Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Konzeption und Entwicklung eines Prüfmoduls zur Steuerung eines Prüfplatzes für elektrostatische Entladungen

Masterarbeit

Im Studiengang Angewandte Informatik

von

Charles Hervé, Tchoutat Tchabo

04. Dezember 2007

Erstprüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. –Ing. Bernado Wagner
Betreuer: M. Sc. Eric Knauss
Dipl. -Ing. Matthias Grimm

Erklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungskommission vorgelegen.

Hannover, den 04.12.2007

Abstract (Kurzfassung)

Im automobilen Bereich hat man regelmäßig mit elektronischen Bauteilen zu tun. Bei der Fertigung und Handhabung dieser elektronischen Bauteile, auch bei ihrer Montage in Kraftfahrzeug können elektrostatische Entladungen (electrostatic discharge kurz: *ESD*) entstehen. Dies kann zur Beschädigung oder Störung der Funktionalität von Bauteilen führen. Um die Widerstandfähigkeit der Baugruppen gegen ESD zu erhöhen, werden bei WABCO in der Hardware-Entwicklung Schutzmassnahmen entwickelt, um die Wirkung des ESD abzumildern. Ein ESD-Test dient dazu, Baugruppen auf Empfindlichkeit gegen Elektrostatische Entladung zu überprüfen. Dabei werden alle Anschlusspins eines Steuergeräts getestet, indem sie vor und nach der Belastung mit elektrischen Entladungen durchgemessen werden. Ein Programm, das einen automatischen ESD-Test steuert wurde schon entwickelt, hat aber bereits Grenzen erreicht, an dem die Wartbarkeit und Erweiterbarkeit nicht mehr gewährleistet ist. Deshalb wird in dieser Arbeit ein Prüfmodul entwickelt, das eine ESD Prüfung einer elektronischen Baugruppe nach einem vorgegeben Testablauf steuert. Dabei sollen passende Szenarien gefunden werden und deren Auswirkungen auf den Entwurf sowie daraus abgeleitete systematische Abnahmetests gezeigt werden. Abnahmetests und Priorisierung der Anforderungen sollen auf Basis der Szenarien formuliert werden.
In dieser Arbeit wird von den Anforderungen, über Konzept und Entwurf, bis hin zur Implementierung und Test betrachtet um ein wartbare und erweiterbare Software (das Prüfmodul) zu entwickeln.

Diese Arbeit ist in Zusammenarbeit mit der Firma WABCO Development GmbH, Abteilung T2 Elektronik Hardware entstanden. Die Firma WABCO Development GmbH erklärt sich einverstanden, das die Universität Hannover die Arbeit für Lehrzwecke verwendet und veröffentlicht, behält sich jedoch die kommerzielle Nutzung vor. Ansprüche daraus für die Universität Hannover ergeben sich nicht.

Dipl.-Ing Jens Gröger Leader Technologie Center T2	Dipl.-Ing Matthias Grimm Betreuer der Diplomarbeit

Danksagungen

Ich danke meiner Frau Grace für ihre sehr große Unterstützung,
außerdem Herrn Eric Knauss und Matthias Grimm für die interessierte und vorbildliche Betreuung dieser Arbeit sowie allen Mitarbeiter der Firma WABCO, die mir bei Planung und Durchführung zur Seite gestanden haben, insbesondere Herrn Detlev Hagen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Das Unternehmen WABCO

Diese Master-Arbeit wurde vom 20. Mai bis 04 Dezember 2007 in der Elektronik Hardware Entwicklung der WABCO Development GmbH in Hannover durchgeführt. Die Hauptaufgabe dieser Abteilung ist die Entwicklung und die Qualifizierung von elektronischen Steuergeräten, entweder als komplettes Steuergerät oder als Modul z.B. für die Integration in mechatronische Module.
WABCO ist mit etwa 6500 Mitarbeitern weltweit und 2400 am Standort Hannover eine der führenden Hersteller von Fahrzeugregelsystemen, insbesondere elektronischer Bremssysteme wie ABS (Antiblockiersystem) und EBS (Elektronik Bremse System) für Lastkraftwagen. WABCO ist unterteilt in die Business Units (BUs)
– Drucklufterzeugung und Bremsen,
– Antriebsstrang,
– Anhänger-Systeme,
– Fahrdynamik-Regelung und
– PKW-Systeme.

1.2 Aufgabestellung

Die Aufgabe ist die Konzeption und Entwicklung eines Prüfmoduls zur Steuerung eines Prüfplatzes für elektrostatische Entladung. Das Steuerprogramm soll ohne graphische Bedienoberfläche implementiert werden. Diese wird später über eine Schnittstelle angebunden.

Ein wichtiger Bestandteil dieser Arbeit ist, die Verwendung der Methoden des Software Engineering, nämlich von den Anforderungen über Konzept, bis hin zur Implementierung und Test. Dabei sollen Szenarien verwendet werden zur Formulierung von Abnahmetest und Priorisierung der Anforderungen.
Anhand dieser Methode soll eine geeignete Lösung der Ziele dieser Masterarbeit gefunden werden. Die Entwicklungs-Ziele sind:

- Das Ansprechen der Prüfplatzkomponenten über ein Treiberkonzept
- Die Kommunikation zwischen der Bedienoberfläche und dem Prüfmodul über D-Bus
- Die ESD Prüfung einer elektronischen Baugruppe nach einem vorgegebenen Testablauf

Schließlich ist der daraus abgeleitete Entwurf zu nutzen um das Programm umzusetzen, und die Vorgehensweise zu dokumentieren.

1.3 Namenskonvention

In der vorliegenden Masterarbeit werden einige Begriffe und Notation (kursiv geschrieben) verwendet, die hier definiert werden.
Unter *Prüfling*, *Elektronik* oder *DUT* (Device under Test) ist die zu testende elektronische Baugruppe zu verstehen.
Ein *Prüfling* besteht aus Steckern die eine Anzahl von Pins enthalten.
Ein *Prüfpunkt* oder *Prüfposition* ist der Punkt, der mit Prüfpulsen beaufschlagt werden soll.
Ein *Sicherheitspunkt* oder eine *Sicherheitsposition* kurz ***SP*** ist ein Punkt im Raum, von dem andere Punkte kollisionsfrei erreicht werden können. Es gibt für jeden Pin und Stecker einen Sicherheitspunkt.
Pin–Sicherheitspunkt (Sicherheitspunkt eines Pins):
Von ihm aus muss es möglich sein den Steckerpin direkt mit linearer Bewegung anzufahren.
Außerdem muss es möglich sein, jeden anderen Pin–Sicherheitspunkt dieses Steckers kollisionsfrei anzufahren.
Stecker – Sicherheitspunkt (Sicherheitspunkt eines Steckers):
Von ihm aus muss sich jeder Pin–Sicherheitspunkt des Steckers Kollisionsfrei erreichen lassen. Von ihm aus muss sich jeder andere Stecker–Sicherheitspunkt der Elektronik kollisionsfrei anfahren lassen.
Roboter anlernen: Beim Anlernen des Roboters, wird das Koordinatensystem der zu testende Elektronik angelernt. Damit der Roboter weißt, wo die Prüfpositionen des Pruflings sich befinden.
Signal : In der Kommunikation mit Geräte, steht ein *Signal* als einen Befehl, der ans Gerät gesendet wird.
HBM: Steht für Human Body Model.
D-Bus ist ein Nachrichten-Bus-System, das verschiedenen Programmen eine einfache Möglichkeit bietet, miteinander zu kommunizieren (Interprozesskommunikation). [http://www.freedesktop.org/wiki/Software/dbus]. D-Bus ist ein „free Software“ und unter der GPL (GNU Public License) veröffentlicht.
GUI: Graphical User Interface, auf Deutsch graphische Benutzerschnittstelle, ist unsere „Mensch-Maschine-Schnittstelle“ und wird auch als graphische Bedienoberfläche genannt.

1.4 Gliederung

Die vorliegende Arbeit gliedert sich in sechs Kapitel auf:

- In Kapitel 1 (das aktuell Kapitel) wird die Firma, wo diese Arbeit gemacht wurde, präsentiert. Außerdem wird die Aufgabe eingeführt, die Namenkonvention dieser Arbeit und der Ablauf der Arbeit erläutert.
- In Kapitel 2 wird definiert was eine Elektrostatische Entladung ist, der ESD-Prüfplatz präsentiert und die Durchführung eines ESD-Tests beschrieben.
- In Kapitel 3 wird das zu entwickelnden System von seiner Umgebung abgegrenzt, die Prozesse und Anforderungen des Systems ermittelt und ein Qualitätsmodell des Systems aufgebaut.
- In Kapitel 4 wird die Architektur des Prüfmoduls ausgesucht, die Szenarien des Systems beschrieben und die wichtigen Konzepte des Prüfprogramms erläutert, insbesondere das Treiber- und D-Bus-Konzept.
- In Kapitel 5 wird anhand der Konzepte die technische Realisierung erläutert. Die Software und Hilfsmittel, die bei der Implementierung benutzt wurden, werden benannt. Dann werden ein Test des entwickelten Systems und die Ergebnisse präsentiert.
- Schließlich wird in Kapitel 6 eine Zusammenfassung über die gesammelten Ergebnisse und den Beitrag dieser Arbeit gegeben. Die Arbeit endet mit einem Ausblick über mögliche Erweiterungen.

Kapitel 2

Allgemeine Grundlagen

2.1 Elektrostatische Entladung (ESD)

Elektrostatische Entladung (engl. electrostatic discharge kurzer *ESD*), entsteht wenn ein aufgeladener Körper in Kontakt mit einem leitenden Gegenstand tritt. Dabei wird eine Potenzialdifferenz ausgeglichen, was einen sehr kurzen hohen elektrischen Stromimpuls zur Folge hat.
Die Potentialdifferenz entsteht durch Ladungstrennung, die z.B. beim Laufen über einen Teppichboden, Bewegen auf einem Stuhl, Ein- oder Aussteigen aus Autos auftritt.
Im automobilen Bereich, indem die Anzahl von Steuergeräten immer weiter wächst, können elektrostatische Entladungen zur Beschädigung oder Störung dieser elektronischen Bauteile führen. Elektrostatischen Entladungen treten besonders bei der Handhabung elektronischer Bauteile in der Produktion auf, ist aber auch beim täglichen Umgang mit Fahrzeugen kein seltenes Ereignis. Die Auswirkungen elektrostatischer Entladungen werden zu einem immer größeren Qualitätsrisiko in der Herstellung und Verarbeitung elektronischer Bauelemente deswegen spielt die ESD in der Hardware-Entwicklung bei WABCO einer sehr großen Rolle. Hier werden Steuergeräte am ESD - Prüfplatz einem Test unterzogen um sie gegen diese gefährlichen elektrostatischen Entladungen resistent zu machen.

2.2 Der ESD-Prüfplatz

Ein ESD- Prüfplatz dient dazu die Empfindlichkeit von elektronischen Baugruppen gegen ESD zu prüfen. Ursprünglich wurden bei WABCO alle ESD-Tests manuell durchgeführt, aber wegen des hohen Kosten- und Arbeitsaufwands, wurde ein automatischer ESD-Prüfplatz aufgebaut. Dieser besteht aus einem 6-Achsroboter (siehe Abbildung 2.1). Dieser Roboter führt die wesentlichen Bewegungen zur Durchführung eines Tests aus. Der Prüfplatz ist aus folgenden Komponenten aufgebaut:

- Einem Prüfkopf, der an der Roboterhand angebracht ist, bestehend aus:
    - Einer ESD- Prüfpistole, die die elektrostatische Ladung ermöglicht.
    - Einer 1MOhm Entladungsspitze, für die Entladung
    - Einer Messspitze, zur Messung der Impedanz vor und nach der Entladung.
- Einer Robotersteuerung, welche den Roboter steuert. Sie kommuniziert mit dem Roboter über eine serielle Schnittstelle. Sie ermöglicht auch eine manuell Steuerung des Roboters durch ein Handbedienteil.
- Einem ESD-Generator zur Erzeugung der Prüfspannungen. Er kommuniziert mit dem Steuerrechner über eine serielle Schnittstelle.
- Einem ESD-Detektor, der die Entladung detektiert und mit dem Steuerrechner über die parallele Schnittstelle kommuniziert.
- Einer Wetterstation, sie übermittelt die aktuell Temperatur, Luftfeuchtigkeit und Luftdruck.
- Einem Testtisch, der nach ISO 10605 ausgeführt ist und auf dem der Prüfling (siehe Abbildung 2.3) montiert ist.

- Einer RLC-Messbrücke zur Impedanzmessung, die mit dem Steuerrechner über eine GPIB Schnittstelle verbunden ist.
- Dem Steuerrechner, auf dem das Prüfmodul laufen soll. Er steuert den gesamten Prüfablauf und kommuniziert mit allen aktiven Prüfplatzkomponenten.
- Dem Roboterrechner, der den Roboter steuert. Er wartet auf Befehle vom Steuerrechner, führt die aus und meldet die den Zustand der Ausführung zurück.

Abbildung 2.1: ESD Prüfplatz

2.4 Der ESD-Test

Der ESD-Test wird nach der internationalen Norm ISO 10605 durchgeführt. Im Testablauf simuliert man die elektrostatische Entladung, die zwischen einem elektrostatisch aufgeladenen Menschen und der zu testenden Elektronik stattfinden kann. Als Grundlage dient ein Test nach dem Human Body Model (HBM), wie er in der ISO 10605 beschrieben ist. Kennzeichnende Größen sind dabei die Kapazität (C) eines Menschen, sein Hautwiderstand (R) und die Spannung (U) auf die er sich aufladen kann. (siehe Abbildung 2.2).

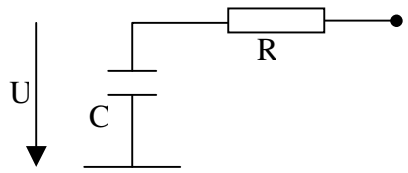


Abbildung 2.2: Ersatzschaltbild des Human Body Models

Man unterscheidet zwei Typen von den Tests:

- Der *Package Handling Test*. Hier wird die Widerstandsfähigkeit gegen Entladungen überprüft, die bei der Handhabung, z.B. beim Auspacken oder Montage der Geräte, auftreten können.
- Der *Modul Test*. Hier wird Widerstandsfähigkeit gegen Entladungen überprüft, die auf eine Elektronik während des Betriebs auftreten kann.

Im folgende wird den Ablauf eines automatischen Tests aus Sicht des Prüfmoduls beschrieben. Die neue Generation des automatischen Tests besteht aus zwei voneinander unabhängigen Teilen: dem Client (der Bedienoberfläche) und dem Server (dem Prüfmodul). Der automatische Test besteht aus zwei Phasen: Der Testvorbereitung und der Durchführung des ESD Tests.

2.4.1 Die Testvorbereitung

In dieser Phase muss der Bearbeiter den ESD Test vorbereiten, indem er einige Informationen zusammenträgt und aufbereitet. Der Bearbeiter wird diese Informationen an den Server mit Hilfe des Clients übermitteln. Diese Informationen sind die Eingabedaten oder Testdaten des Prüfmoduls und unterteilen sich wie folgt:

- Die Elektronikdefinition enthält Information über die zu testende Elektronik:
    - Allgemein Information über die Elektronik
    - Beschreibung des Koordinatensystems der Elektronik
    - Die Liste der Stecker
    - Name der Elektronik
    - Teilenummer
    - Eine freie Beschreibung
- Die Steckerdefinition enthält Information über den Stecker und seine physikalischen Abmessungen. Sie muss für jeden Stecker angegeben werden. Diese Information sind:
    - Die Nummer des Steckers
    - Der Name des Steckers
    - Eine freie Beschreibung
    - Der Ursprung des Steckers mit seinen Koordinaten x, y, z. Dieser Ursprung ist üblicherweise die Spitze des ersten Pins des Steckers.
    - Der Sicherheitspunkt des Steckers mit seinen Koordinaten x, y, z.
    - Die Orientierung des Steckers an der Elektronik, mit den Winkeln A, B, C.
    - Die Anzahl der Pins
    - Die Liste der Pins, für jede Pin wird folgenden Information angegeben:

        - Die Nummer des Pins
        - Die Sicherheitsposition des Pins
        - Die Position des Pins
- Die Testdefinition: Sie enthält Informationen über die Konfiguration des Tests und den Testablauf. Diese Angaben bestimmen was und wie genau eine Elektronik getestet wird. Sie besteht aus:
    - Dem Standard, der dem Testablauf zugrunde liegt
    - Dem Kapazitätswert des zu verwendenden Human Body Models
    - Dem Widerstandswert des zu verwendenden Human Body Models
    - Dem Mindestabstand zwischen Testimpulsen bei Kontaktentladung
    - Dem Mindestabstand zwischen Testimpulsen bei Luftentladung
    - Der Annäherungsgeschwindigkeit des Prüfkopfes bei Luftentladung
    - Der Messfrequenz der RLC Messbrücke
    - Der Wechselspannungsamplitude des Messsignals
    - Dem Gleichspannungspegel des Messsignals
    - Dem Typ der Ersatzschaltung (Seriell oder Parallel)
    - Den einzelnen Testschritten. Für jeden Testschritte wird festgelegt:
        - Die Art der Entladung
        - Die Anzahl der Prüfpulse pro Prüfposition
        - Die verwendete Prüfspannung
        - Ein Flag, das angibt, ob es eine Messung in diesem Testschritt geben soll oder nicht.
        - Ein Flag, das angibt, ob es eine Erdung in diesem Testschritt geben wird oder nicht.

Nach der Vorbereitung der Daten, muss der Bearbeiter den Prüfling auf dem ESD Testtisch montieren und das Basiskoordinatesystem anlernen. Der Anlernvorgang ist eine vom Steuerprogramm unabhängige Prozedur, und dient dazu dem Roboter die Lage des Prüflings mitzuteilen. (siehe Abbildung 2.3).

2.4.2 Durchführung des ESD Tests

Bevor der Bearbeiter den ESD Test starten kann, müssen einige vom Steuerprogramm unabhängige Vorbedingungen erfüllt sein:

- Das Handbedienteil des Roboters muss korrekt in seiner Halterung eingesteckt sein
- Die Tür der Sicherheitszelle des Roboters muss richtig geschlossen werden
- Ein Knopf, der die Fahrfreigabe des Roboters auslöst, muss gedruckt werden

Der Bearbeiter startet den Test mit Hilfe des Clients, indem er durch den Client die Testdaten an das Prüfmoduls übermittelt. Während des Tests, kann das Prüfmodul bestimmte Handlungen (z.B. Prüfspitze wechseln, Roboter anlernen oder Human-Body-Model konfigurieren) vom Bearbeiter verlangen um den Test fortzusetzen. Der Bearbeiter bekommt diese Forderung über den Client mitgeteilt. Wenn er die Tätigkeit ausgeführt hat, setzt das Programm den Test fort.
Das Prüfprogramm gibt am Ende des Tests einer Reportdatei aus, in dem die Ergebnisse des Tests festgehalten sind.

Abbildung 2.3: Beispiel eines Prüflings

2.5 Zusammenfassung

In diesem Kapitel wurde der Aufbau des ESD-Prüfplatzes dargestellt, den das Prüfprogramm benötigt um einen automatischen ESD Test durchführen zu können. Es wurde auch beschrieben was ein ESD-Test ist und wie man einen automatischen ESD-Test durchführt. Besonders wichtig ist Abschnitt 2.4.1 zur Vorbereitung eines ESD Tests, in dem die benötigten Eingabedaten dargestellt wurden.

Kapitel 3

Anforderungen

Um eine wartbare und erweiterbare Software zu implementieren, muss dies bereits bei der Spezifikation der Anforderungen von Anfang an berücksichtigt werden. Deshalb wird hier mit Hilfe der Abnahmekriterien ein Qualitätsmodel definiert. Darin werden die Qualitätsziele der Software klar definiert und eine Liste der Stakeholder ermittelt. Die Erhebung der Anforderungen in diesem Kapitel zeigt wie sich die Anforderungen in diesem System unterteilen und welche Stakeholder es gibt. Diese Anforderungen ergeben sich durch die Befragung des Experten und den Benutzern des ESD-Tests bei WABCO. Hier werden die Ziele des Prüfmoduls festgelegt, in denen die umgebende Hardware eine wesentliche Rolle spielt.
Die daraus abgeleiteten Anforderungen lassen sich in verschiedenen Gruppen unterteilen, wie in Abschnitt 3.5 beschrieben.

3.1 Softwareziele

„Erstrebenswerter Zustand, der in der Zukunft liegt und dessen Eintritt von bestimmten Handlungen und Unterlassungen abhängt, der also nicht automatisch eintritt“[GA00].
Hier ist wichtig zunächst die Zielsetzungen dieser Software zu erläutern. Wir wollen in dieser Arbeit ein altes System ersetzen. Dies hat schon seine Grenzen erreicht, so dass es nur schwer erweiterbar und wartbar ist. Die neue Software soll wartbar und erweiterbar sein. Sie soll die Nachbarsysteme durch Treiber ansprechen. Sie soll mit der Bedienoberfläche über D-Bus kommunizieren. Sie soll den ESD-Test eines Prüflings nach einem vorgegebenen Testablauf durchführen. Es soll explizit keine Bedienoberfläche entwickelt werden.

3.2 Stakeholderliste

„Stakeholder sind alle Personen, die von der Systementwicklung und natürlich
auch vom Einsatz und Betrieb des Systems betroffen sind“. [H&R02]
Die Klassifizierung der Stakeholder kann nach ihren Rollen erfolgen [H&R02]. Das Prüfprogramm, das eher die Bedürfnisse von Fachleuten erfüllen soll, hat nur wenige Stakeholder:

Der ESD Experte: Er definiert die fachliche Ziele des Prüfmoduls und die Erwartung an die Software um ein besseren ESD Test durchzuführen.

Die Anwender: Sie nutzen das Prüfmodul um ihre Elektronik zu testen. Sie möchten ihren Test mit so vielen Optionen wie möglich, entsprechend ihren Anforderungen konfigurieren können.

Das Wartungspersonal: es möchte dem Prüfmodul einen neuen Treiber oder weitere Funktionalität hinzufügen. Der ESD Experte spielt diese Rolle bei WABCO.

3.3 Softwarekontext

Das in dieser Arbeit zu entwickelnde System ist das Steuerprogramm des Prüfplatzes. Das Steuerprogramm soll mit Komponenten des in Kapitel 2 beschriebenen Prüfplatzes kommunizieren. Ein Bestandteil dieses Systems ist das D-Bus Interface mit dem das Prüfprogramm die Informationen mit der Bedienoberfläche austauscht. Diese intensive Interaktion des Prüfprogramms mit der Umgebung erhöht die Komplexität und die Menge der Flussdiagramme um das System zu beschreiben. Deshalb ist es wichtig hier das System von seiner Umgebung abzugrenzen. Das in der strukturierten Analyse [DeM79] eingeführt Kontextdiagramm hilft zu klären, was außerhalb und was innerhalb der Systemgrenzen liegt. Das Kontextdiagramm stellt die oberste Hierarchie ebene des zu entwickelnden Systems dar. Es gibt zwei Arten der Kontextbildung [H&R02], um das Kontextdiagramm explizit zu beschreiben: den logischen und den physikalischen Kontext.
Beim logischen Kontext liegt der Fokus auf der Kommunikation mit den Nachbarsystemen, beim physikalischen Kontext auf den physikalischen Kanälen und Übertragungsmedien, die das System mit den Nachbarsystemen verbindet.

3.3.1 Logische Kontext

Um die Kommunikation mit den Nachbarsystemen zu beschreiben hilft uns an dieser Stelle der logische Kontext. Es gibt drei Typen von Kontextdiagrammen [H&R02] abhängig von den Schnittstellen, die den logischen Kontext darstellen: Kontextabgrenzung mittels Use-Case-Diagramm, Kontextabgrenzung mittels Klassendiagramm, Kontextabgrenzung mittels Sequenzdiagramm. Von Anfang an war es nicht klar welches dieser Diagramme das zu entwickelnde Systems am bestens darstellen kann. Deshalb wird im Weiteren eine Abgrenzung des Systems Prüfmodul mit jedem Diagramm durchgeführt. Am Ende wird das bessere Diagramm ausgewählt. Um zu diesen Diagrammen zu gelangen müssen zuerst die Nachbarsysteme identifiziert werden.

Identifizierung von Nachbarsystemen

Nachbarsystemen sind alle Systeme, die nicht zu unserem System gehören aber Informationen mit unserem System austauchen. Diese Nachbarsysteme werden abhängig von ihrem Zweck stereotypisiert:

- Nachbarsysteme, die Aufgabe für das Prüfprogramm erledigen und Ergebnisse zurückliefern, oder Information für den Benutzer anzeigen, werden als « Device » stereotypisiert.
- Nachbarsysteme, die für das Prüfprogramm ein Gerät steuern, werden als « System » stereotypisiert
- Nachbarsysteme, die für das Prüfprogramm Eigenschaften der Umgebung wie Temperatur, Luftfeuchtigkeit und Luftdruck erfassen können, werden als « Sensor » stereotypisiert

Die Tabelle 3.1 zeigt die betroffenen Nachbarsystemen des zu entwickelten Systems mit ihren Rollen, Interface und Stereotypen.

Tabelle 3.1: Nachbarsysteme

Nachbarsystem	*Rolle*	*Interface*	*Stereotyp*
Die Bedienoberfläche	sendet Steuerbefehle an das Prüfprogramm und erhält Anweisung und Meldungen des Prüfprogramms	D-Bus GUI	« Device »
Die Robotersteuerung	nimmt Befehle des Prüfprogramms an den Roboter entgegen und meldet den Status der Befehle an das Prüfprogramm zurück	Serielle Schnittestelle RS 232 und Parallel Port	« System »
Die Wetterstation	ermittelt die aktuelle Temperatur, Luftdruck und Luftfeuchtigkeit an das Prüfprogramm zur Bestimmung der Randbedingungen des Tests	Serielle Schnittestelle RS 232	« Sensor »
Die RLC Messbrücke	misst für das Prüfprogramm die Impedanz des zu testenden Pins, und liefert das Ergebnis an das Prüfprogramm zurück	GPIB und Parallel Port	« Device »
Der ESD Detektor	liefert dem Prüfmodul parallel zum ESD Generator das Ergebnis, ob eine Entladung erfolgt ist oder nicht	Parallel Port	« Device »
Der ESD Generator	auf Anweisung des Prüfprogramms löst er einen ESD-Impuls aus und meldet dem Prüfprogramm zurück ob eine Entladung stattgefunden hat oder nicht	Serielle Schnittestelle RS 232 und Parallel Port	« Device »

Kontextabgrenzung mittels Use-Case-Diagramm

Mittels die von Ivar Jacobson vorgeschlagene Use-Case-Diagramm Notation, wird hier die grundsätzliche Kommunikation zwischen unserem System und seinen Nachbarsysteme dargestellt. Die Abgrenzung unseres Systems mittels Use-Case-Diagramm besteht aus:

- dem Prüfmodul in Form eines Rechtecks,
- der Prozesse des Prüfmoduls in Form von Ellipsen,
- Assoziationen zwischen Nachbarsystemen und den Prozessen,
- einem Akteur, der als Strichmännchen dargestellt wird. Über die Bedienoberfläche startet er das Prüfmodul, und sendet Steuerbefehle an das Prüfmodul.
- andere Nachbarsysteme werden als Klasse dargestellt.

