann in der XML-Datei `Configuration.xml` sowohl auf der Ebene des Elements `Karte`, als auch auf der Ebene des Elementes `PiggyBack` vorkommen.

Die Struktur der XML-Datei `LabCarKatalog.xml` sieht im Groben folgendermaßen aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE LabCarKatalog SYSTEM "LabCarKatalog.dtd"> .....
<LabCarKatalog>
    <SysTeil KeyBestellnr="F 00K 001 368">
        <SysTeilBez> ES4100.1 Chassis VME64x </SysTeilBez>
        <Einsatzgebiet> .... </Einsatzgebiet>
        <Einsatzgebiet> .... </Einsatzgebiet>
        .....
        <Eigenschaft> ..... </Eigenschaft>
        <Eigenschaft> ..... </Eigenschaft>
        .....
        <TechDatum> ..... </TechDatum>
        <TechDatum> ..... </TechDatum>
        .....
        <Bestellinfo> ..... </Bestellinfo>
        .....
        <Vertriebsinfo>
            <Vertriebstext>
                ---- text for sales and distribution ----
            </Vertriebstext>
            <Preis>950</Preis>
        </Vertriebsinfo>
    </SysTeil>
    .....
    <SysTeil KeyBestellnr="..."> ..... </SysTeil>
    .....
</LabCarKatalog>
```

6.2.5 Dokumenttyp-Definitionen

Die Elemente, Verschachtelungsregeln, Attribute und Wertebereiche werden in XML über die Dokumenttyp-Definitionen (Engl.: Document Type Definition - Abkürzung: DTD) festgelegt.

Die DTD `LabCarKatalog.dtd` sieht wie folgt aus:

```
<!ELEMENT LabCarKatalog (SysTeil*)>
<!ELEMENT SysTeil (SysTeilBez, Einsatzgebiet+, Eigenschaft+,
TechDatum+, Bestellinfo+, Vertriebsinfo)>
<!ATTLIST SysTeil
KeyBestellnr CDATA #REQUIRED
>
<!ELEMENT SysTeilBez (#PCDATA)>
<!ELEMENT Einsatzgebiet (EgBeschreibung, Beispiel*)>
<!ELEMENT EgBeschreibung (#PCDATA)>
<!ELEMENT Beispiel (#PCDATA)>
<!ELEMENT Eigenschaft (EigenschaftBez,
EigenschaftSpezifizierung*)>
<!ELEMENT EigenschaftBez (#PCDATA)>
<!ELEMENT EigenschaftSpezifizierung (#PCDATA)>
<!ELEMENT TechDatum (TechDatBezeichnung, Merkmal+)>
<!ELEMENT TechDatBezeichnung (#PCDATA)>
<!ELEMENT Merkmal (Merkmalbezeichnung, Wert)>
<!ELEMENT Merkmalbezeichnung (#PCDATA)>
<!ELEMENT Wert (#PCDATA)>
<!ELEMENT Bestellinfo (BestellBez, KurzBez, Bestellnr)>
<!ELEMENT BestellBez (#PCDATA)>
<!ELEMENT KurzBez (#PCDATA)>
<!ELEMENT Bestellnr (#PCDATA)>
<!ELEMENT Vertriebsinfo (Vertriebstext, Preis)>
<!ELEMENT Vertriebstext (#PCDATA)>
<!ELEMENT Preis (#PCDATA)>
```

Nachfolgend ist die DTD `Requirements.dtd` abgebildet:

```
<!ENTITY % Signaldaten SYSTEM "Signal.dtd">
%Signaldaten;

<!ELEMENT Requirements (Signal*)>
<!ATTLIST Requirements
IdNr CDATA #REQUIRED
Date CDATA #REQUIRED
Time CDATA #REQUIRED
>
```

Die Datei `Requirements.dtd` enthält einen Verweis auf eine DTD `Signal.dtd`. Die DTD `Signal.dtd` ist folgendermaßen aufgebaut:

```
<!ELEMENT Signal (SignalDescription, IOSpecifications?,
LESimulation)>
<!ATTLIST Signal
signal_name CDATA #REQUIRED
signal_pin CDATA #REQUIRED
signal_type (IO | RS422 | ETK | CAN | FT-CAN) #REQUIRED
>

<!ELEMENT SignalDescription (#PCDATA)>

<!ELEMENT IOSpecifications EMPTY>
<!ATTLIST IOSpecifications
io_resolution (digital | analog | analog-hi-res) #REQUIRED
io_samplingrate CDATA #REQUIRED
io_direction (input | output | bi-directional) #REQUIRED
io_currentperm CDATA #REQUIRED
io_currentpeak CDATA #REQUIRED
io_voltage CDATA #REQUIRED
>

.....
```

```
.....
<!ELEMENT LESimulation (LoadType, ShortCircuit, OpenCircuit)>
<!ELEMENT LoadType EMPTY>
<!ATTLIST LoadType
load_type (No_Load | Original_Load | Common_Rail_Diesel |
Gasoline_Direct_Injection | Piezo_Injection) #REQUIRED
>
<!ELEMENT ShortCircuit EMPTY>
<!ATTLIST ShortCircuit
short_ground (ja) #IMPLIED
short_ubatt (ja) #IMPLIED
short_p2p (ja) #IMPLIED
short_inline (ja) #IMPLIED
>
<!ELEMENT OpenCircuit EMPTY>
<!ATTLIST OpenCircuit
open_ground (ja) #IMPLIED
open_ubatt (ja) #IMPLIED
open_p2p (ja) #IMPLIED
open_inline (ja) #IMPLIED
>
```

Die DTD `Konfiguration.dtd` entspricht in ihrer Struktur dem weiter oben bereits beschriebenen Entity Relationship Diagramms für die Konfiguration. In der folgenden Abbildung ist das ERD »LabCar-Konfiguration« noch einmal dargestellt. Allerdings sind hier alle Kardinalitäten die für die DTD von Bedeutung sind, rot markiert.

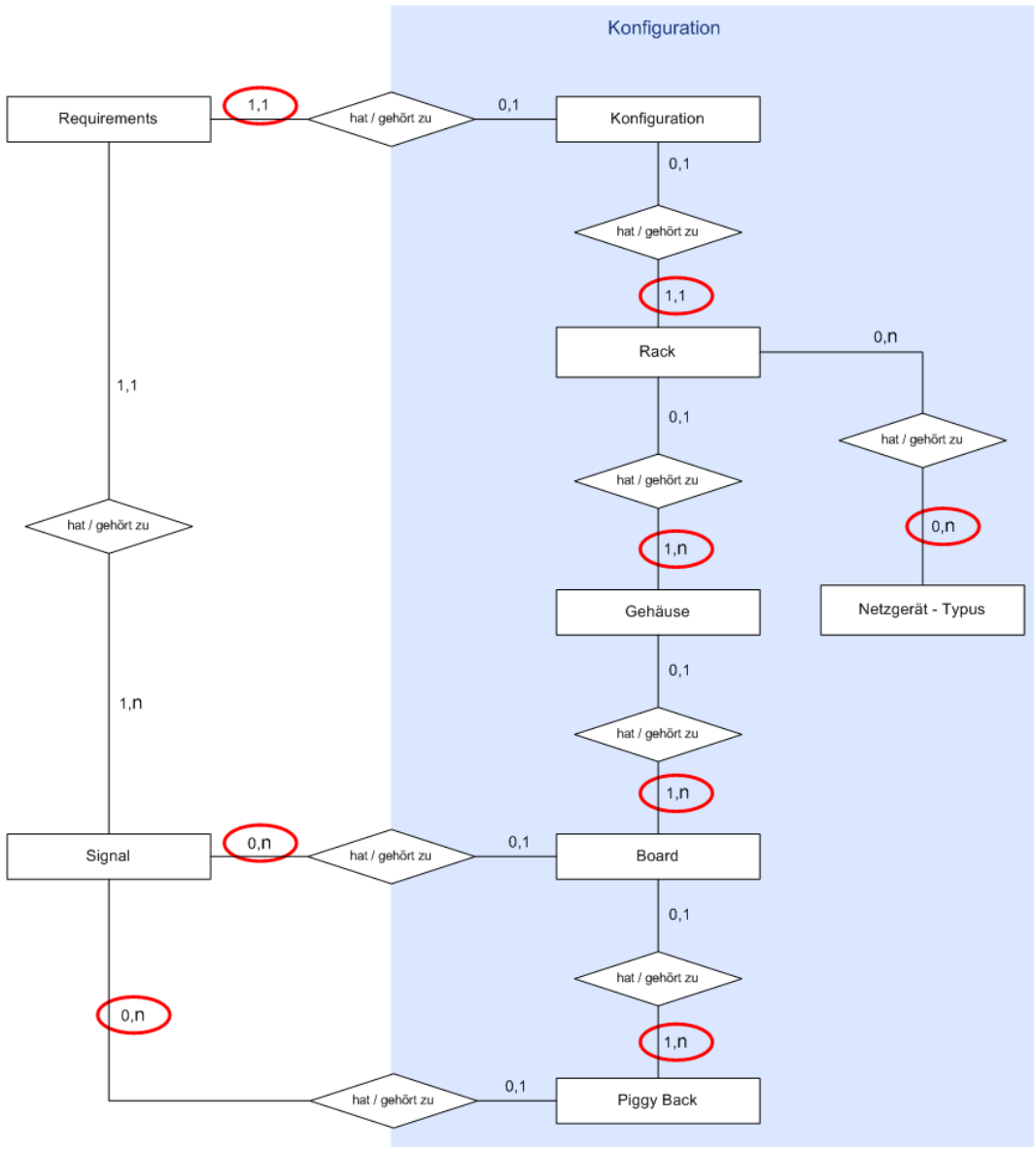

Abbildung 40: ERD - LabCar - Konfiguration *(mit markierten Kardinalitäten)*

In XML gibt es dabei folgende Möglichkeiten die Kardinalitäten aus dem ERD abzubilden:

Zeichen in XML	**Bedeutung**	**entspricht der Kardinalität im ERD**
kein Zeichen	genau einmal	(1,1)
?	kein Mal oder einmal	(0,1)
*	kein Mal, einmal oder mehrmals	(0,n)
+	einmal oder mehrmals	(1,n)

Der XML-Konfigurationsdatei liegt folgende DTD zugrunde:

```
<!ELEMENT Configuration (Rack, Kommentar?, XMLRequirementsRessource,
ImagesRessource, LabCarComponentRessource)>
<!ATTLIST Configuration
      IdNr CDATA #REQUIRED
      Date CDATA #REQUIRED
      Time CDATA #REQUIRED
      Gewicht CDATA #REQUIRED
      Strom CDATA #REQUIRED
      Kommentar CDATA #IMPLIED