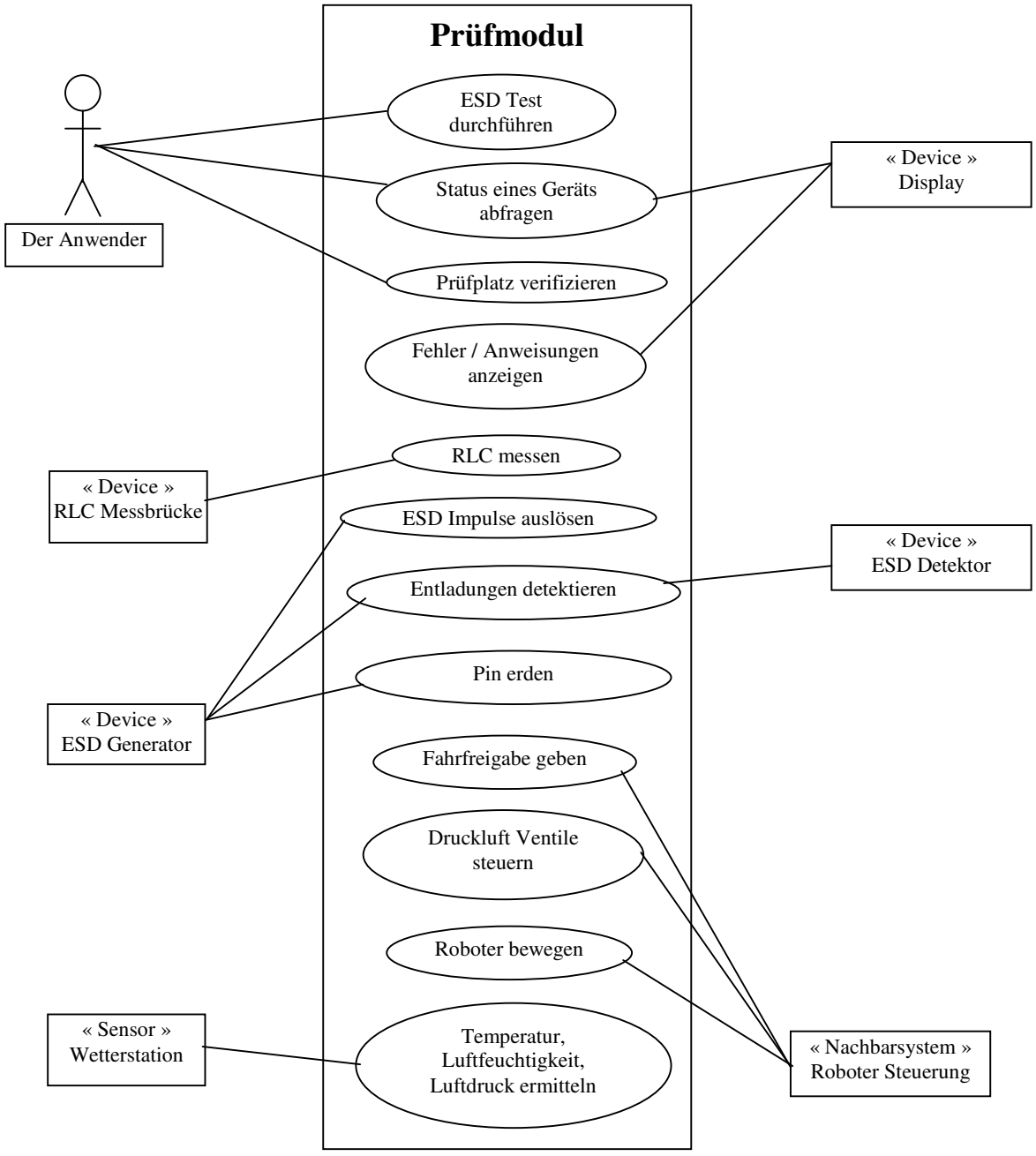


Abbildung 3.1: Kontextabgrenzung mittels Use-Case-Diagramm

Kontextabgrenzung mittels Klassendiagramm

Durch die Abgrenzung des Prüfmoduls mittels Klassendiagramm wird modelliert welche Informationen, das Prüfprogramm mit den Nachbarsystemen austauscht und über welche Schnittstelle die Informationen ausgetauscht werden. Das Klassendiagramm des zu erstellende Prüfmoduls lässt sich wie folgt abbilden:

- Die zentrale Klasse repräsentiert hierbei das Prüfmodul als Black Box
- Die Nachbarsysteme sind als Klassen mit Ihren Schnittstellen mit dem Prüfmodul dargestellt
- Die Assoziation für Ein- und Ausgaben sind gerichtet, und mit den Daten oder Ereignissen, die in das Prüfmodul ein oder ausfließen beschriftet.
- Die Multiplizitäten, die die Anzahl des Auftretens von Nachbarsysteme in Interaktion mit dem Prüfmodul.

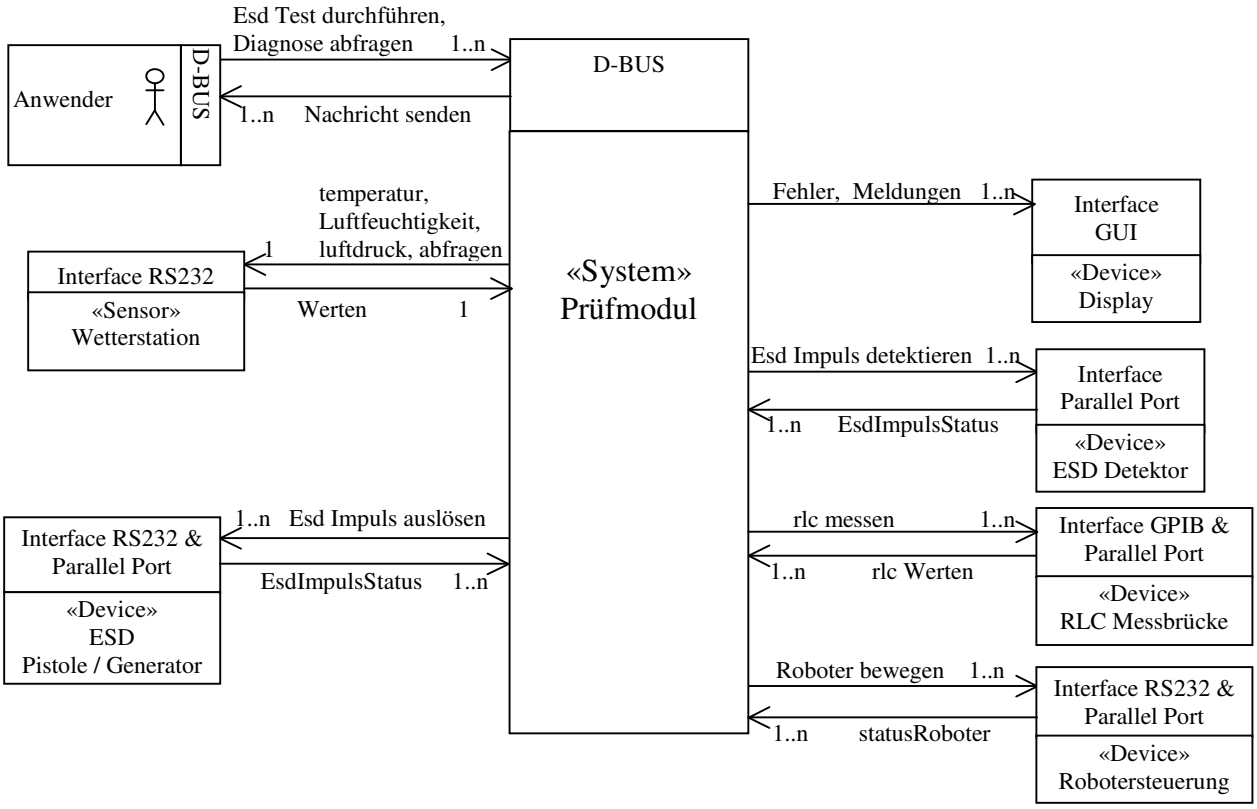


Abbildung 3.2: Kontextabgrenzung mittels Klassendiagramm

Kontextabgrenzung mittels Sequenzdiagramm

Um das zu erstellende System mit einem Sequenzdiagramm abzugrenzen, ist es erforderlich ein mögliches Szenario darzustellen, indem alle Nachbarsystem beteiligt sind. Das in Abbildung 3.3 ausgewählte Szenario stellt eine mögliche Reihenfolge von Interaktionen zwischen dem Prüfprogramm und den Nachbarsysteme, die währen ein ESD Test auftreten können, dar.

Das Prüfprogramm und alle seine Nachbarsysteme sind in dem Sequenzdiagramm als Objekte auf den Lebenslinien notiert. Das zentrale Objekt ist das Prüfmodul, und ist grau markiert. Die Pfeile bzw. die Buchstaben an den Nachrichten zeigen die Richtung der Interaktion bzw. die Sender der Nachrichten zwischen zwei Objekten.

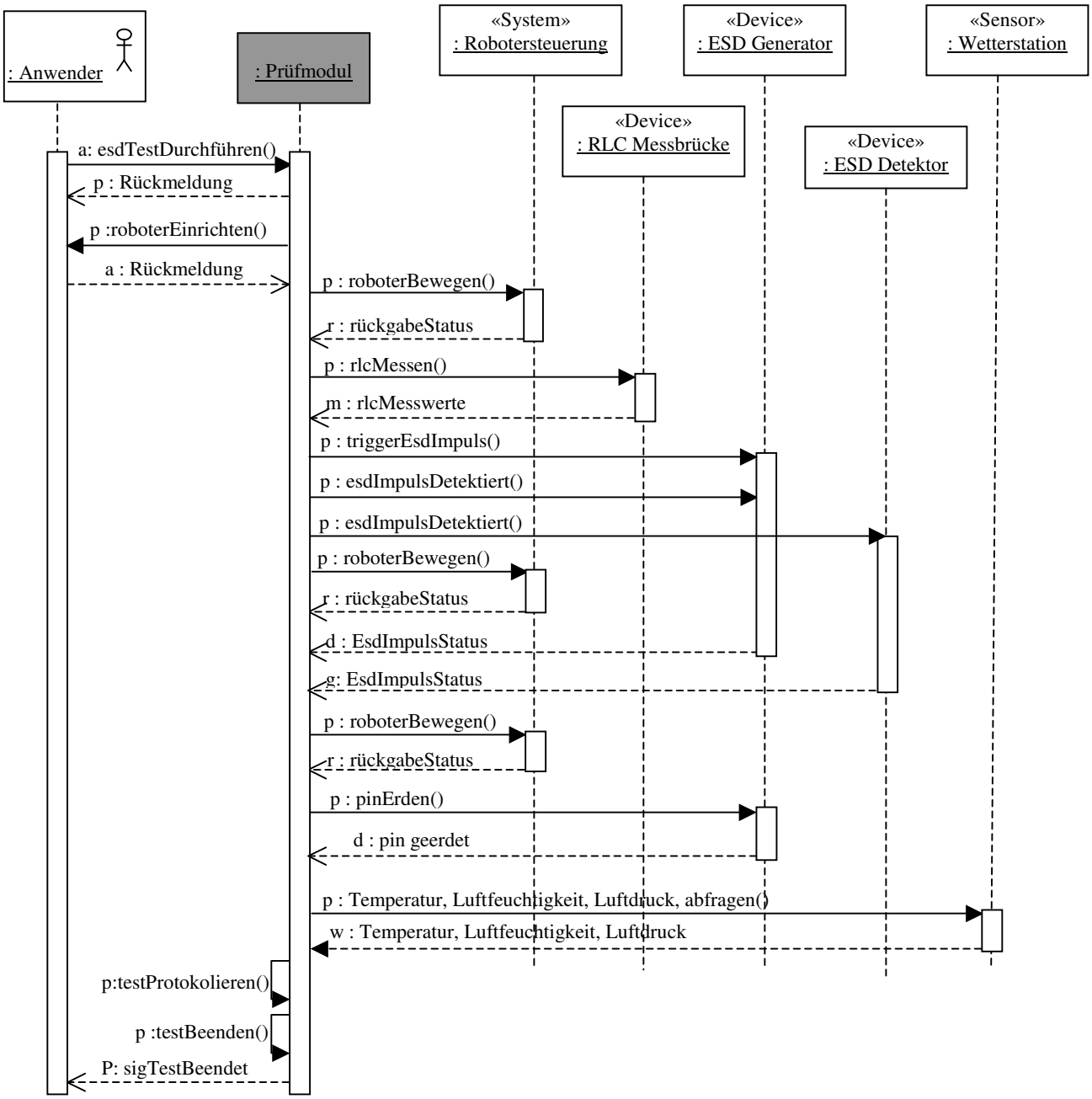


Abbildung 3.3: Kontextabgrenzung mittels Sequenzdiagramm

Auswahl der logischen Kontextabgrenzung

Nachdem das zu entwickelnde System mit den drei oben dargestellten Methoden logische abgegrenzt ist, wird die logische Kontextabgrenzung ausgewählt, die die beste Übersicht unseres Systems mit seiner Umgebung darstellt. Die Tabelle 3.2, stellt die wichtigen Kriterien (Zeilen), um eine logische Abgrenzung des Prüfmoduls darzustellen und die mögliche Abgrenzungen (Spalten).

Tabelle 3.2: Auswahl der logischen Kontextabgrenzung.

	Use-Case-Diagramm	***Klassendiagramm***	***Sequenzdiagramm***
Nachbarsysteme	ja	ja	ja
Schnittstelle	nein	ja	nein
Richtung der Kommunikation	nein	ja	ja
Assoziation	ja	ja	ja
Reihenfolge der Interaktionen	nein	nein	ja
Multiplizitäten	nein	ja	nein

Nach der Analyse der Tabelle 3.2 ist es deutlich, dass das Klassendiagramm die Mehrheit der Kriterien trifft. An zweiter Stelle kommt das Sequenzdiagramm, das uns ein Schritt weiter in der Analyse unseres Systems bringt. Das Sequenzdiagramm stellt die Reihenfolge der Interaktion dar. Aber an dieser Stelle kann die Reihenfolge der Interaktionen noch nicht genau modelliert werden.
Für das Prüfprogramm, das mit Hilfe von externen Geräten einen Test durchführen muss, ist es im ersten Überblick relevant zu wissen:

- mit welchen Geräten oder Systemen kommuniziert das Prüfprogramm
- mit welcher Schnittstelle diese Geräte oder Systeme angesprochen werden
- welche Informationen zwischen dem Prüfprogramm und den Geräten oder Systemen fließen
- in welche Richtung diese Informationen fließen
- Wie oft ein Gerät oder ein System in Kommunikation mit dem Prüfprogramm auftritt.

So zeigt das Klassendiagramm die beste logische Kontextabgrenzung des Prüfmoduls, da es alle die oben genanten Kriterien im Bezug auf Tabelle 3.2 abdeckt.

3.3.2 Physikalischer Kontext

In Abschnitte 3.3.1 wurde das Klassendiagramm als bestes Mittel zur Darstellung der logischen Kontextabgrenzung ausgewählt. Jetzt werden die Verbindungskanäle zwischen den Nachbarsystemen und dem Prüfmodul betrachtet. Um den physikalischen Kontext abzugrenzen, wird erst eine Tabelle aufgebaut, um die Informationen systematisch zu hinterfragen. In dieser Tabelle (siehe Tabelle 3.3) sind für jedes Nachbarsystem alle Ein- und Ausgaben und die verwendende Technologieschnittstelle zur Kommunikation mit dem Prüfprogramm eingetragen.

Tabelle 3.3: Logische Ein-/Ausgabe und Medien für das Prüfmodul

Nachbarsystem	***Ein- / Ausgabe***	***Medium***
Bearbeiter	Eingabebefehle	Mause und Tastatur
	Hinweise und Meldungen	Bildschirm
RLC Messbrücke	Messimpuls / Messergebnisse	GPIB & Parallel Port
ESD Generator	ESD Impuls	RS 232
	Status ESD Impuls	Parallel Port
ESD Detektor	Statue ESD Impuls	Parallel Port
Robotersteuerung	Signal / Befehle	RS 232
	Fahrfreigabe	Parallel Port
Wetterstation	Daten: Temperatur, Luftfeuchtigkeit, Luftdruck.	RS 232

Zum besseren Überblick wird das Verteilungsdiagramm der UML [H&R02] benutzen. Das Diagramm (siehe Abbildung 3.4) zeigt die Schnittstellen mit den externen Komponenten des Systems in Doppelanführungsstrichen.

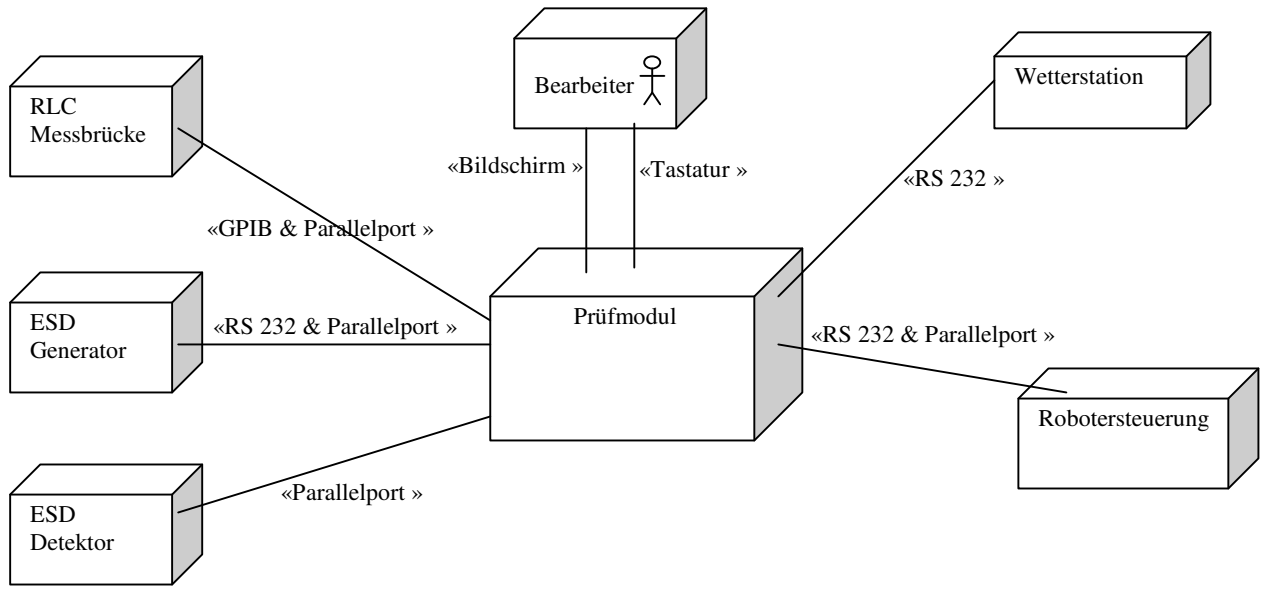


Abbildung 3.4: Der Physikalische Kontext als Verteilungsdiagramm

3.4 Systemprozesse

In diesem Abschnitt werden Systemprozesse ermittelt, die unser System beinhaltet. Um zu diesen Systemprozessen zu kommen ist es wichtig, erst die möglichen Initiatoren dieser Prozesse zu definieren, also die Akteure des Systems. Als Akteure haben wir hier:

- Den Bearbeiter. Hier ist anzumerken, dass der Bearbeiter keinen direkten Kontakt mit dem System Prüfmodul hat, sondern mit dem Prüfmodul ausschließlich über die Benutzeroberfläche kommuniziert. Die vom Prüfmodul gelösten Aufgaben sind vom Bearbeiter initiiert. Der Bearbeiter nutzt also die Benutzerschnittstelle um seine Interessen an das Prüfmodul zu übergeben. Deswegen wird hier der Mensch als Akteur mit modelliert. Er kann folgenden Aktion durchführen:
    - führt einen ESD Test durch
    - fragt den Status eines Geräts ab
    - verifiziert den Prüfplatz
- Ein Sensor, die Wetterstation übermittelt Temperatur, Luftfeuchtigkeit und Luftdruck an das System.
- Die Ein- und Ausgabegeräte:
    - die RLC Messbrücke misst Widerstand und Kapazität eines Pins.
    - der ESD Generator löst ESD und Erdung aus und detektiert den ESD Impuls.
    - der ESD Detektor detektiert den ESD Impuls.
- Die Nachbarsysteme:
    - die Robotersteuerung, die die Kommunikation zwischen dem Prüfprogramm und dem Roboter steuert
    - die Benutzeroberfläche, die die Testdaten an das Prüfmodul bereit stellt und Wünsche des Bearbeiters ans Prüfprogramm übergibt.

Nachdem wir alle möglichen Initiatoren von Prozessen gesammelt haben, nutzen wir die in [H&R02] dargestellte Technik, um den Sachverhalt unseres Systems zu beschreiben. Dazu werden nur die Basis-Systemprozesse (Top-Level-Use-Cases) des Prüfmoduls und ihre Initiatoren betrachtet. Die Erweiterungsprozesse, die zur Durchführung eines Basis-Systemprozesses beitragen, werden nicht dargestellt und als Teil des Basis-Systemprozesses angenommen. Abbildung 3.5 zeigt die Systemprozesse als Anwendungsfälle unseres Systems.

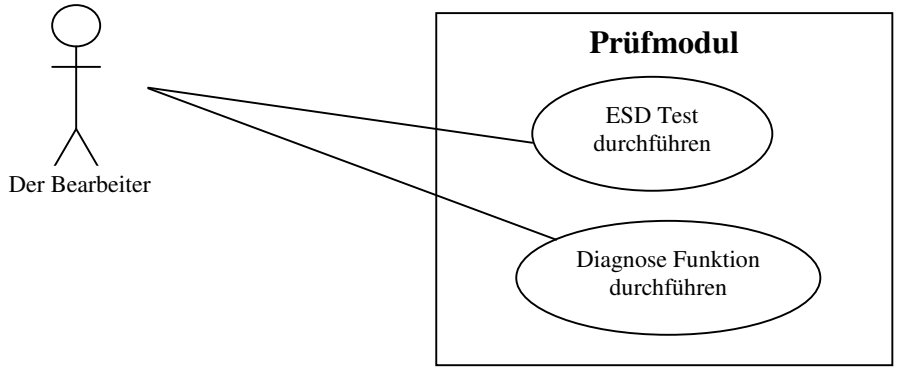


Abbildung 3.5: Systemprozesse

Nachdem die Systemprozesse gefunden wurden, werden wir die in [H&R02] präsentierte Use-Case-Beschreibung erstellen um diese Systemprozesse naher zu spezifizieren. Die Tabelle 3.4 bzw. 3.5 illustrieren die Use-Case-Beschreibung: ESD Test durchführen bzw. Diagnose Funktion durchführen. Besonders in diesen Tabellen ist nicht nur die Intention der

Systemumgebung gegenüber der Reaktion des Systems zu sehen, sondern auch die Intention des Systems gegenüber der Reaktion der Systemumgebung.

Tabelle 3.4: Use-Case-Beschreibung ESD Test durchführen

Name	ESD Test Durchführen	
Akteur	Bearbeiter	
Auslösendes Ereignis	Bearbeiter sendet Testdatenpaket an das Prüfmodul	
Kurzbeschreibung	Das Prüfmodul soll einen ESD Test nach dem in der Testdefinition vorgegebene Testablauf durchführen	
Vorbedingungen	Der Test muss schon vorbereitet sein	
Essenzielle Schritte	**Intention der Systemumgebung**	**Reaktion des Systems**
	Bearbeiter startet das Prüfmodul	Prüfmodul wartet auf Testdaten
	Bearbeiter will einen Test durchführen	Prüfmodul führt den Test durch
	Bearbeiter will den Test anhalten	Prüfmodul hält den Test an
	Bearbeiter will den Test stoppen	Prüfmodul stoppt den Test
	Bearbeiter will den Test fortsetzen	Prüfmoduls setzt den Test fort
	Intention des Systems	**Reaktion der Systemumgebung**
	Prüfmodul will den Status des Prüfplatzes überprüfen	ESD Generator übermittelt seinen Status
		RLC Messbrücke übermittelt seinen Status
		Robotersteuerung übermittelt sein Status
		ESD Detektor übermittelt seinen Status
	Prüfmodul will Roboter bewegen	Robotersteuerung bewegt den Roboter
	Prüfmodul will RLC Messbrücke konfigurieren	RLC Messbrücke passt Konfigurationswerte an
	Prüfmodul will „Einrichtung des Roboters“	Bearbeiter richtet den Roboter ein
	Prüfmodul will neue Prüfspitze benutzen	Bearbeiter wechselt Spitze
	Prüfmodul will R & C Werten Messen	RLC Messbrücke misst und liefert R & C Werte
	Prüfmodul will ESD Impuls auslösen	ESD Generator löst ESD Impuls aus
	Prüfmodul will ESD Impuls detektieren	ESD Generator detektiert ESD Impuls
		ESD Detektor detektiert ESD Impuls
	Prüfmodul will Temperatur, Luftfeuchtigkeit, Luftdruck erfassen	Wetterstation liefert die Werte
Ausnahmefälle	Defektbedingte Einschränkung der Funktionalität des Prüfmoduls	
Nachbedingungen	Prüfmodul erzeugt einen Report des Test und fährt den Roboter in Initialposition	

Tabelle 3.4: Use-Case-Beschreibung Diagnose Funktion durchführen

Name	Diagnose Funktion durchführen	
Akteur	Der Bearbeiter	
Auslösende Ereignis	Bearbeiter wählt Status des ESD Generators aus	
Kurzbeschreibung	Der Bearbeiter möchte überprüfen ob der ESD Generator für den Test funktionsfähig ist	
Vorbedingung	Das Prüfmodul ist gestartet	
Essenzielle Schritte	**Intention der Systemumgebung**	**Reaktion des Systems**
	Bearbeiter startet das Prüfmodul	Prüfmodul wartet auf Befehle des Bearbeiters
	Bearbeiter will Status des ESD Generators prüfen	Prüfmodul prüft Status der ESD Generator
	Intention des Systems	**Reaktion der Systemumgebung**
	Prüfmodul will Status des ESD Generators prüfen	ESD Generator liefert seinen Status an das Prüfmodul zurück
Ausnahmefall		
Nachbedingungen	Anzeigen des Status des ESD Generators	

Die Tabelle 3.4: Use-Case-Beschreibung Diagnose Funktion durchführen kann zur Prüfung aller anderen Geräte des Prüfplatzes angewendet werden.

3.5 Anforderungsermittlung

Die Anforderungen lassen sich mit Hilfe mehrere Techniken erheben [RUPP04]. Abhängig vom System und den Randbedingungen des Projekts gibt es passende Techniken zur Ermittlung der Anforderungen. Die Erhebung der Anforderungen hängt auch von der Hardware ab, die als Nachbarsysteme in der Software beteiligt sind. Das Prüfmodul lässt sich wie folgt klassifizieren:

- Ein zu erweiterndes Altsystem.
- Ein eingebettetes System.
- Ein System mit vielen parallelen Prozessen.

Als Ermittlungstechniken kommen in Frage:

Das Interview: mit dem Experten, der mit dem alten System vertraut ist und die Ziele des neuen Systems kennt.

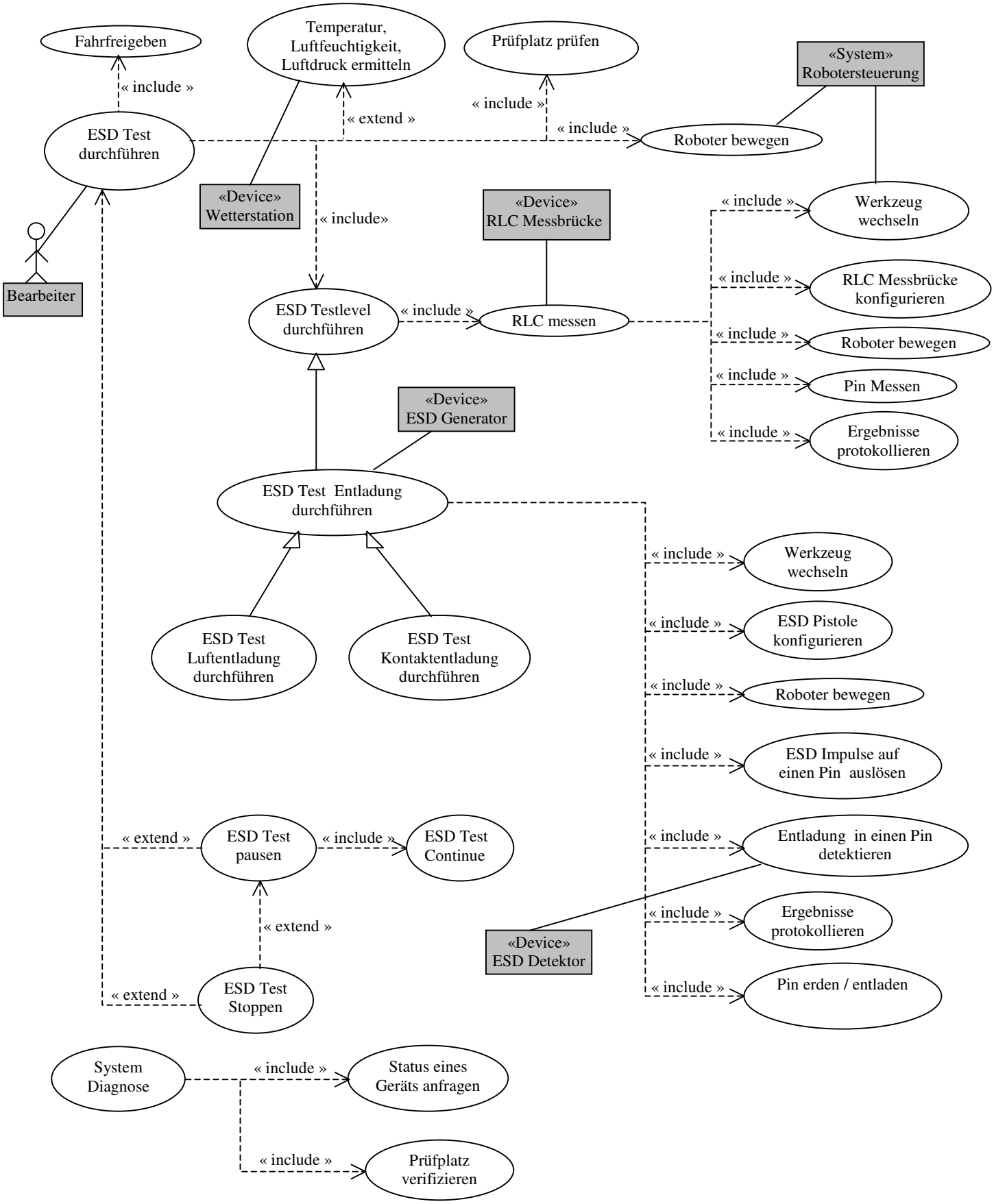
Die Systemarchäologie: Man leitet vom Altsystem die Anforderungen an das neue System ab. Auf diese Weise haben wir in dieser Arbeit sehr schnell die Basis-Anforderungen an das neue System definieren können. Dabei wurden auch die Mängel des alten Systems berücksichtigt.