>
<!ELEMENT Rack (Netzgeraet?, Gehaeuse+)>
<!ATTLIST Rack
      BestellNr CDATA #REQUIRED
      Preis CDATA #REQUIRED
>
<!ELEMENT Netzgeraet EMPTY>
<!ATTLIST Netzgeraet
      BestellNr CDATA #REQUIRED
      Preis CDATA #REQUIRED
>
<!ELEMENT Gehaeuse (Karte+)>
<!ATTLIST Gehaeuse
      BestellNr CDATA #REQUIRED
      Preis CDATA #REQUIRED
>
<!ELEMENT Karte (SignalKey*, PiggyBack*)>
<!ATTLIST Karte
      BestellNr CDATA #REQUIRED
      Preis CDATA #REQUIRED
>
<!ELEMENT PiggyBack (SignalKey*)>
<!ATTLIST PiggyBack
      BestellNr CDATA #REQUIRED
      Preis CDATA #REQUIRED
>
<!ELEMENT SignalKey EMPTY>
<!ATTLIST SignalKey
      KurzBez CDATA #REQUIRED
>
<!ELEMENT Kommentar (#PCDATA)>
<!ELEMENT XMLRequirementsRessource EMPTY>
<!ATTLIST XMLRequirementsRessource
      SourceDatei CDATA #REQUIRED
>
<!ELEMENT ImagesRessource EMPTY>
<!ATTLIST ImagesRessource
      SourceFolder CDATA #REQUIRED
>
<!ELEMENT LabCarComponentRessource EMPTY>
<!ATTLIST LabCarComponentRessource
      SourceDatei CDATA #REQUIRED
>
```


Abbildung 41: DTD - Configuration

Eine XML-Datei `configuration.xml` verfügt über genau ein Rack und bezieht sich auf genau eine XML-Datei `requirements.xml`. Eine XML-Datei `configuration.xml` verfügt über ein oder auch kein Netzgerät und beinhaltet mindestens ein Gehäuse (Einschubssystem). Ein Gehäuse hat mindestens eine Karte. Auf einer Karte können sich kein, ein oder mehrere Piggy-Backs befinden. Eine Karte und ein Piggy-Back können sowohl über keinen, einen oder mehrere Signalausgänge `SignalKey` verfügen.

Der Preis für jede einzelne LabCar-Komponente wurde hier in der `configuration.dtd` noch einmal eingefügt. Die Preise kommen aus dem LabCar-Katalog. Da die Preise sich aber ändern können und somit zeitlich abhängig sind, wird der Preis, der zum Zeitpunkt der Konfiguration gültig ist, in der Datei `configuration.xml` eingetragen.

6.2.6 Umsetzung komplex-komplexer Beziehungen in XML

Das in Kapitel 5.4 »Datenmodell für den LabCar-Konfigurator« unter Abbildung 34 »ERD-LabCar-Komponententypen« dargestellte Entity-Relationship-Diagramm beinhaltet so genannte komplex-komplexe Beziehungen oder auch **m:n**-Beziehung. Wie diese Beziehung in XML dargestellt wurden, wird nachfolgend beispielhaft an der Beziehung zwischen Gehäuse-Typus und Board-Typus dargestellt.

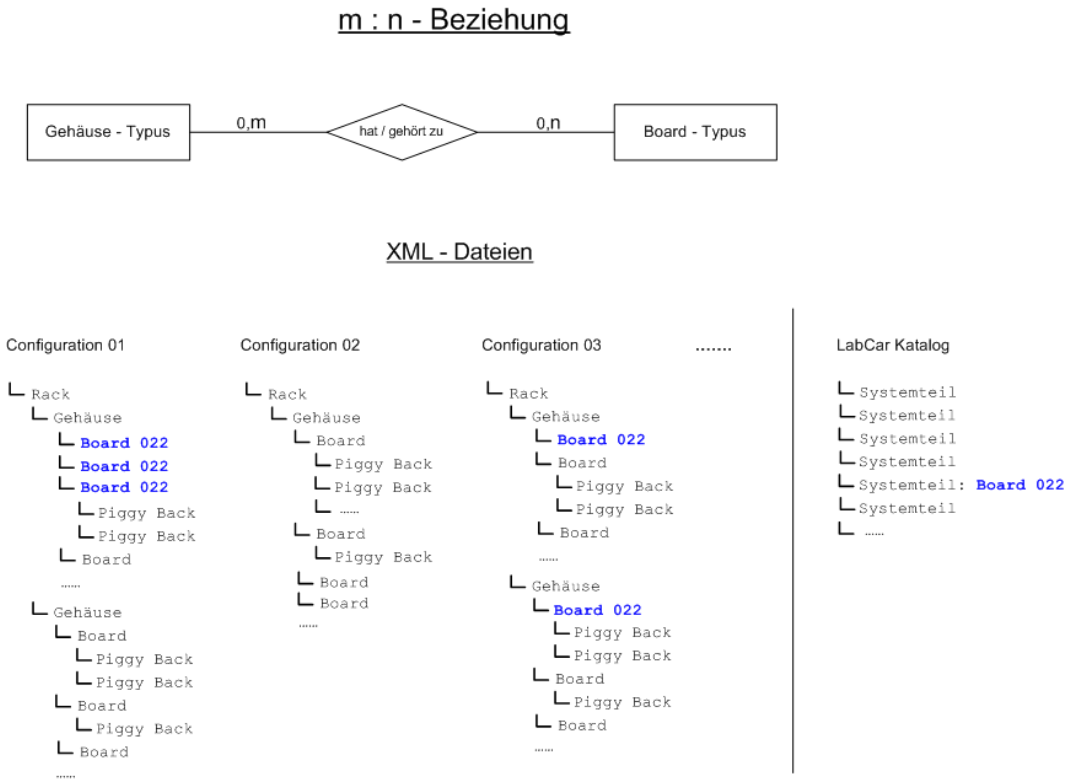

Abbildung 42: **m:n** - Beziehung in XML

Die **m:n**-Beziehung bedeutet in diesem abgebildeten Fall, dass einerseits ein Gehäusetyp einen bestimmten Kartentyp (Board-Typus) kein Mal, einmal oder mehrmals haben kann. Andererseits kann ein bestimmter Kartentyp (Board-Typus) kein Mal, einmal oder mehrmals zu einem bestimmten Gehäusetyp gehören.

In XML wird dies folgendermaßen abgebildet: Für jede durchgeführte Konfiguration existiert eine eigene XML-Konfigurationsdatei. Der LabCar-Ka-

talog wird ebenfalls durch eine XML-Datei abgebildet. Der Kartentyp (Board-Typus) 022 ist im LabCar-Katalog als Typus einmal vorhanden.

In einen bestimmten Gehäusetyp kann der Kartentyp (Board-Typus) 022 kein Mal, einmal oder mehrmals eingebaut werden. Beispielhaft sind in der obigen Abbildung drei Konfigurationsdateien dargestellt. In »Configuration 01« ist der Kartentyp 022 drei Mal eingebaut, in »Configuration 02« ist kein Kartentyp 022 eingebaut und in »Configuration 03« ist der Kartentyp 022 jeweils einmal pro Gehäuse integriert.

Irgendein anderer Kartentyp, beispielsweise Board-Typus 099, kann möglicherweise in keinem, oder nur in einem ganz bestimmten Gehäusetyp oder in mehreren Gehäusetypen eingebaut sein.

6.2.7 Tree-View-Stylesheet

Das Tree-View-Stylesheet wurde als Push-Stylesheet[36] implementiert. Dies bedeutet, die Daten werden in der Reihenfolge und hierarchischen Struktur, wie sie in der XML-Ausgangsdatei `configuration.xml` vorliegen, dargestellt. Dabei wird das XSLT-Element `<xsl:template>` verwendet. Das XSLT-Element `<xsl:template>` dient zur Definition von Schablonen (Engl.: template) für die Übersetzung vom Quellbaum in den Ergebnisbaum. Dazu wird angegeben, welcher Knoten des Ausgangsbaumes in welches Konstrukt des Ergebnisbaumes übersetzt werden soll. Beim LabCar-Konfigurator wird der XML-Code des Ausgangsbaumes in HTML-Code für die Ausgabe des Ergebnisbaumes im Web-Browser umgewandelt.

Für jeden Knoten des Quelldokumentes, der im Ergebnisbaum dargestellt werden soll, wird eine Schablone oder ein Template definiert beginnend beim Wurzelknoten. Über das XSLT-Element `<xsl:apply-templates>` werden andere definierte Templates angewendet. Auf diese Weise lassen sich Abhängigkeiten und Reihenfolgen bei der Anwendung von Templates steuern.

Das XSLT-Element `<xsl:stylesheet ...>` `</xsl:stylesheet>` markiert den Anfang und das Ende des Stylesheets.

---

[36] Das Prinzip, das Push-Stylesheets zugrunde liegt, wird in Kapitel 3.11 »Darstellung des Konfigurationsergebnisses mit XSL« näher erläutert.

Die Struktur des Tree-View-Stylesheets sieht folgendermaßen aus:

```
<xsl:stylesheet ..... >
.....
<xsl:template match="Configuration">
.....
<xsl:apply-templates/>
.....
</xsl:template>
.....
<xsl:template match="Rack">
.....
<xsl:apply-templates/>
.....
</xsl:template>
.....
<xsl:template match="Gehaeuse"> ..... </xsl:template>
.....
<xsl:template match="Karte"> ..... </xsl:template>
.....
<xsl:template match="PiggyBack"> ..... </xsl:template>
.....
</xsl:stylesheet>
```

Auf Elemente, die nicht direkt in der XML-Quelldatei `configuration.xml` vorliegen, wird über einen Schlüssel verwiesen. Die Umsetzung sieht folgendermaßen aus:

```
.....
<xsl:template match="SignalKey">
<xsl:variable name="this_signal_key">
<xsl:value-of select="@KurzBez"/>
</xsl:variable>

.....
```

```
.....
<xsl:apply-templates
select="document(/Configuration/XMLRequirementsRessource/
@SourceDatei) //Signal[@signal_name=$this_signal_key] "/>
</xsl:template>

.....
```

Das XSLT-Element `<xsl:value-of select= "... >` erzeugt eine gespeicherte Zeichenkette an der aktuellen Position im Ausgabebaum. Bei der gespeicherten Zeichenkette kann es sich um den Inhalt eines Knotens der XML-Datei handeln oder um den Inhalt einer zuvor definierten Variablen. Im obigen Beispiel wird der aktuelle Wert des Attributes `@KurzBez` am Knoten `SignalKey` in den Ausgabebaum eingefügt.

Über das XSLT-Element `<xsl:variable>` wird der aktuelle Signalschlüssel als Variable `this_signal_key` definiert. Auf den Wert dieser Variablen kann im weiteren Verlauf zugegriffen werden.

Das XSLT-Element `<xsl:apply-templates>` wendet innerhalb des Templates `SignalKey` andere Templates an, die ebenfalls mit `<xsl:template>` definiert sind.

Die X-Path-Funktion `document()` erlaubt es, XML-Ausgangsdaten aus anderen XML-Dateien als der, in der das Stylesheet eingebunden ist, mit in den Ergebnisbaum zu übernehmen. Damit können die Signale aus der zugehörigen XML-Datei »Requirements« eingebunden werden. Über die X-Path-Syntax `[@signal_name=$this_signal_key]` wird eine Vergleichsoperation durchgeführt. Der aktuelle Signalschlüssel der Ausgangsdatei »Konfiguration«, dessen Wert in der Variablen `this_signal_key` abgelegt ist, wird mit den Signalschlüsseln in der zugehörigen eingebundenen Datei »Requirements«, die dort am Knoten `Signal` im Attribut `@signal_name` liegen, verglichen. Ist der entprechende Signalschlüssel mit demselben Wert gefunden, wird das Signal in den Ergebnisbaum eingebunden.

Die Bilder (Thumbnails) für die LabCar-Komponenten sind in einem separaten Bilderverzeichnis abgelegt. Die Benennung der Bilder erfolgt entsprechend der eindeutigen Bestellnummer, die im Attribut `BestellNr` enthalten ist. Die Bilder sind im JPEG-Format abgelegt. Der Zugriff auf die Bilder wurde wie folgt implementiert:

```
.....
<img>
<xsl:attribute name="src">
<xsl:value-of
select="concat(/Configuration/ImagesRessource/@SourceFolder,
@BestellNr, '.jpg')"/>
</xsl:attribute>
</img>
.....
```

Über das XSLT-Element `xsl:attribute` kann im Ergebnisbaum ein Attribut gesetzt werden. Für das Source-Attribut des HTML-Image-Tags wird damit für jeden Knoten der jeweilige Bezug zum entsprechenden Bild hergestellt. `/Configuration/ImagesRessource/` verweist auf den Knoten in der Ausgangsdatei, an dem das Attribut `@SourceFolder` liegt. Das Attribut `SourceFolder` enthält den Pfad zum entprechenden Bilderverzeichnis.

Das Attribut `@BestellNr` liefert die eindeutige Bestellnummer des aktuellen Knotens. Der Knoten entpricht hier einer LabCar-Hardware-Komponenten.

Über die X-Path-Funktion `concat()` werden alle Einzelwerte zu einer Zeichenkette zusammengefasst.

6.2.8 Component-List-Stylesheet

Das Component-List-Stylesheet ist als Pull-Stylesheet[37] implementiert. Bei dieser Art von Stylesheets gibt es nur ein Template. Das Template setzt am Wurzelknoten all der Informationen an, die man haben möchte.

Über das XSLT-Element `<xsl:for-each>` werden alle Informationen gesucht und herausgezogen (Engl.: to pull). `<xsl:for-each>` stellt eine Schleifenanweisung innerhalb einer Template-Definition dar. Alle aufeinanderfolgenden Knoten innerhalb eines Knotensets werden der Reihe nach abgearbeitet. So lassen sich beispielsweise alle Elemente `Karte` oder alle Elemente `PiggyBack` direkt innerhalb der übergeordneteten Template-Definition

[37] Das Prinzip, das Pull-Stylesheets zugrunde liegt, wird in Kapitel 3.11 »Darstellung des Konfigurationsergebnisses mit XSL« näher erläutert.

`Configuration` abarbeiten, ohne dass noch einmal die untergeordneten Template-Definitionen für die Elemente `Karte` oder `PiggyBack` aufgerufen werden müssen.

Pull-Stylesheets sind daher in der Regel kürzer und einfacher zu lesen, weil das Layout des Stylesheets dem gewünschten Layout der Ausgabe sehr ähnlich ist.

Bei der Codierung eines Pull-Stylesheets nimmt man einfach eine Vorlage der gewünschten Ausgabe und fügt sie in das übergeordnete Template ein. Dann ersetzt man alle Werte in der Vorlage mit dem entsprechenden Code für die Extraktion der Werte aus dem Quelldokument.

Das Component-List-Stylesheet sieht in seiner Struktur folgendermaßen aus. Der HTML-Code wurde der Übersichtlichkeit halber weggelassen, mit Ausnahme der Rahmen-Tags `<html></html>`, `<head></head>` und `<body></body>`.

```
<xsl:stylesheet ..... >

.....
<xsl:key name="piggyback_key" match="PiggyBack"
use="@BestellNr"/>
.....
<xsl:template match="Configuration">
.....
<html>
<head/>
<body style=" ..... >

.....
<xsl:value-of select="@Date"/>
<xsl:value-of select="@Time"/>

.....
<!-- Racks -->
<xsl:for-each select="Rack">
.....
</xsl:for-each>
.....
```

```
.....
<!-- Chassis -->
<xsl:for-each select="Rack/Gehaeuse">
.....
</xsl:for-each>
.....
<!-- Boards -->
<xsl:for-each select="Rack/Gehaeuse/Karte">
.....
</xsl:for-each>
.....
<!-- Piggy Backs -->
<xsl:for-each select="Rack/Gehaeuse/Karte/PiggyBack .....
.....
</xsl:for-each>
.....

<!-- Gesamtsummen ausgeben -->
.....
<xsl:call-template name="rechne_total"> .....
<xsl:call-template name="rechne_tax"> .....
<xsl:call-template name="rechne_total_incl_tax"> .....
.....

</body>
</html>
</xsl:template>
.....

<xsl:template match="LabCarKatalog"> .....
.....
<xsl:template name="rechne_total"> .....
<xsl:template name="rechne_tax"> .....
<xsl:template name="rechne_total_incl_tax"> .....
.....

</xsl:stylesheet>
```

Der Bereich für die Karten, also das was in der oben dargestellten Struktur des Component-List-Stylesheets nach dem Kommentar `<!-- Boards -->` folgt, ist folgendermaßen aufgebaut:

```
<!-- Boards -->
.......
<xsl:for-each select="Rack/Gehaeuse/Karte">
<xsl:sort select="@BestellNr"/>
.......
<xsl:variable name="this">
<xsl:value-of select="@BestellNr"/>
</xsl:variable>
.......
<xsl:apply-templates
select="document(/Configuration/LabCarComponentRessource/
@SourceDatei)//SysTeil[ @KeyBestellnr=$this] "/>
.......
<xsl:value-of select="format-number(@Preis ,
'######0.00')"/>.--
.......
</xsl:for-each>
.......
```

Bei der Ausgabe der `Boards` wird eine Sortierung vorgenommen. Die Sortierung erfolgt über `<xsl:sort select="@BestellNr"/>`. Dabei wird nach Reihenfolge der Knoteninhalte sortiert. Über das Anweisungsattribut `select` wird das XML-Attribut `BestellNr` ausgewählt. Die Sortierung erfolgt über das XML-Attribut `BestellNr`.

Über `<xsl:variable name="this">` wird die Variable `this` definiert, welche die aktuelle `BestellNr` enthält.

Über `document(/Configuration/LabCarComponentRessource/@SourceDatei)//SysTeil[@KeyBestellnr=$this]` wird die Verbindung zur entsprechenden Bestellnummer in `LabCarKatalog.xml` hergestellt. Damit können dann die gewünschten Knoteninhalte aus dem LabCar-Katalog dargestellt werden. Dabei wird wieder die bereits im Stylesheet »Tree View« beschrie-

bene XPath-Funktion `document()` verwendet. Über die XPath-Syntax `[@KeyBestellnr=$this]` wird die aktuelle Bestellnummer, die in der Variablen `this` gespeichert ist, mit den Bestellnummern `KeyBestellnr` in `LabCarKatalog.xml` verglichen. Ist dieselbe Bestellnummer gefunden, wird die Verbindung zum entsprechenden Knoten in `LabCarKatalog.xml` hergestellt. Dadurch können die benötigten Informationen für das entprechende LabCar-Hardware-Komponententeil aus dem LabCar-Katalog in den Ergebnisbaum eingefügt werden.

Der Bereich für die Piggy-Backs ist wie folgt aufgebaut:

```
<!-- Piggy Backs -->
.....
<xsl:for-each select="Rack/Gehaeuse/Karte/PiggyBack
[count(. | key('piggyback_key',@BestellNr)[1]) = 1]">
<xsl:sort select="@BestellNr"/>
.....
<xsl:variable name="this">
<xsl:value-of select="@BestellNr"/>
</xsl:variable>
.....
<xsl:variable name="anzahl">
<xsl:value-of select="count(key('piggyback_key',@BestellNr))"/>
</xsl:variable>
.....
<xsl:value-of select="$anzahl"/> * .....
<xsl:value-of select="@BestellNr"/>
.....
<xsl:apply-templates
select="document(/Configuration/LabCarComponentRessource/
@SourceDatei)//SysTeil[ @KeyBestellnr= $this]"/>
.....
<xsl:value-of
select="format-number(@Preis , '######0.00')"/>.--
.....
<xsl:value-of
select="format-number($anzahl*@Preis , '######0.00')"/>.--
.....
</xsl:for-each>
.....
```

Bei der Ausgabe der Piggy-Backs wird zusätzlich zur Sortierung noch eine Gruppierung und Zusammenfassung vorgenommen.

Zu Beginn des Stylsheets »Component List« wird über das XSLT-Element `xsl:key` mit der Anweisung `<xsl:key name="piggyback_key" match="PiggyBack" use="@BestellNr" />` ein Zugriffsschlüssel definiert. Der Schlüssel hat den Namen `piggyback_key`. Er setzt am Knoten-Set `Piggy Back` an und greift dort auf das Attribut `BestellNr` zu. Damit kann über diesen Zugriffsschlüssel auf alle Piggy-Backs mit der gleichen Bestellnummer zugegriffen werden.

Mit `<xsl:for-each select="Rack/Gehaeuse/Karte/PiggyBack [count(. | key('piggyback_key',@BestellNr)[1]) = 1]">` werden alle ersten Vertreter der vorhandenen Piggy-Back-Sorten ausgewählt. Mit [*Bedingung*] wird die Knotenmenge eingeschränkt. Die *Bedingung* lautet: `count(. | key('piggyback_key',@BestellNr)[1]) = 1`. Mit dieser *Bedingung* wird getestet, ob die aus den zu vergleichenden Knoten erzeugte Menge genau ein Element enthält. Ist nur ein Knoten enthalten, handelt es sich um ein und denselben Knoten, nämlich in diesem Fall den ersten Knoten.

Über die XPath-Funktion `count()` wird ermittelt, wie viele Knoten auf der Ebene unterhalb eines Knotensets enthalten sind. Als Argument erwartet `count()` ein Knotenset. Der Punkt `.` steht für »self-node«, und damit für den aktuellen Knoten selbst. Der senkrechte Strich `|` vereinigt Knotenmengen. Es handelt sich hierbei um eine echte Mengenvereinigung. Die XPath-Funktion `key()`wählt den ersten Knoten aus, der für einen benannten Schlüssel `piggyback_key` denselben Wert wie die `BestellNr` des aktuellen Knotens besitzt.

Weitere Erläuterungen und Beispiele zum Thema »Gruppieren mit nur einem Vertreter« finden sich im Skript »XML-Halbkurs« von Oliver Becker (2003) im Kapitel »Gruppieren (IV)«.

Mit `<xsl:sort>` lassen sich die einzelnen Gruppenvertreter in jede gewünschte Reihenfolge bringen.

`<xsl:variable name="anzahl">` definiert die Variable `anzahl`. Über `count (key('piggyback_key',@BestellNr))` wird gezählt, wie oft das jeweilige Piggy-Back auftritt. Dabei wird wiederum der Zugriffsschlüssel `piggyback _key` benutzt. Der für `anzahl` errechnete Wert wird dann über `<xsl:value-`

of select="$anzahl"/> ausgegeben. Mit $anzahl*@Preis wird die Zwischensumme für das aktuelle Piggy-Back berechnet.

Mit der XPath-Funktion document() werden über die Variable this Knoteninhalte aus dem LabCar-Katalog LabCarKatalog.xml in die Ausgabe eingebunden.

Für die Darstellung der Werte aus dem LabCar-Katalog werden im Component-List-Stylesheet die entsprechenden Templates für die jeweiligen Knoten in LabCarKatalog.xml definiert.

```
<xsl:template match="LabCarKatalog">
.....
<xsl:apply-templates/>
.....
</xsl:template>
.....
<xsl:template match="SysTeil">
<xsl:apply-templates/>
</xsl:template>
.....
<xsl:template match="SysTeilBez">
<xsl:value-of select="."/>
</xsl:template>
.....
<xsl:template match="Vertriebsinfo">
<xsl:value-of select="Vertriebstext"/>
</xsl:template>
.....
```

Für die Berechnung der Summen und der Mehrwertsteuer werden innerhalb des Stylesheets »Component-List« eigene Templates definiert.

Diese Templates werden dann mit <xsl:call-template name="...> über ihren entsprechenden Namen aufgerufen. Als Wert wird mit <xsl:with-param name="liste" select="...> eine Werte-Liste in Form eines Knotensets übergeben, über das die Berechnung laufen soll.

```
.....
<xsl:call-template name="rechne_total">
<xsl:with-param name="liste"
select="/descendant-or-self::node()/*[attribute::Preis]"/>.--
</xsl:call-template>
.....
<xsl:call-template name="rechne_tax">
<xsl:with-param name="liste"
select="/descendant-or-self::node()/*[attribute::Preis]"/>.--
</xsl:call-template>
.....
<xsl:call-template name="rechne_total_incl_tax">
<xsl:with-param name="liste"
select="/descendant-or-self::node()/*[attribute::Preis]"/>.--
</xsl:call-template>
.....
```

Mit `select="/descendant-or-self::node()/*[attribute::Preis]"/>` werden der Inhalt des Attributs `Preis` des aktuellen Knotens und alle Attribute `Preis` der untergeordneten Knoten adressiert. Die Auswahl aller Attribute `Preis` wird durch das Wildcard-Zeichen[38] `/*` erreicht.

Die Berechnungen `rechne_total`, `rechne_tax` und `rechne_total_incl_tax` sind als rekursive Funktionen implementiert.

Rekursion bedeutet Selbstbezüglichkeit (Lat.: recurrere = zurücklaufen). Sie tritt immer dann auf, wenn etwas auf sich selbst verweist.

Als Rekursion bezeichnet man den Aufruf oder die Definition einer Funktion durch sich selbst. Ohne geeignete Abbruchbedingung geraten solche rückbezüglichen Aufrufe leicht in einen so genannten infiniten Regress, was bedeutet, dass das Programm nicht mehr reagiert und auch nicht beendet wird. (vgl. WIKIPEDIA 2004, Stichwort: »Rekursion«)

Die Definition von rekursiv festgelegten Funktionen ist eine grundsätzliche Vorgehensweise in der funktionalen Programmierung. Ausgehend von gegebenen Funktionen (beispielsweise der Summen-Funktion) werden neue

[38] Das englische Wort »Wildcard« steht für Platzhalter.

Funktionen definiert, mit Hilfe derer weitere Funktionen definiert werden können.

```
<!-- Rekursion total -->
.....
<xsl:template name="rechne_total">
<xsl:param name="liste"/>
<xsl:choose>
<xsl:when test="$liste">
<xsl:variable name="summe">
<xsl:call-template name="rechne_total">
<xsl:with-param name="liste" select="$liste[position()!=1]"/>
</xsl:call-template>
</xsl:variable>
<xsl:value-of select="format-number($summe + $liste/@Preis ,
'#######0.00')"/>
</xsl:when>
<xsl:otherwise>0</xsl:otherwise>
</xsl:choose>
</xsl:template>
.....
```

Mit `<xsl:param name="liste"/>` wird `liste` als lokale Variable innerhalb des Templates für die Berechnung definiert. Beim Aufruf des Templates mit `<xsl:call-template name="rechne_total">` wird über `<xsl:with-param name="liste" select=.../>` eine Knotenmenge mit den Preisen übergeben.

`<xsl:choose>` bildet den Rahmen für eine Reihe von Abfragen. `<xsl:when>` definiert innerhalb von `<xsl:choose>` eine Bedingung. Mit `<xsl:when test = "$liste">` wird über das Attribut `test` abgefragt, ob `liste` Werte enthält. Enthält `liste` Werte, dann wird eine rekursive Aufsummierung vorgenommen. Hierbei wird dasselbe Template `rechne_total` erneut über `<xsl:call-template name="rechne_total">` aufgerufen. Mit `<xsl:with-param name="liste" select="$liste[position()!=1]"/>` werden alle Knoten der Knotemenge, bis auf den ersten Knoten ausgewählt. Die X-Path-Funktion `position()`ermittelt der wievielte Knoten ein aktueller Knoten in einem Knotenset ist. Alle Attribute `Preis` der ausgewählten Knotenmenge werden aufsummiert, vom ersten bis zum letzten. Das Summenergebnis

wird über <xsl:variable name="summe"> der Variablen summe zugewiesen.

Über <xsl:value-of select="format-number($summe + $liste/@Preis, '#######0.00')"/> wird jeweils der aktuelle Preis im Knotenset mit dem aktuellen Wert der Variablen summe aufsummiert. Dies geschieht über $summe + $liste/@Preis. Über die X-Path-Funktion format-number() wird die Ausgabe dann als zweistellige Kommazahl formatiert.

Für den Fall, dass beim Aufruf des Templates kein Wert für den Preis übergeben wird, wird <xsl:otherwise>0</xsl:otherwise> durchgeführt. Dies bedeutet, dass als Rückgabewert 0 übergeben wird.

Die Berechnungen rechne_tax und rechne_total_incl_tax unterscheiden sich lediglich in der Berechnung vom Template rechne_total.

```
<!-- Rekursion tax -->
.......
<xsl:template name="rechne_tax"> .......
<xsl:value-of select="format-number($summe + $liste/@Preis *
0.16 , '#######0.00')"/> .......
</xsl:template>
.......