3.6 Ergebnisse der Anforderungsermittlung

Nach der Ermittlung der Anforderungen, in denen die Ziele, die Stakeholders, die Systemprozesse und der Kontext unseres Systems festgelegt wurden, haben wir eine Liste der Anforderungen gesammelt. Es gibt eine große Auswahl von Methoden um die Anforderungen zu formulieren und klassifizieren [H&C02] [RUPP04], die wir hier benutzen, um die Anforderungen zu definieren. Im Folgenden werden wir diese Klassifizierung der Anforderung erläutern. Eine komplette Liste ist in Abschnitt A.1 der Anhang zu finden.

3.6.1 Funktionale Anforderungen und Use-Case-Beziehungen

Hier wurde während der Anforderungsermittlung festgelegt, was das Prüfmodul tun muss, um eine Elektronik zu testen. Besonders interessant war hier wie das Prüfmodul mit seinen Nachbarsystemen kommunizieren soll. Daraus ist ein Entscheidung für ein Konzept entstanden: die Hardware Komponenten sollen durch ein Treiber-Konzept angesprochen werden. Anderer Punkte waren das Einfügen neuer Funktionalitäten um den ESD Test flexibler zu machen, und die Schwächen des alten Systems zu beseitigen.
Im Folgenden werden wir diese funktionale Anforderung mit einem Use-Case-Beziehungen Diagramm veranschaulichen. Das Use-Case-Beziehungen (siehe Abbildung 3.4) beinhaltet nicht alle funktionale Anforderungen des Prüfmoduls, sondern nur die wichtigsten um einen ESD Test durchzuführen.
Bemerkenswert ist hier, dass der Anwender der Initiator des Hauptprozesses ist.



Abbildungen 3.5: Use-Case-Beziehungen

3.6.2 Technische Anforderungen

Das zu erstellende Prüfmodul soll komplette in „C“ implementiert werden, dabei ist die Bibliothek Glib zu verwenden. Die Entwicklungsumgebung ist ein Linux Betriebssystem. Der Aufbau der Kommunikation mit D-Bus soll mit libdbus-glib API implementiert werden. Die Kommunikation mit Hardware Komponenten erfolgt über Schnittestellen die schon in Abschnitte 3.5 erläutert wurden.

3.6.3 Technologische Anforderungen

Ein Teil der zu entwickeln Software ist das D-Bus Interface, das die Kommunikation mit dem Client ermöglichen wird. Mehr allgemeine Erläuterungen zum D-Bus finden Sie im Abschnitt 4.5, Anwendung in dieser Arbeit finden Sie im Abschnitt 5.2.

3.6.4 Qualitätsanforderungen

Die Qualitätsanforderungen spielen eine sehr große Rolle in unserem Projekt. Eine elektronische Baugruppe, die nicht richtig getestet wurde, kann in Betrieb zu Ausfällen führen.
Die DIN EN ISO 66272 [DIN94] unterteilt die Qualitätsanforderungen in fünf Merkmale. Von diesen fünf kommen einige für unser System in Frage:

- Funktionalität: das Prüfmodul soll einen ESD Test richtig und ordnungsgemäß durchführen, und ausreichend Funktionalität für einen Test nach ISO 10605 bieten.
- Zuverlässigkeit: es soll möglich sein einen Test fehlerfrei durchzuführen ohne Risiko den Prüfling zu beschädigen.
- Benutzbarkeit: die Bedienung des Systems soll leicht verständlich sein.
- Effizient: das System soll eine möglich kurze Reaktionszeit haben.
- Änderbarkeit: es soll möglich sein einfach neue Hardware oder Funktionen zu dem System hinzufugen.

Besonders wichtig ist, dass das Prüfmodul über Funktionalitäten verfügt, die den ESD-Test, über die Auswahl von Parametern flexibel machen kann. Auch, dass das neue Programm Lücken des alten Programms schließt. Das Hinzufügen neuen Funktionen zum Prüfmodul und die Entfernung von Fehler, die beim alten Programm auftraten, waren wichtig für viele Nutzer des Prüfplatzes. Am wichtigsten, war hier die Änderbarkeit und Zuverlässigkeit, die häufig während der Interviews angesprochen wurden. Die Hauptziele dieses Systems sind, eine wartbare und erweiterbare Software zu bekommen.

3.7 Qualitätsmodell

Die in vorherige Abschnitte definierten Qualitätsanforderungen sollen getestet werden. Interessant zu wissen ist hier, wie die Qualitätsziele unseres Systems erreicht werden sollen. Dafür nutzen wir ein Verfahren genannt 3-Schritt-Schema für Q-Modell [Kurt07], um zu sehen in wie fern, wir die prinzipiellen Ziele unseres Systems erfüllen. Dabei wird eine Priorisierung der Qualitätsanforderungen gemacht werden. Die folgende Tabelle 3.5 zeigt das Qualitätsmodell des zu entwickelnden Systems. In dieser Tabelle sind die Qualitätsanforderungen nach Prioritäten absteigend angeordnet:

- Als erste kommt die Qualitätsanorderung mit der höchsten Priorität: Die Korrektheit. Da die Hauptziele des Prüfmoduls die Prüfung einer Elektronik nach dem eingegebenen Testablauf und die Erfassung eines Testreports sind.
- Auf der zweiten Position kommt die Erweiterbarkeit und Wartbarkeit. Das Prüfmodul arbeitet mit externen Geräten, um den Test durchzuführen. Um diese Zusammenarbeit zu vereinfachen wird für jedes externe Gerät mit dem das Prüfmodul arbeiten soll, ein Treiber erstellt, der dynamisch geladen werden kann. Dies soll es ebenfalls ermöglichen leicht ein neues Gerät, ggf. mit einem neuen Treiber, einzusetzen.
- Die dritte Priorität ist von der Betriebsicherheit und Zuverlässigkeit besetzt. Unter Betriebsicherheit und der Zuverlässigkeit ist hier zu verstehen, dass das Prüfmodul die Prüfung einer Elektronik steuert, ohne die zu testenden Elektronik zu gefährden (Kollision).
- Die Flexibilität besetzt die vierte Priorität. Diese Qualitätsanforderung soll ermöglichen, dass der Test unterbrochen und dann wieder fortgesetzt werden kann ohne dass der Verlauf des Tests gestört wird.
- Die letzte Priorität in der Tabelle ist die Effizienz. Diese Qualitätsanforderung garantiert, dass die Kommunikation des Prüfmoduls mit den externen Geräten mit minimaler Verzögerungszeit ablaufen soll.

Tabelle 3.5: Verfahren 3-Schritt-Schema für Q-Modelle

Qualitätsziele des Prüfprogramms	***Qualitätsaspekts***	***Maßstab für Q-Aspekte***	***Konkrete Prüfgegenstände und Sollwerte***	***Prüfungsdurchführung***
Korrektheit	ESD Prüfung einer elektronischen Baugruppe nach einem vorgegebene Testablauf durchführen	Wie weit kann das Prüfmodul eine ESD Prüfung einer Elektronischen Baugruppe nach einem vorgegebenen Testablauf erfüllen?	- Vollständigkeit der Erfüllung eines vorgegebenen Testablaufs {ja, nein, Prozent }	- Elektrische Baugruppe am Prüfplatz anlegen - Testablauf festlegen - Prüfmodul starten - Ergebnis sammeln (Testreport)
Erweiterbarkeit & Wartbarkeit	Neuen Treiber einsetzen	Wie viel Aufwand braucht einen Bearbeiter um einen neuen Treiber zu programmieren	Höhe des Aufwandes { viel, wenig, ein Bisschen } Zeit, Arbeitsstunden	- Treiber mit Hilfe des Tutorials aus der Arbeit erstellen und einsetzen, Zeit protokollieren. - 2 Arbeitstage: ein Bisschen - 4 Arbeitstage wenig - 1 Arbeitsmonat viel
	Treiber dynamisch ladbar	- Wie weit kann das Prüfmodul selbständig aus einem Verzeichnis ein Treiber laden und den passenden Geräteschnittstelle zuordnen?	Verbindung mit dem richtigen Gerät	- Treiber in einem Verzeichnis anlegen - Prüfmodul starten - Status eines Geräts abfragen Die Identifikation des Gerätes auslesen
Betriebsicherheit & Zuverlässigkeit	- Bei Stopp wird der Prüfkopf auf einer sichere Position gefahren	Position des Prüfkopfes, von der aus kollisionsfrei in die Initialeposition fahren kann	Test starten, Stopp drücken. Überprüfen, ob die Initialeposition des Prüfkopfes kollisionsfrei erreicht wird.	- Prüfling anlegen - Prüfmodul starten - Stopp auslösen - Prüfung fortsetzen
Flexibilität	Unterbrechung eines laufenden Tests durch eine Sonderaufforderung (Aufgabe)	Lückenlose Fortsetzung des Tests nach der Sonderaufgabe	Doppelte oder fehlende Testschritte (Ja / Nein)	- Anfordern von Sondersaufgaben (Werkzeugwechsel, Selbstverifikation) - Prüfen des Testablaufes auf Lücken oder Doppelte.
Effizienz	Verzögerungsfreie Interaktion mit Klienten	Reaktionszeit, Antwortzeit.	Dauer der Reaktionszeit { Lang, kurz, durchschnittlich }	- Senden von Kommandos und Messen der Antwortzeit

3.8 Zusammenfassung

In diesem Kapitel wurden die Softwareziele des Prüfmoduls noch einmal erläutert, die Stakeholder benannt, den Umfang und die Anforderungen des Prüfmoduls ermittelt. Ein wichtiger Punkt dieses Kapitels war die Abgrenzung des Kontextes unseres Systems. Es wurde eine logische und physikalische Kontextabgrenzung des zu erstellenden Systems gemacht. Bei der logischen Kontextabgrenzung, war es nicht von Anfang an einfach festzustellen welches Diagramm einen besseren Überblick des zu entwickelnden Systems geben würde. Daher wurden drei verschiedene Diagrammtypen verglichen: Klassendiagramm, Use-Case-Diagramm und Sequenzdiagramm. Für das Prüfprogramm hat sich dabei das Klassendiagramm am besten bewährt, da es die Kommunikation des Prüfmoduls mit seinen externen Geräten besser zeigt. Bei der physikalischen Kontextabgrenzung wurden die Hardwarerandbedingungen unseres Systems erfasst. Noch wichtiger war die Ermittlung der Systemprozesse des Prüfmoduls, die mittels Use-Case-Beschreibung veranschaulicht wurden. Am wichtigsten war die Erhebung der Anforderungen an das zu entwickelnden System, dabei wurde durch das Interview und die Systemarchäologie eine Liste von Anforderungen erstellt, die im Laufe der Zeit angepasst wurde. Schließlich wurde ein Qualitätsmodel (Q-Modell) aus Anforderungen abgeleitet, um das Anpassen der Umsetzung mit den Anforderungen später kontrollieren zu können.

Kapitel 4

Konzept

In dem vorherigen Kapitel wurde beschrieben mit welchen Komponenten und über welche Schnittstelle das in dieser Arbeit entwickelte System kommuniziert. Jetzt wollen wir tiefer gehen und genau die Informationen, die ausgetauscht werden, präsentieren. Es wird in diesem Kapitel die Interaktion zwischen den Komponenten des zu entwickelten Systems definiert. Ein Konzept wird erstellt um die definierten Anforderungen umsetzen zu können und die Wartbarkeit und Erweiterbarkeit des Prüfmoduls zu gewährleisten.
Dieser Abschnitt umfasst die in der vorliegenden Arbeit verwendete Architektur, den verwendeten Mechanismus um Daten auszutauschen, einen präzisen Ablauf des Prüfmoduls und das Konzept, mit dem das Prüfmodul mit seinen Nachbarsystemen kommuniziert.

4.1 Architektur

Die neue Generation des Verwaltungssystems des Prüfplatzes hat eine Client – Server Architektur (siehe Abbildung 4.1). Der Client und der Server werden unabhängig von einander entwickelt. In dieser Arbeit wurde nur der Server, genannt Prüfmodul oder Prüfprogramm, entwickelt. Der Client ist die Bedienoberfläche. Der Datenaustausch zwischen dem Client und dem Server wird über D-Bus ermöglicht.
D-Bus ist ein Nachricht-Bus-System, das die Kommunikation zwischen der Bedienoberfläche und dem Prüfprogramm ermöglicht.

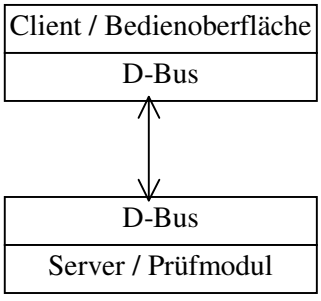


Abbildung 4.1: Client – Server Architektur

4.1.1 Software-Architektur

Die Client – Server Architektur zeigt schon wie das System Prüfmodul mit der Bedienoberfläche kommuniziert. Außer der Bedienoberfläche, kommuniziert das Prüfmodul mit externen Komponenten. In diesem Abschnitt werden drei Architekturen

der Kommunikation des Prüfmoduls mit seinen externen Komponenten dargestellt. Eine ist die Architektur des alten Systems und die zwei anderen sind die möglichen Architekturen des zu entwickelten Systems. Unter diesen letzen beiden Architekturen wird die Architektur ausgesucht, die die Modularität, Wartbarkeit und Erweiterbarkeit des Prüfprogramms am besten ermöglicht.

Die Bestehende Architektur

Diese Architektur stellt vor, wie das alte Steuerprogramm des Prüfplatzes mit seinen externen Komponenten kommuniziert. Die Kommunikation mit jeder externen Komponente des Prüfmoduls wird durch ein Modul hergestellt. Das Prüfprogramm verfügt über einen Timer, der regelmäßig Anfragen entsprechend ihrer Priorität und ihre Anordnung in der Warteschlange einfügt. Der Timer nimmt das oberste Element der Warteschlange und übergibt es nacheinander an alle Module. Das zuständige Modul bearbeitet die Anfrage und trägt die Ergebnisse in die Anfrage ein. Jedes Modul kann weitere Elemente in die Warteschlange einfügen. Die Bearbeitung der Anfragen durch die Module geschieht asynchron.

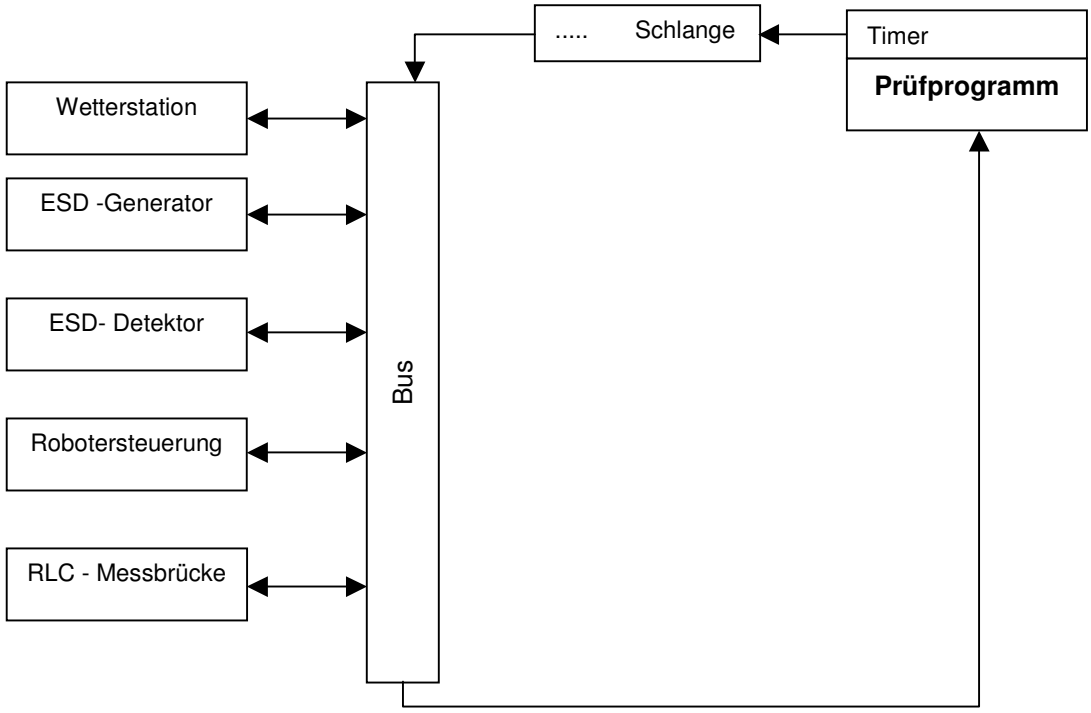


Abbildung 4.1: Bestehende Architektur

In dieser Architektur übernimmt jedes Modul die Verantwortung für die Kommunikation mit seiner Hardware. Die Schnittstelle mit denen jedes Modul mit seiner Hardware kommuniziert sind in der Tabelle 4.1 dargestellt. Diese Architektur hat nur einen Vorteil und viele Nachteile. Eine mögliche Lösung dieser Nachteile wird in den kommenden Architekturen dargestellt. Die Vor- und Nachteile der bestehenden Architektur sind:
Vorteil:

- kein Modul ist in der Lage das Prüfmodul zu blockieren (Timeout Überwachung)

Nachteile:

- Alle Module sollen die übergebende Nachricht bearbeiten (Zeitkosten)
- Mögliche teilweise Ausführung von Aktion, wenn ein Stopp oder Pause Befehl erfolgt.
- Timer wird bis zum Ende des Programms durchgeführt
- Wiederherstellung schwierig zu machen
- Alle Module sehen die übergebenden Nachrichten auch wenn sie nicht für es bestimmt ist
- Ein Modul kann dann Daten oder Nachrichten modifizieren, die für ein anderes Modul bestimmt sind
- Komplexe Bearbeitung der Nachricht, weil Nachrichten sofort weitergeschickt werden müssen und nicht erst dann, wenn die Aufgabe erledigt ist. Der Bus kann nur eine Botschaft gleichzeitig transportieren.

Architekturen *Stern 1* mit High- und Low-Level Funktionsmodul

Die Architektur *Stern 1* erfüllt eine der definierten Anforderungen des in der vorliegenden Arbeit entwickelten Systems, nämlich das Ansprechen der Hardwarekomponenten des Prüfplatzes durch Treiber. Diese Treiber stellen Funktionalitäten ihrer Hardware an das Prüfmodul zur Verfügung. Die Tabelle 4.1 zeigt die Hardwarekomponenten, ihre logische Ein.- und Ausgaben und mit welcher Schnittstelle sie kommunizieren.

Tabelle 4.1: logische Ein-/Ausgabe und die Medien der High- und Low-Level Funktionsmodul

Hardwarekomponenten	***Ein- / Ausgabe***	***Medium***
RLC- Messbrücke	Signal, Werte, Masse-Relais	GPIB & Parallelport
ESD- Generator	Signal & Druckluft Ventile	RS232 & Parallelport
Wetterstation	Messwerte / Signal	RS 232
Robotersteuerung	Signal & Fahrfreigabe	RS232 & Parallelport
ESD- Detector	Signal	Parallelport

Aus dieser Tabelle, entsteht die Architektur *Stern 1* mit High- und Low-Level Funktionsmodul (siehe Abbildung 4.2). In dieser Architektur, gibt es Hardwarefunktionen, die direkt durch den Treiber der Hardware aufgerufen werden. Dies sind High-Level Funktionsmodule. Außerdem gibt es auch Treiber, deren Funktionen, nur indirekt über Funktionen einen anderen Treiber aufgerufen werden. Diese indirekte Kommunikation des Prüfmoduls mit seiner Hardware beschreibt die Low-Level Funktionsmodule. Ein Beispiel des Low-Level Funktionsmoduls ist das Aufrufen der „Fahrfreigabe“ des Roboters. Die Fahrfreigabe in dieser Architektur ist eine Funktion der Robotersteuerung und gehört dem Treiber Robotersteuerung. Aber die Aktivierung der Fahrfreigabe erfolgt durch den Parallelport. So wird noch eine Funktion des Parallelportstreibers aufgerufen um die Funktion Fahrfreigabe Komplette auszuführen. Ein anderes Beispiel in dieser Architektur ist der ESD – Detektor Treiber,

der alle seine Funktionalitäten durch die Funktionen des Parallelport Treibers ergänzen muss.

Tr. Steht in der Abbildung für Treiber

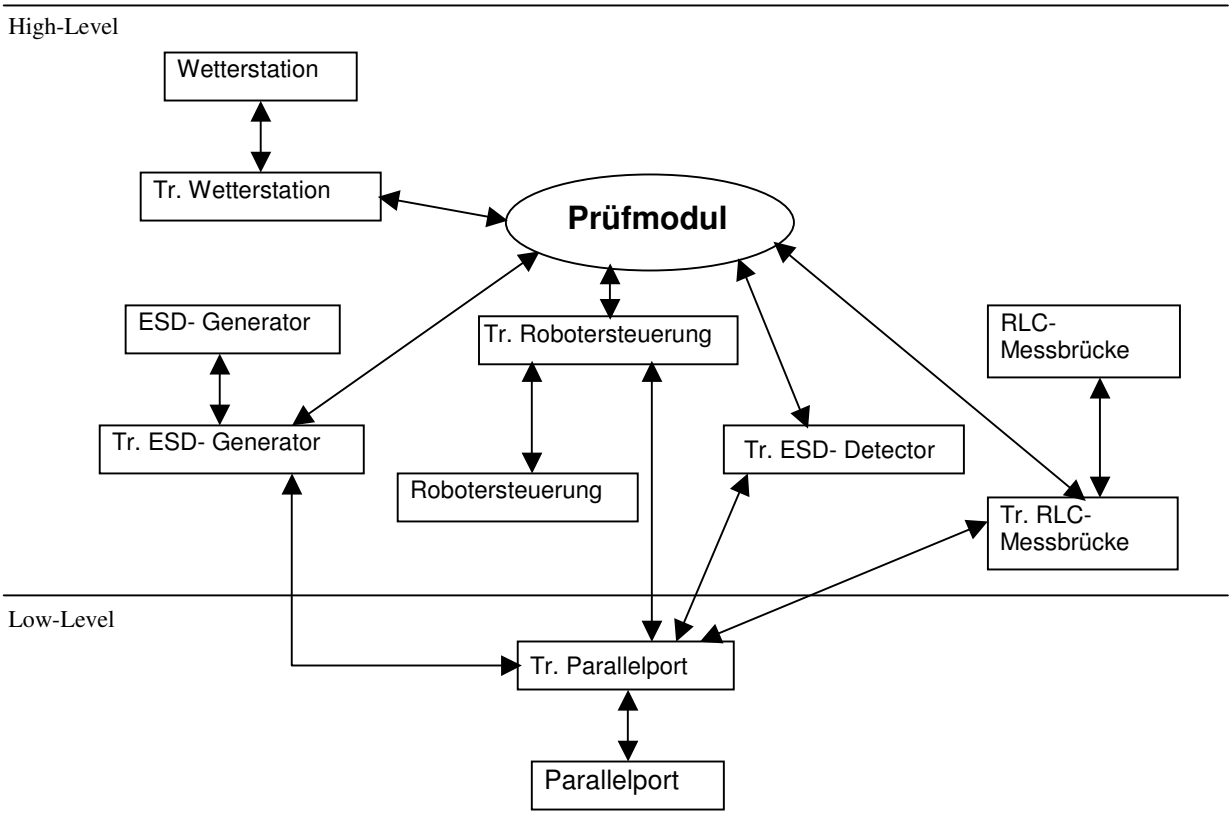


Abbildung 4.2: Architektur *Stern 1* mit High- und Low-Level Funktionsmodul.

Die Vor- und Nachteile diese Architektur sind:
Vorteile:

- Ein Modul kann nur Daten oder Nachrichten bearbeiten, die für es bestimmt sind.
- Bessere Verfolgbarkeit der Zustände der Prozesse.
- Das Prüfmodul übermittelt einen Befehl direkt an das zuständige Modul, kein anderes Modul bekommt den Befehl.

Nachteile:

- Ansprechen der Funktionalitäten einiger Module durch den Treiber des Parallelports. Erhöhter Datenverkehr des Prüfmoduls.
- Funktionalität eines Gerätetreibers, die nicht durch das zugehörige Gerät ausgeführt werden.
- Komplexere Architektur, Wartbarkeit und Erweiterbarkeit schwierig
- Viel Aufwand beim Einsetzen neuer Treiber, die mit dem Parallelport kommunizieren müssen, weil man den Treiber des Parallelports auch berücksichtigen muss.
- Bestimmte Funktionalitäten die man für den Messablauf braucht, führen zu erhöhtem Materialverschleiß am Prüfplatz. Z.B. muss während einer RLC – Messung die Masseverbindung über ein Relais hergestellt werden. Da der Treiber der Messbrücke keinen Überblick über die nächsten Prüfschritte hat, muss er für jede Messung den Massekontakt schließen und nach der Messung

wieder öffnen, auch wenn der nächste Testschritte wieder eine Impedanzmessung ist. Dies führt zu einem unnötigen hohen Verschleiß im Relais.
- Kommunikation mit der Hardware durch mehrere Treiber. Um das Signal „Fahrfreigabe“ anzusteuern muss man den Robotersteuerungs- und den Parallelporttreiber ansprechen.
- Um einen neuen Treiber, der mit dem Parallelport kommunizieren muss, einzusetzen, muss man den Code von drei Modulen ändern anstatt nur von zweien wie bei der unteren Architektur. Das erfordert mehr Programmieraufwand.

Architektur *Stern 2* mit Ein-Level Funktionsmodul

In der Architektur *Stern 2* werden die Hardwarekomponenten neue aufgeteilt. Es wird hier eine Verfeinerung der Architektur *Stern 1* gemacht. Die Tabelle 4.2 zeigt die neue Aufteilung der Hardwarekomponenten mit ihren Ein- oder Ausgabe und ihrer Schnittstelle. In dieser neuer Modellierung ist der ESD Detektor nicht mehr als Hardwarekomponente implementiert. Er wird durch den Parallelport ersetzt, der in Low-Level der Architektur *Stern 2*, die Funktionalitäten des ESD Detektor übernahmt.

Tabelle 4.2: logische Ein-/Ausgabe und die Medien der Ein-Level Funktionsmodul

Hardwarekomponenten	***Ein- / Ausgabe***	***Medium***
RLC- Messbrücke	Signal / Werte	GPIB
ESD- Generator	Signal	RS232
Wetterstation	Messwerte / Signal	RS 232
Robotersteuerung	Signal	RS232
Parallelport	Signal, Druckluft Ventile, Fahrfreigabe	Parallelport

Aus dieser Tabelle ergibt sich die Architektur *Stern 2* mit Ein-Level Funktionsmodul (siehe Abbildung 4.3). Das Prüfmodul hier als zentrales Element ruft jede einzelne Funktionalität seiner Hardwarekomponente durch den zuständigen Treiber auf.