<!-- Rekursion total_incl_tax -->
.......
<xsl:template name="rechne_total_incl_tax"> .......
<xsl:value-of select="format-number($summe + $liste/@Preis *
1.16 ,'#######0.00')"/> .......
</xsl:template>
.......
```

6.3 Dynamischer Prototyp

Beim dynamischen Prototyp werden Benutzereingaben berücksichtigt und entsprechend verarbeitet. Der dynamische Prototyp enthält außer der Datei `help.htm` keine statischen HTML-Dateien mehr, sondern nur noch PHP-Dateien. Die PHP-Dateien enthalten PHP-Skripte, die abhängig von den Eingaben des Benutzers, serverseitig verarbeitet werden.

Mit dem dynamischen Prototyp steht eine Benutzungsoberfläche zur Verfügung, welche die Eingaben des Benutzers serverseitig verarbeitet und welche auf die individuellen Eingaben des jeweiligen Benutzers reagiert. Außerdem gibt es beim dynamischen Prototyp für jeden einzelnen Benutzer ein individuelles Session Management.

6.3.1 Excel-Tabellen im CSV-Format

Im Bereich der ETAS GmbH steht kein Server zur Verfügung, auf dem Tabellen direkt über ein Tabellenkalkulationsprogramm verarbeitet werden können. Tabellenkalkulationsprogramme, wie zum Beispiel Excel, sind typischerweise Client-Anwendungen. Eine serverseitige Verarbeitung von Tabellen ist bei ETAS derzeit nicht möglich. Aus diesem Grund musste für Verarbeitung von Daten aus Tabellen eine eigene Lösung gefunden werden.

Die Lösung für die serverseitige Verarbeitung besteht in einem universell lesbaren und unabhängigen Format. Für den LabCar-Konfigurator wurde das CSV-Format (Engl.: Comma Separated Value - Abkürzung: CSV) ausgewählt. »Comma-Separated« bedeutet, dass die einzelnen Werte einer Tabellenzelle jeweils durch ein Semikolon voneinander getrennt werden. Das Ende einer Tabellenzeile wird durch ein Zeilenumbruchzeichen markiert. In Excel besteht die Möglichkeit, eine Excel-Tabelle direkt im CSV-Format abzuspeichern.

Die Verarbeitung von Tabellen im CSV-Format wird von PHP unterstützt. In PHP sind bereits Funktionen für die Verarbeitung von CSV-Dateien vorhanden. Der LabCar-Konfigurator verwendet die PHP-Funktion `fgetcsv()`. Diese Funktion liest Daten aus einer CSV-Datei in ein PHP-Array[39] ein.

[39] Das englische Wort »Array« steht für Datenfeld.

6.3.2 Verzeichnisstruktur auf dem Web-Server

Die Verzeichnisstruktur auf dem Web-Server ist folgendermaßen aufgebaut:

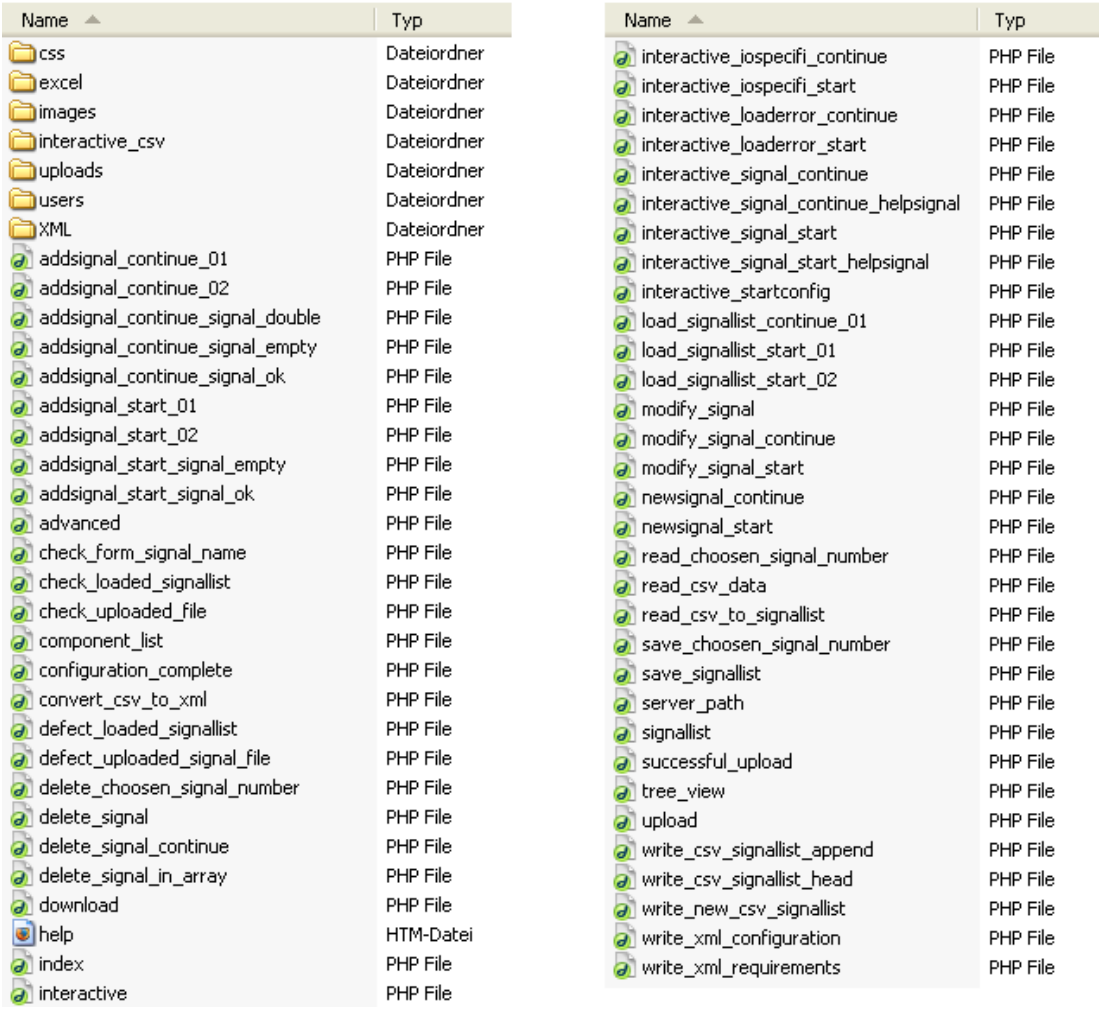

Abbildung 43: Verzeichnisstruktur auf dem Webserver

Im Unterverzeichnis `css` liegt die zentrale HTML-Stylessheet-Datei `styles.css`. Im Unterverzeichnis `excel` liegt die Excel-Tabellenvorlage `signal_template.xls`. Im Verzeichnis `images` liegen die Bilder für die Gestaltung der Benutzungsoberfläche.

Im Verzeichnis `interactive_csv` liegen die aktuellen Signallisten, die von Benutzern über die interaktive Eingabe erstellt wurden beziehungsweise aktuell erstellt werden. Die Dateien sind als `interactive_signallist_`

xxx.csv benannt. Der Dateina- me enthält den jeweiligen Benutzerschlüssel[40] xxx, in Form einer dreistelligen Zahl. Es wird nur dann eine Datei interactive_signallist_xxx.csv angelegt, wenn der Benutzer mit dem entsprechenden Benutzerschlüssel auch den Weg der Eingabe »Interactive« benutzt hat.

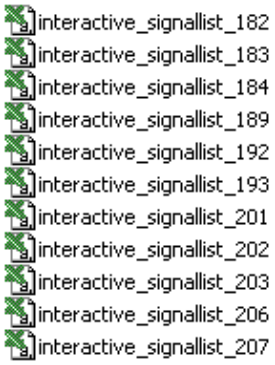

Abbildung 44: Ausschnitt aus dem Verzeichnis interactive_csv

Das Verzeichnis uploads sieht ähnlich aus. Hier befinden sich die CSV-Dateien, die Benutzer über die Eingabemöglichkeit »Advanced« in das System geladen haben. Die Dateien sind als uploaded_signal_xxx.csv benannt, wobei wiederum der zugehörige Benutzerschlüssel xxx im Dateiname integriert ist.

Im Verzeichnis users befindet sich die Datei UserId.txt. In dieser Datei ist der nächste freie Benutzerschlüssel eingetragen. Meldet sich ein neuer Benutzer am Server an, wird der Benutzerschlüssel aus UserId.txt gelesen, danach wird der Benutzerschlüssel um eins hochgezählt und in die Datei UserId.txt geschrieben. Außerdem befinden sich im Verzeichnis users für jeden aktuellen Benutzer eine Datei ChoosenSignalNumber_xxx.txt. Hier wird die Nummer des Signals vermerkt, das der Benutzer mit dem entsprechenden Benutzerschlüssel xxx aktuell aus der interaktiven Signalliste bei »Interactive« im Status-Bereich ausgewählt hat.

[40] Um für jeden einzelnen Benutzer eine Sitzung (Engl.: Session) verwalten zu können, ist die Vergabe eines Benutzerschlüssels notwendig. Siehe hierzu auch Kapitel 3.6 »Session-Management«. In diesem Kapitel ist die Vergabe des Benutzerschlüssels in Form eines Flussdiagramms dargestellt.

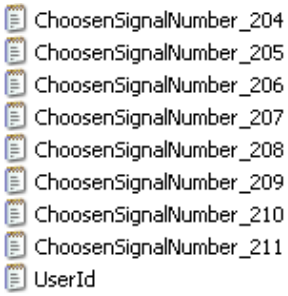

Abbildung 45: Ausschnitt aus dem Verzeichnis `users`

Das Verzeichnis `XML` besitzt eigene Unterverzeichnisse und hat folgende Struktur:

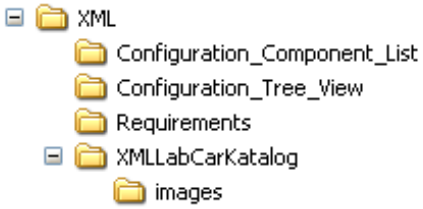

Abbildung 46: Ausschnitt aus dem Verzeichnis `XML`

Im Verzeichnis `Requirements` sind die aktuellen XML-Dateien `requirements_xxx.xml` für die Anforderungen der jeweiligen Benutzer, die sie in Form von Signallisten definiert haben, enthalten. Der Dateiname jeder XML-Datei »Requirements« enthält auch den zugehörigen Benutzerschlüssel `xxx`. Im Verzeichnis `Requirements` sind außerdem die Dateien für die Dokumenttyp-Definition `Requirements.dtd` und `Signal.dtd` abgelegt.

Im Verzeichnis `Configuration_Component_List` sind die aktuellen XML-Konfigurationsergebnisdateien `configuration_xxx.xml` abgelegt. Der Dateiname jeder XML-Konfigurationsergebnisdatei enthält ebenfalls den zugehörigen Benutzerschlüssel `xxx`. Im Verzeichnis `Configuration_Component_List` befindet sich außerdem die Datei `Configuration.dtd` und das XML-Style- sheet `Configuration.xsl` für die Darstellung der Bestellliste »Component List«.

Im Verzeichnis `Configuration_Tree_View` befinden sich dieselben aktuellen XML-Konfigurationsergebnisdateien `configuration_xxx.xml` noch einmal. Ebenso die Datei `Configuration.dtd` für die Dokumenttyp-Definitionen. Die Datei `Configuration.xsl` enthält hier jedoch das XML-Stylesheet für die Darstellung der Aufbaustruktur des konfigurierten LabCars - »Tree View«. Aus programmtechnischen Gründen erwies sich die Lösung der doppelten Datenhaltung hier als die sinnvollste und einfachste Lösung. Die XML-Dateien haben einen sehr geringeren Speicherbedarf. Durch die doppelte Datenhaltung können die beiden Darstellungsmöglichkeiten »Component List« und »Tree View« schnell und umkompliziert an den anfragenden Web-Client geschickt werden.

Im Verzeichnis `LabCarKatalog` ist die Datei `LabCarKatalog.xml` abgelegt. Sie beinhaltet den kompletten LabCar-Katalog mit den aktuellen LabCar-Hardware-Komponenten, den Komponentenschlüsseln, den Komponentenbeschreibungen und jeweiligen Preisen. Außerdem befindet sich hier wiederum die zugehörige Dokumenttyp-Definition `LabCarKatalog.dtd`. In einem Unterverzeichnis `images` befinden sich die Thumbnails (kleine Bilder) für die einzelnen LabCar-Hardware-Komponenten. Die Bildernamen enthalten dabei den jeweiligen Komponentenschlüssel. Dadurch können die Bilder den entsprechenden LabCar-Hardware-Komponenten zugeordnet werden.

6.3.3 PHP-Dateien und ihre Funktionen

In der nachfolgenden Tabelle werden die einzelnen PHP-Dateien und die darin enthaltenen Skriptfunktionen kurz beschrieben. Manche PHP- Skripte sind direkt in den HTML-Darstellungscode eingebettet. Andere PHP-Dateien inkludieren PHP-Dateien, welche wiederum HTML-Code enthalten, und manche PHP-Dateien beinhalten ausschließlich Funktionalität in Form von PHP-Code.

Die Bezeichnung »..._`start`_...« in den Dateinamen steht dabei für die Einstiegsebene (1. Ebene) im Bereich »Interactive«. Auf dieser Ebene wird noch keine Signalliste im Status-Bereich angezeigt.

Die Bezeichnung »..._`continue`_...« in den Dateinamen steht für die Fortsetzungsebene (2. Ebene) im Bereich »Interactive«. Auf dieser Ebene gibt es dann eine Signalliste, die im Status-Bereich angezeigt wird.

PHP-Datei	Funktionalität
index.php	Vergabe eines Wertes für die Variable user_id. Aus der Datei UserId.txt im Verzeichnis users wird die nächste freie user_id eingelesen. Danach wird der Wert in der Datei UserId.txt um 1 erhöht. Bei einem Wert von 999 wird die user_id auf 1 zurückgesetzt. Die Variable choosen_signal_number wird mit dem Wert -1 belegt, das bedeutet, es wurde noch kein Signal aus der Signalliste ausgewählt.
interactive.php	Definition der Variablenwerte für die HTML-Formulare, Wertzuweisung für die HTML-Formularvariablen und Weitergabe der user_id.
interactive_signal_start.php	Weitergabe der HTML-Formularvariablenwerte und der user_id.
interactive_signal_start_ helpsignal.php	Weitergabe der HTML-Formularvariablenwerte und der user_id.
interactive_signal_continue.php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
interactive_signal_continue_ helpsignal.php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
interactive_iospecifi_start.php	Weitergabe der HTML-Formularvariablenwerte und der user_id.
interactive_iospecifi_continue. php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
interactive_loaderror_start.php	Weitergabe der HTML-Formularvariablenwerte und der user_id.
interactive_loaderror_continue. php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
addsignal_start_01.php	Prüft, ob ein Signalschlüssel eingegeben wurde. Ist kein Signalsschlüssel vorhanden wird addsignal_start_signal_empty.php inkludiert. Ist ein korrekter Signalschlüssel vorhanden, wird addsignal_start_signal_ok.php inkludiert.

PHP-Datei	Funktionalität
addsignal_start_signal_empty.php	Fehlermeldung über fehlenden Signalschlüssel und Weitergabe der HTML-Formularvariablenwerte und der user_id.
addsignal_start_signal_ok.php	Weitergabe der HTML-Formularvariablenwerte und der user_id und Aufruf der Funktion write_csv_signallist_head.php. Anforderung einer Benutzerbestätigung.
addsignal_start_02.php	Weitergabe der HTML-Formularvariablenwerte und der user_id und Aufruf der Funktion write_csv_signallist_append.php. Zusätzlich wird user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
addsignal_continue_01.php	Prüft, ob der Benutzer einen Signalschlüssel eingegeben hat. check_form_signal_name.php wird inkludiert. Prüft den Signalschlüssel auf Eindeutigkeit und erzeugt eine Kontrollvariable. Wurde kein Signalschlüssel eingegeben, wird addsignal_continue_signal_empty .php inkludiert. Ist der eingegebene Signalschlüssel nicht eindeutig wird addsignal_continue_signal_double .php inkludiert. Ist ein eindeutiger Signalschlüssel vorhanden, wird addsignal_continue_signal_ok.php inkludiert.
addsignal_continue_signal_empty .php	Eine Fehlermeldung über fehlenden Signalschlüssel wird ausgegeben. Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
addsignal_continue_signal_double .php	Eine Fehlermeldung über nicht eindeutigen, bereits vorhandenen Signalschlüssel wird ausgegeben. Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
addsignal_continue_signal_ok.php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben. Eine Benutzerbestätigung wird angefordert.

PHP-Datei	Funktionalität
addsignal_continue_02.php	Weitergabe der HTML-Formularvariablenwerte und der user_id und Aufruf der Funktion write_csv_signallist_append.php. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
check_form_signal_name.php	Prüft, ob der eingegebene Signalschlüssel eindeutig ist und belegt eine Kontrollvariable mit dem Wert true oder false.
write_csv_signallist_head.php	Die Datei interactive_signallist_xxx.csv im Verzeichnis interactive_csv erzeugen und den Tabellenkopf in die Datei schreiben. xxx steht für die zugehörige user_id, die damit im Dateiname integriert ist.
write_csv_signallist_append.php	Aktuelles Signal in die zugehörige Datei interactive_signallist_xxx.csv schreiben. xxx steht für die zugehörige user_id, die damit im Dateiname integriert ist.
newsignal_start.php	Neue Vorbelegung der HTML-Formularvariablenwerte und Weitergabe der user_id.
newsignal_continue.php	Neue Vorbelegung der HTML-Formularvariablenwerte und Weitergabe der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
modify_signal_start.php	Weitergabe der HTML-Formularvariablenwerte und der user_id.
modify_signal_continue.php	Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.

PHP-Datei	Funktionalität
modify_signal.php	Das ausgewählte Signal wird aus der Signalliste im Statusbereich gelöscht. modify_signal.php inkludiert die PHP-Dateien read_csv_data.php und read_choosen_signal_number.php. Werte des ausgewählten Signals werden in die HTML-Formularvariablenwerte eingelesen und an die nächste Seite übergeben. Über write_new_csv_signallist.php wird eine Signalliste im CSV-Format geschrieben und über delete_choosen_signal_number.php wird die ausgewählte Signalnummer gelöscht. Wenn eine Signalliste mit mindestens einem Signal vorhanden ist, wird modify_signal_continue.php inkludiert, ansonsten wird modify_signal_start.php inkludiert.
load_signallist_start_01.php	Hier kann eine Signalliste im CSV-Format hochgeladen werden. Die Upload-Schaltfläche leitet weiter zu check_loaded_signallist.php. Neue Vorbelegung HTML-Formularvariablenwerte und Weitergabe der user_id.
load_signallist_start_02.php	Neue Vorbelegung der HTML-Formularvariablenwerte und Weitergabe der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
load_signallist_continue_01.php	Neue Vorbelegung der HTML-Formularvariablenwerte und Weitergabe der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
defect_loaded_signallist.php	Ausgabe einer Meldung, dass die hochgeladene Signalliste nicht korrekt war. Neue Vorbelegung der HTML-Formularvariablenwerte und Weitergabe der user_id.

PHP-Datei	Funktionalität
check_loaded_signallist.php	Überprüft, ob die hochgeladene Signalliste den korrekten Tabellenkopf enthält. Ist der Tabellenkopf nicht korrekt, wird defect_loaded_signallist.php inkludiert. Ist der Tabellenkopf korrekt, wird load_signallist_start_02.php inkludiert. Hier ist prototypisch nur die Kontrolle auf einen korrekten Tabellenkopf implementiert. An dieser Stelle sollen später auch die eingetragenen Signaldaten auf Plausibilität geprüft werden.
save_signallist.php	Ermöglicht dem Benutzer, die aktuelle Signalliste lokal auf seinem Computer zu speichern. Weitergabe der HTML-Formularvariablenwerte und der user_id. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.
signallist.php	Sofern es eine Signalliste mit mindestens einem Signal gibt, wird die Datei signallist.php im Inline-Frame im Statusbereich referenziert. Bei der Referenzierung wird jedesmal die user_id und die choosen_signal_number übergeben. signallist.php inkludiert save_choosen_signal_number.php und read_csv_to_signallist.php. Der HTML-Code für die Ausgabe aller Signale in einer Signallliste wird dann über eine FOR-Schleife erzeugt.
read_csv_to_signallist.php.	Liest die Signale aus der zugehörigen CSV-Datei für signallist.php ein.
read_choosen_signal_number.php	Liest die aktuell ausgewählte Signalnummer aus der zugehörigen Datei ChoosenSignalNumber_xxx.txt.

PHP-Datei	Funktionalität
save_choosen_signal_number.php	Schreibt die aktuell ausgewählte Signalnummer in die zugehörige Datei ChoosenSignalNumber_xxx.txt. xxx steht für die zugehörige user_id, die damit im Dateiname integriert ist.
delete_choosen_signal_number.php	Löscht die aktuell ausgewählte Signalnummer aus der zugehörigen Datei ChoosenSignalNumber_xxx.txt.
read_csv_data.php	Liest die Signaldaten aus der CSV-Datei in ein PHP-Array.
delete_signal_in_array.php	Das zu löschende, aktuelle Signal wird aus dem PHP-Array gelöscht.
write_new_csv_signallist.php	Erstellt eine neue Signalliste im CSV-Format. Die aktuellen Signaldaten werden dabei aus einem PHP-Array übernommen.
delete_signal.php	Löscht das ausgewählte Signal aus der Signalliste. Hierfür werden read_choosen_signal_number.php, delete_signal_in_array.php, write_new_csv_signallist.php und delete_choosen_signal_number.php inkludiert. Die Variablenwerte des zu löschenden Signals werden in den HTML-Formularvariablen zwischengespeichert und es wird eine Löschbestätigung des Benutzers angefordert. Bei Eingabe einer Löschbestätigung durch den Benutzer wird zur Datei delete_signal_continue.php weitergeleitet. Zusätzlich wird die user_id und die choosen_signal_number an das Inline-Frame im Statusbereich übergeben.

PHP-Datei	Funktionalität
delete_signal_continue.php	Inkludiert werden read_csv_data.php, read_choosen_signal_number.php, delete_signal_in_array.php, write_new_csv_signallist.php und delete_choosen_signal_number.php. Damit werden die Variablenwerte des zu löschenden Signals aus der Datei interactive_signallist_xxx.csv endgültig gelöscht. Wenn noch mindestens ein Signal vorhanden ist, wird newsignal_continue.php inkludiert, ansonsten wird newsignal_start.php inkludiert. Weitergabe der user_id.
advanced.php	Weitergabe der user_id.
download.php	Erzeugt einen HTML-Link, über welchen die Excel-Tabellenvorlage, die auf dem Server liegt, clientseitig geöffnet gespeichert werden kann. Weitergabe der Variablen user_id.
upload.php	Hier kann eine Excel-Tabelle im CSV-Format hochgeladen werden. Startet der Benutzer den Upload, wird die Datei check_uploaded_file.php aufgerufen. Weitergabe der Variablen user_id.
successful_upload.php	Rückmeldung an den Benutzer, dass die Signalliste erfolgreich hochgeladen wurde und die Konfiguration jetzt gestartet werden kann. Weitergabe der user_id.
defect_uploaded_signal_file.php	Rückmeldung an den Benutzer, dass die hochgeladene Signalliste nicht korrekt ist. Weitergabe der user_id.
convert_csv_to_xml.php	Inkludiert die beiden Dateien read_csv_data.php und write_xml_requirements.php.

PHP-Datei	Funktionalität
check_uploaded_file.php	Speichert die hochgeladene Datei auf dem Server im Verzeichnis uploads unter dem Namen uploaded_signallist_xxx.csv. Für xxx wird wiederum die zugehörige user_id eingetragen. Prüft, ob der Tabellenkopf mit der Excel-Tabellenvorlage übereinstimmt und ruft dann convert_csv_to_xml.php auf. Hier ist prototypisch wiederum nur die Kontrolle für einen korrekten Tabellenkopf implementiert. An dieser Stelle sollen später auch die eingetragenen Signaldaten auf Plausibilität geprüft werden. Wenn die hochgeladene CSV-Datei korrekt war wird successful_upload.php inkludiert. War die hochgeladene CSV-Datei nicht korrekt wird defect_uploaded_signal_file.php inkludiert. Außerdem wird write_xml_configuration.php inkludiert.
write_xml_requirements.php	Inkludiert die Datei read_csv_data.php. Liest die Signaldaten aus der CSV-Datei und schreibt die Signaldaten in eine XML-Datei. Diese XML-Datei wird dann im XML-Unterverzeichnis Requirements als requierements_xxx.xml gespeichert. Für xxx wird die entsprechende user_id eingetragen um die Datei dem entsprechenden Benutzer zuzuordnen.
interactive_startconfig.php	Liest aus der Datei interactive_signallist_xxx.csv die Signalliste. Erzeugt über convert_csv_to_xml.php die Datei requierements_xxx.xml. Beim vorliegenden Prototyp wird hier die PHP-Datei write_xml_configuration.php inkludiert, welche die Datei configuration_xxx.xml erzeugt. Weitergabe der user_id.

PHP-Datei	**Funktionalität**
`write_xml_configuration.php`	Schreibt zwei Konfigurationsergebnisdateien `configuration_xxx.xml`. Eine in das XML_Unterverzeichnis `Configuration_Tree_View` und dieselbe noch einmal in das XML-Unterverzeichnis `Configuration_Component_List`. `xxx` steht wiederum für die entsprechende `user_id`. Damit wird die Datei `configuration_xxx.xml` dem zugehörigen Benutzer zugeordnet. Beim vorliegenden Prototyp fehlt noch das zentrale Konfigurationsprogramm. Ersatzweise wird deshalb hier eine Konfigurationsergebnisdatei mit fest vorgegebenen LabCar-Komponententeilen geschrieben. Die jeweils ersten sieben benutzerdefinierten Signale werden dabei an fest vorgegebenen Stellen, in der Reihenfolge ihrer Eingabe, eingefügt.
`configuration_complete.php`	Weitergabe der `user_id`.
`tree_view.php`	Aufruf und Darstellung der zughörigen `configuration_xxx.xml` aus dem Verzeichnis `Configuration_Tree_View`. Die Zuordnung erfolgt über die `user_id`, die im Dateinamen über `xxx` integriert ist. Die Darstellung erfolgt über ein HTML-Inline-Frame `<iframe> ... </iframe>`.
`component_list.php`	Aufruf und Darstellung der zughörigen `configuration_xxx.xml` aus dem Verzeichnis `Configuration_Component_List`. Die Zuordnung erfolgt über die `user_id`, die im Dateinamen über `xxx` integriert ist. Die Darstellung erfolgt über ein HTML-Inline-Frame `<iframe> ... </iframe>`.

PHP-Datei	Funktionalität
`server_path.php`	Diese Datei enthält den Verzeichnispfad zum Serververzeichnis, in dem der LabCar-Konfigurator abgelegt ist. Diese Datei wird von all jenen PHP-Dateien inkludiert, in denen der Server-Verzeichnispfad benötigt wird.

6.3.4 Weitergabe von Variablenwerten

Sämtliche Benutzereingaben werden in versteckten Input-Feldern von einem Teilformular zum nächsten weitergegeben. In `interactive.php` werden zuerst alle versteckten Input-Felder mit Werten belegt. Die Belegung wird innerhalb eines HTML-Formulars vorgenommen. Für das Formular `<form method="post" name="...">` wird die Methode für die Weitergabe und ein Name definiert. `method="post"` besagt, dass die Werte an die folgende Seite mit der POST-Methode übergeben werden.