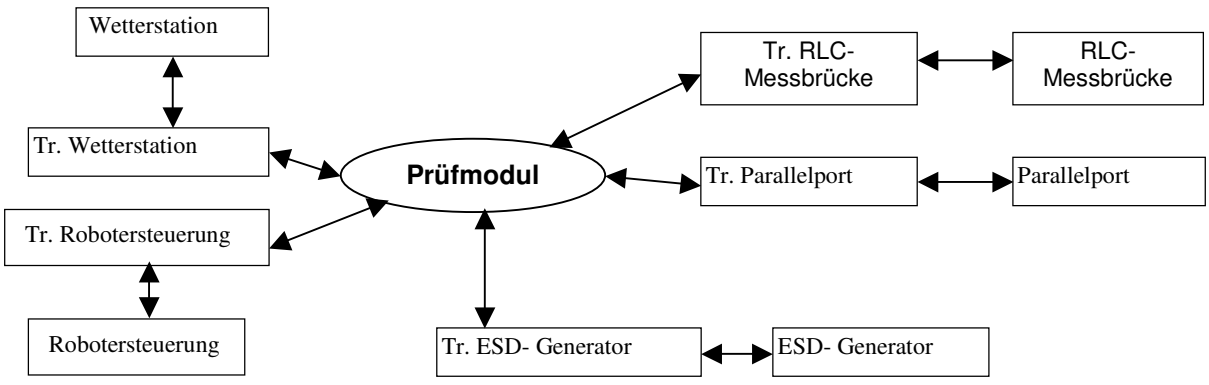


Abbildung 4.3: Architektur *Stern 2* mit Ein-Level Funktionsmodul.

Die Vor- und Nachteile diese Architektur sind:
Vorteile:

- Das Prüfprogramm kann jede Hardware indirekt durch seinen Treiber ansprechen.
- Das Prüfmodul übermittelt einen Befehl direkt an den zuständigen Treiber, kein anderer Treiber bekommt den Befehl.
- Ein Modul kann nur Daten oder Nachrichten bearbeiten, die für es bestimmt sind.
- Die Architektur ist übersichtlich, jede Kommunikation mit einer Hardware erfolgt durch einen Treiber. Erweiterbarkeit und Wartbarkeit sind einfacher.
- Bessere Verfolgbarkeit der Zustände der Prozesse.
- Beim Umsetzen neuer Funktionalität, die vom Parallelport abhängig ist, braucht man nur den Parallelporttreiber und das Prüfmodul anzupassen und nicht noch zusätzlich einen neuen Treiber umzusetzen.
- Neue Treiber einfacher einsetzbar, weil jede Hardware durch genau einen Treiber ansprechbar ist, der alle seine Funktionen besitzt.
- Neue Funktionalitäten von Treibern einfacher umsetzbar, da die Änderung nur bei einem Treiber erfolgt und nicht bei mehreren.
- Das Treiber-Konzept wird dadurch besser beschrieben.
- Alle Treiber funktionieren über das zentrale Modul (Prüfmodul).

Nachteile:

- Das Prüfmodul übernimmt mehr Aufgaben. Da es nur die Funktionen der Treiber nutzt und den ganzen Ablauf des Prüfmoduls steuert.

Auswahl der Architektur

Nach den oben dargestellten Architekturen wird die in dieser Arbeit entwickelnde Software die Architektur *Stern 2* verwendet. Die Darstellung von Vorteilen und Nachteilen jeder Architektur zeigt, dass die Architektur *Stern 2* mehr Modularität unseres Systems und eine bessere Anwendung des Treiberkonzepts bietet. Die Verteilung der Treiber in ein Funktionsmodul hilft dabei die Wartbarkeit und Erweiterbarkeit des Prüfmoduls zu erreichen.

Die Abbildung 4.4 zeigt, die neue Architektur des Steuerprogramms des Prüfplatzes. Der Module-Loader lädt alle Treiber der externen Geräte. Beim Laden werden alle Funktionen jedes einzelnen Treibers an das Prüfmodul übergeben. Durch diese Funktionen kann das Prüfmodul während des Tests jedes externe Gerät ansprechen. Mit dem D-Bus Interface wird die Kommunikation mit der Bedienoberfläche gesteuert. Nachdem das Prüfprogramm gestartet ist, werden die Treiber geladen. Dann läuft das Prüfprogramm in einer Endlos-Schleife. In dieser Schleife wird das Prüfprogramm Testdaten über D-Bus geliefert bekommen und ESD-Tests durchführen. Das Prüfprogramm sendet auch über D-Bus Meldungen an die Bedienoberfläche zurück.
Es wird gezeigt wie das Prüfmodul die Basis Funktionalitäten jedes Treibers anspricht. Jeder Treiber hat spezielle Funktionen, die für ihn spezifisch sind.

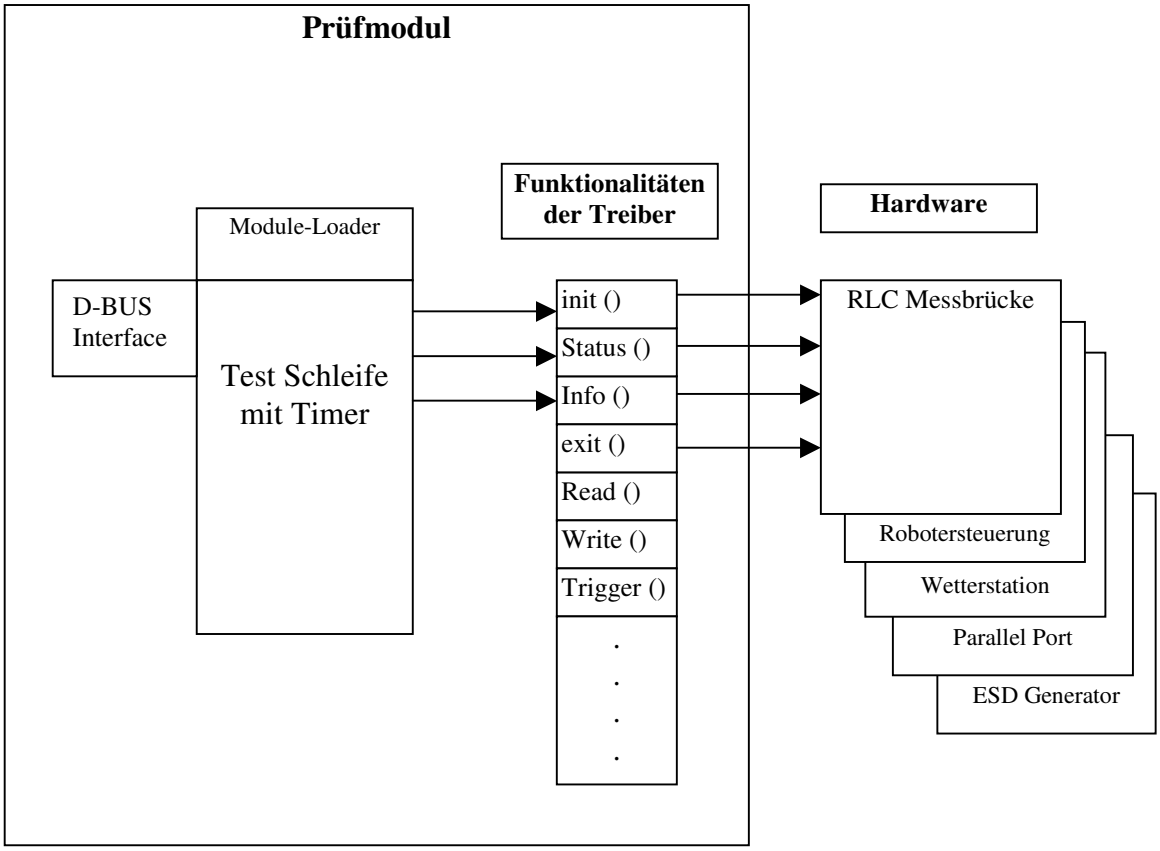


Abbildung 4.4: Neue Architektur des Steuerprogramms des Prüfplatzes

4.2 Abläufe präzisieren (Aktivitätsdiagramm)

In diesem Abschnitt werden die Aktivitäten aus den Test-Prozessen des Prüfmoduls in Form von Aktivitätsdiagramm modelliert. Die UML-Ausdruckmittel werden verwendet um diese Aktivitäten darzustellen. Das System Prüfmodul hat zwei Hauptprozesse: ESD-Test durchführen und Diagnose Funktion durchführen. Die Aktivitäten dieser Prozesse werden mit Hilfe von Aktivitätsdiagrammen aus der UML2 dargestellt.
Um diese Aktivitätsdiagramme besser zu verstehen werden einige verwendete Notationen erläutern:
Die Bedingung „*pin.next = NULL*“ bedeutet, dass der letzte Pin des aktuellen Steckers schon bearbeitet wurde; und „*pin.next <> NULL*“ bedeutet, dass noch Pins zur Bearbeitung anstehen.
Die Bedingung „*plug.next = NULL*“ bedeutet, dass der letzte Stecker der Elektronik schon bearbeitet wurde; und „*plug.next <> NULL*“ bedeutet, dass noch Stecker zur Bearbeitung anstehen.
Die Bedingung „*testlevel.next = NULL*“ bedeutet, dass der letzte TestLevel der Testdefinition schon bearbeitet wurde; und „*testlevel.next <> NULL*“ bedeutet, dass noch Testlevel zur Bearbeitung anstehen.

Die g*rau markierte Aktivitäten* werden in einem anderen Aktivitätsdiagramm beschrieben.

Bei der Durchführung des ESD-Tests, ist anzumerken, dass der Roboter an seiner Hand die ESD-Pistole und drei Prüfspitzen trägt: die Messspitze, die Erdungsspitze, und die Kontaktentladungsspitze oder die Luftentladungsspitze je nach Testlevel. Das Prüfprogramm achtet darauf, dass sich die für den Test richtige Spitze am ESD-Tester befindet und steuert den Roboter zu den zu testenden Positionen.
Abbildung 4.5 zeigt den Ablauf des Prüfmoduls. Nach dem Starten lädt das Prüfmodul zunächst die Treiber, initialisiert diese und öffnet einen Port über D-Bus. Sollte bei einem diesen Schritte ein Fehler auftreten, beendet sich das Prüfprogramm. Ansonsten wartet es ab jetzt auf Befehle des Benutzers über D-Bus. Diese Befehle können sein:

- Ein Startbefehl mit den Testdaten um den ESD-Test durchzuführen,
- Ein Stoppbefehl um den ESD-Test zu unterbrechen,
- Ein Pausebefehl um den ESD-Test anzuhalten,
- Ein Fortfahrensbefehl um den ESD-Test fortzusetzen.

Tritt bei der Durchführung des Tests ein Fehler auf, wird der Fehler über D-Bus an den Bearbeiter übermittelt und der Test unterbrochen.

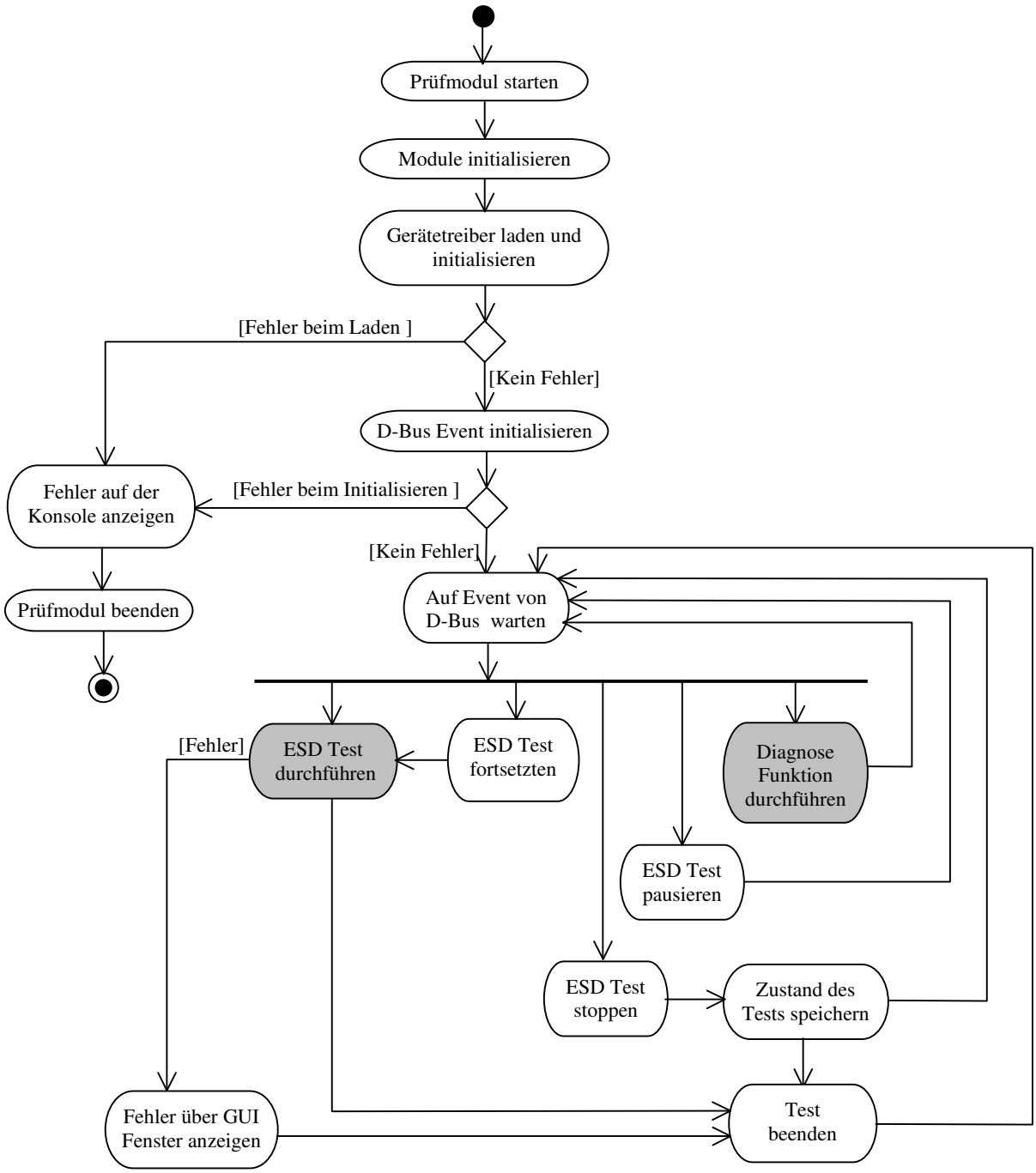


Abbildung 4.5: Aktivitätsdiagramm Ablauf des Prüfmoduls

Die Abbildung 4.6 zeigt den Ablauf eines ESD-Tests. Während dieses Ablaufs, muss der Bearbeiter einige Tätigkeiten unter Anweisung des Prüfprogramms erledigen:

- „Test vorbereiten“ : der Bearbeiter soll, den Roboter anlernen.
- „Test konfigurieren“ : der Bearbeiter soll das passende HBM und Prüfspitze am Roboter montieren.

Nachdem der Bearbeiter die Aufforderung des Prüfprogramms erledig hat, wird der Client über D-Bus eine Quittierung an das Prüfprogramm übermittelt.
In einem ESD Testlevel können folgenden Kombinationen aufftreten:

- Eine Messung

- Eine Kontaktentladung mit Erdung
- Eine Kontakentladung mit Erdung und anschließende Impedanzmessung
- Eine Luftentladung mit Erdung
- Eine Luftentladung mit Erdung und anschließende Impedanzmessung

Diese Kombination werden später mit Aktivitätsdiagramme beschrieben.

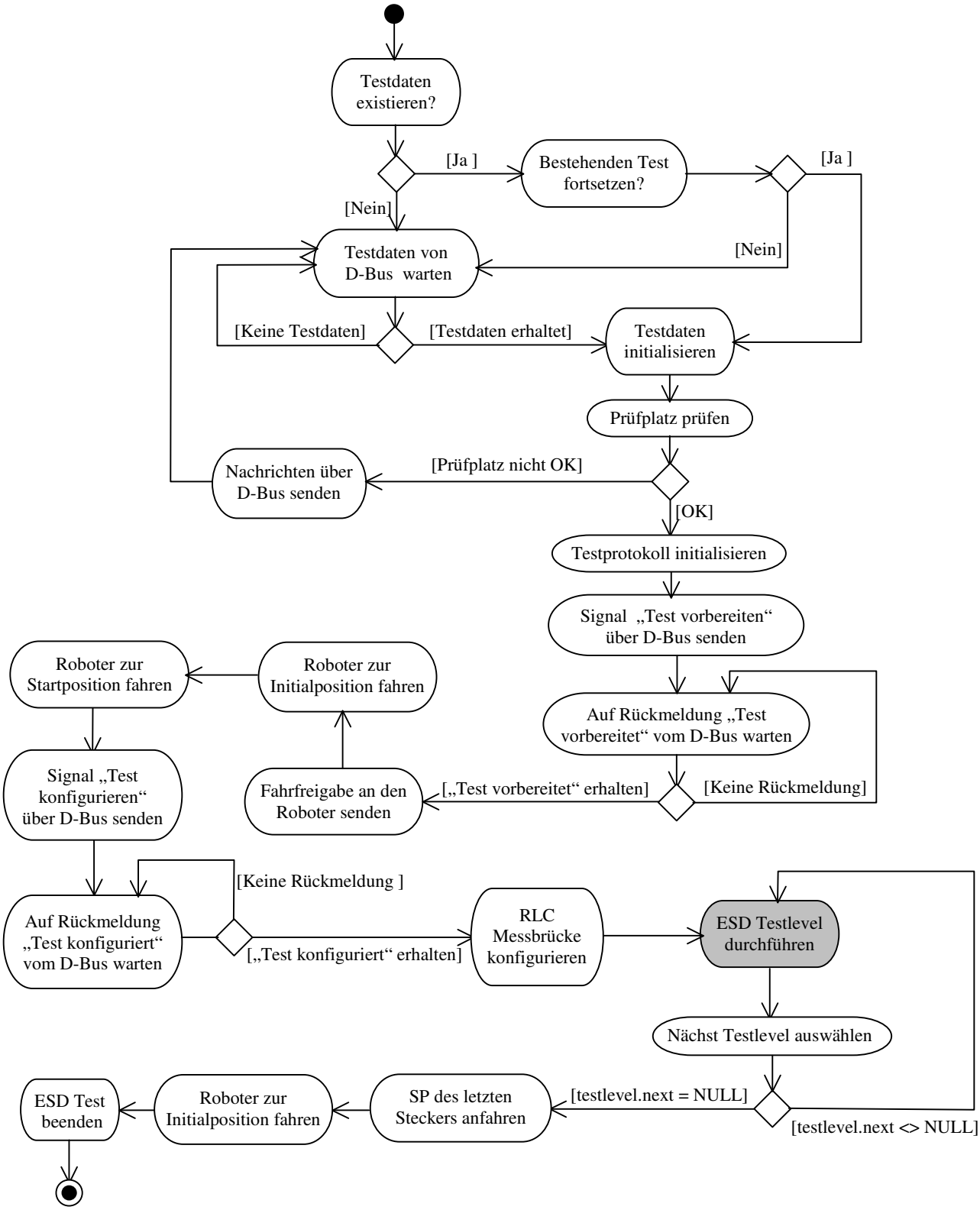


Abbildung 4.6: Aktivitätsdiagramm ESD-Test durchführen

In einem ESD-Test wird vor und nach der Störbeaufschlagung eine Impedanzmessung durchgeführt. Vor dem Messvorgang wird das Masse-Relais zugeschaltet und danach ausgeschaltet. Bei der Messung werden Widerstand und Kapazität jedes einzelnen Pins des Prüflings gemessen und die Werte ins Prüfprotokoll geschrieben. Die Abbildung 4.7 zeigt den Ablauf einer RLC Messung. Der Roboter benötigt sein Messwerkzeug um den Test durchzuführen.

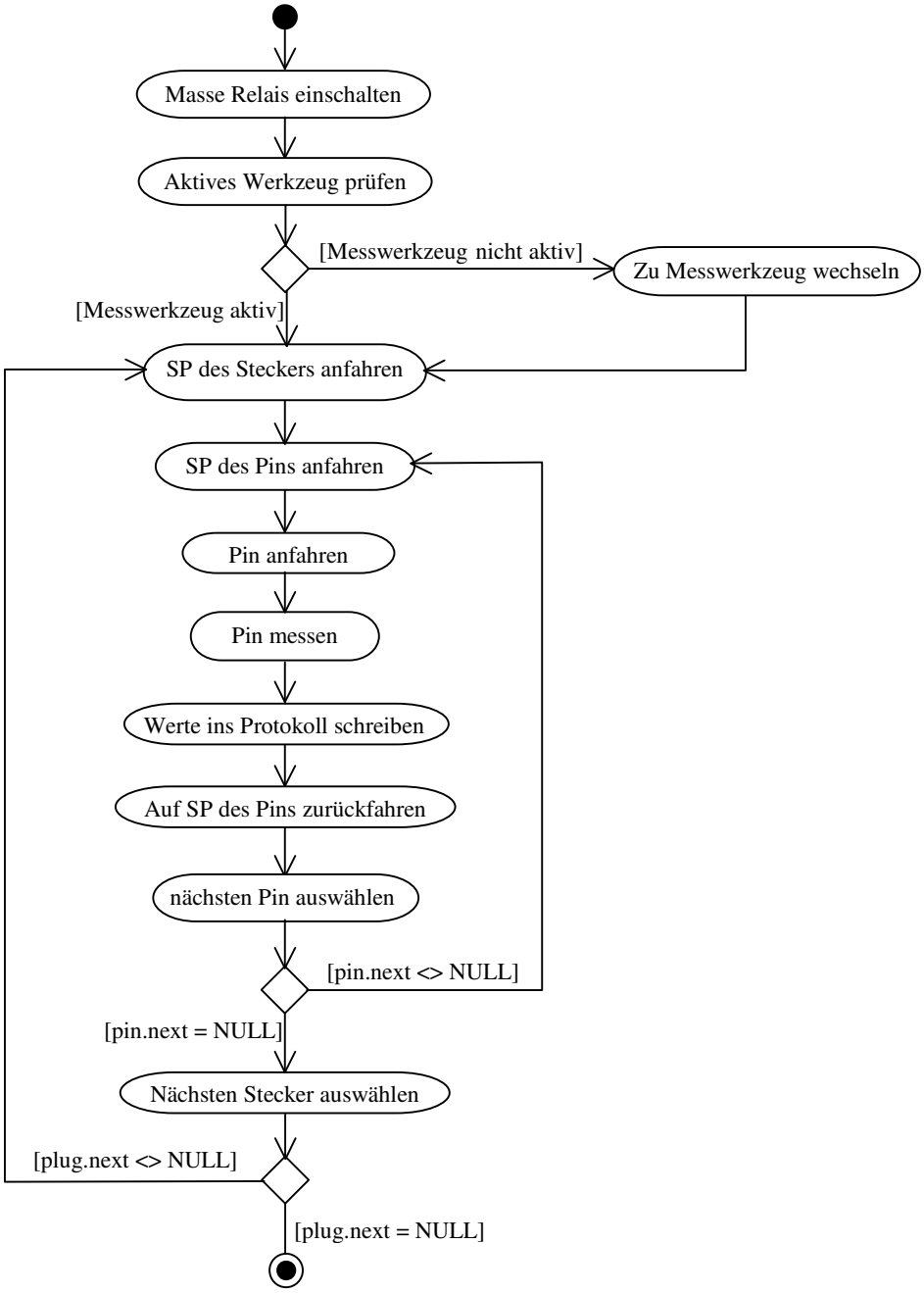


Abbildung 4.7: Aktivitätsdiagramm RLC Messung

Eine der zwei Entladungstypen, die während des Tests durchgeführt werden, ist die ESD-Kontaktentladung. Die Konfiguration des ESD-Generators stellt den Testtyp und die Spannung, mit der der Pin aufgeladen wird, ein. Während der Kontaktentladung wird jeder Pin direkt mit der Prüfspitze berührt bevor der ESD-Impuls ausgelöst wird.

Anschließen wird den Pin durch eine Erdung entladen. Dieser Vorgang wird nach ISO 10605 üblicherweise drei Mal für jeden Pin durchgeführt. Die Abbildung 4.8 zeigt den Ablauf einer ESD – Kontaktentladung. In dieser Abbildung sind folgende Notationen zu beachten:
bi : ist die Anzahl der benötigten Prüfimpulse pro Pin und
ci : ist die Anzahl der schon ausgelösten Prüfimpulse für einen Pin.

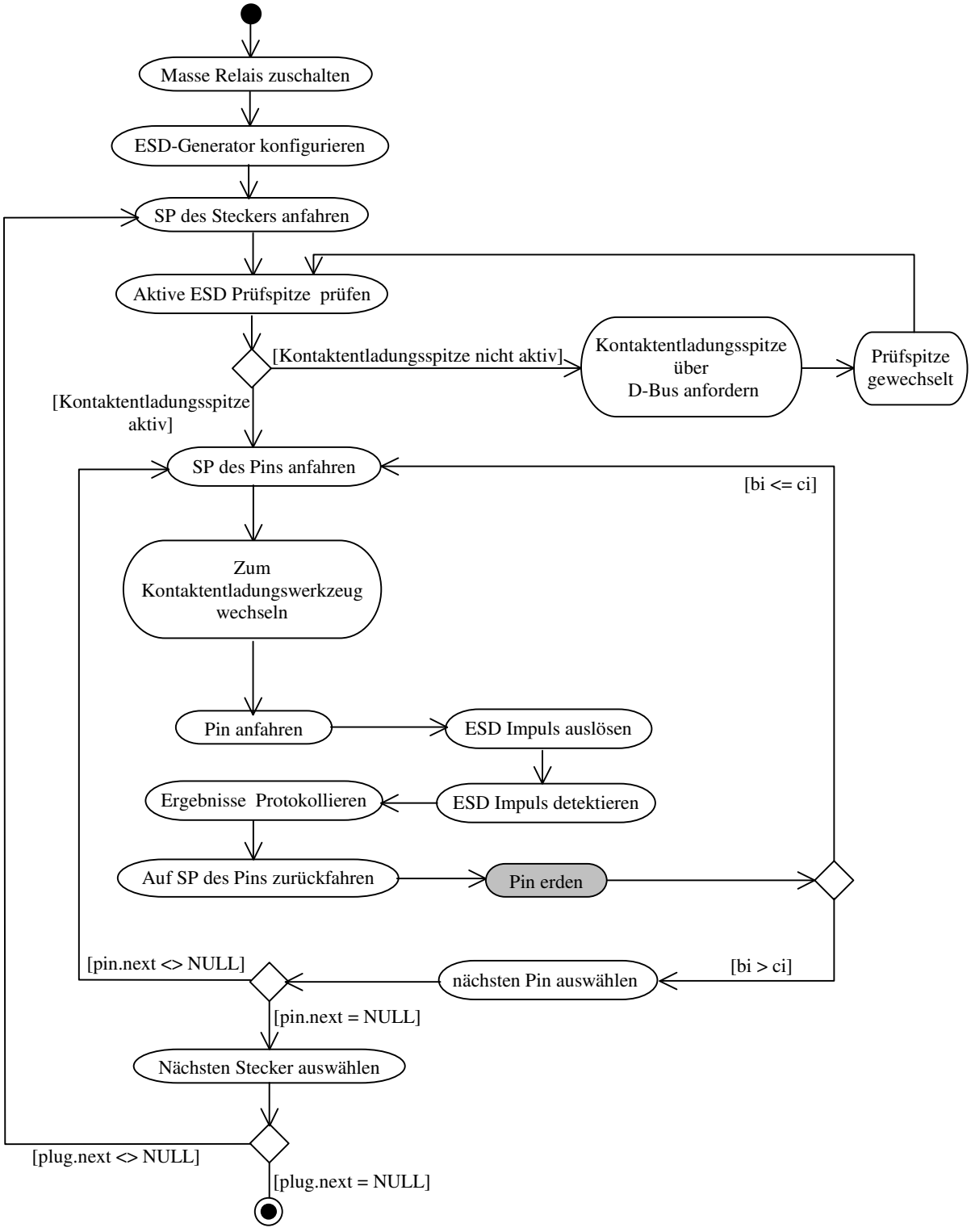


Abbildung 4.8: Aktivitätsdiagramm ESD Kontaktentladung

Der zweite Typ von Entladung ist die Luftentladung. Die Konfiguration des ESD-Generators stellt den Testtyp (Luftentladung) und die Spannung, mit der der Pin aufgeladen wird, ein. Bei dieser Entladung nähert der Roboter die Luftentladungsspitze an den Pin an bis eine Entladung stattfindet oder der Pin erreicht wird. Danach wird der Pin geerdet. Dieser Vorgang wird nach ISO 10605 üblicherweise drei Mal für jeden Pin durchgeführt. Die Abbildung 4.9 zeigt den Ablauft einer ESD – Luftentladung. In dieser Abbildung sind folgende Notationen zu beachten:
bi : ist die Anzahlt der benötigten Prüfimpuls pro Pin und
ci : ist die Anzahle der schon gelöste Prüfimpuls für einen Pin.