```
<form method="post" name="...">
.......
<input type="hidden" name="user_id" value=<?php echo
$user_id;?>>
<input type="hidden" name="choosen_signal_number" value=-1>
<input type="hidden" name="actual_signal_index" value="">
<input type="hidden" name="form_signal_visited" value=false>
<input type="hidden" name="form_iospecifi_visited" value=false>
<input type="hidden" name="form_loaderror_visited" value=false>
<input type="hidden" name="form_signal_name" value="">
<input type="hidden" name="form_signal_pin" value="">
<input type="hidden" name="form_signal_description" value="">
<input type="hidden" name="form_signal_type_selected"
value="IO">
.......
```

```
.......
<input type="hidden" name="form_io_resolution_selected"
value="analog">
.......
<input type="hidden" name="form_load_type_selected"
value="No_Load">
<input type="hidden" name="form_short_ground_checked"
value=false>
<input type="hidden" name="form_short_ubatt_checked"
value=false>
.......
</form>
```

Das Formular umfasst den Content-Bereich und die Navigationsschaltflächen im Navigationsbereich. Wird irgendeine Schaltfläche in diesem Bereich angeklickt, so werden die aktuellen Werte aller Formularvariablen über die POST-Methode automatisch an die nächste Seite weitergegeben. Dies betrifft auch die Werte aller versteckten Input-Felder `<input type= "hidden" name="..." value="...">`.

Versteckte Input-Felder kontrollieren beispielsweise den Besuch der einzelnen Teilformulare und haben die Funktion von Kontrollvariablen. Über diese Kontrollvariablen prüft das System, ob der Benutzer alle notwendigen Formularfelder ausgefüllt hat. Beim vorliegenden Prototyp ist nur diese einfache Form der Kontrolle implementiert. Diese kann bei der Weiterentwicklung des Prototyps noch ausgeweitet und verfeinert werden, indem beispielsweise alle eingegebenen Werte auf Plausibilität überprüft werden.

Die `user_id` wurde bereits in `index.php` vergeben. Deshalb wird sie hier direkt über den eingebetteten PHP-Code `<?php echo $user_id;?>` von der aufrufenden Seite übernommen.

Die Werte aus dem aufrufenden Formular stehen über die POST-Methode im autoglobalen Feld `$_POST[...]` zur Verfügung. Für die weiteren Seiten sieht die Weitergabe der HTML-Formularvaribalenwerte und der `user_id` folgendermaßen aus.

```
input type="hidden" name="user_id"
value=<?php echo $user_id;?>>
<input type="hidden" name="choosen_signal_number"
value=<?php echo $choosen_signal_number;?>>
<input type="hidden" name="actual_signal_index"
value="<?php echo $_POST["actual_signal_index"];?>">
<input type="hidden" name="form_signal_visited"
value="<?php echo $_POST["form_signal_visited"];?>">
.....
```

Werden die Werte in der nächsten Seite wiederum in ein Input-Feld übernommen, so funktioniert die Wertübergabe problemlos. Für den Signalname sieht dies beispielsweise so aus.

```
<input type="text" name="form_signal_name"
value="<?php echo $_POST["form_signal_name"];?>">
```

Für die Wertübergabe spielt es keine Rolle ob das Input-Feld vom Typ `hidden` oder `text` ist. Ändert sich allerdings das Feld vom Input-Feld `<input>` in einen Textarea-Bereich `<textarea>` oder in ein Auswahl-Feld `<select>`, so funktioniert die Wertübergabe nicht mehr ohne Weiteres. Hier wird Javascript für die Wertübergabe eingesetzt.

Bei den Ankreuzfeldern `<input type="checkbox" name="... >` im dritten Teilformular wird ebenfalls Javascript eingesetzt und mit PHP kombiniert.

Abhängig von den Werten, die der Benutzer eingegeben hat, können über PHP Bedingungen definiert werden oder über PHP eine Verarbeitung und Aufbereitung von Werten vorgenommen werden. Nachfolgend sind beispielhaft Fragmente aus PHP-Funktionen dargestellt.

6.3.5 interactive_signal_continue.php

Bei `interactive_signal_continue.php` wird geprüft, welcher Wert für den Signaltyp von der aufrufenden Seite übergeben wurde. Entsprechend wird

dieser Wert dann im HTML-Tag <select> über <option selected ...> vorausgewählt oder nicht vorausgewählt.

```
<select name="SelectSignalType" onChange="SaveSignalType()"
...>
<?php
if ($form_signal_type_selected == "IO") {
echo "<option selected value='IO'>IO</option>";
} else {
echo "<option value='IO'>IO</option>";
}
if ($form_signal_type_selected == "RS422") {
echo "<option selected value='RS422'>RS422</option>";
} else {
echo "<option value='RS422'>RS422</option>";
}
.....
?>
</select>
```

Wird der Signaltyp durch den Benutzer verändert wird über das HTML-Attribut onChange automatisch die Javascript-Funktion SaveSignalType() aufgerufen.

6.3.6 delete_signal_continue.php

Nachdem das aktuelle Signal gelöscht wurde, prüft delete_signal_continue.php, ob noch mehr als ein Signal und damit auch eine Signalliste vorhanden ist.

```
if ( $number_of_signals > 1 ) {
include 'newsignal_continue.php';
}else{
include 'newsignal_start.php';
}
```

Entsprechend wird dann newsignal_continue.php oder newsignal_start.php aufgerufen.

6.3.7 addsignal_continue_01.php

Bei addsignal_continue_01.php sind die Bedingungen für die Ausgabe der Fehlermeldungen folgendermaßen implementiert.

```
<?php
include 'check_form_signal_name.php';
if ($form_signal_name == '') {
include 'addsignal_continue_signal_empty.php';
}
if ($signal_name_is_double) {
include 'addsignal_continue_signal_double.php';
}
if ($signal_name_is_double == false and $form_signal_name != '')
{
include 'addsignal_continue_signal_ok.php';
}
?>
```

6.3.8 signallist.php

In der Datei signallist.php werden die Signale aus der Datei interactive _signallist_xxx.csv in ein PHP-Array eingelesen. Außerdem wird gezählt, wieviele Signale die Signalliste enthält. Über eine FOR-Schleife werden die Signale dann der Reihe nach ausgegeben. Dabei wird auf das PHP-Array zugegriffen, in welches die Signaldaten zuvor eingelesen wurden.

```
<?php
for ($i=1 ; $i <= $number_of_signals ; $i++) {
// Signal ausgeben
echo ...
echo '<input ...
if ($i == $choosen_signal_number) {echo ' checked ';}
echo ...
echo $signal_name[$i];
echo ...
echo $signal_pin[$i];
echo
echo $signal_description[$i];
echo ...
echo $signal_type[$i];
echo ...
}
?>
```

6.3.9 Einbinden der Signalliste

Die Einbindung der Signalliste im Status-Bereich erfolgt über ein HTML-Inline-Frame `<iframe src="...>`. Dabei werden die `user_id` und die `choosen_signal_number` übergeben.

```
<iframe src="signallist.php?
user_id=<?php echo $user_id;?>&
choosen_signal_number=<?php echo $choosen_signal_number;?>
... >
```

6.3.10 Schreiben der XML-Dateien

Zum Zeitpunkt der Implementierung des Prototyps befanden sich viele PHP-Funktionen und PHP-Klassen, die XML verarbeiten, noch im Entwicklungsstadium. Im PHP-Handbuch (The PHP Group 2004, Stichwort:

»PHP Handbuch«) waren Warnungen vermerkt, dass sich diese Klassen und Funktionen noch in einem experimentellen Stadium befinden, und dass sie sich jederzeit noch verändern können.

Erst seit PHP Version 4 stehen einige hilfreiche PHP-Funktionen für das Erzeugen von XML-Dateien mit Validierung gegen die DTD zur Verfügung. Hierfür ist es jedoch notwendig, auf dem Web-Server zusätzlich entsprechende Programm-Bibliotheken zu installieren.

Zum Zeitpunkt der Implementierung des Prototyps war auf dem Intranet-Server von ETAS lediglich PHP Version 3.4 installiert. Eine Installation von zusätzlichen Bibliotheken war zu diesem Zeitpunkt nicht möglich. Aus diesem Grund wurde die Ausgabe der XML-Datei in ganz einfacher Form über die direkte Ausgabe einer Zeichenkette umgesetzt. Bei der Weiterentwicklung des Prototyp kann hier später eine Automatisierung bei der Erzeugung der XML-Dateien über die entsprechenden PHP-Funktionen und PHP-Klassen erreicht werden. Automatisches Parsen und automatisierte Validierung gegen die DTD sind hier denkbare Erweiterungsmöglichkeiten.

6.3.11 Ausgabe der XML-Dateien

Die Darstellung erfolgt über ein HTML-Inline-Frame `<iframe src="...>`. Dort wird über das Attribut `src` direkt die XML-Konfigurationsdatei des jeweiligen Benutzers eingebunden.

```
<iframe
<?php echo src=\"...configuration_".$user_id.".xml\"";?>
... >
```

Das zugehörige XSL-Stylsheet ist jeweils in der XML-Datei »Konfiguration« referenziert. Der XML-Parser im Web-Browser setzt die Ausgabe, so wie sie im zugehörigen XSL-Stylsheet implementiert ist, um.

6.3.12 Javascript

Javascript ist eine Programmiersprache. Javascripte werden direkt in der HTML-Datei oder in einer separaten Datei notiert. Javascripte werden vom

Web-Browser zur Laufzeit interpretiert. Web-Browser besitzen hierfür eine eigene Interpreter-Software.

Um die Bedienung der Benutzungsoberfläche für den Benutzer komfortabler zu machen, wurden verschiedene Javascripte und Javascript-Funktionen implementiert.

6.3.13 Javascript-Funktionen des LabCar-Konfigurators

Folgende Javascript-Funktionen wurden implementiert.

Javascript-Funktion	**Funktionalität**
`SaveSignalDescription()`	Übergibt die Signalbeschreibung aus dem Textbereich-Feld `<textarea>` an das entsprechende versteckte Input-Feld
`SaveSignalType()`	Übergibt den gewählten Signaltyp an das entsprechende versteckte Input-Feld. Außerdem wird eine, dem Signaltyp entsprechende Vorbelegung der `action` für das Formular `FormSignal` vorgenommen.
`SaveResolution()`	Übergibt die gewählte Signalauflösung an das entsprechende versteckte Input-Feld.
`SaveDirection()`	Übergibt die gewählte Signalrichtung an das entsprechende versteckte Input-Feld.
`SaveVoltage()`	Übergibt die gewählte Signalspannung an das entsprechende versteckte Input-Feld.
`SaveLoadType()`	Übergibt den gewählten Lasttyp an das entsprechende versteckte Input-Feld.
`SaveShortGround()`	Speichert im zugehörigen versteckten Input-Feld, ob `ShortGround` angekreuzt wurde oder nicht.
`SaveShortUbatt()`	Speichert im zugehörigen versteckten Input-Feld, ob `ShortUbatt` angekreuzt wurde oder nicht.

Javascript-Funktion	Funktionalität
SaveShortP2P()	Speichert im zugehörigen versteckten Input-Feld, ob ShortP2P angekreuzt wurde oder nicht.
SaveShortInline()	Speichert im zugehörigen versteckten Input-Feld, ob ShortInline angekreuzt wurde oder nicht.
SaveOpenGround()	Speichert im zugehörigen versteckten Input-Feld, ob OpenGround angekreuzt wurde oder nicht.
SaveOpenUbatt()	Speichert im entsprechenden versteckten Input-Feld, ob OpenUbatt angekreuzt wurde oder nicht.
SaveOpenP2P()	Speichert im zugehörigen versteckten Input-Feld, ob OpenP2P angekreuzt wurde oder nicht.
SaveOpenInline()	Speichert im zugehörigen versteckten Input-Feld, ob OpenInline angekreuzt wurde oder nicht.

Nachfolgend sind beispielhaft einige Javascripte und Fragemente aus Javascript-Funktionen dargestellt.

6.3.14 SaveSignalDescription()

Die Javascript-Funktion SaveSignalDescription() übergibt den Wert aus dem Textarea-Bereich <textarea> an das versteckte Input-Feld form_signal_description.

```
function SaveSignalDescription() {
this.FormSignal.form_signal_description.value = document.
FormSignal.textarea_signal_description.value;
}
```

Die Funktion SaveSignalDescription() wird jeweils im HTML-Tag <textarea> aufgerufen.

```
<textarea name="textarea_signal_description"
onchange="SaveSignalDescription()" ... > ... </textarea>
```

6.3.15 SaveSignalType()

Die Javascript-Funktion SaveSignalType() prüft in einer FOR-Schleife, ob aus der vorhandenen Werteliste für den Signaltyp vom Benutzer ein neuer Wert ausgewählt wurde. Die Listenwerte sind durchnummeriert, beginnend mit dem Wert 0, und werden der Reihe nach auf selected == true geprüft. Der ausgewählte Wert wird dann an das versteckte Input-Feld form_signal_type_selected übergeben.

Abhängig davon, welcher Wert für den Signaltyp ausgewählt wurde, muss auch das Attribut action für das Formular entsprechend neu belegt werden. Dies ist für die Next-Schaltfläche von Bedeutung. Wurde »IO« ausgewählt leitet die Next-Schaltfläche zu interactive_iospecifi_continue .php weiter, ansonsten wird zu interactive_loaderror_continue.php weitergeleitet.

```
function SaveSignalType() {
for(i=0;i<document.FormSignal.SelectSignalType.length;++i) {
if(document.FormSignal.SelectSignalType.options[i].
selected == true) {
this.FormSignal.form_signal_type_selected.value = document.
FormSignal.SelectSignalType.options[i].value;
if (this.FormSignal.form_signal_type_selected.value != "IO") {
this.FormSignal.action = "interactive_loaderror_continue.php";
}
if (this.FormSignal.form_signal_type_selected.value == "IO") {
this.FormSignal.action = "interactive_iospecifi_continue.php";
}

....
```

Die Funktion SaveSignalType() wird jeweils im HTML-Tag <select> aufgerufen.

```
<select name="SelectSignalType" onChange="SaveSignalType()" ...
>
```

6.3.16 Integriertes Javascript im Formular »FormSignal«

Im ersten Teilformular FormSignal ist direkt nach dem HTML-Tag <select> ... </select> für die Auswahlliste des Signaltyps folgendes Javascript integriert.

```
<script type="text/javascript"><!--
if (this.FormSignal.SelectSignalType.selectedIndex == 0) {
this.FormSignal.action = "interactive_iospecifi_continue.php";
}
if (this.FormSignal.SelectSignalType.selectedIndex != 0) {
this.FormSignal.action = "interactive_loaderror_continue.php";
}
//--></script>
```

Dieses Javascript prüft beim Erzeugen der HTML-Ausgabe, ob es sich beim ausgewählten Wert um »IO« handelt oder nicht. Entsprechend wird dann das HTML-Attribut action mit interactive_iospecifi_continue.php oder interactive_loaderror_continue.php belegt.

6.3.17 SaveShortGround()

Die Javascript-Funktion SaveShortGround() übergibt den Wert des Checkbox-Input-Feldes form_short_ground an das versteckte Input-Feld form_short_ground_checked.

Hat form_short_ground.checked den Wert true, dann wird auch der Wert des versteckten Input-Feldes form_short_ground_checked auf true gesetzt.

Hat die Eigenschaft `form_short_ground.checked` den Wert `false`, dann wird der Wert des versteckten Input-Feldes `form_short_ground_checked` ebenfalls auf `false` gesetzt.