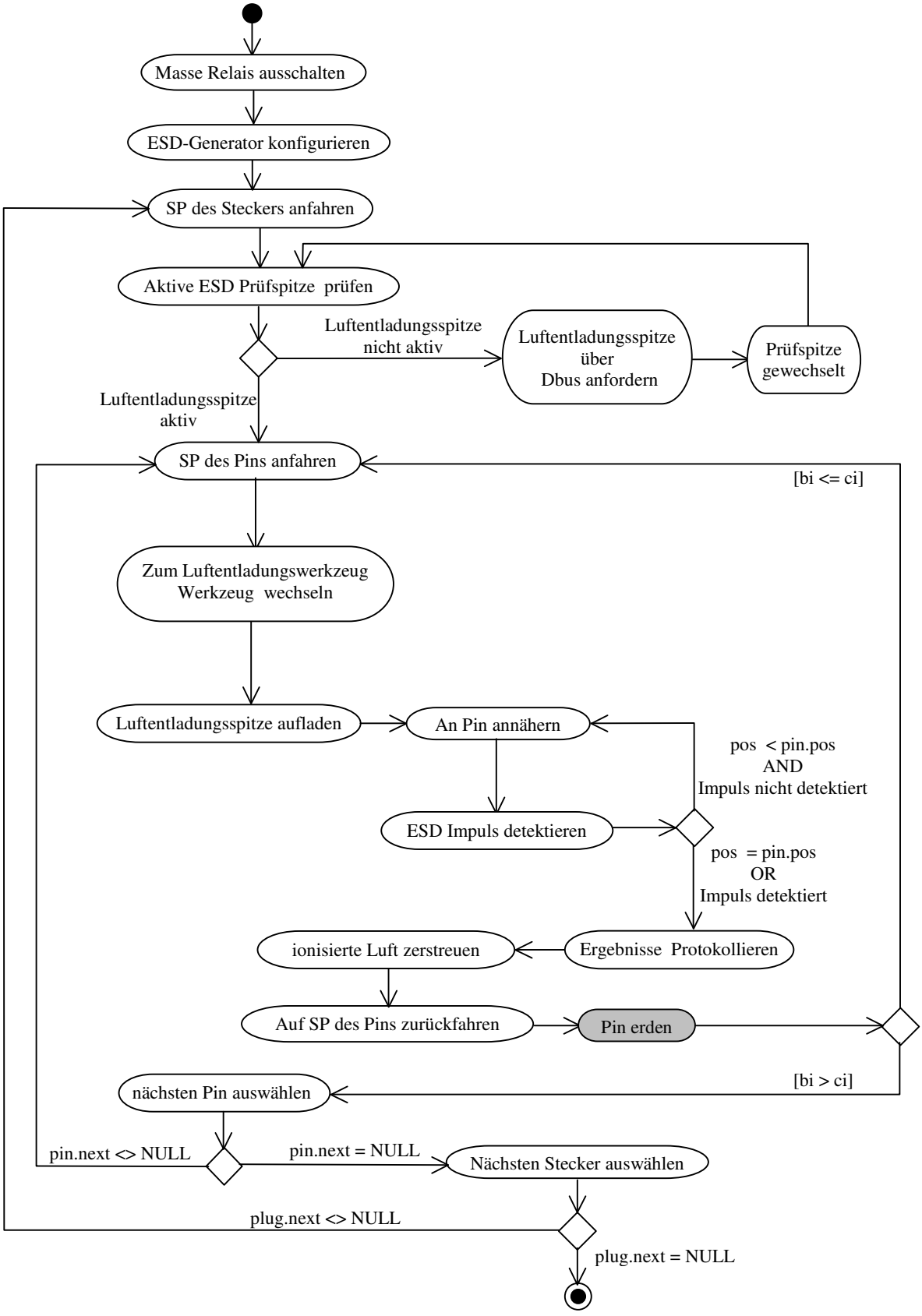


Abbildung 4.9: Aktivitätsdiagramm ESD Luftentladung

Nach jeder Aufladung des Pins wird diesen geerdet. Die Abbildung 4.10 zeigt den Ablauf einer Erdung.

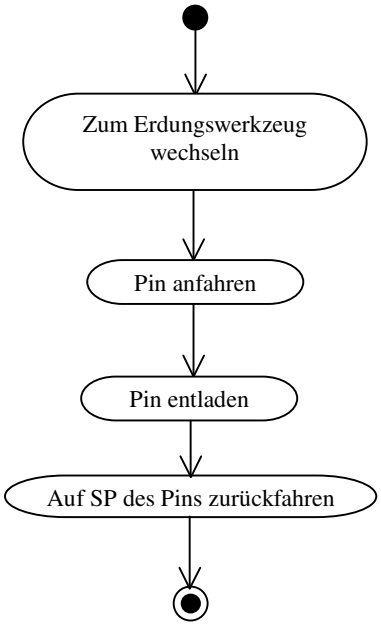


Abbildung 4.10: Aktivitätsdiagramm Pin erden

Eine andere Funktionalität, die das Prüfprogramm bietet ist die System-Diagnose des Prüfplatzes. Durch die System-Diagnose kann der Bearbeiter den Status oder die Info über Name, Version und Baujahr eines Geräts abfragen. Der Bearbeiter sendet mit Hilfe des Clients eine Nachricht über D-Bus an das Prüfprogramm. Die Nachricht enthält den Namen des Geräts, dessen Status gewünscht wird. Die Abbildung 4.11 zeigt den Ablauf der System-Diagnose des Prüfplatzes.

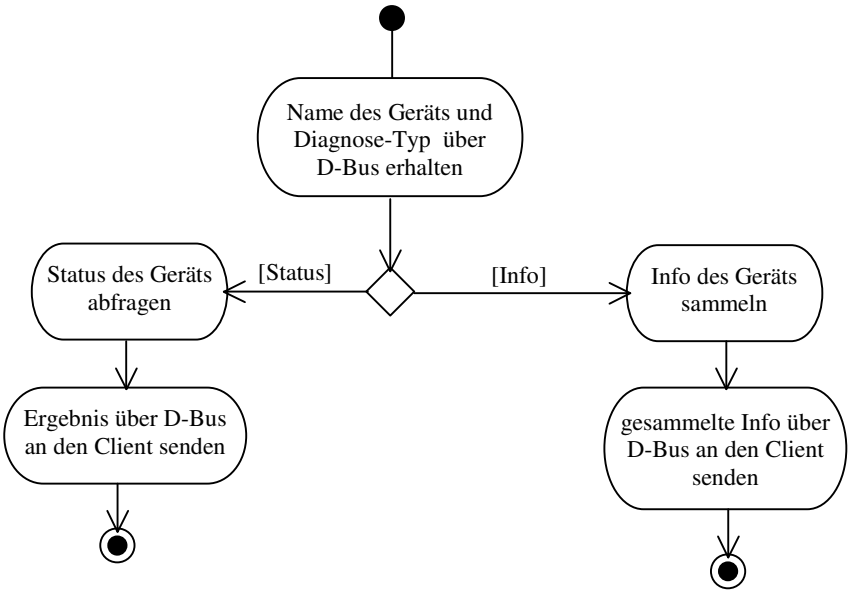


Abbildung 4.11: Aktivitätsdiagramm Prüfplatz Diagnose

4.3 Datenmodell

Nachdem die Abläufe der Aktivitäten des Prüfmoduls modelliert wurden, werden in diesem Abschnitt die Klassen des zu entwickelnden Systems modelliert. Um das Datenmodell übersichtlicher zu machen, wird erst das Datenmodell der Treibers modelliert, dann das Datenmodell des Prüfmoduls dargestellt. Die UML-Notation eines Klassendiagramms wird dafür verwendet um diese Datenmodell zu modellieren.

Die Abbildung 4.12 zeigt das Klassendiagramm des Datenmodells der Treiber. Es werden nicht alle Methoden der Treiber dargestellt, sondern nur die Methoden, die für den Ablauf des Tests benötigt werden. Das Attribut „base“ der Treiber ist eine Struktur, welche die internen Daten der Treiber enthält. Diese Daten werden nicht hier dargestellt.

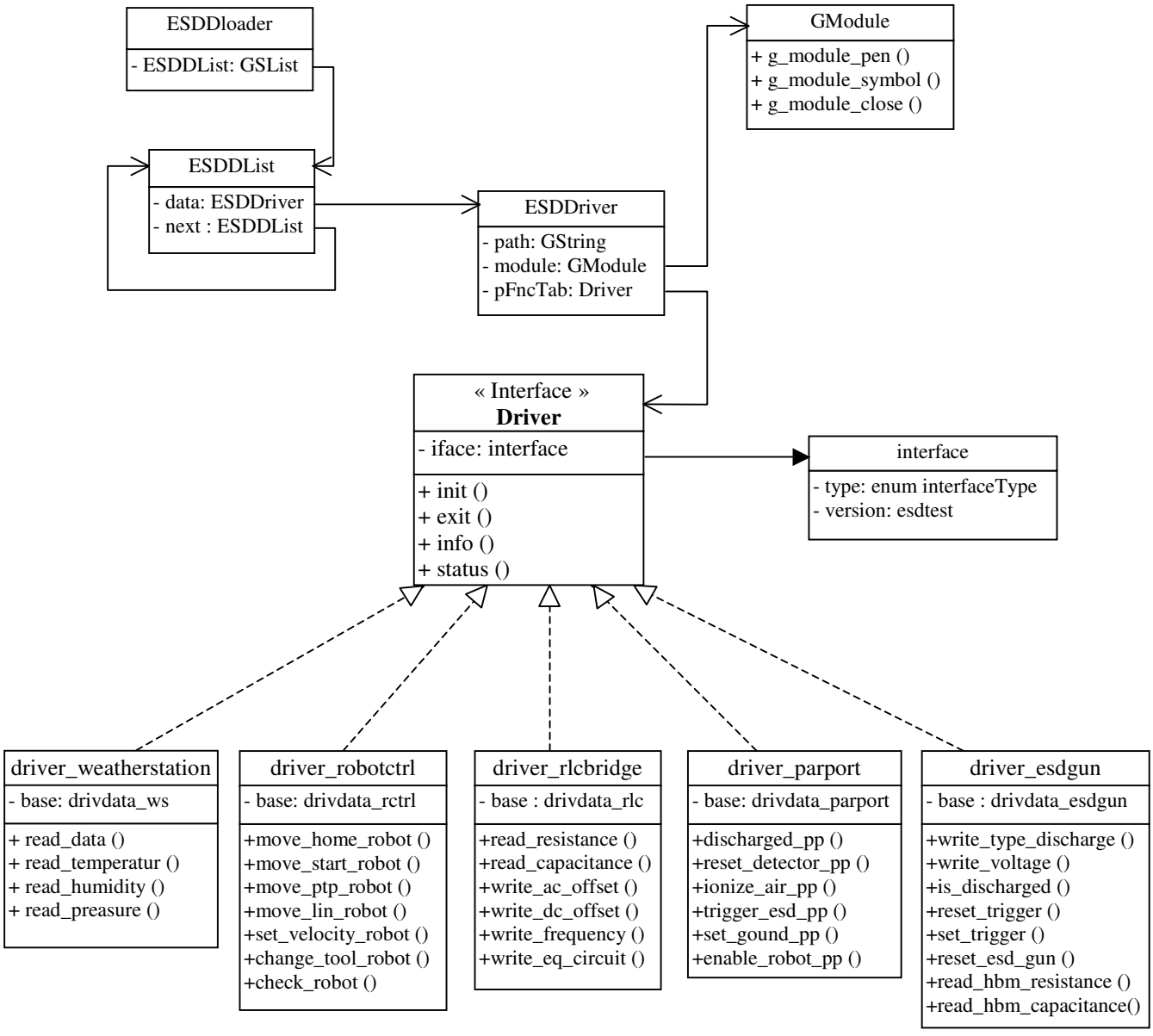


Abbildung 4.12: Klassendiagramm des Datenmodells der Treiber

In Abbildung 4.13 ist das Datenmodell des Prüfprogramms zu sehen. Das Datenmodell modelliert den Steuerfluss von den Eingabedaten bis zu den Ausgabedaten des ESD-Tests. Das Datenmodell zeigt auch wie die Treiber die Testdaten benutzen um den Test durchzuführen.

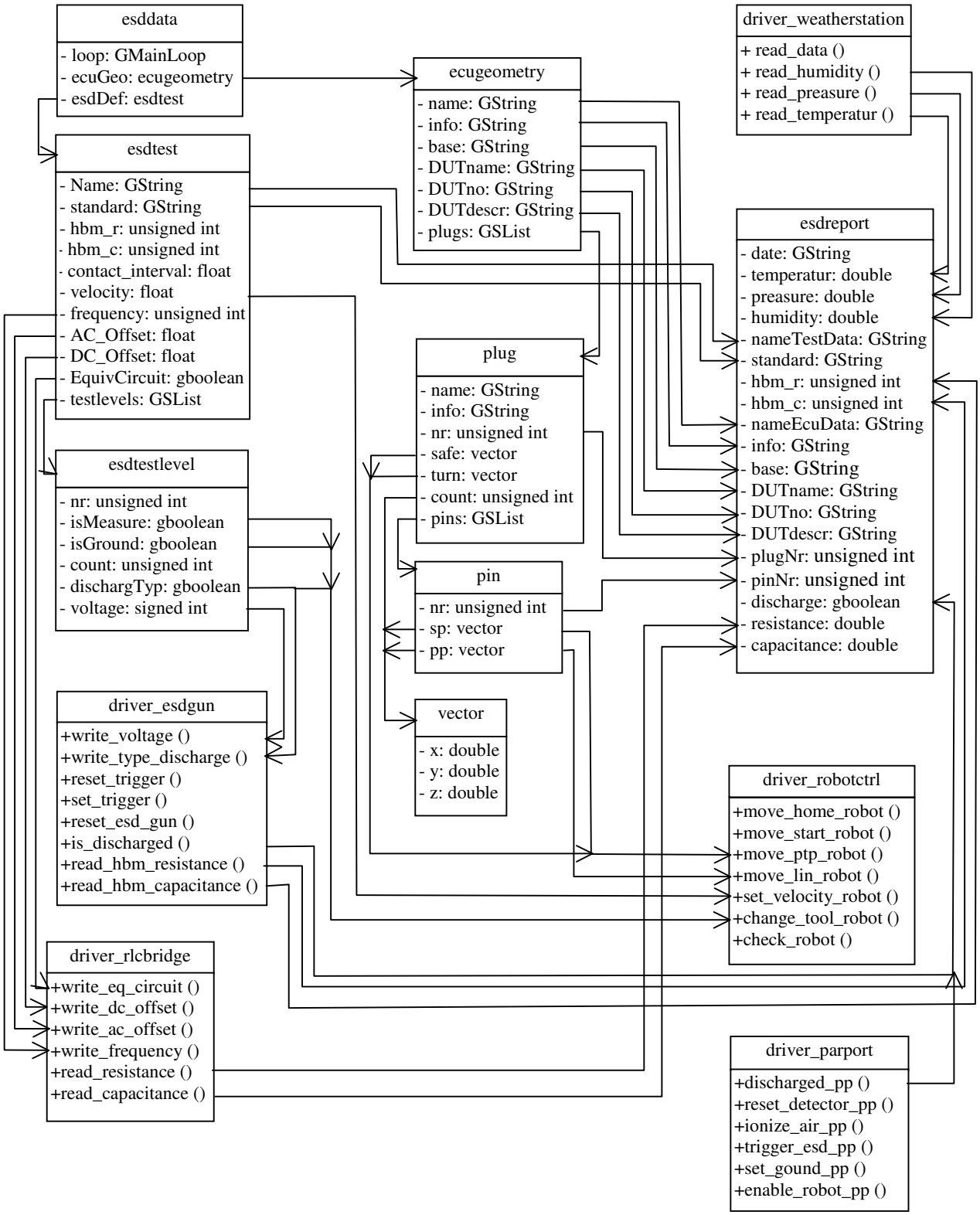


Abbildung 4.13: Klassendiagramm des Datenmodells des Prüfmoduls

Um das Verständnis des oben dargestellten Datenmodells zu vereinfachen, werden einige Klassen der Datenmodelle erläutert.
Die Tabelle 4.1 stellt die Eingabedaten der Definition eines ESD-Tests dar.

Tabelle 4.1: Testdefinition

Name	esdtest
Definition	Speicherplatz, in dem die vom Benutzer festgelegten Konfiguration und Ablauf des Tests definiert ist.
Attribute	- name: GString (Name des Tests) - standard: GString (Name der Norm) - hbm_r: unsigned int (Widerstandwert in Ohm) - hbm_c: unsigned int (Kapazitätswert in pF) - contact_interval: float (Zeit in Sekunden) - air_interval: float (Zeit in Sekunden) - air_velocity: float (Geschwindigkeit in m/s im Bereich 0,1 – 2,0 m/s) - frequency: unsigned int (Frequenz der RLC - Messbrücke) - AC_Offset: float (Wechselspannungs-Testsignal) - DC_Offset: float (Gleichspannungs-Testsignal) - EquivCircuit: gboolean (Ersatzschaltung der RLC - Messbrücke Seriell oder Parallel) - testlevels: GSList (Liste der ESD Testlevel siehe Tabelle 4.2)

Tabelle 4.2: Beschreibung der Eingabedaten eines Testlevels

Name	esdtestlevel
Definition	Speicherplatz, in dem den vom Benutzer festgelegten Schrittablauf des Test gespeichert ist.
Attribute	- nr: unsigned int (Bezeichnung des Testlevels) - isMeasure: gboolean - isGround: gboolean - count: unsigned int (Anzahl von Prüfimpuls pro Pin) - voltage: signed int (Prüfspannung in Volt)

Die Tabelle 4.3 zeigt die Eingabedaten einer Elektronik, die getestet wird. Diese Daten werden benutzt um die Prüfpunkte der Elektronik zu treffen.

Tabelle 4.3: Beschreibung der Daten der Elektronikdefinition

Name	ecugeometry
Definition	Speicherplatz, in dem Information über eine Elektronik gespeichert ist.
Attribute	- name: GString (Name der Elektronik) - info: GString (Beschreibung) - base: GString (Beschreibung Basiskoordinatensystem) - plugs: GSList (Liste der Stecker / plugs der Elektronik siehe Tabelle 4.4) - DUTname: GString (Name der ECU) - DUTno: GString (Nummer der ECU) - DUTdescr: GString (Beschreibung der ECU)

Tabelle 4.4: Beschreibung der Eingabedaten eines Steckers

Name	plug
Definition	Speicherplatz, in dem die Beschreibung einen einzelnen Stecker gespeichert ist.
Attribute	- nr: unsigned int (Nummer des Steckers) - name: GString (Name des Steckers) - info: GString (Freie Beschreibung) - safe: vector (Koordinate des SP des Steckers) - turn: vector (Rotation Winkels des Steckers) - count: unsigned int (Anzahl der Pins) - pins: GSList (Liste von Pins des Steckers siehe Tabelle 4.5)

Tabelle 4.5: Beschreibung der Eingabedaten eines Steckers

Name	pin
Definition	Speicherplatz, in dem ID und Koordinaten der Position und Sicherheitsposition des Pins gespeichert ist.
Attribute	- nr: unsigned int (Bezeichnung des Pins) - pp: vector (Koordinaten des Steckerpins) - sp: vector (Koordinaten des SP des Steckerpins)

Die Tabellen 4.6 und 4.7 zeigen die gemeinsam Daten der Treiber. Das Interface enthält den Typ (RLC-Messbrücke, Robotersteuerung, ESD-Generator, Wetterstation, Parallelport) und die Version (eins, zwei, …) des Treibers.

Tabelle 4.6: Beschreibung des Interfaces eines Treibers

Name	interface
Definition	Speichert Angabe über den Typ und die Version des Treibers
Attribute	type: enum interfaceType (Liste der Treiber die das Prüfmodul benötigt) version: int

Die Tabelle 4.7 zeigt die Basismethoden eines Treibers.

Tabelle 4.7: Beschreibung der Basismethoden eines Treiber

Name	Basis_Member
Definition	Basis Funktionalitäten, die alle Treiber erfüllen müssen.
Member	- Init(): Initialisiert alle Datenstrukturen des Treibers. Muss aufgerufen werden bevor der Treiber benutzt werden kann - Exit(): Datenstrukturen aufräumen. Der Treiber kann nach dem Aufruf nicht mehr benutzt werden. - Status(): prüft ob der Treiber und die angeschlossene Hardware einsatzbereit ist, versucht Einsatzbereitschaft herzustellen und gibt das Ergebnis zurück (Fehlernummer) - Info(): Gibt Treibername, Version, Gerätename und Status zurück

4.4 Das Treiberkonzept

Die neue Version des Prüfprogramms implementiert ein Treibermodell. Diese Treiber sind Bibliotheken von Funktionen der Geräte des Prüfplatzes. Das Steuerprogramm des Prüfplatzes benutzt eine große Anzahl von Funktionen der Geräte. Bei der Durchführung des ESD-Tests werden diese Funktionen sehr häufig aufgerufen. Die Treiber ermöglichen es Code wieder zu verwenden. Die Wiederverwendung von Code hilft dabei die Wartbarkeit und Erweiterbarkeit des Steuerprogramms zu erreichen. So werden die Fehler bei der Implementierung reduziert, da die Suche nach Fehlern nur im neu erstellten Code gemacht werden muss. Außerdem kann beim Hinzufügen neuer Funktionalität auf bestehenden und bereits getesteten Code zurückgegriffen werden.
Der Funktionscode der Gerätetreiber wird als verteilte Bibliothek aufgebaut. Diese Bibliotheken haben als Extension „.so“ und werden in dieser Arbeit mit der Application Programming Interface (*API*) GLib und GMODULE realisiert. Das Prüfprogramm soll die Treiber automatisch laden, die Mindestfähigkeit eines Treibers automatisch finden und Treiber und deren Klassen einbinden können. Um diese Anforderungen zu erfüllen, läuft das Einbinden der Treiber nach folgendem Schema ab: Beim Laden der Treiber, werden alle Klassen von Treiberfunktionen unabhängig von ihren zugehörigen Treibern geladen. Das in der Tabelle 4.6 definiert „interface“ für alle Treiber löst das Problem der Einbindung der Treiber und ihrer Klassen. Durch das Attribut „Typ“ wird jede Funktionsklasse dem richtigen Treiber zugeordnet. Die Mindestfähigkeit eines Treibers wird durch seine Version ermittelt. Das Attribut „version“ der Klasse „interface“ speichert die Version eines Treibers. Umso größer die Version eines Treibers ist, desto aktueller ist der Treiber. Wenn es schon zwei Treiber derselben Klasse gibt, wird der Treiber mit der höchsten Version geladen und der anderer entladen. So werden Konflikte aufgelöst.

4.5 D-Bus-Konzept

Das in dieser Arbeit entwickelte System soll mit Hilfe des Nachrichten-Bus-Systems D-Bus mit der Bedienoberfläche kommunizieren. D-Bus ist ein System für die Interprozesskommunikation (Inter Process Communication, *IPC*), es stellt die Infrastruktur bereit, damit sich Anwendungen untereinander und mit Teilen des Betriebssystems unterhalten können. Die Architektur von D-Bus besteht aus mehrere Schichten:

- Einer Bibliothek „libdbus“, die es ermöglicht, zwischen zwei Applikation eine Verbindung für den Austausch von Nachrichten zu erstellen.
- Einem ausführbaren Bus-Daemon für Nachrichten, über den sich andere Applikation verbinden können.
- Einer Hülle von Bibliotheken (oder Bindings) basierend auf bestimmten Anwendungs-Frameworks. Z.B. libdbus-qt und libdbus-glib. In dieser Arbeit wird libdbus-glib verwendet.

Mit D-Bus kann das Prüfprogramm bei der Kommunikation mit der Bedienoberfläche, sowohl die Rolle des Servers als auch die des Clients einnehmen. Als Server bietet das Prüfprogramm Services an, indem es Befehle des Clients entgegennimmt, ausführt und das Ergebnis zurückmeldet. Als Client sendet es Befehle an die Bedienoberfläche und warte auf die Rückmeldung der Bedienoberfläche, die in diesem Fall als Server arbeitet.

Via D-Bus können Applikationen unterschiedlicher Programmiersprachen oder Systeme miteinander kommunizieren.
Im Weiteren wird das D-Bus Konzept und die Tools auf denen D-Bus basiert, erläutert.

Bus

Um eine Kommunikation mit dem System aufzubauen, muss zuerst eine Verbindung mit dem Bus erstellt werden. Es gibt die „dbus library“ für eine Punkt-zu-Punkt Kommunikation zwischen zwei Prozessen und den „dbus daemon“, der zum Transport von Nachrichten dient. In dieser Arbeit wird hauptsächlich der „dbus daemon“ verwendet. In diesem System können auch mehrere Busse gleichzeitig aktiv sein. Jeder Bus hat eine Adresse, die beschreibt, wie die Verbindung zum Bus hergestellt wird. Es kann mehrere Verbindungen zu einem einzelnen Bus geben. Die Verbindungen werden durch ihren Namen unterschieden.

Objekte

Es gibt „Native Object“ für low-Level D-Bus Protokoll and „Object Path“ für high-Level. Durch ein Objekt kann der Client-Prozess seine Services am Bus anbieten. Ein Client ist in der Lage beliebig viele Objekte zu erstellen. Jedes Objekt lässt sich über seinen Namen oder Pfad und seinen Interface-Namen identifizieren. Ein Objekt-Name besteht aus Bezeichnungen getrennt mit Schrägstrichen. Ein Interface-Name besteht aus Bezeichnungen getrennt mit Punkten. Die Anwendung von D-Bus sieht mehr wie die objektorientierte Programmierung. Jedes Objekt hat Methoden und Signals die später definiert werden.

Proxies

Objekte auf dem Bus können durch Stellvertreter, so genannte Proxies, erreicht werden. Ein Proxy Objekt vertritt ein passendes „native Objekt“, und es erlaubt ein Fernobjekt in einem anderen Prozess darzustellen.

Nachrichten

Über D-Bus werden Nachrichten zwischen Prozesse ausgetauscht. Im „high-level binding“ wird nicht direkt mit Nachrichten gearbeitet. Es gibt vier Typen von Nachrichten:

- Methode „ call messages“ rufen eine Methode einen entfernten Objekt auf
- Methode „return messages“ geben das Ergebnis der aufgerufenen Methode zurück.
- Fehlermeldung gibt eine Ausnahme zurück, die von der aufgerufenen Methode verursacht wurde,
- Signalnachrichten sind Mitteilungen, dass ein bestimmtes Signal ausgesendet wurde

Jede Nachricht besteht aus:

- Einer Überschrift: Sie enthält Felder für den Absenderbusnamen, den Zielbusnamen, den Methodennamen, den Signalnamen und so weiter.
- Einem Körper: Dieser enthält die Parameter der zu sendenden Nachricht.

Methode

Eine Methode in D-Bus besteht aus zwei Nachrichten: Eine Nachricht *Aufruf* die von Prozess A zu Prozess B geschickt wird, und eine Nachricht *Rückmeldung* von Prozess B zu Prozess A. Diese beiden Nachrichten werden über den *Bus daemon* gesendet. Die Aufrufnachricht enthält alle Argumente der Methode. Der Antwortnachricht kann einen Hinweis auf einen Fehler, oder zurückgeschickte Daten der Methode enthalten. Eine Methode kann synchron oder asynchron aufgerufen werden.
Wenn ein Client eine Abfrage zu einem Objekt sendet, sieht er die Abfrage als den Aufruf eine Methode eines Objekts. So wird das Objekt aufgefordert, die Aktion durchzuführen.

Signal

Ein Signal in D-Bus besteht aus einer einzigen Nachricht, gesandt von einem Prozess zu eine beliebigen Anzahl von anderen Prozessen. Das Signal ist einseitig ausgesendet. Es kann Argumente beinhalten, hat aber keine Rückgabewerte.
Client Prozesse können ein Interesse an Signalen eines bestimmten Objekts registrieren. Wann immer ein Objekt ein Signal ausstrahlt, empfangen alle interessierten Klienten eine Kopie des Signals.

Interfaces

Jedes Objekt unterstützt bestimmte Methoden und kann bestimmte Signale aussenden. Diese Methoden und Signale sind als *Mitglieder* des Objekts bekannt. Alle Mitglieder eines Objekts werden in einem Interface definiert. Wie in „Java“, indem irgendeine Anzahl von Klassen das gleiche Interface implementiert.
Wenn ein Client eine Methode aufruft oder auf einem Signal hört, muss er das Objekt und das Mitglied auf dem es sich bezieht, angeben. Zusätzlich zum Objekt und Mitglied, soll der Client das Interface auch nennen, in dem dieses Mitglied spezifiziert wurde.

Introstpection

Um die Programmierung zu vereinfachen, bietet D-Bus die Möglichkeit Methoden und Signale in XML Format in einer Datei zu beschreiben. Diese Datei ist eine „Introspection XML“ Datei, die das Interface eines Objekts beschreibt. Diese XML-Datei wird verwenden um zwei Header-Dateien zu generieren:

- Eine Header-Datei mit den benötigten Prototypfunktionen des Objekts für den Server
- Eine Header-Datei mit den benötigten Prototypfunktionen des Objekts für den Client

Dies vereinfacht die Benutzung von D-Bus Objekten. Die Abbildung 4.13 zeigt ein Beispiel der „Introspection XML“ Datei, die das Objekt „MyObjekt“ darstellt. Das Objekt hat ein Methode „MyMethod“ und ein Signal „MySignal“. Diese XML Datei besteht aus folgenden Komponenten:

- node name: Bestimmt den Namen des Objekts
- interface name: Bestimmt den Namen des Interfaces
- method name: Bestimmt den Namen der Methode
- signal name: Bestimmt den Namen des Signals
- arg: Bestimmt folgenden Komponente für ein Argument:

- type: Ein ASCII Buchstabe, der den Typ des Arguments bestimmt
- name: Bestimmt den Namen des Arguments
- direction: Bestimmt, ob es ein Eingabe- oder Ausgabe Parameter ist

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/com/example/MyObject">
  <interface name="com.example.MyObject">
    <method name="MyMethod">
      <arg type="s" name="str" direction="in" />
      <arg type="d" name="zahl" direction="in" />
      <arg type="d" name="d_ret" direction="out" />
    </method>
    <signal name="MySignal">
      <arg type="b" name="new_value" />
    </signal>
  </interface>
</node>
```

Abbildung 4.13: Beispiel einer Introspection XML Datei

4.6 Zusammenfassung

In diesem Kapitel wurde der Aufbau des neuen Systems veranschaulicht. Die Architektur des neuen Steuerprogramms des Prüfplatzes wurde erläutert. Am wichtigsten war die Auswahl einer Architektur für die Kommunikation des Prüfprogramms mit den Geräten des Prüfplatzes. Dazu wurde ein Vergleich der Kommunikationsarchitektur des bestehenden Steuerprogramms mit zwei möglichen neuen Kommunikationsarchitekturen durchgeführt. Auf Grund dieses Vergleichs hat sich die Kommunikationsarchitektur *Stern 2* mit Ein-Level Funktionsmodul durchgesetzt. Diese Architektur hat den Vorteil, die Anforderung des Treiber-Konzepts besser umzusetzen. Die Aktivitäten der Prozesse des Prüfmoduls sowie der Steuerfluss zwischen diesen wurden mittels Aktivitätsdiagramme modelliert. Das Datenmodell des Systems wurde dargestellt und am Ende das Treiber- und D-Bus Konzept erläutert.

Kapitel 5

Implementierung

In dem Kapitel 4 wurde das Konzept der in dieser Arbeit zu entwickelnden Prüfmoduls vorgestellt. In diesem Kapitel wird die Umsetzung dieses Konzepts erläutert. Es wird gezeigt wie das D-Bus und Treiber Konzept sich implementieren lassen. Wie schon erwähnt, soll das Prüfprogramm mit den Geräten des Prüfplatzes kommunizieren, um einen Test durchzuführen. Es wird gezeigt, wie diese Kommunikationen in der Praxis hergestellt werden. Die Probleme, die während der Implementierung aufgetreten sind, werden erläutert.
Die Implementierung wird auf Grund des Qualitätsmodells in Abschnitt 3.7 bewertet. Er wird auch eine Auswertung des Stands der Umsetzung im Vergleich zu den vorgegebenen Anforderungen gegeben.

5.1 Systemumgebung und Komponente

Die neue Generation des Steuerprogramms des Prüfplatzes, hat eine Client / Server Architektur. In dieser Arbeit soll nur das Backend (der Server) dieser Architektur implementiert werden. Dieser Implementierung ist unabhängig vom Frontend (der Client), welches später implementiert werden soll. Der Server Teil des neuen Steuerprogramms bekommt die Eingabedaten (Testdaten) und Steuerbefehle vom Client über D-Bus geliefert. Dieser Server kommuniziert mit den Geräten des Prüfplatzes über bestimmte Schnittstellen um den Test durchzuführen. Während des Tests kommuniziert das Prüfprogramm mit dem Bearbeiter über eine GUI um Fehler des Systems anzuzeigen. Das Prüfprogramm schickt auch Anweisungen für den Benutzer über D-Bus an den Client. Um dies zu zeigen wurde ein kleiner Client aufgebaut um Befehle an den Server zu schicken und auch Nachrichten vom Server angezeigt zu bekommen.

5.1.1 Programmiersprache

Das Prüfmodul wird komplett in „C“ programmiert. Die Umsetzung der Treiber wurde mit Hilfe der Bibliothek GMODULE und GLib realisiert. Diese Bibliotheken bieten Funktionen um Objektdateien (plug-ins) dynamisch zu laden. Die Implementierung von D-Bus wurde mit der Bibliothek libdbus-glib durchgeführt. Dieses Paket stellt die GLib-basierten Bibliotheken für Anwendungen, welche die GLib-Schnittstelle für D-Bus benutzen, bereit. Diese Bibliotheken, die während der Implementierung verwendet wurden, sind unter der GPL veröffentlicht.

5.1.2 Schnittstellen

Die Kommunikation des Prüfprogramms mit den Geräten, erfolgt über bestimmte Schnittstellen. Diese Schnittstellen werden im Folgenden zusammen mit den Geräten beschrieben. Es wird auch gezeigt wie die Verbindungen aufgebaut werden.

GPIB Schnittstelle

GPIB (*General Purpose Interface Bus*) ist eine short-range, digitale Kommunikationen Busspezifikation, die seit über 30 Jahre im Gebrauch ist. Dieser Bus wurde von der Firma Hewllet-Packard (*HP*) entwickelt und ist unter der Norm IEEE-488 bekannt. Der Bus IEEE-488 setzt 16 Leitungssignalen ein:

- Acht bidirektionale für Datenübertragung
- Drei für Handshake
- Fünf für Busmanagement
- Plus acht Grundrückleitungen

GPIB wird hauptsächlich für die Verbindung zwischen einem Messgerät und einem Computer benutzt.
Der GPIB Bus wird in diesem Projekt für die Verbindung zwischen dem Steuerrechner und der RLC Messbrücke verwendet. Die RLC Messbrücke misst unter Anweisung des Prüfmoduls die Impedanz eines Pins und liefert das Ergebnis zurück. Diese Kommunikation des Prüfprogramms mit der RLC Messbrücke über GPIB wird durch den Treiber „driver_rlcbridge“ unterstützt. Dieser Treiber benutzt die Bibliothek „ni488.h“ von National Instruments[1] um die Verbindung mit der RLC Messbrücke herzustellen.

Serielle Schnittstelle (RS-232)

Die RS-232 Schnittstelle wurde in den frühen 1960ern von einem US-amerikanischen Standardisierungskommitee eingeführt (RS steht dabei für *Radio Sector).* Sie ist fast auf allen Computern verfügbar und wird als „COM Port“ bezeichnet. RS-232 wurde ursprünglich geschaffen, um Computerterminals (DTE - *data terminal equipment*) an langsame Modems (DCE - *data communication equipment*) anzuschließen. RS-232 benutzt für die Datenübertragung ein einfaches asynchrones serielles Verfahren. Seriell bedeutet, dass die einzelnen Bits des zu übertragenden Bytes nacheinander über eine einzige Datenleitung geschoben werden. Asynchron heißt, dass es keine Taktleitung gibt, die dem Datenempfänger genau sagt, wann das nächste Bit auf der Datenleitung liegt. So ein Verfahren kann nur funktionieren, wenn Sender und Empfänger mit genau dem gleichen internen Takt arbeiten, und wenn der Empfänger gesagt bekommt, wann das erste Bit genau anfängt (Synchronisation).
Alle RS232-Leitungen (mit Ausnahme der Masseleitung) arbeiten mit den Spannungspegeln +12V (für eine logische '0') und -12V (für eine logische '1'). (Erlaubt sind jeweils 5V..15V.) Der Datenempfänger erwartet eine Spannung von über +3V für eine „0“ und von unter -3V für eine „1“. Für die Steuerleitungen (Handshakeleitungen) bedeutet ON einen hohen Pegel (+5V … +15V) und OFF einen negativen Pegel (-5V ... -15V).
In diesem Projekt wurde die RS-232 Schnittstelle für die Verbindung zwischen: dem Steuerrechner und dem ESD Generator, dem Steuerrechner und dem Roboterrechner und dem Steuerrechner und der Wetterstation eingesetzt. Diese Verbindungen werden von den jeweiligen folgenden Treibern gesteuert: „driver_esdgun“, „driver_robotctrl“ und „driver_weatherstation“. Diese Treiber werden mit den Bibliotheken „fcntl.h“ und „sys/ioctl.h“ der standard „C“ implementiert. Diese Bibliotheken bieten Funktionen um die

[1] Nationale Instruments zählt zu den führenden Herstellern von GPIB-(IEEE 488)-Schnittstellen und -Modulen und bietet Anwendern eine reiche Auswahl an GPIB-Produkten.

Kommunikation mit der RS-232 Schnittstelle abzuwickeln. Damit die Kommunikation mit dem Gerät erfolgt, wird für jede Treiber definiert:

- Ein Baudrate: Datenrate in Baud (in Bit pro Sekunde angegeben).
- Einen Port: Port mit dem das Gerät mit dem Steuerrechner verbunden ist. Dieser Port muss von einem Gerät zum anderem unterschiedlich sein, da sonst Konflikte während der Kommunikation auftreten.

Parallel-Port

Der Parallel-Port (auch: Parallele Schnittstelle) bezeichnet einen digitalen Ein- oder Ausgang eines Computers oder eines Peripheriegerätes. Bei der Datenübertragung über eine parallele Schnittstelle werden mehrere Bits gleichzeitig – also parallel – übertragen. Der Primärgebrauch des Parallel Ports ist es, Drucker an Computer anschließen.
Der Einsatz des Parallel-Ports in dem in dieser Arbeit entwickelnden System geschieht zur Steuerung von Signalen, Druckluftventilen, und zur Fahrfreigabe. Die Verbindung des Prüfprogramms mit dem Parallel Port wird durch den Treiber „driver_parport“ unterstützt. Dieser Treiber wird mit Hilfe der Funktionen in „linux/ppdev.h“ und „sys/ioctl.h“ der standard „C“ Library implementiert.

5.2 Umsetzung

Nachdem die Vorrausetzungen zur Kommunikation mit den Geräten des Prüfplatzes und D-Bus geschaffen sind, wird in diesem Abschnitt erläutert, wie das System umgesetzt wurde. Es wurde für den Aufbau des Prüfprogramms wie folgt implementiert:

- zunächst ein Treiber für jedes Gerät,
- zweitens ein Modul zum Laden der Treiber,
- drittens die Kommunikation mit D-Bus
- und am Ende der Ablauf des Prüfmoduls.

Im Folgenden wird die technische Umsetzung der Kommunikation mit den Treibern und mit D-Bus im Detail beschrieben.

5.2.1 Kommunikation mit den Treiber

Jeder Treiber soll die Funktionen seines entsprechenden Geräts implementieren. Alle Funktionen sollen synchron abgerufen werden, im Vergleich zum alten Steuerprogramm, bei dem einige Funktionen asynchron aufgerufen wurden.
Die Kommunikation jedes Treibers mit seinem Gerät läuft wie folgt ab:

- Verbindung herstellen: Ohne diese Verbindung kann der Treiber keine Funktion des Geräts aufrufen. Nach der Herstellung dieser Verbindung, wird ein File Descriptor für die Kommunikation mit dem Gerät erzeugt. Der File Descriptor ist die Schnittstelle zum Betriebssystem, um eine zugehörige Hardwareschnittstelle (GPIB, RS232, Parallel-Port …) benutzen zu können. Der File Descriptor ist ein ganze Zahl größer oder gleich „0“, wenn die Verbindung erfolgreich hergestellt wurde, und kleiner „0“ wenn nicht.
- Funktion aufrufen: Jetzt kann der Treiber mit Hilfe des File Descriptors Funktionen des Geräts aufrufen.

- Verbindung trennen: Am Ende der Kommunikation kann der Verbindung getrennt werden.

Die Funktionen der Treiber werden in einer Struktur, wie in Abschnitt 4.3 beschrieben ist, definiert. Die Treiber Dateien werden mit bestimmten Kompilierungsoptionen übersetzt, um die verteilten Bibliotheken mit Extension „.so“ zu erzeugen (z.B. driver_rlcbridge.so). Diese verteilten Bibliotheken repräsentieren jetzt die Treiber, die dynamisch durch ein Modul genannt „module_loader“ geladen werden. Dieses Modul benutzt die *API* von GMODULE und GLib um die Treiber dynamisch zu laden. Es wird erst mal mit der Funktion g_module_supported() überprüft, ob die Plattform ein dynamisches Laden unterstützt. Dann wird für jeden Treiber die Funktion g_module_open() aufgerufen, der einen Treiber öffnet. Dann kann mit der Funktion g_module_symbol() jedes Symbol (z.B. Funktionsnamen oder Name einer Variable) eines Treibers gefunden und importiert werden. Nachdem alle nötigen Symbole des Treibers importiert sind, kann der Treiber mit der Funktion g_module_close() geschlossen werden.

5.2.2 Kommunikation mit D-Bus

Die Kommunikation wurde mit D-Bus in der Version 1.0.1 und der API „libdbus-glib“ realisiert. In dieser Arbeit wurde D-Bus mit den „higher-level bindings“ eingesetzt. Die Kommunikation in D-Bus findet zwischen einem Server, der passt auf ankommenden Anschlüssen wartet und einem Client, der sich zu denn Server verbindet, statt. Für beide Client und Server, muss erst eine Verbindung an den Bus hergestellt werden, um eine Kommunikation zu realisieren. Eine Verbindung zum Bus sieht wie folgt aus:

```
dbus_g_bus_get (DBUS_BUS_SESSION, &error);
```

Die Kommunikation des Prüfmoduls mit D-Bus erfolgt in drei Fällen, die im Folgenden beschrieben werden:

Server Prüfmodul für Testdaten

Das Prüfmodul nach seinem Start wartet auf Testdaten, vom Client (der Benutzeroberfläche) um den Test zu starten. Als Server sollte das Prüfmodul ein Objekt implementiert, das die Methode zum Starten des Tests zur Verfügung stellt. Diese Methode wird vom Client aufgerufen werden. Wie in Abschnitt 4.5 erklärt, wird eine XML Datei geschrieben, um die von dem Objekt exportierte Methode „StartEsdTest“ zu beschreiben. Diese Methode enthält, alle Parameter, die das Prüfprogramm braucht, um den Test durchzuführen und seinen Rückgabewert. Dieser Rückgabewert teilt dem Client mit, ob die Methode ausgeführt wurde oder nicht.

Es ist auch anzumerken, dass D-Bus seinen eigenen Datentyp besitzt um Daten zu repräsentieren. In XML Datei wird auf bestimmte Formatierung entsprechend dem Datentypen verwendet. Einige Beispiele sind:

- "u" steht für „unsigned integer“ im „C Standard“
- " (id) " steht für eine Struktur mit einem Integer und einem Double als Felder im „C Standard“
- Ein Array wird durch den ASCII-Code "a" gefolgt von dem Typ der Daten repräsentiert. Z.B "ad" steht für ein Array von Double im „C Standard“

Die Abbildung 5.1 zeigt die XML Datei, die das Interface „esd.esdtest.esdtestdata“ definiert. Die Datei zeigt das Objekt „esdtestdata“, das die Methode „StartEsdTest“ beschreibt.

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/esd/esdtest/esdtestdata">
  <interface name="esd.esdtest.esdtestdata">
    <!--Remote Methods of the Pruefprogramms from the GUI Application-->
    <method name="StartEsdTest">
      <!--Input ECUGeometryData-->
          <arg type="s" name="ecuname" direction="in"/>
          <arg type="s" name="info" direction="in"/>
          <arg type="s" name="base" direction="in"/>
          <arg type="s" name="DUTname" direction="in"/>
          <arg type="s" name="DUTno" direction="in"/>
          <arg type="s" name="DUTdescr" direction="in"/>
          <!-- ecugeometry -->
          <!--arg type="a(uss(iddd)(iddd)uua(u(iddd)(iddd)))" direction="in"/ -->
          <!--Input TestDefinitionData-->
          <arg type="s" name="Standard" direction="in"/>
          <arg type="u" name="HBM_C" direction="in"/>
          <arg type="u" name="HBM_R" direction="in"/>
          <arg type="d" name="IntervalContact" direction="in"/>
          <arg type="d" name="IntervalAir" direction="in"/>
          <arg type="d" name="VelocityAir" direction="in"/>
          <arg type="i" name="Frequency" direction="in"/>
          <arg type="d" name="AC_Offset" direction="in"/>
          <arg type="d" name="DC_Offset" direction="in"/>
          <arg type="b" name="EquivCircuit" direction="in"/>
          <arg type="a(ubuibb)" name="TestLevel" direction="in"/>
          <!--Output EsdTestStart-->
          <arg type="b" name="isEsdStart" direction="out"/>
    </method>
  </interface>
</node>
```

Abbildung 5.1: Datei GuiTestData.xml des Interfaces „esd.esdtest.esdtestdata“

Bei der Durchführung des Befehls „dbus-binding-tool –mode=glib-server“ wird eine Header-Datei erzeugt. Diese Header-Datei enthält die Prototypfunktion der Methode „StartEsdTest“ und wird in Serverdatei eingefügt. Der Befehl sieht seitens des Servers wie Folge aus:

dbus-binding-tool --mode=glib-server --prefix=gui_data --output=GuiTestDataGlue.h GuiTestData.xml

--*prefix=gui_data***:** Diese Option gibt das Präfix der Methode aus. In diesem Fall heißt die Methode gui_data_start_esd_test

--*output=GuiTestDataGlue.h:* Diese Option gibt aus den Dateiname der Header-Datei

Das Objekt „esdtestdate“ wird instanziert mit dem Aufruf g_object_new(). Eine Registrierung des Objekts mit dem Bus wird durch den Aufruf dbus_g_connection_register_g_object() gemacht. Mit dem folgenden Befehl wird ein neues Proxy für ein entferntes angelegt.

dbus_g_proxy_new_for_name (bus,
DBUS_SERVICE_DBUS,
DBUS_PATH_DBUS,
DBUS_INTERFACE_DBUS) ;

Dieses Proxy erlaubt uns bestehende Funktionen des entfernten Services aufzurufen. Dieser entfernte Service ist der D-Bus-Daemon selbst. Sobald der Proxy angelegt ist, kann der Name (esd/esdtest/esdtestdata) unter dem das Prüfprogramm auf D-Bus sichtbar seine soll mit der folgenden Funktion beim D-Bus-Daemon registriert werden:

org_freedesktop_DBus_request_name(). Dies ist ein Aufruf der Methode „RequestName“, die im „org.freedesktop.DBus interface“ definiert ist.

Das Prüfmodul ist Server für Befehle des Bearbeiters

In diesem Fall, ist das Prüfmodul der Server und wird Befehle der Bedienoberfläche ausführen und Signale aussenden. Die Methoden repräsentieren die Wünsche des Bearbeiters bezüglich des Ablaufs des Tests. Z.B den Wunsch den Test anzuhalten. wird durch die Methode „TestPause“ dem Prüfmodul gemeldet. Das Prüfprogramm wird ein Signal über D-Bus aussenden falls ein Fehler beim Test auftritt und den Status des Testablaufs an den Bearbeiter melden. Die Abbildung 5.2 zeigt die XML-Datei, die das Interface "esd.esdtest.daemonevent" definiert. Die Datei definiert die Methoden und Signale des Prüfprogramms für den Client (Die Bedienoberfläche).

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/esd/esdtest/daemonevent">
  <interface name="esd.esdtest.daemonevent">
        <!--Methods and Signals for the Prueprogramm from GUI Application-->
    <method name="TestPause">
          <arg type="b" name="isPaused" direction="out"/>
    </method>
    <method name="TestStop">
          <arg type="b" name="isStoped" direction="out"/>
    </method>
    <method name="TestContinue">
          <arg type="b" name="isContinued" direction="out"/>
    </method>
    <method name="TestNew">
          <arg type="b" name="isNew" direction="out"/>
    </method>
    <!--Output Information and status about each device of the test station-->
    <method name="StatusPruefplatz">
          <arg type="a{ss}" name="Status" direction="out"/>
    </method>
    <method name="DeviceConf">
          <arg type="s" name="Device" direction="in"/>
          <arg type="a{ss}" name="Conf" direction="out"/>
    </method>
    <method name="EmitError">
    </method>
    <!--method name="TestStatus"-->
    <!--/method -->
    <!-- Signal to output the appearing faults during the test -->
    <signal name="DisplayError">
          <arg type="s" name="ErrorText"/>
    </signal>
    <!-- output the status any states of the test, also the end of the test -->
    <!--signal name="ReportTestStatus" -->
          <!--arg type="a{su}" name="TestStatus"/ -->
    <!--/signal -->
  </interface>
</node>
```

Abbildung 5.2: EsdDaemonEvent.xml des Interfaces "esd.esdtest.daemonevent"

Die Implementierung erfolgt hier genauso wie beim Abschnitt „Prüfmodul ist Server für Testdaten.

Das Prüfmodul ist Client für Anweisungen an den Bearbeiter

In diesem Fall, ist das Prüfmodul der Client. Das Prüfprogramm soll Methoden der Bedienoberfläche aufrufen. Diese Methoden sind Anweisungen des Systems an den Bearbeiter um den ESD Test durchführen zu können. Sie können sein:

- ChangeProbe: Beim Aufruf diese Methode wird der Bearbeiter aufgefordert die Prüfspitze zu wechselt. Der Bearbeiter bekommt angezeigt, eine Nachricht mit dem Probetyp, der am Roboterhand zu montieren ist.
- SetUp: Der Aufruf dieser Methode weist den Bearbeiter an, dass passende HBM einzustellen und die passenden Prüfspitze zu montieren.
- PrepareTest: Der Aufruf diese Methode weist den Bearbeiter an, dass er die Vorbereitungen für den Test durchzuführen hat.

Das System bekommt nach jedem Aufruf eine Rückmeldung, ob der Bearbeiter die Arbeiten ausgeführt hat.
Die Abbildung 5.3 zeigt die XML-Datei, die das Interface "esd.esdtest.useraction" beschreibt. Die Datei definiert die Methoden des Clients (der Bedienoberfläche), die vom Prüfprogramm aufgerufen werden.

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/esd/esdtest/useraction">
  <interface name="esd.esdtest.useraction">
    <!-- Methode to indicate the user which probe is needed for the test -->
    <annotation name="org.freedesktop.DBus.GLib.CSymbol" value="user_action"/>
    <method name="ChangeProbe">
          <annotation name="org.freedesktop.DBus.GLib.CSymbol" value="user_action_change_probe"/>
          <arg type="b" name="NewProbeTyp" direction="in"/>
          <arg type="b" name="isChanged" direction="out"/>
    </method>

    <!-- Methode to indicate the user the configuration for the test -->
    <method name="SetUp">
          <annotation name="org.freedesktop.DBus.GLib.CSymbol" value="user_action_set_up"/>
          <arg type="b" name="ProbeTyp" direction="in"/>
          <arg type="d" name="HBM_R" direction="in"/>
       <arg type="d" name="HBM_C" direction="in"/>
          <arg type="b" name="isSetUp" direction="out"/>
    </method>
    <!-- Methode to tell the user to Prepare Test -->
    <method name="PrepareTest">
          <arg type="s" name="Instructions" direction="in"/>
          <arg type="b" name="isMade" direction="out"/>
    </method>
  </interface>
</node>
```

Abbildung 5.3: GuiProcess.xml: Beschreibung der Methoden des Clients für das System

Bei der Ausnutzung der XML Introspection Datei seitens des Clients wird die passende Header-Datei mit Prototypfunktion zur Benutzung des Remote-Dbus Objekts, automatisch generiert. Bei der Ausführung des Befehls: dbus-binding-tool --mode=glib-client wird eine Header-Datei, die Inline-Funktion der Methode beinhaltet, generiert. Für den Clientteil des Prüfprogramms wird der Befehl wie folgt ausgeführt:
dbus-binding-tool --mode=glib-client --output=GuiProcessBindings.h --prefix=user_action GuiProcess.xml
Die Header-Datei beinhaltet für jede Methode eine Inline Funktion mit Parametern für: einen blockierenden Aufruf, einen nicht - blockierenden Aufruf und einen nicht blockierenden Rückruf.
Beim Client, wird nach der Verbindung mit dem Bus, der Proxy angelegt dann die entfernten Methoden aufgerufen. Die oben beschriebene XML Datei beschreibt folgende Methoden:
esd_esdtest_useraction_set_up()
esd_esdtest_useraction_change_probe()
esd_esdtest_useraction_prepare_test()
Jede diese Methode hat als Parameter den Proxy und seine Argumente.

5.3 Schwierigkeiten

Bei der Realisierung dieser Arbeit, war die Implementierung am meisten von Problemen begleitet. Diese Probleme haben dazu geführt, dass der geplante Zeitplan nicht ganz wie geplant eingehalten wurde. Auf Grund dieser Probleme wurden Maßnahmen getroffen, die letzt endlich die Art der Realisierung des Projekts ausmachen. In diesem Abschnitt werden die wichtigsten diese Probleme erläutert.

Verfügbarkeit der Ressourcen

Am Prüfplatz werden noch ESD-Tests mit dem alten Programm durchgeführt. Das Problem daran ist, das ein ESD-Test bis zu drei Tage dauern kann. Wegen dieser Benutzung des Roboters war es nicht immer möglich, das Prüfmodul jeder Zeit zu testen. Im WABCO Intranet gibt es eine Ressourcenplanung, in der die Nutzung des Prüfplatzes eingetragen ist. Mit deren Hilfe konnte die Benutzung des Prüfplatzes vorgeplant werden, oder wenn nötig eine Absprache mit dem zuständigen Bearbeiter getroffen werden.

Fehlen des Clients (Die Bedienoberfläche)

Die vorliegende Arbeit beinhaltet nicht die Implementierung der Bedienoberfläche (der Client). Diese soll erst in zukünftigen Arbeiten entstehen. Es war jedoch unmöglich das Prüfprogramm ohne Client zu testen. Deshalb wurden zwei kleine (Ersatz-) D-Bus Clients mitentwickelt um zu prüfen, ob das System Daten von der Bedienoberfläche empfangen kann. Dabei sollte auch das Datenformat seitens des Clients definiert werden. Es wurde auch ein kleiner Server implementiert um zu prüfen, ob das Prüfprogramm Daten über D-Bus übermitteln kann.

Kommunikation mit Hardwarekomponenten

Die Implementierung der Kommunikation mit den Hardwarekomponenten hatte ebenfalls viele Schwierigkeiten bei der Implementierung gebracht. Um diese Funktionalität zu implementieren war es wichtig, die passende Bibliothek zu kennen, die die Kommunikation unterstützt. Außerdem muss bekannt sein, wie die Hardware auf Befehle reagiert. Das Timeout Management hat auch zur Schwierigkeiten bei der Implementierung der Kommunikation mit den Geräten geführt. Es mussten passende Timeout-Zeiten gefunden werden. Das war besonders auffällig beim Ansprechen des Roboters, bei dem die Timeouts für einige Befehle des Roboters unterschiedlich sind.

System D-Bus

Das Verständnis des Nachricht-Systems D-Bus war nicht einfach, weil es dazu bislang keine ausführliche Dokumentation gibt. Das D-Bus-System, welches im Jahr 2004-2005 entwickelt wurde, hat nur eine unvollständige Anleitung im Internet. Ein großes Problem war die Übertragung von komplexen Datenstrukturen. D-Bus besitzt seine eigenen Datentypen, die aus einer Menge von Basistypen und mehreren Containertypen besteht. Das heißt, die Daten die über D-Bus transportiert werden, müssen in dieses Datenformat umgewandelt werden. Das ist vor allem bei mehrfach verschachtelt Strukturen wie „a(uss(iddd)(iddd))“ problematisch. Für die Repräsentierung solcher Daten im D-Bus Typsystem wurde bis zum Ende der Implementierung kein passendes Format gefunden. Damit das Prüfprogramm die kompletten Testdaten zum Starten eines ESD-Tests sammeln kann, wurde eine Funktion zur Ausfüllung der Testdaten aufgebaut. Diese Funktion übergibt an das Prüfprogramm die Steckerbeschreibung des Prüflings.[E1]

Kompilieren der Treiber

Es war geplant, das ganze System mit „Automake“ und „Libtool“ zu kompilieren. Aber die Erzeugung von Bibliothek (Extension „.so“) hat mit dem Kompilierungstool nicht funktioniert. Deshalb werden die Treiber in diesem Projekt manuell kompiliert.

5.4 Test

Die entwickelte Software wurde laufend während der Implementierung getestet. Es wurden besonders beim D-Bus, für uns eine total neue Welt, erst kleine Prototypen mit Client und Server aufgebaut und getestet. Es wurden Prototypen mit Ein- und Ausgabe für Methoden und Signale getestet. Für das Treiber-Konzept wurden ebenfalls kleine Prototypen geschrieben und getestet. Ein kleiner Treiber mit einem Modul zum Abrufen der Funktionen des Treibers wurde geschrieben und geprüft. Ein Prototypmodul zum automatischen Laden von Treiber wurde auch implementiert und getestet.
Später wurde D-Bus Client-Server mit echtem Interface entwickelt und getestet. Es wurde auch dann für jedes einzelne Gerät ein Treiber implementiert und die Funktionen getestet. Dann wurde ein Modul, das alle Treiber laden und nach ihrer Version sortieren kann, implementiert und getestet.
Am Ende wurden alle Module im System eins nach dem anderen integriert und weiter getestet. Anschließend wurde der Ablauf implementiert und getestet.
Im Folgenden werden mit Hilfe der Anforderungsliste, der Szenarien und dem im Abschnitt 3.7 definierten Qualitätsmodell einige der durchgeführten Tests ausführlich dokumentiert.

5.4.1 Aus der Spezifikation abgeleitete Tests

In diesem Abschnitt werden ein paar Black-Box-Tests die bei der Implementierung des Systems durchgeführt wurden, dokumentiert. Diese Tests dienen dazu, die tatsächliche Verhaltene das implementierte System mit den erwarteten Verhalten, die in der Anforderungsliste beschrieben sind, zu vergleichen.

Test 1: Treiber RLC-Messbrücke soll Information über Impedanz (R, C) liefern

Fall 1:

Eingabe: Ein Kondensator ist an der RLC-Messbrücke angeschlossen, Einstellung der RLC-Messbrücke auf Seriell
Sollwerte: Widerstand- und Kapazitätswert des Kondensators, die auf dem Display der RLC Messbrücke angezeigt werden: Kapazität = 923,6 µF und Widerstand= 0,0951 Ω
Testdurchführung: Aufrufen der Funktionen read_resitance() und read_capacitance() des Treibers RLC-Messbrücke. Ausgabe der Ergebnisse auf der Konsole und Vergleichen mit den Werten auf dem Display der RLC-Messbrücke
Ergebnisse: read_capacitance() lieferte 9,45e-08, und read_resitance() lieferte: 0,0968 Ω

Fall 2:

Eingabe: Ein Kondensator ist an der RLC-Messbrücke angeschlossen, Einstellung der RLC-Messbrücke auf „Parallel“
Sollwerte: Widerstand- und Kapazitätswert des Kondensators, die auf dem Display der RLC Messbrücke angezeigt werden. Zwar: Kapazität = 703,7 µF und Widerstand= 0,419 Ω

Testdurchführung: Aufrufen der Funktionen read_resitance() und read_capacitance() des Treibers RLC-Messbrücke. Ausgabe der Funktionen auf der Konsole und Vergleichen mit den Werten auf dem Display der RLC-Messbrücke
Ergebnisse: read_capacitance() lieferte 7,12e-08, und read_resitance() lieferte: 0,423 Ω. Diese Werte stimmen mit den erwarteten Werten überein.

Test 2: Das Prüfprogramm soll automatisch die Mindestfähigkeiten eines Treibers finden.

Eingabe: Drei Treiber werden in Treiberverzeichnis gespeichert: „driver_rlcbridge.so“, „driver_parport.so“ und „driver_parport2.so“. jede Treiber besitzt eine Basisvariable Interface, die den Typ und die Version des Treibers enthält. Für jeden dieser Treiber sind eingetragen:

- Für den Treiber „driver_rlcbridge.so“ Typ = RLC_METER und Version = 1,
- für den Treiber „driver_parport.so“ Typ = PARPORT und Version = 1 und
- für den Treiber „driver_parport2.so“ Typ = PARPORT und Version = 6.

Hier ist anzumerken, dass wir zwei Treiber für den Parallel-Port haben.
Sollwerte: Von diesen drei Treibern sollen nur die Treiber „driver_rlcbridge.so“ und „driver_parport2.so“ geladen werden. Der Treiber „driver_parport.so“ wird nicht geladen, weil er die Mindestfähigkeiten nicht erfüllt, im Vergleich zu dem anderen Treiber „driver_parport2.so“ für den parallelen Port, der eine größere Versionsnummer hat.
Testdurchführung: Funktion „load_modules()“ aufrufen, Typ und Version der geladene Treiber auf der Konsole ausgeben
Ergebnisse: Auf der Konsole hatten wir folgende Ausgabe: Typ = RLC_METER, Version = 1 und Typ = PARPORT, Version = 6. Der Test wurde erfolgreich abgeschlossen.

Test 3: Das Prüfprogramm soll über D-Bus mit dem Client kommunizieren.

Eingabe: Ein D-Bus Client und ein D-Bus Server, der Client schickt eine Nachricht an den Server. Die Nachricht hat folgendem Inhalt: ein String Testname = "Test1", ein Integer Pinanzahl=32; und „Ack“ eine boolesche Variable, die aussagt ob die Nachricht empfang wurde oder nicht.
Sollwerte: Nachdem der Client die Nachricht gesendet hat, sollte seine Variable „Ack“ den Wert TRUE haben. Der Server sollte die Nachricht mit den Werte „Test1“ und „32“ geliefert bekommen.
Testdurchführung: Den D-Bus Server in einer Konsole ausführen, dann den D-Bus Client in einer anderen Konsole ausführen. Der Wert „Ack“ seitens des Clients auf der Konsole anzeigen lassen und die Werte der gelieferten Nachricht seitens des Servers ausgeben.
Ergebnisse: In der Konsole des Client lässt sich der Wert TRUE lesen, und in der Konsole des Servers die Werte strValue = "Test1", intValue=32; Der Test ist erfolgreich.

Im Anhang A.1 bitte finden Sie die Liste der Anforderungen mit den Ergebnissen der jeweiligen Tests.

5.4.2 Vom Qualitätsmodell abgeleitete Tests

In diesem Abschnitt werden der Qualitätsziele des Prüfprogramms überprüft. Bewertet wird, ob die Qualitätsziele des zu entwickelten System erreicht wurden oder nicht.

Die Korrektheit:

Der Test der Korrektheit wurde mit Hilfe das in [R&S07] definierte White-Box-Verfahren durchgeführt. Die definierten Aktivitätsdiagramme werden hier benutzt, um die einzelnen Schritte des Tests zu überprüfen. Bei der Durchführung dieses Tests, kommen die Augen des Betrachters zum Einsatz. Man sollte überprüfen, ob der Roboter, genau dem Prüfablauf folgt. Wegen der Zeit, die man braucht um eine komplette elektronische Baugruppe zu testen, wird hier nur ein minimaler Test definiert. Es wird die Durchführung der ESD-Prüfung einer elektronischen Baugruppe nach einem vorgegebene Testablauf geprüft.

Fall 1:

Eingabe: Als Eingabedaten werden die in Abbildung T.1 der Anhang repräsentierten Testdaten angewendet. Diese Testdaten definieren nach ISO 10605 einen Testablauf mit folgenden Informationen wie in Abschnitt 2.4.1:

- Ein Prüfling, bei dem nur zwei Stecker und drei Pins pro Stecker geprüft werden
- Eine Impedanzmessung
- Zwei Kontaktentladungen mit drei Prüfpulsen pro Prüfposition gefolgt von einer Impedanzmessung
- Zwei Luftentladungen mit zwei Prüfpulsen pro Prüfposition gefolgt von einer Impedanzmessung

Sollwerte: Der Prüfablauf soll genau nach den Aktivitätsdiagrammen aus den Abbildungen 4.6 bis 4.10 erfolgen. Ein Testreport für den Test soll automatisch erstellt werden. Der Inhalt dieses Testreports soll mit dem definierten Testdaten übereinstimmen.
Testdurchführung: Die zu testende elektronische Baugruppe wurde am Testtisch montiert. Das Basiskoordinatesystem wurde angelernt. Die Geschwindigkeit des Roboters wurde auf 50% reduziert, um genau zu sehen, was der Roboter tut. Der Testablauf wurde im Modul Client festgelegt. Das Prüfmodul wurde gestartet und die Testdaten durch den Client gesendet. Jetzt muss der Tester die Bewegungen des Roboters beobachten und mit den Aktivitätsdiagrammen abgleichen. Bei der Impedanzmessung wurde geprüft, ob der Roboter exakt zwei Stecker und pro Stecker nur drei Pins angefahren hat. Bei der Luft- und Kontaktentladungen wurde auch geprüft, ob der Roboter sich entsprechend dem vorgegebenen Testablauf und Aktivitätsdiagrammen bewegt hat. Der Tester kann dabei hören, wenn ein Triggerimpuls oder die Fahrfreigabe gesetzt wurde. Außerdem wird durch Beobachtung der LED-Anzeige des ESD-Detektors auch geprüft, wann das Masserelais umgeschaltet wurde. Am Ende, wird der Ablauf der in den Testdaten definiert wurde mit dem resultierenden Testreport (siehe Abbildung T.2 im Anhang) verglichen. [E2]

Ergebnisse: Durch den Test konnte die Korrektheit (in diesem Beispiel) erfolgreich nachgewiesen werden. Der im Anhang (Abbildung T.2) angehängte Testreport stimmt mit den Testdaten überein und zeigt die Ergebnisse des gesamten Tests.[E3]

Fall 2:

Im diesem zweiten Fall wird überprüft, ob der Ablauf der Testdurchführung ebenfalls mit den Aktivitätsdiagrammen auch bei anderen Testdaten übereinstimmt. Der Ablauf des Tests wurde daher wie im ersten Fall durchgeführt. Als Eingabedaten wurde die in Abbildung T.3 der Anhang definierte Testdaten genommen. Auch dieser Test verlief erfolgreich, die Abbildung T.4 der Anhang zeigt den Testreport dieses Falls.

Die Erweiterbarkeit und Wartbarkeit

Zwei Qualitätsaspekte, die zur Erweiterbarkeit und Wartbarkeit der in dieser Arbeit entwickelten Software beitragen, wurden in Qualitäts-Modellen beschrieben: das Einsetzen eines neuen Treibers und das dynamische Laden von Treibern.
Das Einsetzen neuer Treiber: Dieser Qualitätsaspekt wurde nicht getestet, weil in dieser Phase des Projekts schon alle Treiber implementiert waren und es keinen zusätzliche Treiber gab, mit dem man einen Test hätte durchführen können. Nach Meinung eines Experten würden allerdings zwei Tage ausreichen, um mit den Konzepten dieser Arbeit einen neuen Treiber einzusetzen. Damit wäre laut dem Qualitätsmodell die Erweiterbarkeit in diesem Punkt zufrieden stellend gewährleistet, da Andere mit wenig Aufwand neue Treiber hinzufügen können.
Das dynamische Laden von Treibern: Dieser Qualitätsaspekt wurde mit dem Black-Box-Verfahren wie folgt getestet:

Fall 1:

Eingabe: Alle Treiber, die das Prüfprogramm braucht.
Sollwerte: Das Prüfprogramm soll den Namen des geladenen Treibers und dessen Status und Eigenschaften des zugehörigen Gerätes auf der Konsole anzeigen. Außerdem sollte das Prüfprogramm die Meldung „Test Station is OK“ auf der Konsole anzeigen.
Testdurchführung: Alle Treiber, die das Prüfprogramm braucht um einen Test durchzuführen, wurden im Treiberverzeichnis gespeichert. Dann wurde das Prüfprogramm gestartet, die Testdaten durch den Client gesendet und die Ausgabe auf der Konsole gelesen.
Ergebnisse: Das Prüfprogramm hat die in der Abbildung T.5 der Anhang gezeigte Information auf der Konsole ausgegeben. Am Ende der Abbildung erkennt man die Meldung „Test Station is OK“, d.h. alle Treiber wurden geladen und erfolgreich getestet.

Fall 2:

Eingabe: Alle Treiber des Prüfprogramms ohne den Treiber der Wetterstation.
Sollwerte: Das Prüfprogramm soll nach dem Aktivitätsdiagramm 4.5 eine Fehlermeldung auf der Konsole anzeigen und das Prüfmodul beenden, wenn alle Treiber nicht initialisiert wurden.
Testdurchführung: Der Treiber „driver_weatherstation.so“ wurde aus dem Treiberverzeichnis gelöscht. Dann wurde das Prüfprogramm gestartet. Der Tester muss dann die Ausgabe auf der Konsole beobachten.

Ergebnisse: Kurz nachdem Starten des Prüfprogramms wurde die Nachricht „The Driver -- driver_weatherstation.so was not load“ auf der Konsole des Clients angezeigt. Damit war dieser Testfall erfolgreich.

Beide Fälle zusammen ergeben, dass der Prüfplatz erweiterbar und wartbar ist.

Die Betriebsicherheit und Zuverlässigkeit

Hier wird geprüft, ob beim Stopp der Roboter auf eine sichere Position fährt. Dieses Qualitätsziel wurde mit den Testdaten der Abbildung T.3 geprüft. Die Augen kommen auch in diesem Fall zum Einsatz.

Eingabe: Ein Prüfling, die Testdaten der Abbildung T.3 der Anhang.
Sollwerte: Nach dem Befehl „Stopp“ soll der Roboter an die Sicherheitsposition des Steckers fahren. Anschließen durch den Befehl „continue“ soll der Roboter an die Initialposition fahren, ohne den Prüfling zu berühren.
Testdurchführung: Den Prüfling am Testtisch anlegen, das Prüfmodul starten. Nachdem der Roboter den ersten Pin angefahren hat, Stopp drücken und beobachten wie der Roboter sich verhält. Dann auf „Weiter“ drücken und wieder beobachten wie sich der Roboter verhält.
Ergebnisse: Nach dem Stoppbefehl fährt der Roboter seine Hand zur Sicherheitsposition des Steckers. Dann wurde der Befehl „continue“ gegeben, woraufhin der Roboter zu seiner Initialposition gefahren ist ohne den Prüfling zu berühren. Danach hat er den Test an der Stelle fortgesetzt, an der er unterbrochen wurde. Damit sind Betriebssicherheit und Zuverlässigkeit laut Qualitätsmodell erfüllt.

Die Flexibilität

Die Flexibilität wurde auch bei der Korrektheit getestet, da die Prüfspitze für die Luftentladung gewechselt worden ist, nachdem der Roboter mit der Kontaktentladung fertig war. Danach wurde der Test lückenlos fortgesetzt.

Die Effizienz

Die Effizienz wurde bei der Korrektheit und die Betriebsicherheit und Zuverlässigkeit getestet. Die Reaktionszeit bei „Pause“ und „Stopp“ ist noch zu lang, da der Roboter nachdem der Befehl gesendet worden war noch ein bis zwei Bewegungen ausgeführt hat, bevor er anhielt. Damit ist die Effizienz nur eingeschränkt gewährleistet.

5.5 Ergebnisse

Eine Analyse der Ergebnisse dieser Tests zeigt, dass die wichtigeren Qualitätsziele des Prüfmoduls erreicht wurden. Nur die Effizienz konnte nicht zufrieden stellend nachgewiesen werden, hat aber auch keine große Priorität.
Es muss auch erwähnt werden, dass der Fall eines plötzlichen Systemausfalls nicht berücksichtigt wurde. Dieser Fall betrifft die Betriebsicherheit und Zuverlässigkeit. Es ging

darum zu wissen, wie der Bearbeiter seinen Test nach diesem Ausfall fortsetzen würde ohne den ganzen Test noch einmal durchführen zu müssen. Eine einfache Lösung dafür ist, dass der Bearbeiter mit Hilfe des Testreports weist, wo sein Test unterbrochen wurde und dem entsprechend seinen neuen Test konfigurieren kann.
Es wurden ungefähr 85 Prozent der Anforderungen getestet und mit Gut bewertet. Von den restlichen nicht erfüllten Anforderungen gibt es mehrere, die aus Zeitgründen nicht realisiert wurden. Dabei wurde aber strikt nach Kundenpriorität vorgegangen, zudem ist auch das Konzept schon vorhanden, so dass die fehlenden Anforderungen leicht nachgebessert werden können.

5.6 Zusammenfassung

In diesem Kapitel wurden technische Details zur Realisierung des Prüfmoduls erläutert. Besonders wichtig waren hier die unterschiedlichen Schnittstellen, die benutzt wurden um die Kommunikation mit den Geräten herzustellen. Herauszustellen ist hier das System D-Bus, das die Interprozess-Kommunikation ermöglicht
In diesem Kapitel wurde auch auf die Probleme, die bei der Entwicklung auftraten und deren Lösung, eingegangen. Am wichtigsten in diesem Kapitel sind die Tests des entwickelten Prüfmoduls. Obwohl die Entwicklung des Prüfmoduls nicht vollständig abgeschlossen ist, zeigen diese Tests, dass die größten und wichtigsten Teile des Projekts realisiert wurden.

Kapitel 6

Schlussfolgerungen

6.1 Zusammenfassung

In dieser Arbeit wurden die Konzeption und die Entwicklung eines Prüfmoduls zur Steuerung eines ESD-Prüfplatzes für elektrostatische Entladungen realisiert. Der Server der neuen Generation des Steuerprogramms des Prüfplatzes wurde präsentiert. Es wurde zunächst (in Abschnitt 2) die allgemeine Grundlage eines elektrostatischen Entladungs-Tests (ESD-Test) geschaffen.
In Abschnitt 3 wurden die Stakeholder identifiziert, die Ziele des Projekts wurden festgelegt und der Kontext des Systems wurde abgegrenzt. Bei der Kontextabgrenzung wurden die Nachbarsysteme des Prüfmoduls identifiziert, zu denen auch die Geräte des Prüfplatzes zählen.
Dann wurde eine logische Kontextabgrenzung ausgewählt und eine entsprechende physikalische Kontextabgrenzung definiert. Entscheidend war hier die Auswahl eines geeigneten Modell-Typs für die logische Kontextabgrenzung, der für dieses System die beste Übersicht verschafft. Aus dem Vergleich von Klassendiagramm, Use-Case-Diagramm und Sequenzdiagramm, hat sich dabei das Klassendiagramm für unser System als am besten geeignet gezeigt. Das liegt daran, dass das Klassendiagramm die Kommunikation des Prüfmoduls mit seinen Nachbarsystemen am besten aufzeigt.
Die Interfaces für die Kommunikation mit den Nachbarsystemen wurden durch physikalische Kontextabgrenzung festgelegt. Es wurden in Abschnitt 3.4 Systemprozesse des entwickelten Systems gefunden. Bei der Erhebung der Anforderungen in Abschnitt 3.5 wurde hauptsächlich auf das Interview mit dem Experten und die Analyse des Bestandsystems zurückgegriffen, um eine Liste der Anforderungen zu erstellen. Von diesen Anforderungen spielten das D-Bus- und Treiber-Konzept eine sehr große Rolle. Aus diesen gesammelten Anforderungen, wurde ein Qualitätsmodel des Prüfmoduls abgeleitet, um die Umsetzung mit den Anforderungen vergleichen zu können.
Beim Konzept in Abschnitt 4 wurde erst die Architektur des Verwaltungssystems des Prüfplatzes definiert: eine Client – Server Architektur. In diese Arbeit wurde das Backend (der Server) entwickelt. Eine Hauptaufgabe dieser Arbeit wurde die Herstellung der Kommunikation des Servers mit dem Client über ein Nachricht-System-Bus: D-Bus. Das D-Bus-System wurde in Abschnitt 4.5 modelliert. Eine zweite Hauptaufgabe war es, die Kommunikation mit den Geräten des Prüfplatzes zu definieren. Dabei wurden zwei Vorgehensmodelle eingeführt und verglichen: die Architektur *Stern 1* mit High- und Low-Level Funktionsmodul und die Architektur *Stern 2* mit Ein-Level Funktionsmodul. Das Ergebnis dieses Vergleichs hat riesige Vorteile für die Architektur *Stern 2* mit Ein-Level Funktionsmodul ergeben. Diese Architektur hat sich als einfacher, übersichtlicher und am besten bewährt erwiesen, um die Anforderungen eines Treiberkonzepts umzusetzen.
Eine Architektur des Servers wurde dann festgelegt, um eine Übersicht des zu entwickelnden Systems zu haben. Um die genaueren Abläufe der System Prozesse des Prüfmoduls darzustellen, wurden Aktivitätsdiagramme verwendet. Das in dieser Arbeit benutzte

Datenmodell wurde in Abschnitt 4.3 festgelegt. Während der Erstellung des Konzepts wurde auch das D-Bus- und Treiber-Konzept sowie ihre Anwendung in diesem Projekt erläutert.
Das definierte Konzept hat sich während der Implementierung in Abschnitt 5 gut bewährt, da die Reihenfolge und das Vorgehen der Implementierung daraus schnell abgeleitet werden konnte. Das System wurde in der Programmiersprache „C“ mit der APIs „GLib“, „GMODULE“ und „libdbus-glib“, die unter der GPL veröffentlicht sind, implementiert. Die Implementierungsphase wurde in zwei Teile getrennt. Die Module für Treiber wurden erstmal implementiert, dann wurde das Prüfmodul, zusammen mit der Integration der Module, selbst implementiert. Ein wichtiger Punkt der Implementierung war das Ansprechen der Geräte des Prüfplatzes, die erläutert wurde. Außerdem war hier die praktische Umsetzung des D-Bus-Systems während der Entwicklung des Prüfmoduls wichtig, die in Abschnitt 4.5 erklärt wurde.
Die Implementierungsphase endet mit der Testphase, in der der Stand der Implementierung im Vergleich zu den definierten Anforderungen und dem Qualitätsmodell ausgewertet wurde. Es wurde eine Testdatenbasis geschaffen und Tests durchgeführt. Die Auswertung der Tests aller Qualitätsziele mit höheren Prioritäten führte zu den erwünschten Ergebnissen. Nur das letzte Qualitätsziel mit der niedrigsten Priorität führte nicht zu dem erwarteten Ergebnis. Allerdings ist die Effizienz für die Durchführung eines ESD-Tests nicht wesentlich, und macht sich aber nicht störend bei der Durchführung eines ESD-Tests bemerkbar. Die Auswertung der Tests der Anforderungen führte in ungefähr 85 Prozent zu den erwarteten Ergebnissen. Die Tests zeigen, dass die wichtigsten Erwartungen dieses Prüfmoduls erreicht wurden: die Implementierung der Treiber, das dynamisch Laden der Treiber und die Kommunikation mit dem Client über D-Bus.

6.2 Ausblick

Das entwickelte Prüfmoduls zur Steuerung eines Prüfplatzes für elektrostatische Entladungen, verfügt über grundlegende Fähigkeiten die zur Durchführung eines ESD-Tests nötig sind. Aus Zeitgründen wurden aber nicht alle Funktionalitäten umgesetzt und genügend Tests durchgeführt, um das Prüfmodul in Betrieb zu nehmen. Zudem benötigt das Prüfmodul als zentrales Verwaltungssystem unbedingt einen Client, um tatsächlich produktiv eingesetzt zu werden. Die Anregungen zur Verbesserung, Erweiterung und Inbetriebnahme das vorgestellte Prüfmodul wären:

- Die Implementierungen fehlender Anforderungen des in dieser Arbeit entwickelten Prüfmoduls. Diese wäre von keinem größeren Aufwand, da das Konzept der Realisierung ein Bestandteil dieser Arbeit ist.
- Ein System zu entwickeln, das den Roboter ermöglichen würde, die Prüfspitze, die am ESD-Generator montiert ist, selbst zu wechseln. Dieses Thema wurde häufig während des Projekts diskutiert. Wenn der Roboter so ein System haben würde, wäre es möglich einen ESD-Test bis zum Ende ohne Unterbrechungen durchführen zu können. Die Unterbrechungen auf Grund des Wechsels der Prüfspitze führen sehr oft zur Verlängerung der Testzeit., da der Bearbeiter regelmäßig zum Roboter gehen muss, um zu überprüfen, ob der Roboter angehalten hat, damit der Benutzer die Prüfspitze.wechselt.
- Die Entwicklung eines Systems, das dem Bearbeiter auf seinem Rechner den aktuellen Status des Testablaufs melden kann. Dieses System könnte die Ergebnisse der in dieser Arbeit mit D-Bus entwickelte Funktion „TestStatus“ benutzen.
- Das wichtigste fehlende Teil ist jedoch die Bedienoberfläche (der Client), des vorgestellten Prüfmoduls.

Quellenverzeichnis

[Barr99] Michael Barr: Programming Embedded Systems in C and C++. O'Reilly &Associates, Inc. 1999

[Coc00] Alistair Cockburn: Writing Effective Use Cases. Addison-Wesley, Boston, 2000.

[Drep] Ulrich, Drepper: How To Write Shared Libraries: August 20, 2006. http://people.redhat.com/~drepper/dsohowto.pdf

[Dem79] Tom Demarco: Structured Analysis and System Specification. Prentice Hall, Inc. 1978, 1979.

[GA00] C. Gernert, N. Ahrend/ IT-Management/ Syst statt Chaos – Ein praxisorientiertes Vorgehensmodell, Olldenburg 2000

[Gott02] © Armin Gottschalk: ESD Automotive Die ESD Kette www.rmctech.de 0233V01: 2002

[Gr01] Graeme Smith: Specifying mode requirements of embedded systems: Software Verification Research Centre University of Queensland, Australia Email: smith@svrc.uq.edu.au ACSC2002

[H&R02] Peter Hruschka, Chrisp Rupp: Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML. Carl Hanser Verlag München Wien 2002.

[Koop96] Philip J. Koopman, Jr: Embedded System Design Issues (the Rest of the Story) Proceedings of the International Conference on Computer Design (ICCD 96)

[Kurt07] Kurt Schneider: Abenteuer Software Qualität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement. Heidelberg dpunkt-Verlag 2007

[LAN] Yann, LANGLAIS: Jouer avec la « libdl.so » : 2004, 2005 : http://chl.be/glmf/ilay.org/yann/articles/dlfcn/#renvoi2

[Lar05] Graig Larman: UML2 und Patterns angewendet – Objektorientierte Softwareentwicklung. Mitp-Verlag/Bonn 2005.

[Oest02] Oestereich, Bernd: Die UML - Kurzreferenz für die Praxis: kurz, bündig, ballastfrei. 2.,überarb. Aufl. – München ; Wien : Oldenbourg, 2002

[R&S07] Chris Rupp & die Sophisten: Requirement engineering und Management – Professionelle, iterative Anforderungs-analyse für die Praxis. Carl Hanser Verlag München Wien 2007.

[Rob06] Williams Rob: Real Time Systems Development. Elsevier 2006.

[Rup04] Chris Rupp. Requirements-Engineering und - Management : professionelle, iterative Anforderungsanalyse für die Praxis. Hanser, 2004.

[W&W03] Matthias Weber and Joachim Weisbrod, DaimlerChrysler Research: Requirements Engineering in Automotive Development: Experiences and Challenges. IEEE Software 2003 http://computer.org/software

[VACIMNVU05] Olivier Vogel, Ingo Arnold, Arif Chughtai, Edmund Ihler, Uwe Mehlig, Thomas Neumann, Markus Völter, Uwe Zdun: Sofware Architektur Grundlage – Konzepte – Praxis. Elsevier GmbH, München 2005.

http://en.wikipedia.org/wiki/IEEE_488

http://de.wikipedia.org/wiki/Parallele_Schnittstelle

Anhang

A.1 Tabellarische Auflistung der Anforderungen

Die Anforderungen werden hier nach unterschiedlicher Gruppe unterteilt, die gleich definiert werden.
Die Arte der Anforderungen, davon zählt hier:
Die funktionalen Anforderungen
Die technischen Anforderungen
Die Anforderung an die Benutzerschnittstelle
Die technologischen Anforderungen
Die Qualitätsanforderung

Für jede Arte von Anforderungen werden die Anforderungen nach Betroffene unterteilt. Die Betroffenen sind hier:
Der Bearbeiter, das Prüfprogramm, der Treiber Wettestation, Robotersteuerung, ESD-Generator, RLC-Messbrücke, Parallel Port.
In der Tabelle der Betroffene werden die Anforderungen nach Priorität angeordnet und Für jede Anforderungen werden die Quelle und der Status der Realisierung der Anforderung angegeben.

Status der Realisierung:
I implementiert
NI nicht implementiert
FN funktioniert nicht
NR nicht relevant
TI teilweise implementiert

Quellen:
E Experten
A Analyse des Systems
M Mitarbeiter der Firma WABCO

Die funktionalen Anforderungen

Der Bearbeiter		
muss das Prüfmodul starten können	E	I
muss Testdaten ans Prüfprogramm liefern können	E	TI
muss den Test Stoppen oder Anhalten können	E	I
muss einen unterbrochenen Test fortsetzen können	E	I
muss den Status eines Geräts abfragen können	E	I
muss ein Selbsttest anregen können	E	NI
muss den Roboter anlernen können	E	NR
muss die Spitze und HBM einstellen und wechseln können	E	NR
muss Daten and System Prüfen können	E	NR
muss den Roboter freigeben können	E	NR

Die vier letzten Anforderungen gehören nicht unserem System, sind aber Tätigkeiten, die der Bearbeiter erledigen muss, damit den ESD-Test vom Prüfprogramm richtig durchgeführt wird.

Das Prüfprogramm		
muss Treiber laden können	E	I
muss die Treiberkonfiguration beim Programmstart laden	E	I
muss die Treiber abfragen können	E	I
soll automatisch die Treiber und deren Klassen erkennen	E	I
soll automatisch Treiber und deren Klassen einbinden (Mechanismus zur Konfliktbewältigung)	E	I
soll die Messdaten des Prüflings von der GUI über D-Bus bekommen	E	FN
muss einen Messablauf entgegennehmen können	E	I
muss Steuerbefehle entgegennehmen können (Start, Stopp, Pause, Continue)	E	I
muss die RLC-Messbrücke konfigurieren können	M	I
soll überprüfen, ob der Benutzer ein korrektes HBM eingestellt hat	E	FN
soll den Status aller angeschlossenen Geräte abfragen können und zur Prüfung durch andere Programmteile bereithalten	E	I
muss den Messablauf nach dem vorgegebenen Testablauf durchführen können	E	I
soll beim Stoppbefehl den Roboter auf die Sicherheitsposition des Steckers	A	I
muss eine Fehlermeldung ausgeben, wenn D-Bus nicht funktioniert	A	I
soll jederzeit den Prüfablauf anhalten oder pausen können	E	I
soll die Konfiguration des Prüfgenerators mit der Sollvorgabe (Spannung, Intervall, Polarität, Type (Luft- oder Kontaktentladung)) vergleichen können	E	FN
soll den ESD – Generatore fordern können einen Puls auszulösen	E	I
muss den Roboter folgenden Bewegungsaufträge geben können: a. Anfahren der Sicherheitspunkte b. Anfahren der Zielpunkte (Pins, Gehäuse) c. Werkzeug wechseln (Entladungsspitze, Messspitze, GND)	E	I
falls der Benutzer einen Test anhält, soll das Prüfprogramm die aktuelle Position beibehalten	E	I
soll die Selbsttestfunktion der ESD Pistole unterstützen (ablaufen lassen und Ergebnissen melden)	E	I
soll die RLC – Konfiguration ins Prüfprotokoll schreiben	E	I
soll alle Aktionen protokollieren (Messablauf)	E	I
muss den Messablauf konfigurieren können (Spannung, Intervall, Polarität, Type (Luft- oder Kontaktentladung))	E	I
soll bei Luftentladung, nach der Erkennung einer Erfolgten Entladung die Roboterbewegung unmittelbar stoppen können	E	I
soll den Prüfkopf bei Luftentladung kontinuierlich annähern können (kein Stotterbetrieb)	E	NI
soll eine Messung nach der Pause oder Stopp fortsetzen können	E	I
soll dem Client melden wenn das HBM nicht korrekt ist	E	I
soll Informationen über eigene Fähigkeiten an den Client liefern kann	E	I
soll eine Messung nach der Pause oder Stopp fortsetzen können	E	I

Das Prüfprogramm		
falls ein Stoppbefehl folgt auf ein Pausebefehl, soll das Prüfprogramm wie beim Stoppbefehl verfahren	E	I
soll die Selbsttestfunktion der ESD Pistole unterstützen (ablaufen lassen und Ergebnissen melden)	E	I
soll den Messablauf mit der Roboterbewegung synchronisieren (z. B. soll auf Roboter warten)	E	I
soll Datum, Uhrzeit des letzten Selbsttests in das Messprotokoll der Messung schreiben	E	NI
soll regelmäßig Statusmeldungen über D-Bus versenden, um anderen Programmen die Verfolgung einer Messung zu ermöglichen	E	I
soll den Betrieb protokollieren (Logmodul)	E	NI
soll Informationen über aktuellen Betriebszustand für den Client zur Verfügung stellen können (Tabelle der Treiber und ihre Status)	E	I
soll automatisch die Server Änderungen am Betriebszustand melden können (per Broadcast)	E	I
soll bei einem Programmabbruch (Not - Aus) selbsttätig den Weg nach seiner Initiale Position finden (über Geometrie der Prüflings und Sicherheitspunkte der Elektronik)	E	NI
soll die Verifikation des Prüfplatzes einmal pro Messtag am Beginn der ersten Messung durchführen können	E	NI
soll keine Koordinatentransformation durchführen	E	I

Treiber Wetterstation		
soll eine Selbsttestfunktion haben	E	I
soll Befehle des Prüfprogramms entgegennehmen können	E	I
soll Informationen über die aktuelle Temperatur liefern können	E	I
soll Informationen über die relative Luftfeuchtigkeit liefern können	E	I
soll Informationen über den aktuellen Luftdruck liefern können	E	I
soll Informationen über eigene Fähigkeiten liefern können	E	I

Treiber Robotersteuerung		
soll eine Selbsttestfunktion haben	E	I
soll Befehle des Prüfprogramms entgegennehmen können	E	I
soll Befehlen des Prüfprogramms für das Steuerprogramm des Roboters aufbereiten und übermitteln	E	I
soll Rückmeldungen vom Roboter auswerten und ggf. weitermelden (Vollzugsmeldung, Fehler)	E	I
soll Information über den Betriebszustand z.B. aktuelle Position, gewähltes Koordinatensystem, und aktuelles Werkzeug liefern	E	I
soll Informationen über eigene Fähigkeiten liefern	E	I

Treiber ESD - Generator		
soll eine Selbsttestfunktion haben	E	I
soll Befehle des Prüfprogramms entgegennehmen können	E	I
soll auf Kommando einen ESD – Puls auslösen können	E	I
soll Informationen über eigene Fähigkeiten liefern können	E	I
soll auf Kommando einen Selbsttest durchführen können	E	I
soll Information über aktuelle Konfiguration liefern können	E	NF
soll Informationen liefern, ob eine Entladung stattgefunden hat oder nicht	E	I

Konfiguration: Spannung, Polarität, Intervall, Typ (Luft / Kontakt)

Treiber RLC – Messbrücke		
soll eine Selbsttestfunktion haben	E	I
soll Befehle des Prüfprogramms entgegennehmen können	E	I
soll Informationen über Impedanz liefern (R, C)	E	I
wenn nötig, soll die RLC – Messbrücke automatisch den Masse-Relais zuschalten	E	I
soll eigene Konfiguration liefern können	E	I
soll Informationen über eigene Fähigkeiten liefern	E	I

Konfiguration: DC-Offset, Frequenz, Ersatzschaltbild, AC-Offset

Treiber Parallel Port		
soll eine Selbsttestfunktion haben	E	I
soll Befehle des Prüfprogramms entgegennehmen können	E	I
soll durch den ESD- Detektor Informationen liefern, ob eine Entladung stattgefunden hat oder nicht	E	I
soll Druckluft ein- oder ausschalten (bei Verbindung des ionisierten Luft im Testsbereich) können	E	I
soll die Fahrfreigabe des Roboters aktivieren können	E	I
soll Informationen über eigene Fähigkeiten liefern können	E	I

Technische Anforderungen

das System muss ausschließlich in der Programmiersprache C und der Bibliothek GLib entwickelt sein	E	I
das System muss auf einem Linux-Betriebsystem laufen	E	I
das System soll auf einem Linux-Betriebssystem entwickelt werden	E	I
Kommunikation mit der Robotersteuerung erfolgt über RS232	E	I
Kommunikation mit der RLC-Messbrücke erfolgt über GPIB	E	I
Kommunikation mit der Wetterstation erfolgt über RS232	E	I
Kommunikation mit der ESD-Generator erfolgt über RS232	E	I
Kommunikation mit dem ESD-Detektor erfolgt über den Parallelport	E	I

Technologische Anforderungen

Das Prüfmodul muss mit dem Client über D-Bus kommunizieren	E	I

Qualitätsanforderungen

ESD Prüfung einer elektronischen Baugruppe nach einem vorgegebenen Testablauf steuern	E	I
Das Prüfmodul darf eine Messung nur starten, wenn der Prüfplatz fehlerfrei ist	E	I
Das Prüfprogramm muss die Prüfplatzkomponenten über ein Treiberkonzept ansprechen	E	I
Das Prüfmodul muss Treibern in Klassen organisieren	E	I
Alle Treiber müssen eine definierte Schnittstelle haben (jede Geräteklasse soll eine individuelle Schnittstelle haben) und gemeinsam ein identische Basis-Schnittstelle haben (Verwaltungszweck)	E	I
Treiber sollen allein verantwortlich für angeschlossen Hardware sein	E	I
Das Prüfmodul soll Status des Generators überwachen und auswerten können (Fehler melden, Prüfdaten ins Protokoll schreiben)	E	I
Das Prüfmodul soll die Möglichkeit bieten, eine Funktion aus verschiedenen Quellen abzufragen und die Ergebnisse diese Funktion sinnvoll zu verknüpfen. (nur ausgewählte Funktionen)	E	I
Das Prüfmodul soll synchron mit den Treiber kommunizieren können	A	I
Das Prüfmodul soll verhindern, dass ein zweites Prüfmodul gestartet wird	E	NI
Das Prüfmodul soll das Übermitteln neuer Testdaten durch den Client während einer laufenden Prüfung verhindern. (auch im Zustand Pause)	E	I
Das Prüfmodul muss wartbar und erweiterbar sein	E	I
Das Prüfmodul muss eine möglich schnelle Antwortzeit haben	E	FN
Das Prüfmodul muss testbar sein: Es muss ein kleiner Client entwickelt werden um das Prüfmodul zu testen	A	I
Das Prüfmodul muss zuverlässig sein	E	TI
Das Prüfmodul muss flexibel sein	E	I
Das Prüfmodul soll selbsttätig entscheiden können, wann ein Selbsttest durchgeführt werden muss und ihn in den normalen Prüfablauf einflechten (zusammenbringen)	E	NI
Das Prüfmodul soll wenn möglich weitere Treiber zur Ansteuerung vom Messtechnik einsetzen, (RLC-Messbrücke, Kennlinienschreiben ...)	E	NI

Gesamte Anforderungen: 104
Implementierte Anforderungen: 88

A.2 Tests

Testdaten und Testreports

```
ECU Name:       File Name ECU geometry
Info:             Prüflingsname
  Stecker 1:      Stecker_18
  Info:           Info Plug 1
  Ursprung:       X =   0,0 ;  Y =   0,0 ;  Z =   0,0
  Drehung:        A =  90,0Â°;  B =  90,0Â°;  C =   0,0Â°
  Sicherh.punkt:  X =  36,0 ;  Y =-100,0 ;  Z =-150,0
  Pins:           3
   1: p.X =  36,0 ; p.Y =   0,0 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =   0,0 ; s.Z = -50,
   2: p.X =  36,0 ; p.Y =   5,5 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =   5,5 ; s.Z = -50,
   3: p.X =  36,0 ; p.Y =  11,0 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =  11,0 ; s.Z = -50,

  Stecker 2:      Stecker_15
  Info:           Info Plug 2
  Ursprung:       X =   0,0 ;  Y =   0,0 ;  Z =   0,0
  Drehung:        A =  90,0Â°;  B =  90,0Â°;  C =   0,0Â°
  Sicherh.punkt:  X =   0,0 ;  Y =-100,0 ;  Z =-150,0
  Pins:           3
   1: p.X =   0,0 ; p.Y =   0,0 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =   0,0 ; s.Z = -50,
   2: p.X =   0,0 ; p.Y =   5,5 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =   5,5 ; s.Z = -50,
   3: p.X =   0,0 ; p.Y =  11,0 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =  11,0 ; s.Z = -50,



Testname:       File Name Esd Test
Standard:         -- ISO 10605 --
HBM:              150 pF, 2000 Ohms
Interval contact: 5,0
Interval air:     0,1
Velocity air:     0,1
frequency:        100000
AC_Offset:        1,0
DC_Offset:        1,0
Equivalent Circuit:     SERIAL
Testlevel:        [ 1] Contact ;  0x ;      0V  ; M
                  [ 2] Contact ;  3x ;      4kV ; G
                  [ 3] Contact ;  3x ;     -4kV ; G ; M
                  [ 4] Air     ;  2x ;      8kV ; G
                  [ 5] Air     ;  2x ;     -8kV ; G ; M
```

Abbildung T.1: Eingabedaten des ersten Tests

Testreport

ESD-o-Matic v0.1.0, automatic ESD test system
Test performed: Fri, 30.November 2007, 14:11

Air conditions
---
temperature: 23,7 deg.C
pressure: 998 hPa
humidity: 26 % rel.

RLC BRIDGE CONFIGURATION

DC_Offset: 1,0 Volt.
AC_Offset: 1,0 Volt.
Frequency: 100000 Hz
Equivalent Circuit : SERIE

ESD Test Procedure

ESD test according: -- ISO 10605 --
ESD test data file: T_File Name Esd Test.txt.
Human Body Model: C = 150pF, R = 2000 Ohm

Device under Test

DuT Family: File Name ECU geometry
Family info: Prüflingsname
Base system: X2.1 -> X1.13 -> X1.15

Device name: DUT Name
Device no.: DUT Nummer
Description: DUT Description

Test level: contact discharge 0 V

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,80e+01; 1,05e-08
1; 2; 1,83e+01; 1,05e-08
1; 3; 1,85e+01; 1,04e-08
2; 1; 5,51e+00; 9,39e-09
2; 2; 7,35e+00; 1,40e-08
2; 3; 1,50e+01; 1,00e-08

Test level: contact discharge 4 kV
1: X1.1 - gen = not discharged - pp = discharged
2: X1.1 - gen = not discharged - pp = discharged
3: X1.1 - gen = not discharged - pp = discharged
1: X1.2 - gen = not discharged - pp = discharged
2: X1.2 - gen = not discharged - pp = discharged
3: X1.2 - gen = not discharged - pp = discharged
1: X1.3 - gen = not discharged - pp = discharged
2: X1.3 - gen = not discharged - pp = discharged
3: X1.3 - gen = not discharged - pp = discharged
1: X2.1 - gen = not discharged - pp = discharged
2: X2.1 - gen = not discharged - pp = discharged
3: X2.1 - gen = not discharged - pp = discharged
1: X2.2 - gen = not discharged - pp = discharged
2: X2.2 - gen = not discharged - pp = discharged
3: X2.2 - gen = not discharged - pp = discharged
1: X2.3 - gen = not discharged - pp = discharged
2: X2.3 - gen = not discharged - pp = discharged
3: X2.3 - gen = not discharged - pp = discharged

Test level: contact discharge -4 kV
1: X1.1 - gen = not discharged - pp = discharged
2: X1.1 - gen = not discharged - pp = discharged
3: X1.1 - gen = not discharged - pp = discharged
1: X1.2 - gen = not discharged - pp = discharged
2: X1.2 - gen = not discharged - pp = discharged
3: X1.2 - gen = not discharged - pp = discharged
1: X1.3 - gen = not discharged - pp = discharged
2: X1.3 - gen = not discharged - pp = discharged
3: X1.3 - gen = not discharged - pp = discharged
1: X2.1 - gen = not discharged - pp = discharged
2: X2.1 - gen = not discharged - pp = discharged
3: X2.1 - gen = not discharged - pp = discharged
1: X2.2 - gen = not discharged - pp = discharged
2: X2.2 - gen = not discharged - pp = discharged
3: X2.2 - gen = not discharged - pp = discharged
1: X2.3 - gen = not discharged - pp = discharged
2: X2.3 - gen = not discharged - pp = discharged
3: X2.3 - gen = not discharged - pp = discharged

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,82e+01; 1,05e-08
1; 2; 1,83e+01; 1,05e-08
1; 3; 1,85e+01; 1,04e-08
2; 1; 5,88e+00; 9,58e-09
2; 2; 7,33e+00; 1,41e-08
2; 3; 1,50e+01; 1,00e-08

Test level: air discharge 8 kV
1: X1.1 - gen = discharged - pp = discharged

2: X1.1 - gen = discharged - pp = discharged
1: X1.2 - gen = discharged - pp = discharged
2: X1.2 - gen = discharged - pp = discharged
1: X1.3 - gen = discharged - pp = discharged
2: X1.3 - gen = discharged - pp = discharged
1: X2.1 - gen = discharged - pp = discharged
2: X2.1 - gen = discharged - pp = discharged
1: X2.2 - gen = discharged - pp = discharged
2: X2.2 - gen = discharged - pp = discharged
1: X2.3 - gen = discharged - pp = discharged
2: X2.3 - gen = discharged - pp = discharged

Test level: air discharge -8 kV
1: X1.1 - gen = not discharged - pp = not discharged
2: X1.1 - gen = not discharged - pp = not discharged
1: X1.2 - gen = not discharged - pp = not discharged
2: X1.2 - gen = not discharged - pp = not discharged
1: X1.3 - gen = not discharged - pp = not discharged
2: X1.3 - gen = not discharged - pp = not discharged
1: X2.1 - gen = not discharged - pp = discharged
2: X2.1 - gen = discharged - pp = discharged
1: X2.2 - gen = discharged - pp = discharged
2: X2.2 - gen = not discharged - pp = not discharged
1: X2.3 - gen = not discharged - pp = not discharged
2: X2.3 - gen = not discharged - pp = not discharged

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,83e+01; 1,05e-08
1; 2; 1,84e+01; 1,05e-08
1; 3; 1,85e+01; 1,04e-08
2; 1; 5,97e+00; 9,58e-09
2; 2; 7,48e+00; 1,44e-08
2; 3; 1,51e+01; 1,00e-08

Abbildung T.2: Testreport des ersten Tests

Bei den Testreporten gelten folgenden Notationen für die Testlevels:
a:Xb.c: steht für Prüfpulse Nummer a über den Pin c des Steckers b
gen: gibt aus, ob über den ESD-Generator eine Entladung detektiert wurde oder nicht
pp: gibt aus, ob über den ESD-Detektor eine Entladung detektiert wurde oder nicht

```
ECU Name:       File Name ECU geometry
Info:             Prüflingsname
  Stecker 1:      Stecker_18
  Info:           Info Plug 1
  Ursprung:       X =   0,0 ;  Y =   0,0 ;  Z =   0,0
  Drehung:        A =  90,0Â°;  B =  90,0Â°;  C =   0,0Â°
  Sicherh.punkt:  X =  36,0 ;  Y =-100,0 ;  Z =-150,0
  Pins:           3
   1: p.X =  36,0 ; p.Y =   0,0 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =   0,0 ; s.Z = -50,0
   2: p.X =  36,0 ; p.Y =   5,5 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =   5,5 ; s.Z = -50,0
   3: p.X =  36,0 ; p.Y =  11,0 ; p.Z =   0,0 / s.X =  36,0 ; s.Y =  11,0 ; s.Z = -50,0

  Stecker 2:      Stecker_15
  Info:           Info Plug 2
  Ursprung:       X =   0,0 ;  Y =   0,0 ;  Z =   0,0
  Drehung:        A =  90,0Â°;  B =  90,0Â°;  C =   0,0Â°
  Sicherh.punkt:  X =   0,0 ;  Y =-100,0 ;  Z =-150,0
  Pins:           3
   1: p.X =   0,0 ; p.Y =   0,0 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =   0,0 ; s.Z = -50,0
   2: p.X =   0,0 ; p.Y =   5,5 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =   5,5 ; s.Z = -50,0
   3: p.X =   0,0 ; p.Y =  11,0 ; p.Z =   0,0 / s.X =   0,0 ; s.Y =  11,0 ; s.Z = -50,0



Testname:       File Name Esd Test
Standard:         -- ISO 10605 --
HBM:              150 pF, 2000 Ohms
Interval contact: 5,0
Interval air:     0,1
Velocity air:     0,1
frequency:        100000
AC_Offset:        1,0
DC_Offset:        1,0
Equivalent Circuit:     SERIAL
Testlevel:        [ 1] Contact ;  0x ;      0V  ; M
                  [ 2] Contact ;  2x ;      2kV ; G
                  [ 3] Contact ;  2x ;     -2kV ; G ; M
                  [ 4] Air     ;  1x ;     15kV ; G
                  [ 5] Air     ;  1x ;    -15kV ; G ; M
```

Abbildung T.3: Eingabedaten des zweiten Tests

Testreport

ESD-o-Matic v0.1.0, automatic ESD test system
Test performed: Fri, 30.November 2007, 17:11

Air conditions
---
temperature: 24.0 deg.C
pressure: 997 hPa
humidity: 25 % rel.

RLC BRIDGE CONFIGURATION

DC_Offset: 1.0 Volt.
AC_Offset: 1.0 Volt.
Frequency: 100000 Hz
Equivalent Circuit : PARALLEL

ESD Test Procedure

ESD test according: -- ISO 10605 --
ESD test data file: T_File Name Esd Test.txt.
Human Body Model: C = 150pF, R = 2000 Ohm

Device under Test

DuT Family: File Name ECU geometry
Family info: ABS-E Basic 4S/4M
Base system: X2.1 -> X1.13 -> X1.15

Device name: DUT Name
Device no.: DUT Nummer
Description: DUT Description

Test level: contact discharge 0 V

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,26e+03; 1,04e-08
1; 2; 1,27e+03; 1,03e-08
1; 3; 1,28e+03; 1,03e-08
2; 1; 4,91e+03; 9,44e-09
2; 2; 1,63e+03; 1,43e-08
2; 3; 1,69e+03; 9,93e-09

Test level: contact discharge 2 kV
1: X1.1 - gen = discharged - pp = discharged
2: X1.1 - gen = discharged - pp = discharged
1: X1.2 - gen = discharged - pp = not discharged

2: X1.2 - gen = discharged - pp = not discharged
1: X1.3 - gen = not discharged - pp = discharged
2: X1.3 - gen = discharged - pp = not discharged
1: X2.1 - gen = discharged - pp = discharged
2: X2.1 - gen = discharged - pp = not discharged
1: X2.2 - gen = discharged - pp = discharged
2: X2.2 - gen = discharged - pp = discharged
1: X2.3 - gen = discharged - pp = discharged
2: X2.3 - gen = discharged - pp = discharged

Test level: contact discharge -2 kV
1: X1.1 - gen = not discharged - pp = discharged
2: X1.1 - gen = not discharged - pp = discharged
1: X1.2 - gen = not discharged - pp = discharged
2: X1.2 - gen = not discharged - pp = discharged
1: X1.3 - gen = not discharged - pp = discharged
2: X1.3 - gen = not discharged - pp = discharged
1: X2.1 - gen = discharged - pp = discharged
2: X2.1 - gen = not discharged - pp = discharged
1: X2.2 - gen = not discharged - pp = discharged
2: X2.2 - gen = not discharged - pp = discharged
1: X2.3 - gen = not discharged - pp = discharged
2: X2.3 - gen = not discharged - pp = discharged

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,27e+03; 1,04e-08
1; 2; 1,28e+03; 1,03e-08
1; 3; 1,29e+03; 1,03e-08
2; 1; 4,86e+03; 9,47e-09
2; 2; 1,64e+03; 1,43e-08
2; 3; 1,70e+03; 9,93e-09

Test level: air discharge 15 kV
1: X1.1 - gen = discharged - pp = discharged
1: X1.2 - gen = discharged - pp = discharged
1: X1.3 - gen = not discharged - pp = discharged
1: X2.1 - gen = not discharged - pp = discharged
1: X2.2 - gen = not discharged - pp = discharged
1: X2.3 - gen = not discharged - pp = discharged

Test level: air discharge -15 kV
1: X1.1 - gen = not discharged - pp = discharged
1: X1.2 - gen = discharged - pp = discharged
1: X1.3 - gen = discharged - pp = discharged
1: X2.1 - gen = not discharged - pp = discharged
1: X2.2 - gen = discharged - pp = discharged
1: X2.3 - gen = discharged - pp = discharged

Plug; Pin; R [Ohm]; C [F]
1; 1; 1,26e+03; 1,04e-08

1; 2; 1,27e+03; 1,03e-08
1; 3; 1,28e+03; 1,03e-08
2; 1; 4,97e+03; 9,44e-09
2; 2; 1,60e+03; 1,43e-08
2; 3; 1,70e+03; 9,93e-09

Abbildung T.4: Testreport des zweiten Tests

```
RLC Driver found
RLC Driver is ON !!!!!
YEAR -> May 1995
STATUS -> ON
VERSION -> V1.6/0000
DRIVER -> driver_rlcbridge
DEVICE -> PM6306
MAKER -> FLUKE
ESD Driver found
ESD Driver is ON !!!!!
YEAR -> 4980875
STATUS -> ON
VERSION -> 34
DRIVER -> driver ESD Generator
DEVICE -> SESD 30000
MAKER -> SCHLOEDER
PAR PORT Driver found
Driver PAR PORT is ON !!!!!
YEAR -> 0000
STATUS -> ON
VERSION -> 1.0
DRIVER -> Parallel Port
DEVICE -> PAR PORT
MAKER -> SYSTEM
ROBOTER CONTROL Driver found
Driver ROBOTER CONTROL is ON !!!!!
YEAR -> May 1995
STATUS -> ON
VERSION -> 1.0
DRIVER -> driver_robotctrl
DEVICE -> KUKA KR6/2
MAKER -> KUKA
WEATHER STATION Driver found
Driver WEATHER STATION is ON !!!!!
YEAR -> 1999
STATUS -> ON
VERSION -> 1.0
DRIVER -> Weatherstation
DEVICE -> Thero-Hydro-Sensor
MAKER -> HEAVY-WEATHER 433MHz
Test Station is OK !!!!
```

Abbildung T.5: Ausgabe nachdem Laden aller Treiber des Prüfprogramms