```
function SaveShortGround() {

if (document.FormLoaderror.form_short_ground.checked) {
this.FormLoaderror.form_short_ground_checked.value = true;
}else{
this.FormLoaderror.form_short_ground_checked.value = false;
}
}
```

Die Funktion `SaveShortGround()` wird jeweils im HTML-Tag `<input type ="checkbox" ... >` aufgerufen.

```
<input type="checkbox" name="form_short_ground"
onChange="SaveShortGround()" ... >
```

6.4 Technische Voraussetzungen

Der entwickelte Prototyp ist für den Microsoft Internet-Explorer Version 5.5 optimiert. Bei älteren Versionen muss der Microsoft-XML-Parser (zum Beispiel der MSXML 4.0 Service Pack 2) nachinstalliert werden. Beim Microsoft Internet-Explorer Version 5.5 ist der Microsoft-XML-Parser bereits enthalten.

Im Browser muss Javascript aktiviert sein. Auf dem Web-Server muss mindestens die PHP-Version 3.4 installiert sein.

6.5 Zur Entwicklung verwendete Werkzeuge

Für die Entwicklung der HTML-Seiten wurde Microsoft Frontpage[41] und Edit-Plus[42] verwendet. Die PHP-Programmierung wurde mit Edit-Plus und Macromedia Dreamweaver[43] durchgeführt. Das Design wurde mit Adobe Photoshop[44] entworfen. Die XML-Dateien, die Dokumenttyp-Definitionen (DTD) und die zugehörigen XSL-Stylesheets wurden mit XML-Spy, einem XML-Editor der Firma Altova[45], umgesetzt.

Für die Entwicklung des dynamischen Prototyps wurde ein lokaler Web-Server installiert. Hierfür wurde der frei verfügbare »AppServ«[46] Version 2.1.0 verwendet. »AppServ« ist ein Komplettpaket und kann schnell und einfach installiert werden. Mit »AppServ« werden unter anderem ein lokaler Web-Server (Apache-Web-Server[47]) und ein PHP-Interpreter[48] installiert.

[41] www.microsoft.com/frontpage

[42] www.editplus.com

[43] www.macromedia.com

[44] www.adobe.de/products/photoshop/main.html

[45] www.altova.com

[46] www.appservnetwork.com

[47] www.apache.org

[48] www.php.net/downloads.php

7 Resümee und Ausblick

In einem abschließenden Ausblick wird auf den Nutzen und die Erweiterungsmöglichkeiten des LabCar-Konfigurators eingegangen. Berücksichtigt werden dabei sowohl technische als auch inhaltliche Aspekte.

7.1 Nutzen

Der Nutzen des LabCar-Konfigurators liegt einerseits in der Unterstützung und Verkürzung des Beratungs- und Verkaufsprozesses. Außerdem kann der LabCar-Konfigurator neuen Mitarbeitern als Lerninstrument dienen. Für Kunden und Interessenten kann der LabCar-Konfigurator im Internet als Informationsquelle eingesetzt werden. Das innerbetriebliche Wissensmanagement wird durch den LabCar-Konfigurator unterstützt. Ein weiterer Nutzen besteht in der Vereinheitlichung der konfigurierten LabCar-Systeme.

7.1.1 Unterstützung des Beratungs- und Verkaufsprozesses

Die Definition der Kundenanforderungen an ein LabCar ist ein Prozess, der sich über einen längeren Zeitraum erstrecken kann. Oft kann der Kunde zu Beginn des Prozesses selbst noch nicht ganz genau eingrenzen, wieviele und welche Signalein- und Signalausgänge er benötigt. In der Praxis erstreckt sich der Beratungsprozess häufig über einen längeren Zeitraum, in welchem mehrere Gespräche geführt werden. Gemeinsam mit dem Kunden gilt es herauszufinden, welche und wieviele Signalein- und Signalausgänge das LabCar haben muss, damit der Kunde die entsprechend gewünschten Tests durchführen kann.

In diesem Beratungsprozess kann der LabCar-Konfigurator als hilfreiches Werkzeug dienen. Über das Abspeichern der Signallisten lassen sich Zwischenstände dokumentieren. Außerdem kann zu jedem Zeitpunkt eine virtuelle Konfiguration durchgeführt werden. Als Ergebnis der virtuellen Konfiguration lassen sich dann der aktuelle Umfang und Aufbau des LabCars visualisieren. Eine Bestellliste der einzelnen LabCar-Hardware-Komponententeile mit Preisen und weitere technische Daten stehen für jede abgelaufene Konfiguration zur Verfügung. Die Ergebnisdokumente stellen eine

hilfreiche Diskussionsgrundlage für den weiteren Verlauf des Beratungs- und Verkaufsprozesses dar.

7.1.2 Lerninstrument für neue Mitarbeiter

Neuen Mitarbeitern steht durch den LabCar-Konfigurator ein sehr mächtiges und nützliches Instrument zur Verfügung. Der LabCar-Konfigurator ermöglicht es dem Mitarbeiter, sich in den Aufbau und die Strukur von LabCars einzuarbeiten. Das Lerninstrument bietet einerseits Hilfe und Unterstützung bei der Definition von Signalein- und Signalausgängen und gibt andererseits Feedback in Form einer Ergebniskonfiguration. Durch die Visualisierung und die Dokumentation, sowohl der Eingaben, als auch des daraus folgenden Ergebnisses, hat der jeweilige Mitarbeiter für sich selbst und auch für eventuelle weitere Diskussionen mit Kollegen konkrete Fakten und Dokumente in der Hand. Dadurch kann er sein Wissen weiter ausbauen und Wissenslücken schließen.

7.1.3 Informationsquelle für Kunden und Interessenten

Sobald der LabCar-Konfigurator auch im Internet zur Verfügung steht, kann er von Kunden und Interessenten als Informationsquelle genutzt werden. Dabei findet der Kunde, beziehungsweise der Interessent, Informationen über das Produkt LabCar, über die Produkteinzelkomponenten, technische Details und die Preise. Außerdem kann sich der Kunde oder Interessent selbst näher mit der Konfiguration von LabCar-Hardware auseinandersetzen. Er kann Signalein- und Signalausgänge und deren Umfang, sowie Lastnachbildungen und Fehlersimulation, beliebig definieren. Nach abgeschlossener Konfiguration bekommt der Kunde Feedback zu seinem individuell konfigurierten LabCar. Dadurch kann er sich selbständig mit der Definition von Signalein- und Signalausgängen vertraut machen. Ein Kunde der schon Vorkenntnisse hat, kann sehr viel genauer und präziser festlegen, was er haben möchte. Durch das bessere Verständnis des Kunden wird der Beratungsprozess verkürzt. Der Kunde kann seine eigenen Bedürfnisse in Bezug auf das LabCar, das er kaufen möchte, besser einschätzen und besser kommunizieren.

Die Präsentation des Produktes LabCar zum Markt hin wird verbessert. Dadurch wird die Kommunikation und das Marketing nach außen hin ge-

stärkt. Die »Public Relations« (Englisch für: Beziehungen nach außen - zur Öffentlichkeit - hin) stellen in der Kommunikationspolitik eines Unternehmens eine wichtige Säule dar. (vgl. Net-Lexikon 2004, Stichwort: »Öffentlichkeitsarbeit«)

7.1.4 Werkzeug für innerbetriebliches Wissensmanagement

Der LabCar-Konfigurator beinhaltet Unternehmenswissen. Unternehmswissen umfasst neben expliziten Quellen auch das implizite Wissen von Mitarbeitern des Unternehmens.[49]

Das Kernprogramm »LabCar-Konfiguration« stellt die Intelligenz des LabCar-Konfigurators dar. In diesem Kernprogramm und in den gesteuerten Signaleingabemasken des Eingabebereichs »Interactive« sind Kenntnisse und Erfahrungen von Ingenieuren und Verkäufern enthalten. Wissen, das sich zum Teil nur implizit in den Köpfen der Mitarbeiter befindet, wird explizit gemacht. Dadurch wird implizites Wissen einzelner auch für andere Mitarbeiter zugänglich gemacht und diesen zur Verfügung gestellt.

Themenbereiche des Wissensmanagements, wie die Wissensverteilung, der Wissenserwerb, die Wissensbewahrung und die Wissensschaffung werden durch den LabCar-Konfigurator unterstützt und möglich gemacht.

Wissensbewahrung ist von größter Bedeutung für ein Unternehmen, wenn beispielsweise Mitarbeiter das Unternehmen verlassen. Der Wissenserwerb ist, vor allem für neue Mitarbeiter oder Mitarbeiter, die sich in diesem Bereich neu einarbeiten, ein wichtiger Prozess. Dieser Einarbeitungsprozess kann durch den LabCar-Konfigurator unterstützt werden. Die Wissensverteilung und die Externalisierung von impliziten Wissen kann außerdem die Basis für die Schaffung von neuem Wissen und somit von Innovation bilden.

7.1.5 Vereinheitlichung konfigurierter LabCar-Systeme

Der LabCar-Konfigurator bietet die Möglichkeit, den Aufbau der LabCar-Systeme einheitlicher zu gestalten. Eine Vereinheiltlichung des Aufbaus der konfigurierten LabCar-Systeme bietet später einen erheblichen Vorteil

[49] siehe hierzu auch: Kapitel 3.8 »Wissensmanagement«

in der Wartung der Systeme. Fehlerhafte und defekte Systemkomponenten können schneller gefunden und lokalisiert werden, wenn die LabCar-Systeme nach bestimmten vorgegebenen und denselben immer wiederkehrenden Strukturen aufgebaut sind.

7.2 Erweiterungsmöglichkeiten

Erweiterungen des LabCar-Konfigurators sind im Bereich der Ausgabe des Konfigurationsergebnisses jederzeit möglich. Durch eine geringfügige Anpassung kann der LabCar-Konfigurator auch für die Konfiguration von Kabelbäumen eingesetzt werden. Durch die Verwendung von UML kann die Implementierung des LabCar-Konfigurators auch in einer objektorientierten Programmiersprache vorgenommen werden.

7.2.1 Weitere Ausgabemöglichkeiten von konfigurierten LabCars

Für die Ausgabe der virtuell konfigurierten LabCars sind jederzeit viele denkbare Erweiterungen möglich. Die Erweiterungen können flexibel an den Bedürfnissen der Nutzer ausgerichtet werden. Listen, in beliebiger Zusammenstellung und nach beliebiger Sortierung, sind möglich.

Die Ausgabe auf einem PDA (Personal Digital Assistent) ist außerdem eine weitere denkbare Möglichkeit.

7.2.2 Konfiguration von Kabelbäumen

Für die Konfiguration von Kabelbäumen wäre ein Konfigurationswerkzeug, wie der LabCar-Konfigurator, ebenfalls eine große Hilfe.

Im Automobilbau bezeichnet man einen Strang von Signalkabeln, die zum Beispiel durch Kabelschellen zusammengefasst sind, als Kabelbaum. Beim Kabelbaum zweigen an verschiedenen definierten Punkten einzelne enthaltene Kabel zu mehreren oder einzeln vom Kabelhauptstrang ab.

Moderne Fahrzeuge verfügen über elektrische Leitungen, die in ihrer Summe eine Entfernung von mehreren Kilometern erreichen können. Bei einer Verlegung einzelner Kabel würden diese ein so großes Volumen einnehmen, dass in der Karosserie kein Platz dafür wäre. Daher werden alle Kabel auf

einer speziellen Werkbank zu einem Kabelbaum kombiniert und miteinander verklammert. Dieser nur noch begrenzt flexible Baum wird dann in das Fahrzeug eingebaut. (vgl. Net-Lexikon 2004, Stichwort: »Kabelbaum«)

Genau wie bei der Konfiguration von LabCars müssen auch bei Kabelbäumen die gewünschten Signalein- und Signalausgänge festgelegt werden. Die Struktur einer Benutzungsoberfläche für einen »Kabelbaum-Konfigurator« würde daher ganz ähnlich aussehen. Der hier entwickelte Prototyp des LabCar-Konfigurators müsste für eine Konfiguration von Kabelbäumen nur geringfügig angepasst werden.

7.2.3 Objektorientierte Programmierung

Mit der Verwendung von UML-Diagrammen beim Entwurf der Architektur des LabCar-Konfigurators, wurde auf der Basis von Standards die Möglichkeit für eine Portierung in eine andere Systemlandschaft mit einer objektorientierten Programmiersprache geschaffen. Die Vorarbeiten beim Entwurf, insbesondere die entworfenen UML-Diagramme, können dabei als Grundlage verwendet werden.

PHP bietet ab Version 5 objektorientierte Programmierung. Java Server Pages sind eine weitere Möglichkeit, objektorientierte Web-Applikationen umzusetzten. Active Server Pages der Firma Microsoft basieren auf Visual Basic, einer objektorientierten Programmiersprache, und stellen eine weitere Möglichkeit für die objektorientierte Umsetzung des LabCar-Konfigurators dar.

Das Zustandsdiagramm (Engl.: State Diagram) für die Signalliste in Kapitel 5.5 beipielsweise zeigt, dass die Signalliste auch als Klasse, im Sinne der objektorientierten Programmierung, umgesetzt werden kann. Dabei wird für den jeweiligen Benutzer ein konkretes Objekt, beziehungsweise eine Instanz der Signalliste, erzeugt. Die Signalliste verfügt über bestimmte Eigenschaften und Attribute. Über Methoden, kann die Signalliste verändert werden, indem beispielsweise Attribute geändert oder Signale hinzugefügt oder gelöscht werden.

7.3 Ausblick

Im Rahmen des internen Qualitätsmanagements bei ETAS wird der LabCar-Konfigurator als ETAS-eigenes Produkt geführt. Der LabCar-Konfigurator befindet sich derzeit noch im ersten Stadium - im Entwicklungsstadium. Eine Entscheidung über eine Weiterentwicklung des LabCar-Konfigurators steht noch aus.

Der LabCar-Konfigurator bietet, wie in Kapitel 7.1 »Nutzen« und den dazugehörigen Unterkapiteln beschrieben, erhebliche Vorteile und Möglichkeiten, die in dieser Form bis jetzt noch nicht zur Verfügung stehen. Die Weiterentwicklung des LabCar-Konfigurators würde dem Unternehmen große Vorteile und erheblichen Nutzen einbringen.

Sicherlich gilt es den Aufwand abzuwägen.

Der LabCar-Konfigurator bietet Potential, die Verkaufs- und Absatzzahlen zu erhöhen und die Kosten für die LabCar-Hardwarekonfiguration zu senken. Der Kundenberatungsprozess kann durch den LabCar-Konfigurator verkürzt werden. Verkäufer können durch den LabCar-Konfigurator besser und gezielter geschult werden. Mit dem LabCar-Konfigurator können außerdem Zwischenstände im Beratungsprozess dokumentiert werden. Auf die Dokumentation der Zwischenstände kann jederzeit wieder zugegriffen werden. Dadurch wird der Verkaufsprozess unterstützt und verkürzt. Dies wirkt sich kostensenkend aus.

Ein LabCar-Konfigurator, der im Internet zur Verfügung steht, gibt den Kunden die Möglichkeit sich zu informieren. Besser informierte Kunden sind für den Beratungs- und Verkaufsprozess von großem Vorteil. Der Beratungsprozess mit einem informierten und sachkundigen Kunden wird schneller und effektiver ablaufen. Dadurch erhöht sich auch die Kundenzufriedenheit.

Durch einen LabCar-Konfigurator im Internet wird das Produkt LabCar mit seinen vielfältigen technischen Details und Möglichkeiten einem größeren Kunden- und Interessentenkreis zugänglich gemacht. Bessere und detailiertere Informationen über ein Produkt sind eine grundlegende Voraussetzung für den besseren Absatz eines Produktes.

Außerdem bietet der LabCar-Konfigurator die Möglichkeit den Aufbau von konfigurierten LabCar-Systemen zu vereinheitlichen. Im Aufbau einheitli-

chere LabCar-Systeme reduzieren später Kosten im Bereich von Wartung und Service.

Die Weiterentwicklung und Einführung des LabCar-Konfigurators wäre für die erfolgreiche Positionierung von LabCar-Testsystemen am Markt sicherlich ein großer Gewinn.

Literaturverzeichnis

Arnheim, R. (1983): Art and Visual Perception: A Psychology of the Creative Eye. Univeristy of California Press.

Arnheim, R. (1989): Visual Thinking. University of California Press.

Balzert, H. (2000): Lehrbuch der Software-Technik. 2. Auflage. Heidelberg - Berlin: Akademischer Verlag.

Bass, L., Clements, P. & Kazman, R. (1998): Software Architecture in Practice. SEI Series in Software Engineering. Boston: Addison Wesley.

Bodoff, S. (2002): Web Applications and Technology. In: Armstrong, E., Bodoff, S., Carson, E., Fisher, M., Green, D. & Haase, K. (Hrsg.). The Java^{TM} Web Services Tutorial. 1. Auflage. Palo Alto (California): Sun Microsystems, Inc.

Bonsiepe, G. (1996): Interface. Design neu begreifen. Mannheim: Bollmann Verlag.

Conallen, J. (2003): Building Web Applications with UML. Web Application Basics. 2nd Edition. Boston: Addison-Wesley.

Cooper, A. (1999): The Inmates are Running the Asylum. Indianapolis: SAMS Verlag.

Köhler, W. (1992): Gestalt Psychology: An Introduction to New Concepts in Modern Psychology. New York: Liveright.

Koffka, K. (1967): Principles of Gestalt Psychology. London: Routledge.

Krause, J. (2000): PHP Grundlagen und Lösungen. München/Wien: Carl Hanser Verlag.

McLaughlin, B. (2000): Java and XML. Sebastopool: O'Reilly & Associates.

Middendorf, S., Singer, R. & Heid, J. (2002): Java^{TM}. Programmierhandbuch und Referenz für die Java^{TM}-2-Plattform. Standard Edition. 3. Auflage. Heidelberg: dpunkt.verlag.

Nonaka, I. & Takeuchi, H. (1997): Die Organisation des Wissens. Wie japanische Unternehmen eine brachliegende Ressource nutzbar machen. Frankfurt: Campus-Verlag.

Sauer, H. (2002): Relationale Datenbanken. Theorie und Praxis. 5. Auflage. Addison-Wesley.

Probst, G. Raub, S. & Romhardt, K., H. (1999): Wissen managen. Wie Unternehmen ihre wertvollste Ressource optimal nutzen. 3. Auflage. Wiesbaden: Gabler-Verlag.

Schäuffele, J. & Zurawka, T. (2003): Automotive Software Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge - 1. Auflage. Wiesbaden: Vieweg & SohnVerlag.

Schütt, P. (2000): Wissensmanagement. Mehrwert durch Wissen. Nutzenpotenziale ermitteln. Den Wissenstransfer organisieren. IBM Unternehmensberatung GmbH (Hrsg.). Falken Verlag.

Shneiderman, B. (2002): User Interface Design. Deutsche Ausgabe. Effektive Interaktion zwischen Mensch und Maschine. Leitfaden für intelligentes Schnittstellendesign. Was Programmierer und Designer über den Anwender wissen müssen. 3. Auflage. Bonn: mitp-Verlag.

Thissen, F. (2003): Kompendium Screen-Design. Effektiv informieren und kommunizieren mit Multimedia - dritte, überabeitete und erweiterte Auflage. Berlin: Springer-Verlag.

Wertheimer, M. (1925): Drei Abhandlungen zur Gestaltpsychologie. Erlangen: Philosophische Akademie.

Firmeninterne Quellen

ETAS - LabCar (2002): Hardware-in-the-Loop-Testsysteme für Steuergeräte.
ETDE CPR/Sar/05.02.

ETAS - LabCar-Hardware (2002): Firmeninternes Bildarchiv.
ETDE/CPR, 2002.

ETAS - LabCar-Project (2002): »Diesel-Management«.
ETDE/CPR-Sar/09.02.

Internetquellen

Becker, O. (2003): XML-Halbkurs. Systems Architecture Group. Computer Science Department. Humboldt-Universität. Berlin.
http://www.informatik.hu-berlin.de/~obecker/Lehre/SS2002/XML/.
(Zugriff am: 08.10.2004).

Bodoff, S. (2002): Web Applications and Technology. In: Armstrong, E., Bodoff, S., Carson, E., Fisher, M., Green, D. & Haase, K. (Hrsg.). The Java^{TM} Web Services Tutorial. 1. Auflage. Palo Alto (California): Sun Microsystems, Inc.
http://java.sun.com/webservices/docs/1.0/tutorial/doc/WebApp.html.
(Zugriff am: 08.09.2004).

Conallen, J. (2003): Building Web Applications with UML. Web Application Basics. 2nd Edition. Boston: Addison-Wesley.
http://www.awprofessional.com/articles/article.asp?p=30610&seqNum=3.
(Zugriff am: 03.09.2004).

Dumke, R. (2004): UML-Tutorial. Otto-von-Guericke-Universität Magdeburg.
http://ivs.cs.uni-magdeburg.de/~dumke/UML/. (Zugriff am: 13.09.2004).

ELAU AG (2004): ELAU Elektronik Automations AG. PacDrive Lexikon. CAN.
http://www.elau.de/Rahmen.asp?Knoten=95#c1. (Zugriff am: 07.09.2004).

E-Matrix Global (2003): E-Matrix Global Ltd. Software for Superior Enterprise Performance. Homepage for EMG Site.
www.ematrixglobal.com/index.html. (Zugriff am: 07.09.2004).

ETAS (2003): ETAS GmbH.
http://de.etasgroup.com.
(Zugriff am: 08.09.2004).

Götz, R. et al. (1997): Datenbank-Seminar Version 1.7. Steinbeis-Transferzentrum. Softwaretechnik Esslingen. 1997.
http://www.stz-softwaretechnik.de/download/skripte/DATABASE.pdf.
(Zugriff am: 13.09.2004).

Haas, M. (2004): Abkürzungen und Begriffe der Ingenieurinformatik. FH Braunschweig/Wolfenbüttel - Fachbereich Elektrotechnik. Sommersemester 2004.
http://public.rz.fh-wolfenbuettel.de/~haas/vl/pdv_vorlesung/abkuerzungen.pdf. (Zugriff am: 07.09.2004).

Herde, G. (2004): IT-Kompaktkurs. Datenbanken. Entity-Relationship, Normalformen, Semantisches Datenmodell. Fachhochschule Deggendorf . Virtuelle Hochschule Bayern .
http://www.bw.fh-deggendorf.de/kurse/db/skripten/skript5.pdf.
(Zugriff am: 13.09.2004).

Kolter Electronic (2004): Kolter Electronic. PC Schnittstellen, RS232, RS422.
http://www.pci-card.com/schnittstellen.html. (Zugriff am: 07.09.2004).

Kronsbein, M. (2004): php-homepage.de. Die deutschsprachige Ressource für PHP und MySQL. 1999-2004.
http://www.php-homepage.de.
(Zugriff am: 12.09.2004).

Krüger, I. (2004): Farbentheorie und Farbgestaltung. Fraunhofer Institut für Integrierte Publikations- und Informationssysteme (IPSI). 1999-2004.
http://www.ipsi.fraunhofer.de/~crueger/farbe/index.html.
(Zugriff am: 08.09.2004).

Middendorf, S., Singer, R. & Heid, J. (2002): Java^{TM}. Programmierhandbuch und Referenz für die Java^{TM}-2-Plattform. Standard Edition. 3. Auflage. Heidelberg: dpunkt.verlag.
http://www.dpunkt.de/java/index.html. (Zugriff am: 07.09.2004).

Motorola (2000): Motorola. Advance Information. Fault Tolerant CAN Interface. Rev: 2.2. Date: 02.Nov.2000.
http://e-www.motorola.com/files/analog/doc/data_sheet/MC33388.pdf.
(Zugriff am: 08.09.2004).

Münz, S. (2001): SELFHTML Version 8.0. 2001.
http://de.selfhtml.org. (Zugriff am: 09.09.2004).

Nehrbass, E. (2004): Seminar zum Thema Datenintegration. Datenintegration mit XML. Technische Universität Darmstadt. Fachbereich Informatik. Sommersemester 2004.
http://www.ipsi.fraunhofer.de/~lehti/seminar2004/Datenintegration_mit_XML.pdf. (Zugriff am: 07.09.2004).

Net-Lexikon (2004): Net-Lexikon.
http://www.lexikon-definition.de. (Zugriff am: 21.09.2004).

Nielsen, J. (1997): Jakob Nielsen's Alertbox for October 1, 1997: How Users Read on the Web.
http://www.useit.com/alertbox/9710a.html. (Zugriff am: 13.09.2004).

QM-Lexikon (2004): Das QM-Lexikon. 1995-2004.
http://www.quality.de/lexikon.htm. (Zugriff am: 21.09.2004).

SAP (2004): SAP Deutschland - Unternehmen .
http://www.sap.com/germany/aboutsap/index.asp. (Zugriff am: 07.09.2004).

TechTarget Network (2004): IT-Specific Encyclopedia. Fault-Tolerant. 2000-2004.
http://whatis.techtarget.com/definition/0,„sid9_gci214456,00.html.
(Zugriff am: 08.09.2004).

The PHP Group (2004): PHP Preprocessor. PHP Handbuch. 2001-2004.
http://www.php.net/manual/de.php. (Zugriff am: 13.09.2004).

WIKIPEDIA (2004): WIKIPEDIA - Die freie Enzyklopädie.
http://de.wikipedia.org/wiki. (Zugriff am: 07.09.2004).

Williams, K. (2002): XML for Data: XSL style sheets: push or pull? A look at two authoring techniques for XSL style sheets and how they stack up for data. May 2002.
http://www-106.ibm.com/developerworks/xml/library/x-xdpshpul.html.
(Zugriff am: 14.09.2004).

Wissen.de (2004): Wissen.de - Lexikon. 2000-2004.
http://www.wissen.de. (Zugriff am: 07.09.2004).

Danksagung

Zunächst möchte ich Herrn Prof. Wolf-Fritz Riekert für seine zahlreichen Hinweise zur inhaltlichen Ausrichtung der Masterarbeit und vor allem für seine Unterstützung und Geduld bei der Erstellung der Masterarbeit danken.

Weiterer Dank geht an Herrn Dr. Peter Bach für seine Bereitschaft zur Zweitkorrektur und die Unterstützung in ETAS-spezifischen und technischen Themenbereichen.

Sowohl Herr Prof. Riekert als auch Herr Dr. Bach haben durch viele Anregungen und auch kritische Bemerkungen diese Arbeit positiv beeinflusst.

Des Weiteren möchte ich mich ganz herzlich bei meiner Familie bedanken, für die Unterstützung und das Verständnis, das sie mir während der Erstellung meiner Masterarbeit entgegengebracht hat.

Roswitha Wettinger Februar 2004

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift