Institut für Softwaretechnologie

Abteilung Software Engineering

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 234

Entwicklung eines Eclipse-Plugin für die erweiterte Methode von STPA

Yannic Sowoidnich

Studiengang:	Softwaretechnik, B.Sc.
Prüfer:	Prof. Dr. rer. Nat. Stefan Wagner
Betreuer:	Asim Abdulkhaleq, M.Sc.
begonnen am:	18.05.2015
beendet am:	17.05.2015
CR-Klassifikation:	D.2.4

Zusammenfassung

Heutige Softwaresysteme werden immer größer und komplexer. Dieser Umstand macht es erforderlich eine zuverlässige Methode zu haben, diese Systeme auf Fehlverhalten oder auf gefährdende Situationen für Menschen zu untersuchen. Diese Arbeit versucht ein Werkzeug zu erstellen, welches zukünftige Unfall- und Sicherheitsanalysen dabei unterstützen kann, die untersuchten Systeme so sicher wie möglich zu gestalten. Die Software wird ein Plugin für die von der Universität entwickelte „XSTAMPP Plattform“, und versucht somit dessen Funktionalität zu erweitern. Dafür wird die erweiterte Methode von STPA als Funktionalität in das Plugin integriert (basierend auf der Arbeit von John Thomas), sowie einige Verbesserungen welche dabei helfen sollen, den Analyseprozess zu automatisieren (basierend auf der Arbeit von Asim Abdulhaleq). Dieses Dokument beschreibt aus welchen Gründen das neue Tool nützlich und erforderlich ist. Außerdem dokumentiert es den Entwicklungsprozess der neuen Software.

Abstract

Softwaresystems grow larger and larger. Over the years they got more complex and difficult to understand. Thats why it is more important than ever to have a reliable method to investigate these systems for any hazardous behaviour. There are some techniques out there to solve this problem, but this thesis tries to implement and improve a new procedure called „extended STPA“. The new Software will come as plugin for the „XSTAMPP Platform“ developed by the University of Stuttgart and tries to extend the functionality of this platform (the ideas for these extended functionalities come from John Thomas and Asim Abdulkhaleq). This document describes the new tool and all its functionalities, as well as why this tool is important to accident analysis.

Inhaltsverzeichnis

Danksagung

Hiermit möchte ich mich bei allen herzlich bedanken, die mich beim Schreiben meiner Bachelorarbeit unterstützt haben.

Danke Ann-Catrin, dafür, dass du Sonne in mein Leben bringst.

In besonderem Maße möchte ich mich bei Asim Abdulkhaleq bedanken, welcher für meine Betreuung zuständig war und an welchen ich mich stets mit meinen Fragen wenden konnte.

Des weiteren möchte ich mich bei Lukas Balzer bedanken, welcher mit seinem Wissen über „*XSTAMPP*“ stets behilflich war. Zu guter Letzt geht mein Dank an die Korrektoren dieses Dokuments.

Abbildungsverzeichnis

Abkürzungsverzeichnis

XSTAMPP	**“Extended STAMP Platform”**
STAMP	**“System-Theoretic Accident Models and Processes”**
CAST	**“Causal Analysis using System-Theory”**
STPA	**“Systems-Theoretic Process Analysis”**
XSTPA	**“Extended STPA”**
LTL	**“Lineare temporale Logik”**
RCP	**“Rich Client Platform”**
GUI	**“Graphical User Interface”**

Begriffsverzeichnis

“Extended STAMP Platform”	Das Programm für welches das Plugin geschrieben werden soll, es kombiniert verschiedene Tools in einer Software.
“System-Theoretic Accident Models and Processes”	Ein Ansatz aus der System-Theorie um komplexe Systeme zu modellieren.
“Causal Analysis using System-Theory”	Ein Ansatz aus der System-Theorie um eine kausale Abhängigkeiten zu bestimmen.
“Systems-Theoretic Process Analysis”	Eine Analysemethode welche auf “STAMP” aufbaut, ermittelt kritische Schwachpunkte innerhalb eines Systems.
“Extended STPA”	Eine verbesserte Version von STPA nach John Thomas und Asim Abdulkhaleq. Versucht mittels automatisierten Prozessen Schwachstellen zu finden.
“Lineare temporale Logik”	Lineare temporale Logik ist eine boolsche Darstellung von Abhängigkeiten.
„Safety Constraint“	Sicherheitseinschränkungen werden benötigt, um den Programmablauf besser kontrollieren zu können.

1 Einleitung

Dieses Kapitel bietet eine kurze Einführung in die Thematik der Arbeit. Es definiert die Frage- und Zielstellung der Bachelorarbeit um dem Leser ein grundlegendes Verständnis für die Herangehensweise, welche bei der Entwicklung des Plug-ins für „*XSTAMPP*“ verwendet wurde, zu geben.

1.1 Überblick

Komplexe Softwaresysteme sind in der heutigen Gesellschaft kaum noch wegzudenken. Sie befinden sich inzwischen überall, man findet sie im handelsüblichen Computer, in Fahrzeugen, smarten Wohneinrichtungen und auch in vielen anderen Sicherheitskritischen Systemen. Eben dieser Umstand, dass Software in vielen Fällen die Sicherheit von Personen (z.B. im Straßenverkehr) gewährleisten muss, macht es notwendig eine zuverlässige Gefahrenanalyse zu erstellen. Denn anders als bei Hardware, nutzt sich Software niemals ab, was zu dem Schluss führt, dass ein mögliches Fehlverhalten mit in die Software „eingebaut“ ist. Es ist also sehr wichtig, alle Gefahrensituationen genau zu definieren, sodass die Software in jeder Situation die richtige Entscheidung treffen kann. Um solche Situationen zu modellieren, wird ein Verfahren aus der Systemtheorie namens „*STAMP*“ (Systems-Theoretic Accident Modeling and Processes) verwendet. Mithilfe von „*STAMP*“ und diversen Analysemethoden („*STPA*“, „*CAST*“) ist es möglich, kritische Schwachpunkte in heutiger Software zu ermitteln. Diese Arbeit versucht die Architektur der von der Universität Stuttgart entwickelten „STAMP-Plattform“ („*XSTAMPP*“) zu analysieren und diese um die erweiterte Methode von „*STPA*“ (Systems-Theoretic Process Analysis) zu erweitern. Mit dieser Erweiterung wird es möglich sein, gefährliche Schwachpunkte in den zu modellierenden Systemen zu finden und entsprechende „Safety Constraints“ zu ermitteln, sodass gefährliche Situationen vermieden oder entsprechend reguliert werden können [4].

1.2 Motivation

Da die Gefahrenanalyse in der Softwareentwicklung eine äußerst wichtige Rolle bei der Entwicklung von sicherheitskritischen Systemen innehat, ist es äußerst sinnvoll eine automatisierte Analyseumgebung zu haben. Die sich noch in der Entwicklung befindende Rich-Client Anwendung „*XSTAMPP*“ ist ein mächtiges Werkzeug, welches solche automatisierten Abläufe und Werkzeuge zur Gefahrenanalyse bietet. „*XSTAMPP*“ soll mit den Funktionen der erweiterten Methode von „*STPA*“ ausgestattet werden, sodass künftig eine noch bessere Analyse der Schwachstellen in diversen Softwaresystemen möglich ist.

1.3 Zielstellung

Es steht die Entwicklung eines Plugins für die erweiterbare Plattform „*XSTAMPP*“ im Vordergrund. Das Plugin soll „*XSTAMPP*“ um folgende Eigenschaften erweitern:

Implementierung der erweiterten Methode von „*STPA*“, basierend auf der Arbeit von John Thomas, sodass die sogenannte Kontext-Tabelle und möglicherweise kritische Szenarien basierend auf den Prozessmodellen erzeugt werden können. Außerdem soll eine Basis geschaffen werden, die gefundenen sicherheitskritischen Kombinationen von Prozessvariablen mit den in „*XSTAMPP* gefundenen „Unsafe Control Actions“ zu verknüpfen.

Die davon abhängigen Datentabellen werden entwickelt und erstellt.

Implementierung einer booleschen Tabelle deren Einträge „And/Or“ Verknüpfungen für jede Prozessvariable enthalten werden (im folgenden „*LTL-Tabelle*“ genannt).

Entwickeln und implementieren einer optimalen Methode, um die Anzahl der möglichen Kombinationen zwischen den Prozessmodellen möglichst gering zu halten, sodass potenzielle Redundanz vermieden wird (Pairwise-Algorithm).

Der Export der Ergebnisse soll als PDF, CSV und PNG möglich sein, angelehnt an die unterliegende „*XSTAMPP*“-Plattform

Außerdem wird eine geeignete Dokumentation für das Plugin erstellt, sodass es künftigen Nutzern einfach fällt, das Tool zu nutzen.

1.4 Aufbau der Bachelorarbeit

Die Arbeit unterteilt sich in die Unterpunkte „Grundlagen“, „Analyse und Entwurf“ und „Implementierung“, gefolgt von den Kapiteln „Ergebnisse“, und „Zusammenfassung und Ausblick“.

- Im Kapitel „Grundlagen“ werden die verschiedenen Aspekte, auf welche „*XSTAMPP*“ aufbaut, vorgestellt, sodass dem Leser die Funktionalität sowie Sinn und Zweck des zu entwickelnden Plugins klar werden.

- Das Kapitel „Verwandte Arbeiten“ klärt darüber auf, wo genau diese Arbeit genau einzuordnen ist.

- Danach richtet sich der Fokus komplett auf den Entwicklungsprozess, beginnend mit der Analyse der Anforderungen, gefolgt vom Entwurf des Systems.

- Im Kapitel Implementierung wird eingehend die entstandene GUI beschrieben, sowie die Funktionalität des Plugins im einzelnen erläutert. Außerdem wird auf verwendete Algorithmen eingegangen, um den Nutzen mancher Funktionen besser zu erklären.

- Das Kapitel „Ergebnisse“ beschreibt das entstandene Produkt und versucht die Funktionalitäten des neuen Plugins zu erläutern.

- Das letzte Kapitel „Zusammenfassung und Ausblick“ bietet einige Ansätze inwiefern „*XSTPA*“ (Extended *STPA*) erweiterbar ist und gibt einen kurzen Überblick über den Verlauf des Projekts.

2 Grundlagen

In diesem Kapitel werden die wichtigsten elementaren Prozesse und Vorgehensweisen vorgestellt, welche in direkter oder indirekter Abhängigkeit zu „*XSTPA*“ stehen und somit alle die Basis für „*XSTAMPP*“ bilden.

2.1 STAMP

„*STAMP*“ steht für „Systems-Theoretic Accident Models and Processes“, es ist also ein Ansatz aus der Systemtheorie für die Modellierung von Prozessen. Eine wichtige Eigenschaft ist dass die Systemtheorie Systeme immer ganzheitlich betrachtet, nicht nur einzelne Komponenten des Systems. Die grundlegende Idee für ein durch Systemtheorie modelliertes System ist, dass es zum einen eine Hierarchie unter den Komponenten gibt und dass diese miteinander kommunizieren können. Außerdem unterliegen diese jeweils gewissen Regeln. Durch diesen Ansatz kann man auch komplexe Inhalte modellieren und betrachten [8].

Nun kann man sich überlegen wie Unfälle in einem solchen System entstehen können. Dies geschieht durch Kommunikation und Interaktion der Komponenten untereinander oder mittels ihrer Umwelt. Um ein Beispiel zu nennen: Ein Auto A soll einem anderen Auto B in gleicher Geschwindigkeit folgen. Nun bremst dass Auto B abrupt ab und die Sensoren von Auto A teilen diese Geschwindigkeitsveränderung von Auto B nicht rechtzeitig dem Geschwindigkeitsregler mit. Somit kann ein Unfall entstehen. Um solche Situationen vermeiden zu können, werden sogenannte „Safety Constraints“ erstellt, welche das Verhalten der einzelnen Komponenten regeln sollen. Dies könnte in obigem Beispiel ein Sicherheitsabstand zum anderen Auto sein. Sobald dieser überschritten wird, muss dass Auto A abbremsen bis es wieder genügend Abstand zu Auto B hat.

Man kann nun also sehen, dass die Systemtheorie (und damit „*STAMP*“) Sicherheit als ein Kontrollproblem betrachtet. Geschieht ein Unfall, so war eine oder mehrere Komponenten nicht gut genug „abgesichert“. Aus diesem Grund modelliert „*STAMP*“ solche Prozesse um Unfälle und Gefahrensituationen nachzuvollziehen und vorzubeugen. Um ein System zu beschreiben durchläuft „*STAMP*“ einige festgelegte Schritte die im folgenden erläutert werden:

- Definieren von „Accidents“ (Unfällen)
- Definieren von „Hazards“ (Gefahrensituationen)
- Definieren von „Safety Requirements“ (Sicherheitsanforderungen) und „Control Actions“ (Regelungsaktionen)

- Aufzeichnen / Modellieren einer passenden Kontrollstruktur
- Definieren von kausalen Abhängigkeiten.

Diese Schritte sind elementar in jedem durch „*STAMP*“ modellierten System. Der nächste Abschnitt behandelt „*STPA*“, welches verwendet wird diese nun modellierten Prozesse zu analysieren und mögliche Gefahrensituationen zu erkennen und abzuwenden [8], [3].

2.2 STPA

„Systems-Theoretic Process Analysis“ („*STPA*“) ist eine gängige Methode welche auf „*STAMP*“ aufbaut, um kritische Gefahren möglichst zuverlässig zu erkennen, sodass Systeme verifiziert werden können. Es analysiert alle gefundenen Hazards auf eventuelle kritische Situationen, damit ein möglichst sicheres und zuverlässiges System entstehen kann.

Wie oben schon genannt, benötigt dieses Verfahren die Vorarbeit welche in „*STAMP*“ verrichtet wird. Die oben beschriebenen Punkte müssen erfüllt sein sodass es möglich ist, mit dem „*STPA*“ Verfahren zu beginnen. „*STPA*“ selbst versucht sogenannte „Control Actions“ welche die Kommunikationsbasis der Systemkomponenten bilden, in verschiedene Kategorien aufzuteilen. Durch diese Kategorisierung wird deutlicher, durch welche Aktionen potentiell gefährliche Szenarien entstehen können. Die folgendenen Punkte beschreiben die möglichen Zuordnungen der „Control Actions“:

- Eine „Control Action“ wird nicht ausgeführt oder ist nicht verfügbar.
- Es wird eine unsichere (oder falsche) „Control Action“ ausgeführt.
- Es wird eine potentiell sichere Control Action zu einem falschen Zeitpunkt ausgeführt.
- Es wird eine potentiell sichere Control Action ausgeführt, aber zu früh abgebrochen.

Diese vier Punkte bilden die Basis für alle möglichen Gefahrensituationen. Nachdem alle „Control Actions“ eingeordnet wurden, muss die zuvor aufgezeichnete Kontrollstruktur um die Prozessmodelle der einzelnen Komponenten erweitert werden. Außerdem sollte überlegt werden, wo solche potentiell gefährlichen bzw. unsicheren „Control Actions“ ausgeführt werden können und wie ebendies verhindert werden kann. Das Verhindern dieser unsicheren Aktionen geschieht durch sogenannte „Safety Constraints“ - also Einschränkungen der Komponenten in gewissen Situationen, sodass es möglichst zu keinen Unfällen bzw. zu falschem Systemverhalten kommen kann [8] [1].

2.3 Extended approach to STPA (XSTPA)

Die erweiterte Methode von „*STPA*“ („*XSTPA*“) nach John Thomas und Asim Abdulkhaleq verbessert „*STPA*“ um einige Funktionen. Außerdem lässt es manche Prozesse vereinfachter ablaufen sodass es dem Sicherheitsanalysten erleichtert wird, das System zu analysieren und Gefahren zu erkennen.

Eine deutliche Verbesserung durch den Ansatz von John Thomas stellt der Kategorisierungsversuch der „Control Actions“ (Regelungsaktion) dar. In „*XSTPA*“ werden sogenannte „Context Tables“ aus den Prozessmodellen der jeweiligen Systemkomponenten abgeleitet, welche dazu dienen die Einstufung der Regelungsaktionen zu verbessern. Ein solcher ist zur Veranschaulichung in Abbildung 2.3.1 grafisch dargestellt [4].

Control Action	Train Motion	Emergency[10]	Train Position	Hazardous control action?		
				If provided any time in this context	If provided too early in this context	If provided too late in this context
Door open command provided	Train is moving	No emergency	(doesn't matter)	Yes	Yes	Yes
Door open command provided	Train is moving	Emergency exists	(doesn't matter)	Yes[11]	Yes	Yes
Door open command provided	Train is stopped	Emergency exists	(doesn't matter)	No	No	Yes
Door open command provided	Train is stopped	No emergency	Not aligned with platform	Yes	Yes	Yes
Door open command provided	Train is stopped	No emergency	Aligned with platform	No	No	No

Abbildung 2.3.1: Context Table - Train Door Controller

Wie man in Abbildung 2.3.1 erkennen kann, ist die Kontext-Tabelle eine Kombination der zu analysierenden „Control Action“ und des Prozessmodells einer Komponente. Der Name der Tabelle ist darauf zurückzuführen, dass jede Wertekombination der unterschiedlichen Prozessvariablen einen gewissen Kontext für die Regelungsaktion darstellt. Denn im Kontext der ersten Zeile (Abbildung 2.3.1) ist der „Door open command“ gefährdend, wohingegen er im Kontext der letzten Zeile als nicht gefährlich eingestuft wird.

Der Vorteil dieser Darstellung ist, dass dem Sicherheitsanalysten kein potentiell gefährliches Szenario entgeht und somit eine möglichst hohe Abdeckung der Gefahrensituationen erreicht wird.

Jedoch ist zu beachten, dass die entstehende Kontexttabelle bei einer großen Anzahl an Variablen auf eine enorme Größe wachsen kann. Das Problematische an dieser Tabelle ist eindeutig, dass sie für einen menschlichen Bearbeiter deutlich zu komplex wäre. Für jede Tabellenzeile müsste sich der Sicherheitsexperte Gedanken darüber machen, ob das aktuell betrachtete Szenario ein sicherheitskritisches wäre oder nicht.

Aus diesem Grund wurde durch Asim Abdulkhaleq für „*XSTPA*“ ein Ansatz entwickelt, welcher es ermöglicht, die Tabellengröße durch Methoden der Kombinatorik deutlich zu reduzieren. Dies ist möglich, da eine Kontexttabelle mit allen denkbaren Einträgen sehr redundant ist (vgl. Abbildung 2.3.1, Diese Tabelle ist nicht mehr redundant, würde man aber in Zeile 1 statt „doesnt' matter“ wieder die Originalwerte für diese Variable verwenden, so wären alle neu generierten Zeilen redundant zueinander). Durch diese Methoden kann eine Tabelle mit ca. 1000 möglichen Einträgen auf eine Größe von ~ 40 Einträgen reduziert werden. Diese Funktionalität wird etwas näher im Kapitel 5.2 erläutert.

Ein weiterer Vorteil, den diese Darstellung mit sich bringt, ist dass die jeweiligen Zeilen leicht in boolesche Formeln umgewandelt werden können, welches das weitere Bearbeiten der potentiell gefährlichen Szenarien erleichtert, in Hinsicht auf zu erstellende „Safety Constraints“. Ein weiterer Vorteil entsteht dadurch, dass dieser „Context Table“ automatisiert erstellt werden kann, sodass sich der Sicherheitsexperte nur noch Gedanken darüber machen muss, ob ein gewisses Szenario sicherheitskritisch ist oder nicht.

2.4 XSTAMPP

„Extended STAMP Platform“ („*XSTAMPP*“) bildet die Schnittstelle für welche das Plugin „*XSTPA*“ entwickelt werden soll. Das Tool „*XSTAMPP*“ selbst ging aus dem von der Universität Stuttgart entwickelten Programm „*A-STPA*“ hervor, welches viele Basisfunktionen für das jetzige „*XSTAMPP*“ geliefert hat. „*XSTAMPP*“ vereinigt durch das dafür sehr gut geeignete „Eclipse Rich Client Platform“ (RCP) Framework viele Tools in einem. Mit enthalten sind die Basisfunktionen von „*XSTAMPP*“, sowie in Zukunft die Plugins „*XSTPA*“ und „*A-CAST*“. Mittels all dieser Funktionalität entsteht ein Tool welches weit in die verschiedenen Bereiche der „*STAMP*“ – Analysemethoden hineinreicht. Das umfasst im Allgemeinen die im Kapitel 2 beschriebenen Analysemethoden sowie die „*STAMP*“ Modellierung.

Das Tool ist sehr benutzerfreundlich gestaltet. Es bietet eine flexible Gestaltung des Layouts, da sich alle geöffneten Tabs beliebig verschieben lassen, sowie, dass sie in der Größe genau auf den jeweiligen Benutzer angepasst werden können. Außerdem bietet „*XSTAMPP*“ eine Navigation welche selbst unerfahrene Benutzer sicher durch die in Kapitel 2.1 genannten Schritte leitet. Wie in Abbildung 2.4.1 zu sehen, ist die grafische Benutzeroberfläche sehr schlicht gehalten, sodass unerfahrene Benutzer nicht verwirrt werden. Hält sich der Benutzer an die Navigation, so durchläuft er alle notwendigen Schritte zum Modellieren des Systems, beginnend mit dem Definieren von „Accidents“ und „Hazards“, gefolgt vom Definieren der „Safety Constraints“ über das Zeichnen der Kontrollstruktur bis hin zum Definieren der kausalen Abhängigkeiten. Es ist leicht zu erkennen, dass „*XSTAMPP*“ weit über die elementaren Schritte eines „*STAMP*“ Modellierungsprozesses hinausgeht.

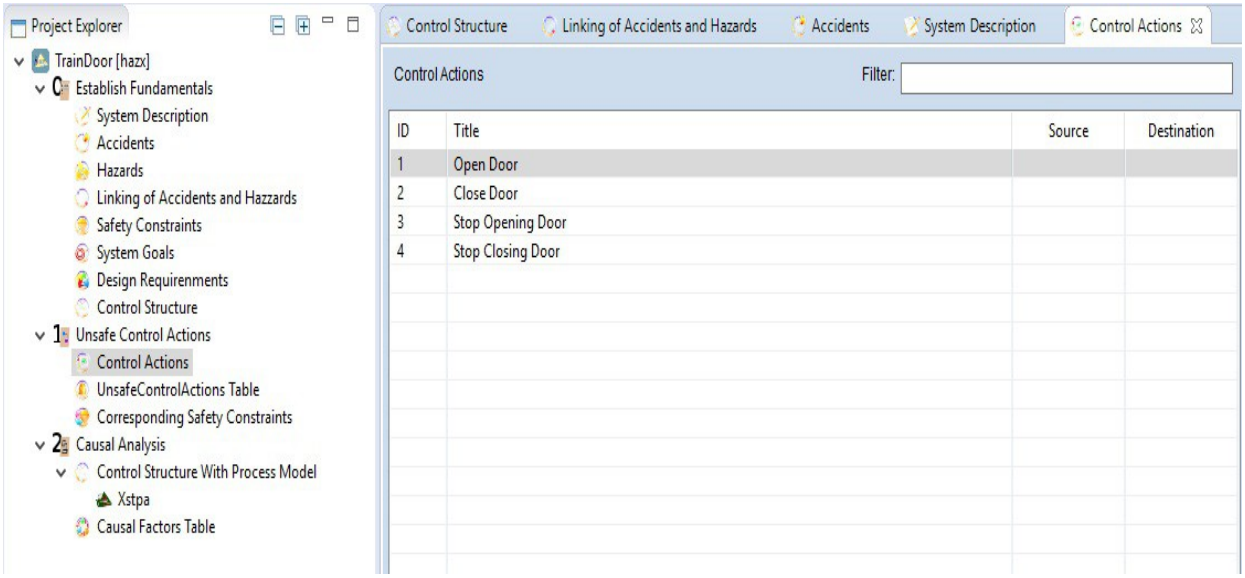

Abbildung 2.4.1: Hier ist die “XSTAMPP” Navigation und die "Control Actions" des aktuell bearbeiteten Systems zu sehen.

2.5 Eclipse Plug-in Development (RCP)

Da die „*XSTAMPP*“ Plattform auf dem „Eclipse Rich Client Platform Framework“ aufbaut, wird das Plugin über dieses in die Software integriert.

Um „*RCP*“ zu verstehen, ist es notwendig bestimmte Begriffe genauer zu definieren. Wenn von sogenannten „Rich Clients“ gesprochen wird, dann ist damit im Allgemeinen eine Anwendung gemeint, welche sowohl eine (interaktive) grafische Benutzeroberfläche bietet, als auch die Anwendungslogik des Tools zur Verfügung stellt. Die „Rich Client Platform“ wird aus allgemeineren bzw. unspezifischeren Funktionen eines Tools gebildet. Man kann sich also vorstellen, dass das Plugin die Basisfunktionalitäten einer Plattform um eine gewisse Spezialisierung erweitert.

Der Vorteil dieser Vorgehensweise kann leicht verständlich in einem Realwelt-Beispiel beschrieben werden. Man kann sich die „Rich Client Platform“ als ein Auto vorstellen, welches außer der Standardausstattung keinerlei Extras besitzt. Jedoch kann man unterschiedliche Zusätze (Musikanlage, Navigationssystem etc.) in das Auto einbauen lassen, um den individuellen Wünschen des Kunden zu entsprechen. Genauso verhält es sich mit RCP-Anwendungen, sie können je nach Bedarf um die gewünschte Funktionalität erweitert werden. Die Verwaltung der Plugins geschieht immer zentralisiert über die Plattform selbst – im Bezug auf die in diesem Dokument beschriebene Arbeit übernimmt „*XSTAMPP*“ also die Aufgabe alle Plugins zu steuern. Dies beinhaltet vor allem die Bereitstellung von „Views“, „Editoren“ und eines Hilfesystems. Des weiteren wird „XSTPA“ auch auf das in „XSTAMPP“ enthaltene Datenmodell zurückgreifen, um anderen Plugins und der Plattform selbst die Möglichkeit zu geben mit diesen Daten zu arbeiten [9].

Die Integration des Plugins geschieht über sogenannte „Extension Points“ in der „Rich Client Platform“ (wobei auch Plugins solche Erweiterungspunkte definieren können) - dies sind vordefinierte Stellen in welche Plugins neue Funktionalitäten einbinden und so mit der „Hauptplattform“ interagieren können. Die „Eclipse IDE“ bietet dem Entwickler außerdem eine leicht zu benutzende „*GUI*“, welche die eigentliche Einbindung von „*XSTPA*“ in die XSTAMPP-Plattform auf einige wenige Mausklicks reduziert [10].

Wegen der genannten Punkte wird es möglich, dem Endanwender ein Produkt zu liefern, welches immer auf dessen Bedürfnisse zugeschnitten ist.

3 Verwandte Arbeiten

In diesem Kapitel soll geklärt werden, wie diese Arbeit einzuordnen ist. Dies geschieht indem Ansätze verwandter Arbeiten aufgezeigt werden, um so ein Verständnis dafür zu schaffen, welche Lücken mit der Entwicklung von „*XSTPA*“ geschlossen werden sollen.

„Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis“ lautet der Titel der Arbeit von John Thomas [4]. Diese Arbeit legt den Grundstein für die Entwicklung des Plugins für „*XSTAMPP*“. Er erarbeitete in seiner These eine formale, mathematische Struktur für das „*STPA*“ Analyseverfahren, welches bis dahin eher einer losen Vorgehensweise unterlag (In Bezug auf festgelegte Prozesse). Diese mathematische Struktur bringt eine gewisse Systematik in „*STPA*“. Durch diese Erweiterung ist John Thomas in der Lage den Analyseprozess durch computergestützte Verfahren zu verbessern, um so große komplexe Systeme besser auf Gefahren analysieren zu können. In seiner Arbeit zeigt er die Vor- und Nachteile verschiedenster Modellierungs- und Analysemethoden auf und ermittelt weshalb „*STPA*“ für heutige Softwaresysteme eines der bestgeeigneten Verfahren ist [4]. Dennoch stellt er fest dass seine Arbeit erweiterbar ist. Sein Ansatz weist für die sicherheitskritischen Szenarien der unterschiedlichen „Control Actions“ oftmals eine hohe Redundanz auf, da sie oftmals dieselben Szenarien enthalten können. Des weiteren stellt er fest, dass das ermittelte Analyseverfahren schwer auf dem zuvor erstellten Modell basiert, was in dieser Hinsicht zu Fehlern führen kann [4].

Diese Arbeit versucht diese theoretischen Erweiterungen von John Thomas in ein Werkzeug zu integrieren, welches zukünftige Unfallanalysen automatisierter und systematischer ablaufen lässt. Des weiteren wird versucht, auf die genannten Verbesserungsvorschläge einzugehen und so die Arbeit von John Thomas nicht nur praktisch umzusetzen, sondern diese zu erweitern, sodass eine Software entsteht, welche zukünftigen Sicherheitsanalysten eine gute Arbeit leisten wird.

4 Analyse und Entwurf

Dieses Kapitel beschäftigt sich mit der Analyse von „*XSTPA*“. Es wurden mehrere Diagramme erstellt um alle Anforderungen so genau wie möglich zu beschreiben.

4.1 User Stories

Im diesem Abschnitt finden sich die Anforderungen in der folgenden Liste in Form von User Stories wieder. Wenn im folgenden von Anwendung gesprochen wird, dann ist dass in „*XSTAMPP*“ integrierte Plugin gemeint, nicht „*XSTAMPP*“ selbst.

Anwendung starten:

- *Als Anwender möchte ich, dass das Plugin sich in einem separaten Tab an der Unterseite des Hauptfensters öffnet.*
- *Als Anwender möchte ich, dass alle abgespeicherten Daten sofort geladen werden, um direkt weiterarbeiten zu können .*
- *Als Anwender möchte ich alle verfügbaren Daten sofort in das Plugin laden, sodass ich jederzeit mit ihnen arbeiten kann.*

Navigation:

- *Als Anwender möchte ich eine Navigation innerhalb des Plugins, welche platzsparend und leicht verständlich ist.*

Prozessmodell:

- *Als Anwender möchte ich die Prozessmodelle, welche in der Kontrollstruktur vorhanden sind, im Plugin in tabellarischer Form dargestellt haben.*
- *Als Anwender möchte ich jedem Wert einer Prozessvariable einen Beschreibung hinzufügen können.*

Control Actions:

- *Als Anwender möchte ich alle „Control Actions“, welche in der Kontrollstruktur vorhanden sind, im Plugin in tabellarischer Form dargestellt haben.*
- *Als Anwender möchte ich jede „Control Action“ als sicherheitskritisch einstufen können, sodass ich die „unkritischen“ nicht betrachten muss.*

- *Als Anwender möchte ich jeder „Control Action“ eine Beschreibung hinzufügen können*

Abhängigkeiten:

- *Als Anwender möchte ich jeder „Control Action“ beliebige Prozessvariablen zuordnen können.*
- *Als Anwender möchte ich jeder Control Action“ einen Kontext zuordnen können (Verfügbar und nicht Verfügbar).*

Kontext-Tabelle:

- *Als Anwender möchte ich verschiedene Optionen haben, die Einträge generieren zu lassen (Combinatorial Testing, Constraints).*
- *Als Anwender möchte ich automatisch die größtmögliche Tabelle generiert bekommen, sodass ich entscheiden kann, welche der Einträge überflüssig sind.*
- *Als Anwender möchte ich die Kontext-Tabelle mit verschiedenen Filtern anzeigen lassen (Alle Einträge, „Only Hazardous“ und „Not Hazardous“).*
- *Als Anwender möchte ich den Kontext der Tabelle einstellen können (Verfügbar und nicht Verfügbar).*
- *Als Anwender möchte ich jede einzelne Kombination der Kontext-Tabelle als gefährlich („Hazardous“) einstufen können.*
- *Als Anwender möchte ich die Tabelle auf „Fehler“ verifizieren lassen (Wenn derselbe Eintrag in „Verfügbar“ und in „nicht Verfügbar“ vorhanden ist).*
- *Als Anwender möchte ich einzelne Einträge manuell löschen können.*
- *Als Anwender möchte ich einzelne Einträge manuell erstellen können.*

Verbesserte Sicherheitsanforderungen (Refined Safety Requirements)

- *Ich möchte alle als „Hazardous“ eingestuften Einträge (Kontextunabhängig) in dieser Ansicht aufgelistet haben.*
- *Ich möchte allen Einträgen einen Kommentar hinzufügen können.*
- *Ich möchte alle Einträge mit den in „XSTAMPP“ erstellten Hazards verknüpfen können.*
- *Ich möchte alle Einträge mit den in „XSTAMPP“ erstellten „Unsafe control actions“ verknüpfen können.*
- *Ich möchte alle Einträge in eine alternative Darstellung umwandeln können (Lineare Temporale Logik).*

Allgemein:

- *Ich möchte ein Menü für mögliche Einstellungen des Plugins.*
- *Ich möchte alle Informationen in „XSTPA“ exportieren können (PDF, PNG, CSV).*

4.2 Use Case Diagramm

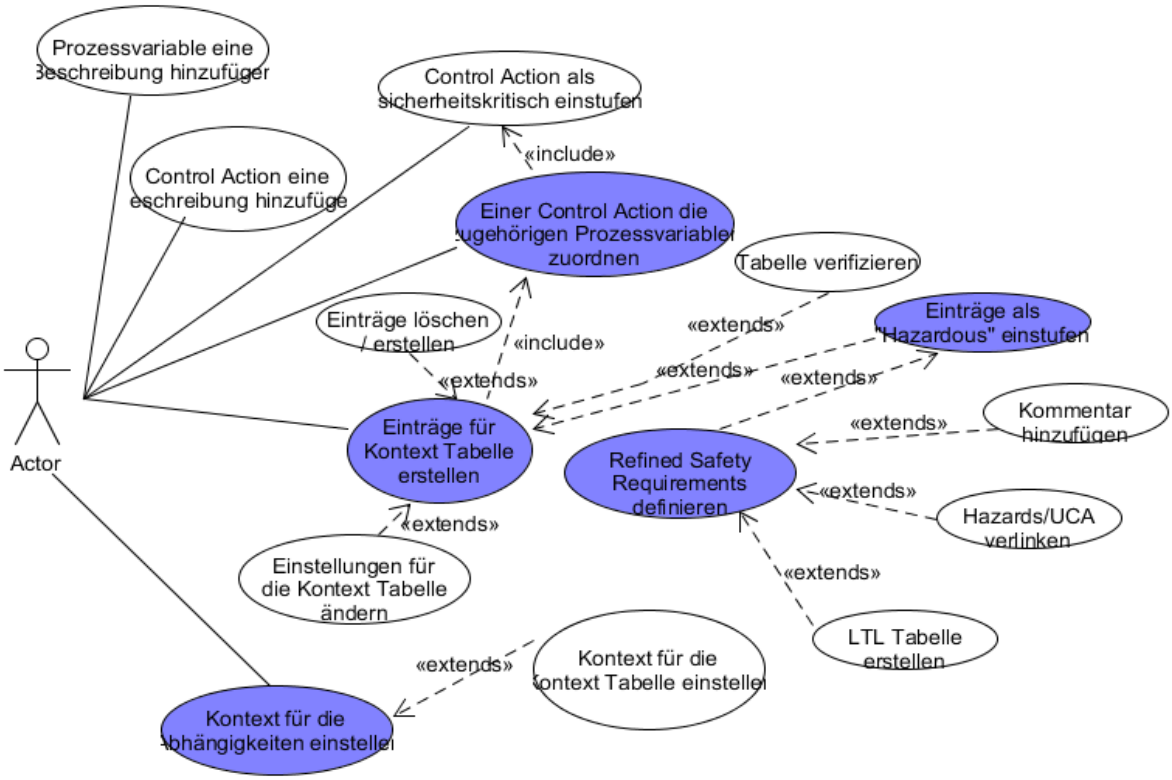

Abbildung 4.2.1: Use Case Diagramm von " XSTPA"

Abbildung 4.2.1 beschreibt einen Teil der in Kapitel 4.1 aufgezählten Punkte in grafischer Darstellung. Aus Gründen der Übersichtlichkeit wurden sehr triviale Punkte (z.B. in die entsprechende Ansicht navigieren) ausgespart.

4.3 Klassendiagramm

Abbildung 4.3.1 soll kein vollständiges Klassendiagramm in Hinsicht auf die verwendeten Methoden und Variablen darstellen. Es stellt vielmehr die wichtigsten Beziehungen und die allgemeine Struktur der vier wichtigsten Klassen dar. Der Aufbau ist an das „MVC Pattern“ angelehnt, jedoch besitzt das Plugin „*XSTPA*“ selbst kein eigenständiges Model. Es bekommt die Daten von der unterliegenden Plattform „*XSTAMPP*“ gestellt. Die Klasse „View“ übernimmt auch zu großen Teilen die Funktionalität eines „Controllers“. Diese Entscheidung wurde aus dem Grund getroffen, das dass Plugin kein eigenes Model besitzt und da es durch seine „geringe“ Größe überdimensioniert gewesen wäre, einen eigenen „Controller“ zu implementieren.

Da es in „*XSTPA*“ hauptsächlich um das Weiterverarbeiten von Informationen, welche aus dem Prozessmodell stammen, geht, wurden die Beziehungen der „View“ und der zuständigen Klassen für ebendiese Informationen dargestellt. Die „View“ bezieht alle wichtigen Daten aus dem DataModel von „*XSTAMPP*“, diese werden in eine eigene Struktur überführt. Diese Struktur ist hierarchisch geprägt, die Klasse „Controller“ enthält alle im untergeordneten „ControlActions“, welche wiederum alle die jeweils verknüpften „Prozessvariablen“ enthalten. Außerdem enthält die Klasse „ControlActions“ auch die Einträge der zugehörigen Kontext-Tabelle.

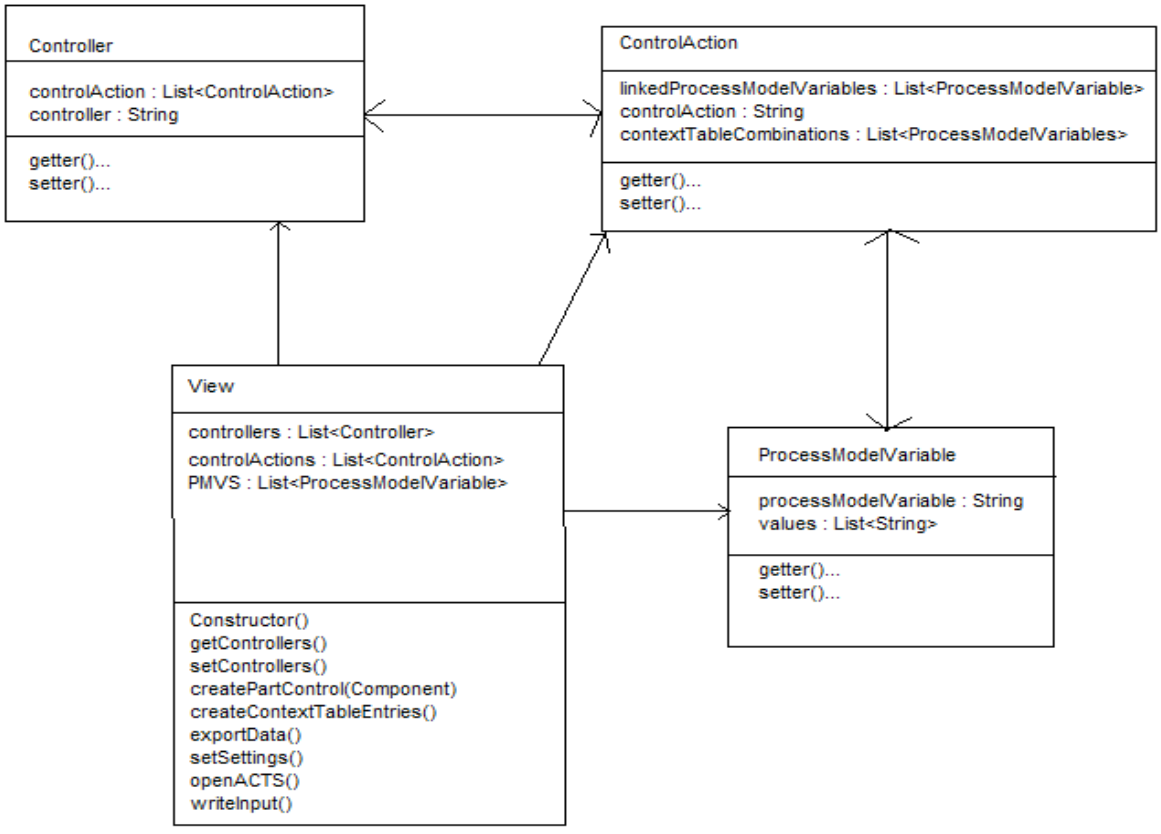

Abbildung 4.3.1: Beziehungen der View mit der allgemeinen Informationsstruktur

4.4 Beispiel (Train Door)

In diesem Abschnitt sollen die Punkte der verschiedenen Unterkapitel des Kapitels 4 in einem kleinen Resümee als Beispiel zusammengefasst werden, sodass der allgemeine „Workflow“ des Programms zum Ausdruck gebracht wird.

Es wird davon ausgegangen, dass die in Kapitel 2.1 genannten Schritte mittels der „*XSTAMPP*“ Plattform bereits durchgeführt wurden und man den in Abbildung 4.4.1 dargestellten „Train Door Controller“ mit dem „*XSTPA*“ Plugin auf gefährliche „Control Actions“ analysieren möchte. Beispielhaft werden die folgenden „Control Actions“ untersucht:

- „Open Door“
- „Close Door“
- Stop opening Door“
- Stop closing Door“

Der Sicherheitsanalyst wird nun eine Kontext-Tabelle erstellen, mit allen möglichen Wertekombinationen der abgebildeten Prozessvariablen („Emergency“, „Door_State“, „Door_Position“, „Train_Motion“ und „Train Position“), sowie der zu untersuchenden „Control Action“. Ein Auszug einer möglichen Kontext-Tabelle für die „Close Door - Control Action“ ist in Abbildung 4.4.2 zu sehen. Nun muss der Anwender sich überlegen, welche der dargestellten Kombinationen als sicherheitskritisch einzustufen sind und welche nicht.

Im dargestellten Schaubild 4.4.2 wäre dass nur die Zeile mit der ID 7, da grundsätzlich alle Zeilen wegfallen, in dem „Train_Motion“ = „Moving“ ist, da die Tür immer sofort geschlossen werden sollte, wenn der Zug sich bewegt. Zeile 13 und 14 fallen weg da keinerlei Gefahr besteht wenn der Zug im Bahnhof steht, sich nicht bewegt und sich keine Person in der Tür befindet. Somit bleiben nur noch die Zeilen 5,6,7 und 8. Die Fahrgäste sollen den Zug nicht verlassen, wenn der Zug nicht im Bahnhof hält und kein Notfall existiert. Aus diesem Grund sollte die Tür also immer schließen, wenn sich keine Fahrgäste in der Tür befinden und kein Notfall vorliegt. Es kann nur dann zu einer Gefahrensituation kommen, wenn sich eine Person in der Tür befindet während diese schließt.

Nachdem die Tabelle erstellt wurde, kann der Sicherheitsanalyst sich daran machen, alle als „gefährlich“ eingestuften Kombinationen zu bearbeiten und entsprechende Sicherheitseinschränkungen für diese Fälle zu definieren. Man erkennt schnell, wie hilfreich die „*STPA*“ Methode ist um mögliche Gefahrensituationen zu erkennen.

Die Tabelle kann zwar sehr schnell sehr groß werden (Für die Summe aller möglichen Wertekombinationen wird das Produkt der Anzahl der jeweiligen Werte gebildet, in diesem Beispiel also

$$2*2*2*2*3=48$$

), jedoch fällt es so deutlich leichter, alle kritischen Szenarien zu erkennen. Durch „*XSTPA*“ kann mittels kombinatorischem Testen die Tabellengröße stark reduziert und gesteuert werden.

Dieser komplette Vorgang muss natürlich für alle „Control Actions“ durchgeführt werden, um eine volle Abdeckung des zu überprüfenden Systems zu erreichen. Außerdem muss auch überprüft werden, ob es zu einer kritischen Situation führen kann, wenn eine dieser Kontrollanweisungen nicht bereit gestellt wird („Control Action Not Provided“). Aus der Kontext-Tabelle können nun die Sicherheitsanforderungen an die jeweilige Komponente des Systems abgeleitet und mit den durch den „*STAMP*“ Prozess definierten „Hazards“ verknüpft werden. In „*XSTPA*“ geschieht dies durch die „Refined Safety Requirements – Tabelle“. Aus dieser kann außerdem die alternative Darstellungsform der sicherheitskritischen Kombinationen abgeleitet werden („*LTL*“) um eine bessere Weiterverarbeitung zu gewährleisten. Damit die gefundenen Szenarien auch außerhalb von „*XSTPA*“ zur Verfügung stehen, können diese als „PDF“, „CSV“ oder „PNG“ exportiert werden.

Es stehen dem Sicherheitsanalysten bei der Erstellung der Kontext-Tabelle einige Hilfsmittel zur Verfügung. Das Plugin bietet einige Tools und Funktionen, welche das Ermitteln von gefährlichen Wertekombinationen erleichtert. Es können zum einen einzelne Szenarien manuell hinzugefügt werden und zum anderen können „überflüssige“ bzw. unkritische Kombinationen gelöscht werden. Wenn der Anwender eine Zeile findet, welche er unter Umständen als sicherheitskritisch einstufen würde, so steht im die Option offen, einzelne Einträge in dieser Zeile als „unwichtig“ zu markieren (In das entsprechende Feld wird der Wert „Don't Care“ gesetzt), sodass nur die Parameter welche schlussendlich zu einer Gefahr führen im Fokus liegen. Außerdem besteht für den Analysten die Möglichkeit, alle Einträge auf ihre Logik zu prüfen, sodass keine redundanten oder sich logisch widersprechende Szenarien auftreten. Der komplette Arbeitsablauf wird vereinfacht in Abbildung 4.4.3 dargestellt.

Dieses Vorgehen stellt den allgemeinen Arbeitsablauf einer solchen Analyse dar. Die genaue Funktionalität der implementierten Funktionen wird nochmals näher in Kapitel 5.1 beschrieben. Das nächste Kapitel zeigt den ersten GUI-Entwurf, der zu Beginn der Entwicklung entstanden ist.

Control Action Provided | Control Action Not Provided

ID	Train_Position	Train_Motion	Door_Position	Door_State	Emergency	Hazardous if provided Anytime	to Early	to late
1	Not_Aligned_With_Platform	Moving	Fully_Open	Person_not_in_Door	Evacuate!	☐	☐	☐
2	Not_Aligned_With_Platform	Moving	Fully_Open	Person_not_in_Door	NoEmergency	☐	☐	☐
3	Not_Aligned_With_Platform	Moving	Fully_Open	Person_in_Door	Evacuate!	☐	☐	☐
4	Not_Aligned_With_Platform	Moving	Fully_Open	Person_in_Door	NoEmergency	☐	☐	☐
5	Not_Aligned_With_Platform	Stopped	Fully_Open	Person_not_in_Door	Evacuate!	☐	☐	☐
6	Not_Aligned_With_Platform	Stopped	Fully_Open	Person_not_in_Door	NoEmergency	☐	☐	☐
7	Not_Aligned_With_Platform	Stopped	Fully_Open	Person_in_Door	Evacuate!	☐	☐	☐
8	Not_Aligned_With_Platform	Stopped	Fully_Open	Person_in_Door	NoEmergency	☐	☐	☐
9	Aligned_With_Platform	Moving	Fully_Open	Person_not_in_Door	Evacuate!	☐	☐	☐
10	Aligned_With_Platform	Moving	Fully_Open	Person_not_in_Door	NoEmergency	☐	☐	☐
11	Aligned_With_Platform	Moving	Fully_Open	Person_in_Door	Evacuate!	☐	☐	☐
12	Aligned_With_Platform	Moving	Fully_Open	Person_in_Door	NoEmergency	☐	☐	☐
13	Aligned_With_Platform	Stopped	Fully_Open	Person_not_in_Door	Evacuate!	☐	☐	☐
14	Aligned_With_Platform	Stopped	Fully_Open	Person_not_in_Door	NoEmergency	☐	☐	☐

Abbildung 4.4.1: Auszug einer Kontext-Tabelle für die "Close Door - Control Action"

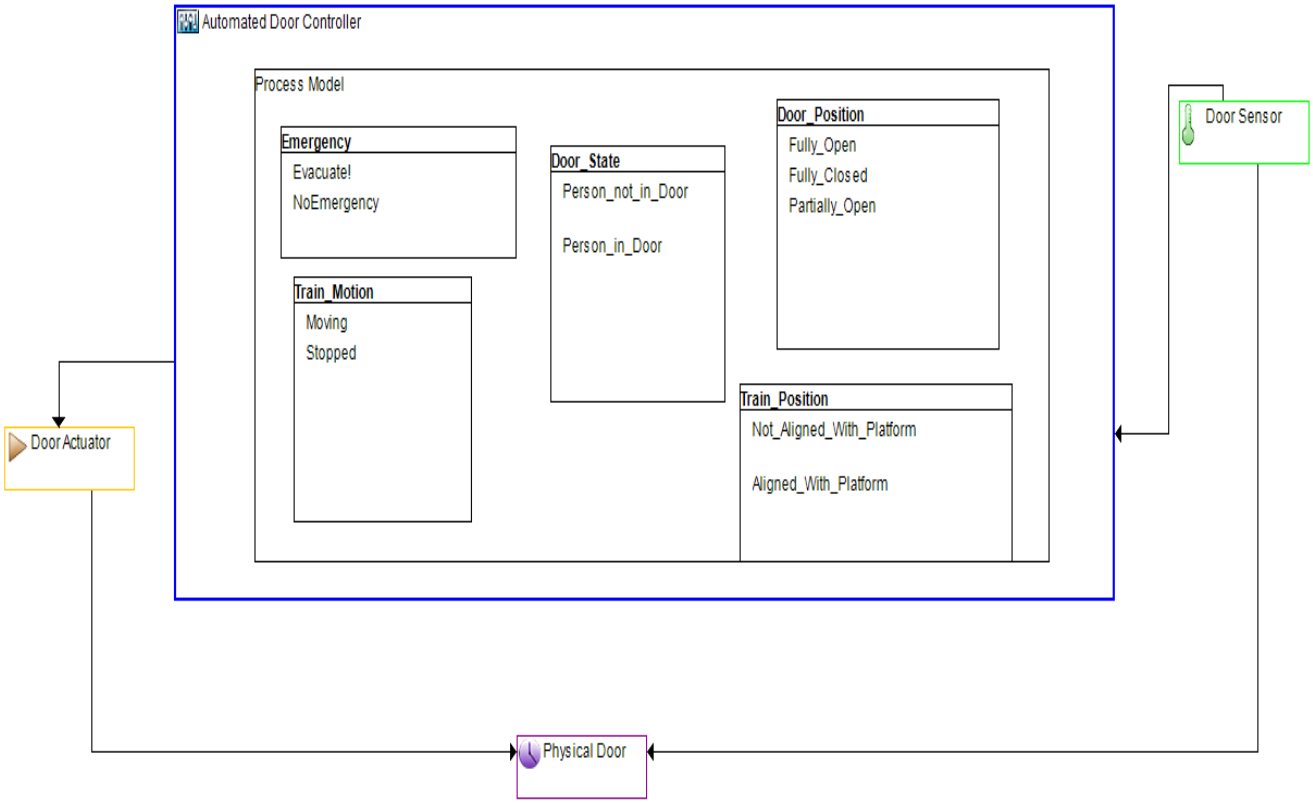

Abbildung 4.4.2: Kontrollstruktur mit Prozessmodell eines "Automated door controllers"

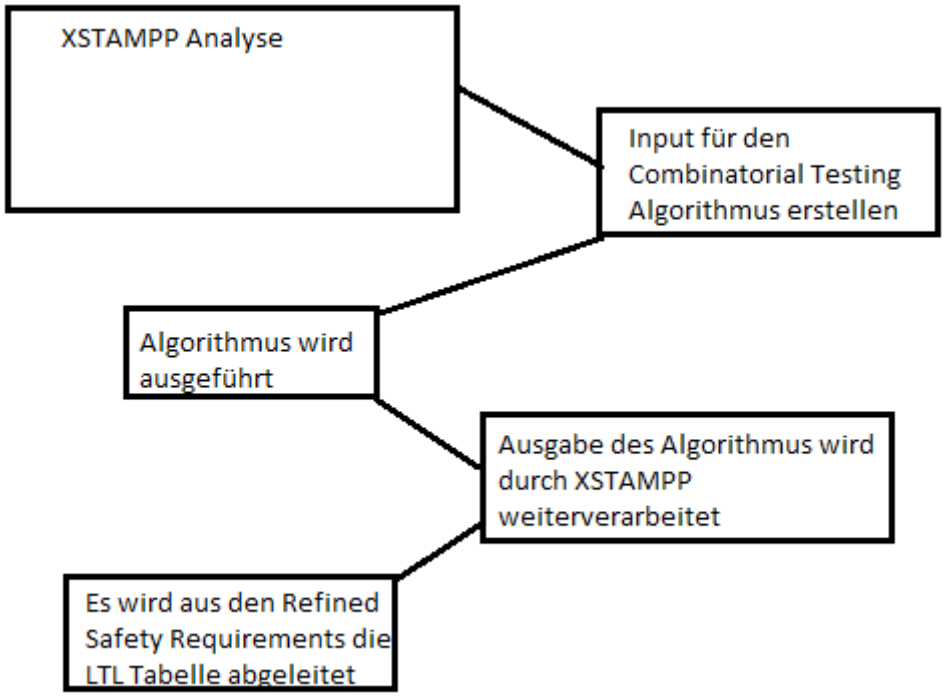

Abbildung 4.4.3: Allgemeine Struktur von XSTPA

4.5 Prototyp der GUI

Dieses Kapitel beschreibt, wie die grafische Benutzeroberfläche entstanden ist und welche Entscheidungen maßgeblich für die Entwicklung ebendieser waren.

Die erste Einschränkung für das Plugin war, dass es nur sehr wenig Platz zur Verfügung haben sollte, es sollte über der „Kontrollstruktur mit Prozessmodell“ von „*XSTAMPP*“ liegen, sodass diese während dem gesamten Analyseprozess noch erkennbar war. Dies machte es zwingend eine sehr platzsparende Navigation zu haben, sowie relativ kleine Fenster für die restliche Funktionalität von „*XSTPA*“. Diese Einschränkung, sowie der oben beschriebene „Workflow“ führte zu einem Prototyp wie er in Abbildung 4.5.1 dargestellt ist.

In der Grafik ist die Startansicht des Plugins skizziert. Die Navigation wurde vertikal entworfen, da dies die beiden Vorteile mit sich brachte, dass es zum einen wenig Platz beanspruchte und zum anderen war klar, dass es innerhalb des Programms in den einzelnen Ansichten weitere Navigationselemente geben würde, so war sichergestellt dass diese sich nicht im Weg sein würden.

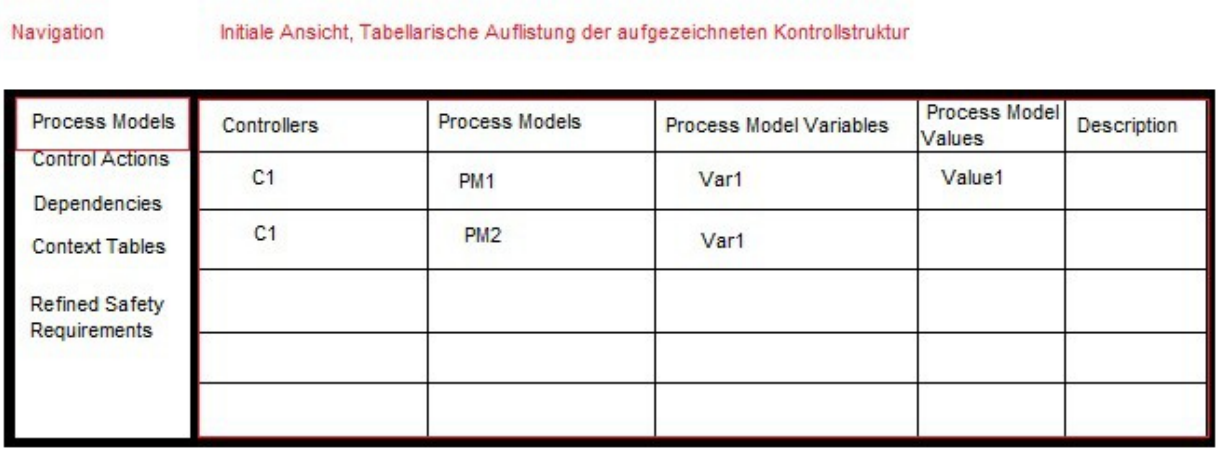

Abbildung 4.5.1: Erster GUI Prototyp von "XSTPA" - Mit Navigation und Initialer Ansicht

Nach dem der generelle Aufbau des Plugins festgelegt war, musste jeder Arbeitsschritt des in Kapitel 4.4 genannten Ablaufs in eine Ansicht transferiert werden. Da „*XSTAMPP*“ viele Controller und Prozessmodelle haben kann und um die intuitive Bedienbarkeit des Plugins zu erhöhen, war es ein naheliegender Schritt nochmals alle Controller mit ihren zugehörigen Prozessmodellen in tabellarischer Form darzustellen und ihnen einen Kommentar, bzw. eine Beschreibung mitzugeben.

Die nächste Ansicht sollte den „Control Actions“ gewidmet sein, da man aus diesen mit den zugehörigen Prozessvariablen den „Context Table“ erstellen muss. Dies ist in der Grafik 4.5.2 zu erkennen. Die dargestellte Tabelle zeigt die Beziehung von den „Control Actions“ mit den zugehörigen Controllern. Man sieht dass in der skizzierten Tabelle manche der Kontrollanweisungen als „Safety Critical“ markiert sind. Dies ist die erste Entscheidung, die der Sicherheitsexperte in „*XSTPA*“ treffen muss. Markiert er eine dieser „Control Actions“ als sicherheitskritisch, so wird sie in den weiteren Ansichten aufgeführt. Ist sie als „nicht kritisch“ eingestuft, so kann mit ihr nicht weitergearbeitet werden. Zusätzlich kann jede „Control Action“ nochmals eine Beschreibung erhalten.

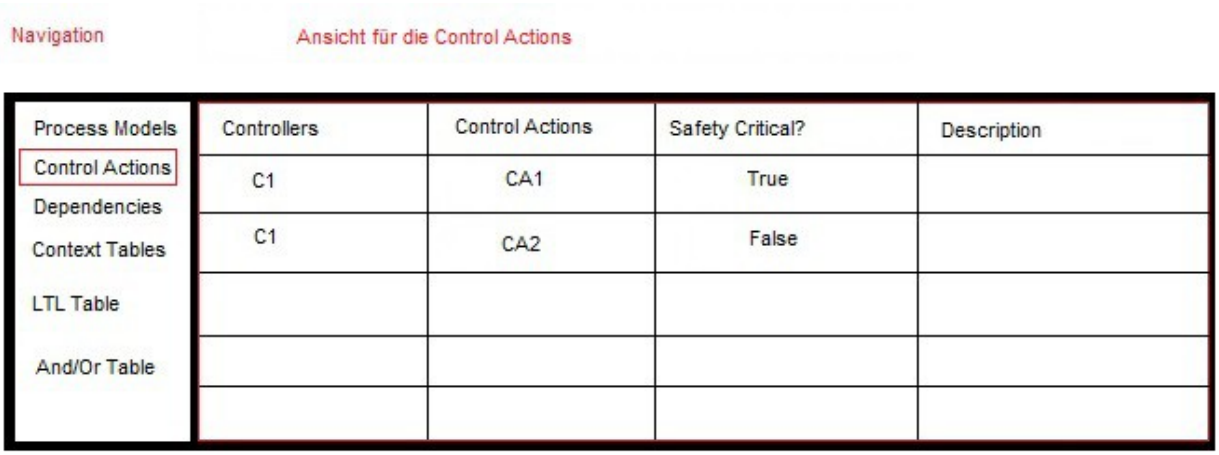

Abbildung 4.5.2: Skizze der Control Actions

Im nächsten Arbeitsschritt wird festgelegt, welche Prozessvariablen welche Kontrollanweisung beeinflussen und ob diese voneinander abhängig sind oder nicht. Das geschieht im Unterpunkt „Dependencies“. Da es in „*XSTAMPP*“ schon einen Punkt „Linking of Accidents and Hazards“ gibt, wurde das Design stark an dieses angepasst, da dieses zum einen sehr intuitiv ist und zum anderen um dem Endanwender einen hohen Wiedererkennungswert zu bieten. Die Kontrollanweisungen können angewählt werden und es ist möglich ihnen über die Buttons die entsprechenden Prozessvariablen zuzuweisen (siehe Abbildung 4.5.3). Die obere Tabelle soll dabei alle verfügbaren Prozessvariablen aufzeigen, wohingegen die untere der beiden Tabellen jene auflistet, welche tatsächlich mit der ausgewählten „Control Action“ verknüpft sind.

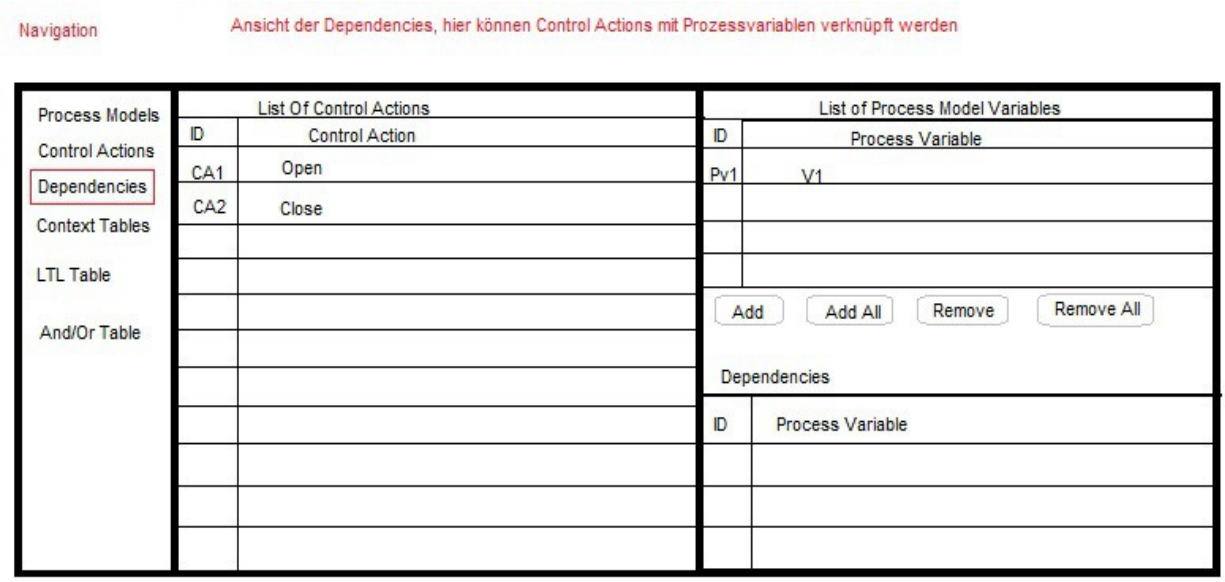

Abbildung 4.5.3: Gui-Prototyp der Dependencies Ansicht

Die folgende Grafik 4.5.4 bietet eine Skizze der Kontext-Tabelle.

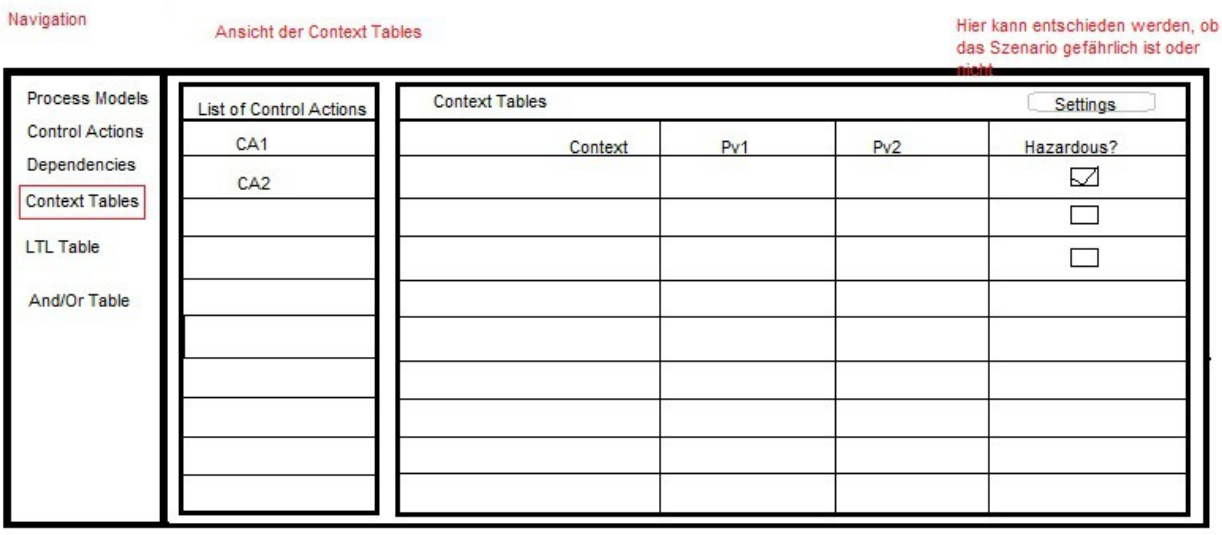

Abbildung 4.5.4: Gui-Prototyp der Kontext-Tabelle

In dieser Tabelle geschieht die meiste Arbeit des Analyseprozesses welcher mit „*XSTPA*“ durchgeführt werden kann. Die Tabelle wird in Abhängigkeit zu den verknüpften Prozessvariablen der angewählten „Control Action“ generiert. Dies kann darin resultieren, dass die Tabelle sehr viele Spalten bekommen kann. Die einzigen „festen“ Spalten sind die Erste und die Letzte.

In der ersten Spalte soll der Kontext der Zeile angegeben sein (z.B. Die Kontrollanweisung wurde rechtzeitig ausgeführt), die letzte Spalte muss vom Sicherheitsexperten mit „gefährlich“ oder „nicht gefährlich“ markiert werden. So kann jedes mögliche Szenario gefunden und eingestuft werden. Aus den mit „hazardous“ markierten Zeilen werden die Einträge für die „LTL Tabelle“ sowie die „Refined Safety“ Tabelle abgeleitet. Der Grund warum diese in keiner der GUI-Skizzen zu sehen ist, ist dass sich diese Ansicht erst während der Entwicklung von „*XSTPA*“ ergeben hat und zum Zeitpunkt der Planung noch nicht vorgesehen war. Dasselbe gilt für den Unterpunkt „And/Or Table“. Dieser wurde während der Entwicklung verworfen, da er zu starke Ähnlichkeiten mit der „*LTL“* Tabelle aufwies, da beide Tabellen („And/Or“, „*LTL*“) vor allem alternative Darstellungsformen der als gefährlich eingestuften Szenarien sind. Die Ansicht der „*LTL*“ Tabelle bietet dem Nutzer keinerlei große Interaktionsmöglichkeiten, da sie vor allem eine Grundlage zur Weiterarbeit für zukünftige Plugins bieten soll. Die einzig geplante Interaktionsmöglichkeit für den Benutzer in dieser Ansicht ist eine Exportfunktion. Es ist noch nicht klar in welchen Formaten die Tabelle genau exportiert werden soll. Geplant ist aber eine direkte Anlehnung an die unterliegende „XSTAMPP-Plattform“, welche die Dateiformate „JPG“, „PDF“ und „CSV“ als Exportmöglichkeit anbietet. Die Abbildung 4.5.5 zeigt das Schema welches die „LTL Tabelle“ verwenden wird. Im nächsten Kapitel werden genauere Erläuterungen der Funktionalität sowie die tatsächliche Umsetzung der „GUI“ folgen.

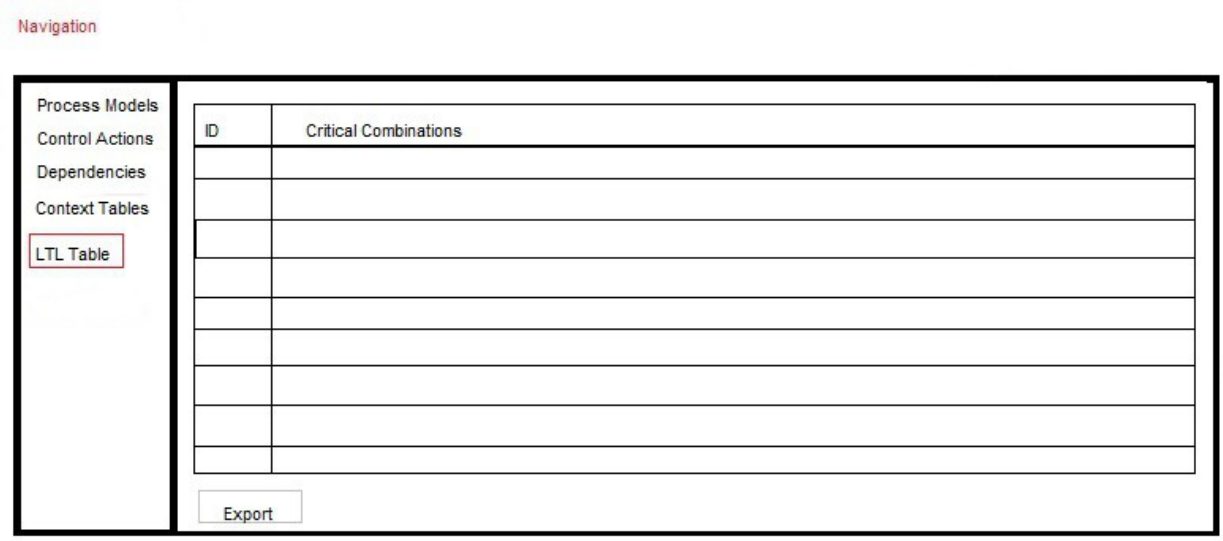

Abbildung 4.5.5: GUI-Prototyp der LTL-Tabelle

5 Implementierung

Dieses Kapitel beschäftigt sich mit dem Implementierungsvorgang von „XSTPA“. Es bietet eine Einsicht in das endgültige Design des Plugins, sowie über verwendete Algorithmen.

5.1 GUI

In diesem Kapitel werden die im vorigen Kapitel „Prototyp der GUI“ gezeigten Skizzen in ihrer umgesetzten Form gezeigt. Des Weiteren werden alle endgültigen Funktionalitäten genauer erklärt. Außerdem werden getroffene Entscheidungen, welche das Design vom geplanten Prototyp abweichen lassen, erläutert, sodass der Leser einen vertieften Einblick in den Implementierungsprozess erhält.

Um „*XSTPA*“ zu öffnen muss man im „Project Explorer“ von „*XSTAMPP*“ zum Unterpunkt „Control Structure with Process Model“ im derzeitigen Projekt navigieren, welcher den Link zum Plugin enthält. Dies ist in der Grafik 5.1.1 zu sehen.

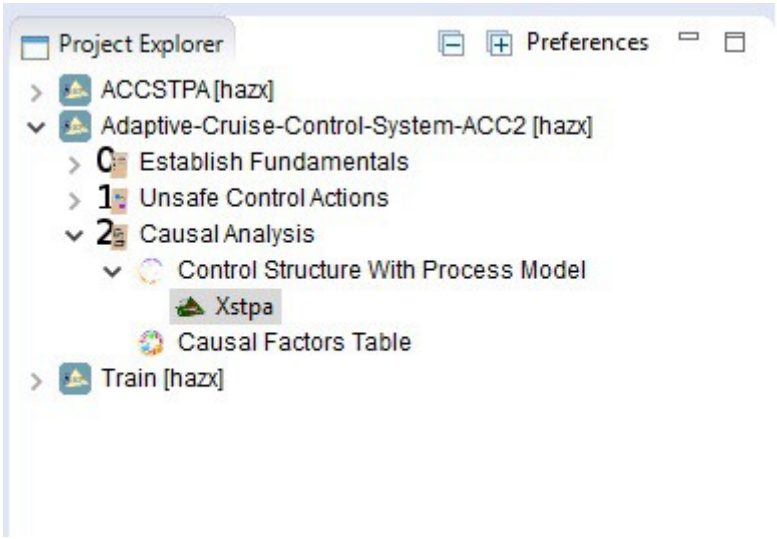

Abbildung 5.1.1: Project Explorer von "*XSTAMPP*", über diesen wird das Plugin aufgerufen

Das Logo von „*XSTPA*“ wurde ausgehend vom Logo von „*XSTAMPP*“ ausgearbeitet. Die allgemeine Form wurde für den Wiedererkennungswert übernommen, die Farben, sowie der Schriftzug im Inneren wurden angepasst (Abbildung 5.1.2).

Abbildung 5.1.2: Logo von "*XSTPA*"

Öffnet man nun das Plugin, so bekommt man die initiale Ansicht von „*XSTPA*“. Diese wurde wie geplant umgesetzt, falls die unterliegende Ansicht der Kontrollstruktur nicht geöffnet ist, wird diese geöffnet, danach legt sich die Ansicht der Prozessmodelle von „*XSTPA*“ darüber.

Wie in Abbildung 5.1.4 demonstriert, ist es möglich einen Kommentar für jede der in der Kontrollstruktur definierten Prozesswerte einen Kommentar zu hinterlassen. Um die Übersichtlichkeit während der Arbeit mit allen Tabellen in „*XSTPA*“ zu verbessern, wurden im Allgemeinen drei unterschiedliche Farben verwendet - Zwei alternierende, um den Wechsel einer Zeile zu kennzeichnen und eine dritte um die aktuell selektierte Zeile hervorzuheben.

Wechselt man nun in die Ansicht „Control Actions“, so sieht man, dass diese Ansicht nicht ganz planmäßig umgesetzt wurde (Abbildung 5.1.3). Es war geplant dass der zugehörige Controller für jede „Control Action“ mit angezeigt werden würde, jedoch unterstützte die unterliegende Datenstruktur von „XSTAMPP“ dieses Vorhaben nicht. Die in der Kontrollstruktur vorhandenen Pfeile (Abbildung 5.1.4) verknüpfen die bestehenden Komponenten nur auf eine optische Art und Weise. Dieses Problem hatte einige Änderungen zur Folge, da zu großen Teilen eine eigene Struktur aufgebaut werden musste, welche nicht auf Schnittstellen von „*XSTAMPP*“ zurückgreifen konnte. Abgesehen von der fehlenden „Controller-Spalte“ wurde in dieser Ansicht alles so umgesetzt wie geplant. Es gibt eine „Checkbox“ welche es dem Sicherheitsexperten ermöglicht die ausgewählte „Control Action“ als sicherheitskritisch einzustufen. Falls sie nicht sicherheitskritisch ist, wird sie in den folgenden Ansichten nicht weiter aufgeführt. Außerdem kann man die durch „*XSTAMPP*“ festgelegte Beschreibung nochmals einsehen oder gegebenenfalls ändern.

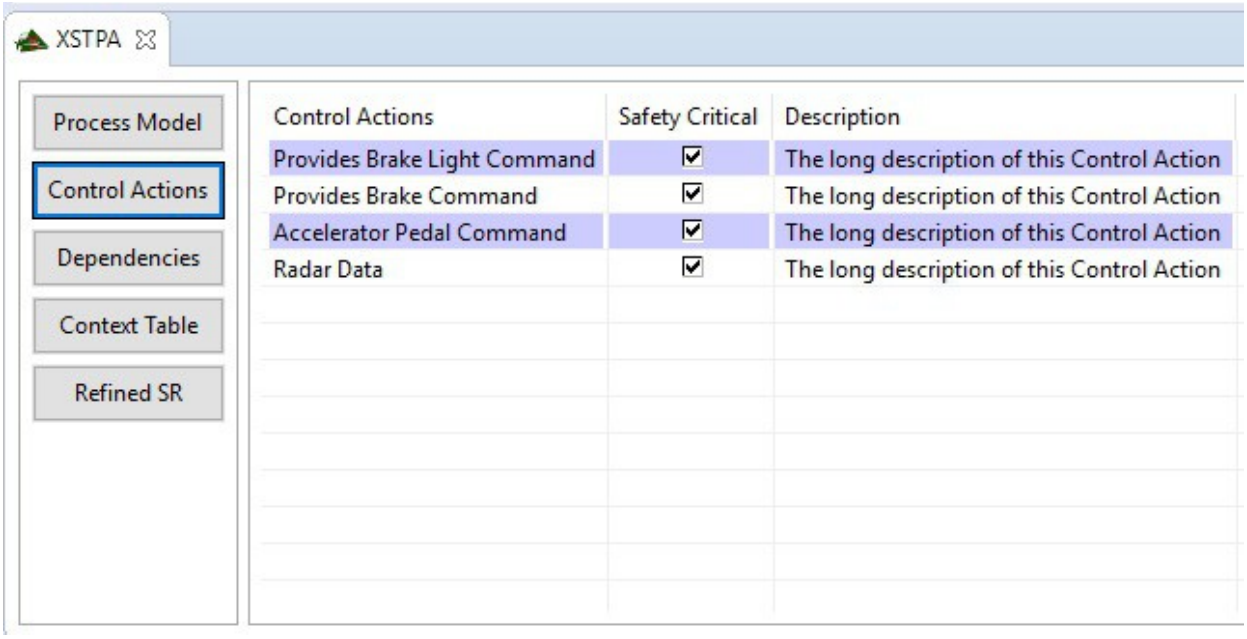

Abbildung 5.1.3: Ansicht der "Control Actions" - Eine Auflistung aller definierten "Control Actions"

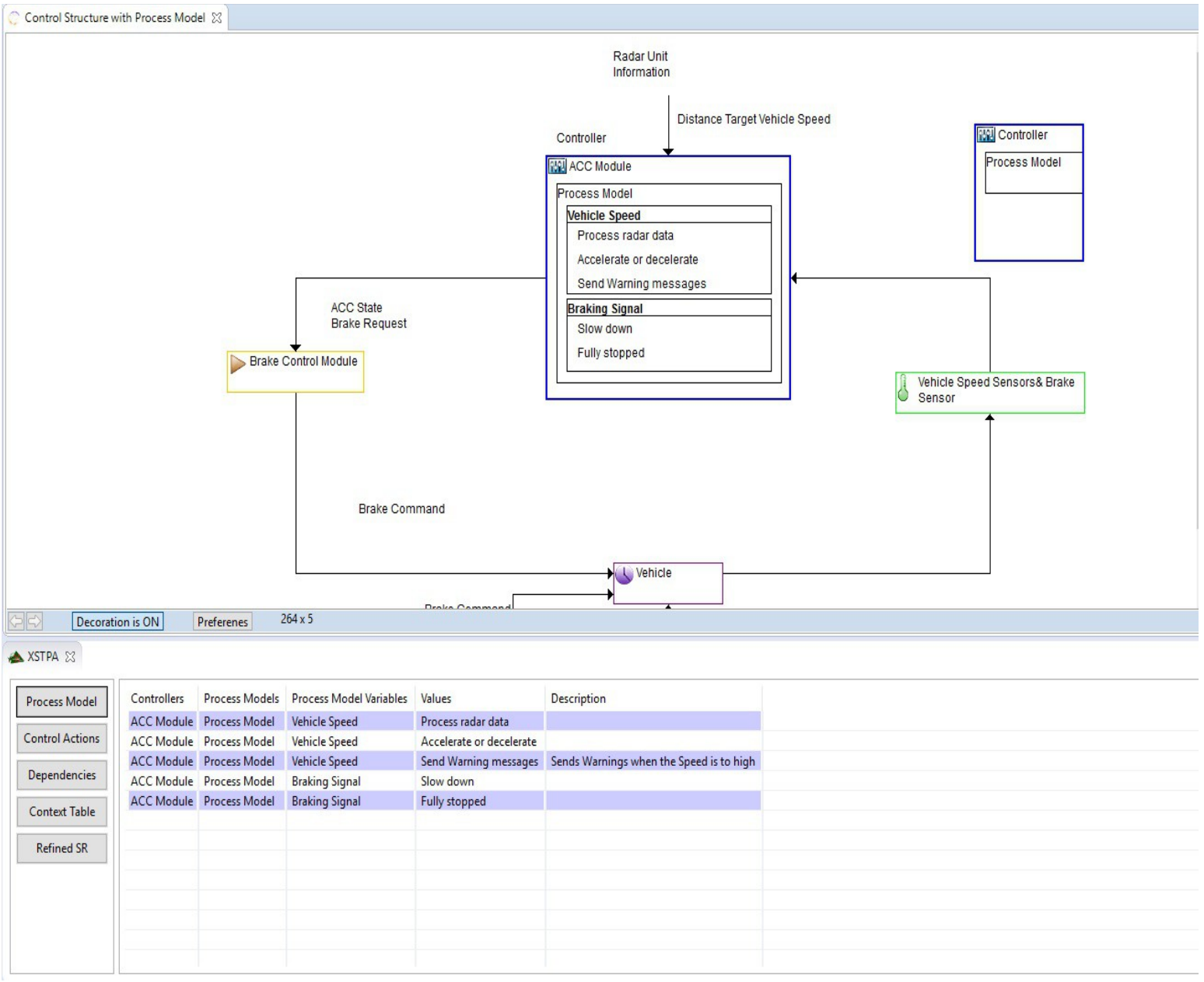

Abbildung 5.1.4: Initiale Ansicht von "*XSTPA*" - Auflistung der Prozessmodelle mit zugehörigen Controllern

Möchte man nun die sicherheitskritischen „Control Actions“ weiter auf die genauen Szenarien untersuchen, so kann man den nächsten Schritt in der „Dependencies“ Ansicht vornehmen (Abbildung 5.1.5). In diesem Fenster werden die „Control Actions“ mit den Prozessvariablen verknüpft. Dies geschieht durch das Selektieren der zu bearbeitenden „Control Action“, dadurch werden alle verfügbaren Prozessvariablen in der Tabelle rechts oben angezeigt (insofern diese „Control Action“ noch nicht bearbeitet wurde). Nun können die Variablen durch die unten stehenden Buttons mit der „Control Action“ verknüpft werden. Hierbei ist darauf zu achten, dass die „Control Actions“ jeweils mit den „richtigen“ Prozessvariablen in Verbindung gebracht werden. Damit ist der bereits oben angesprochene Punkt gemeint, dass „*XSTPA*“ alle möglichen Prozessvariablen aus allen Prozessmodellen und Controllern anzeigt und nicht nur jene, welche laut der Kontrollstruktur zusammengehören (da die in der Kontrollstruktur gemachten Verbindungen nur von einer rein optischen Natur sind). Diese Ansicht wurde gegenüber dem Prototyp leicht abgeändert. Es existieren nun zwei Reiter: „Control Action Provided“ und „Control Action Not Provided“, welche den Kontext der „Control Action“ bestimmen. Dies hat den Grund, dass der Kontext der jeweiligen „Control Action“ eine sehr große Auswirkung für die Nachfolgenden Arbeitschritte hat und deshalb so früh wie möglich festgelegt werden sollte.

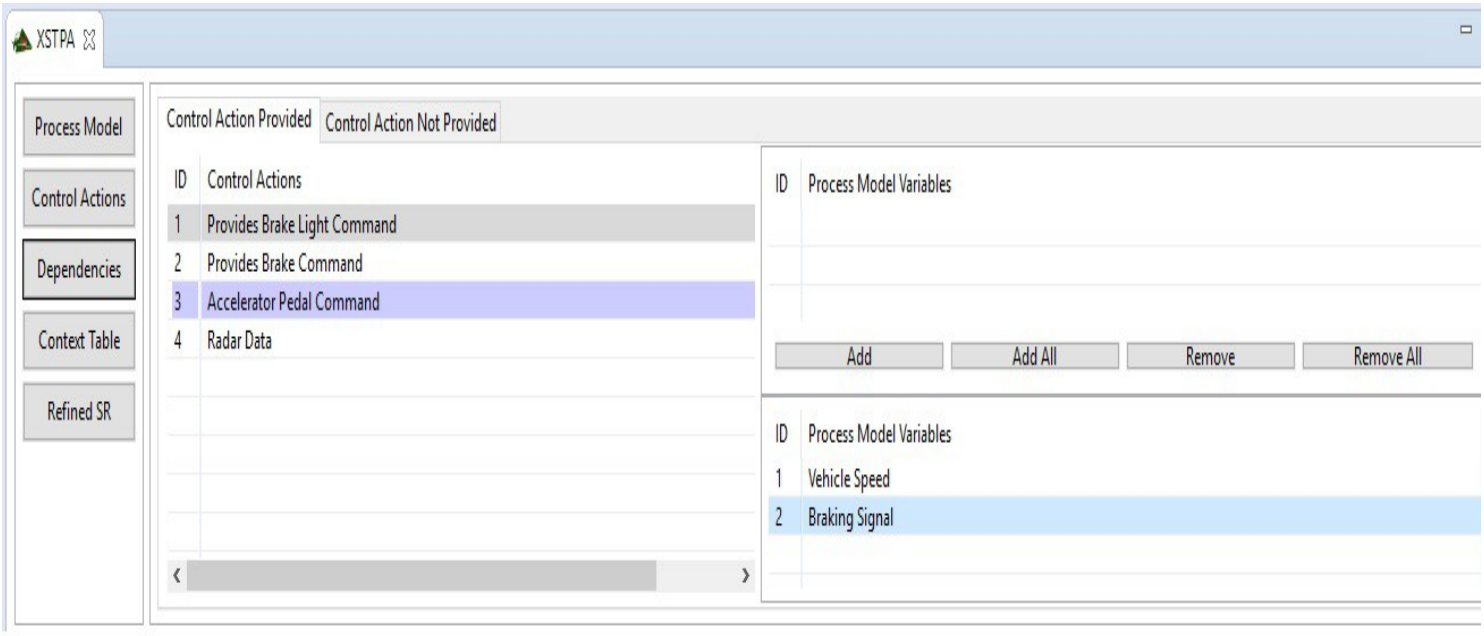

Abbildung 5.1.5: "Dependencies" Ansicht - Ermöglicht die Verknüpfung von “Control Actions” mit den dazugehörigen Prozessvariablen

Möchte man nun eine Kontext-Tabelle für eine (oder mehrere) „Control Actions“ generieren, dann wechselt man in die „Context Table“ Ansicht (Abbildung 5.1.10). Diese ermöglicht es dem Benutzer mittels eines „Combinatorial Testing“-Algorithmus eine sehr auf den Benutzer angepasste Tabelle mit potentiell gefährlichen Szenarien zu erstellen. Möchte man eine Tabelle generieren, so ist es erforderlich eine „Control Action auszuwählen und im Anschluss auf den „Generate“-Button zu klicken. Bei der Erstbenutzung des Programms wird nachgefragt, ob der

Pfad für die „JAR“ Datei gesetzt ist welche den Algorithmus enthält, da beim Erstellen der Tabelle einige Textdateien entstehen und der Algorithmus mit diesen als Parameter aufgerufen wird. Sobald der Pfad gesetzt ist, können die Einträge für die Tabelle generiert werden. Wird der Algorithmus mit den Standardeinstellungen aufgerufen, so wird immer die größtmögliche Tabelle generiert, sodass alle potentiell gefährlichen Szenarien abgebildet sind. Der Benutzer muss diese Einstellungen über den „Settings Button“ anpassen, falls er eine kleinere Tabelle generieren möchte.

Um die Einstellungen anpassen zu können, öffnet sich ein neues Fenster, wie in Abbildung 5.1.6 dargestellt. Dieses Fenster ist in drei Reiter aufgeteilt: der erste nennt sich „General Options“ und enthält, wie der Name schon sagt, allgemeine Einstellungen. Man kann zwischen fünf verschiedenen Algorithmen wählen. Die ersten drei („IPOG“, „IPOG-F“ und „IPOG-F2“) sind für mittelgroße Systeme gedacht (bis zu 20 Eingabeparameter). „IPOG-D“ ist für sehr große Systeme gedacht (mehr als 20 Parameter) und der „Base Choice“ Algorithmus stellt die Teststärke automatisch auf 1, was bedeutet, dass jeder Parameter höchstens einmal verglichen werden muss (sehr schnell, aber nicht sehr genau in der Abdeckung der Testfälle). Die nächste Auswahlmöglichkeit ist eine „Combobox“, welche die Teststärke des Algorithmus beeinflusst. Es kann eine Zahl 1<= t <=6 gewählt werden, sowie die Option „Mixed Strength“. Diese Zahl gibt an, wie oft jeder Parameter mindestens mit einem anderen verglichen werden muss, je höher die Teststärke desto umfangreicher wird das Ergebnis. Die Standardeinstellung ist, falls die Anzahl der Eingabeparameter zwischen 1<= t <=6 liegt, diese Anzahl t als Teststärke für den Algorithmus gewählt wird. Ansonsten wird t=6 als Teststärke verwendet. Die Option „Mixed Strength“ sollte immer dann gewählt werden, wenn unterschiedliche Teststärken im Reiter „Relations“ für die Parameter festgelegt worden sind. Die folgende Combobox unterstützt momentan nur den Modus „Scratch“. Ursprünglich sollte sie einen weiteren Modus („Extend“) unterstützen, jedoch war dieser bis zum Release zu fehlerhaft. Die Einstellung „Scratch“ lässt den Algorithmus auf den kompletten Datensatz laufen, sodass er komplett neu erstellt wird. Die Option „Extend“ hätte einen bestehenden Datensatz erweitern sollen (z.B. wenn ein Parameter hinzukommt, oder entfernt wird), um die Generierung der Testfälle zu beschleunigen. Die letzten beiden Interaktionselemente sind eine „Combobox“, sowie eine „Checkbox“ welche die im Reiter „Constraints“ gemachten Einschränkungen betreffen. Falls die Checkbox „Ignore Constraints“ selektiert ist, so werden alle getroffenen Einschränkungen ignoriert. Die Optionen für „Constraint Handling“ sind zum einen „Forbidden Tuples“ welche für mittlere bis große Systeme optimal ist und „CSP Solver“, welche für sehr kleine Systeme Anwendung findet [2].

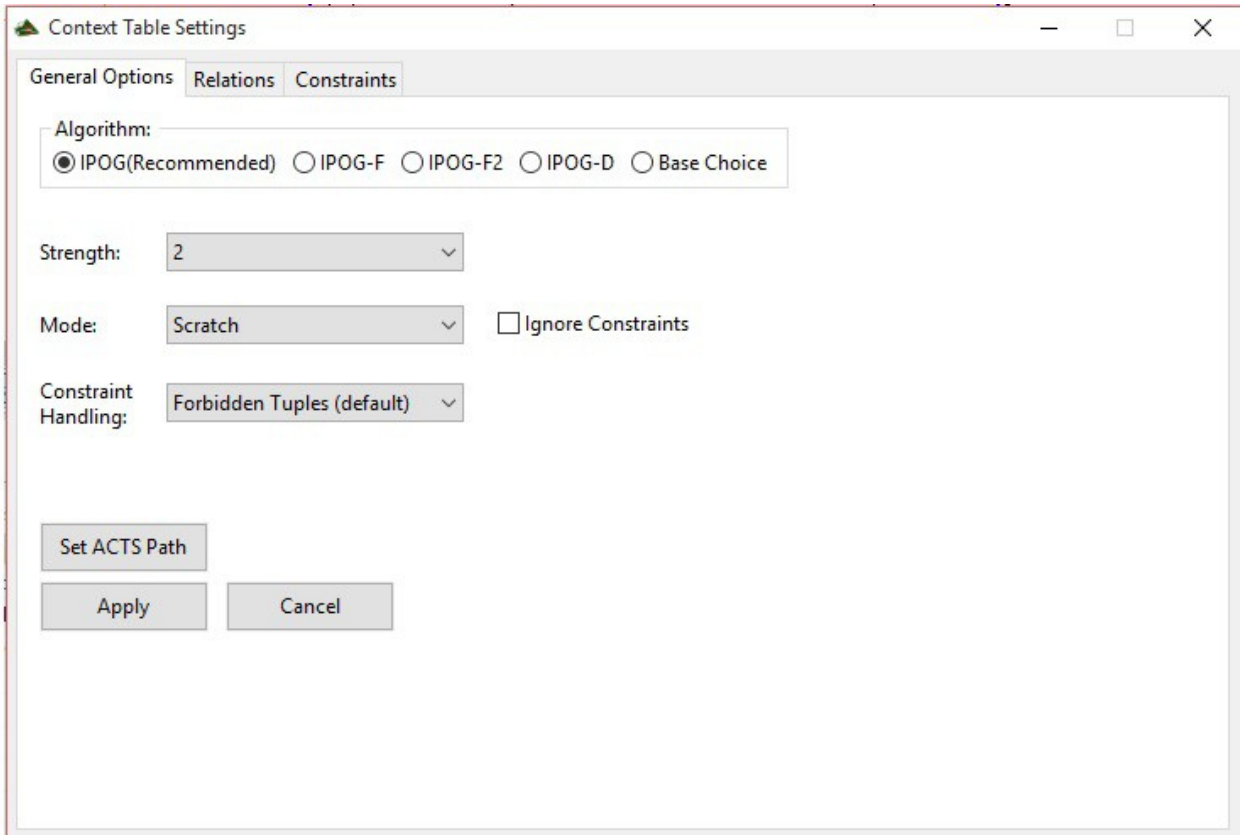

Abbildung 5.1.6: Allgemeine Optionen für den ACTS Algorithmus

Der Reiter „Relations“ (Abbildung 5.1.7) beinhaltet alle Prozessvariablen, welche der momentan ausgewählten „Control Action“ zugewiesen wurden. Diese können nun untereinander mit verschiedenen Stärken verglichen werden. Dazu wählt man alle Parameter welche untereinander in Relation stehen aus und gibt ihnen die gewünschte Stärke (zwischen 1 und 6) . Außerdem muss in den Allgemeinen Optionen „Mixed Strength“ angewählt sein.

In der „Constraints“ Ansicht (Abbildung 5.1.8) findet der Benutzer einen Editor, mit welchem er boolsche Ausdrücke formulieren kann. So können die Parameter weiter verfeinert werden. Es können alle aufgelisteten Zeichen für den Ausdruck benutzt werden, sowie die Parameter mit den zugehörigen Werten. Die „Constraints“ müssen über den Editor jeweils einzeln hinzugefügt werden. Hat der Benutzer alle Einstellungen getroffen, so kann er den Algorithmus über den „Apply Button“, welcher in jedem der Reiter zu finden ist, starten.

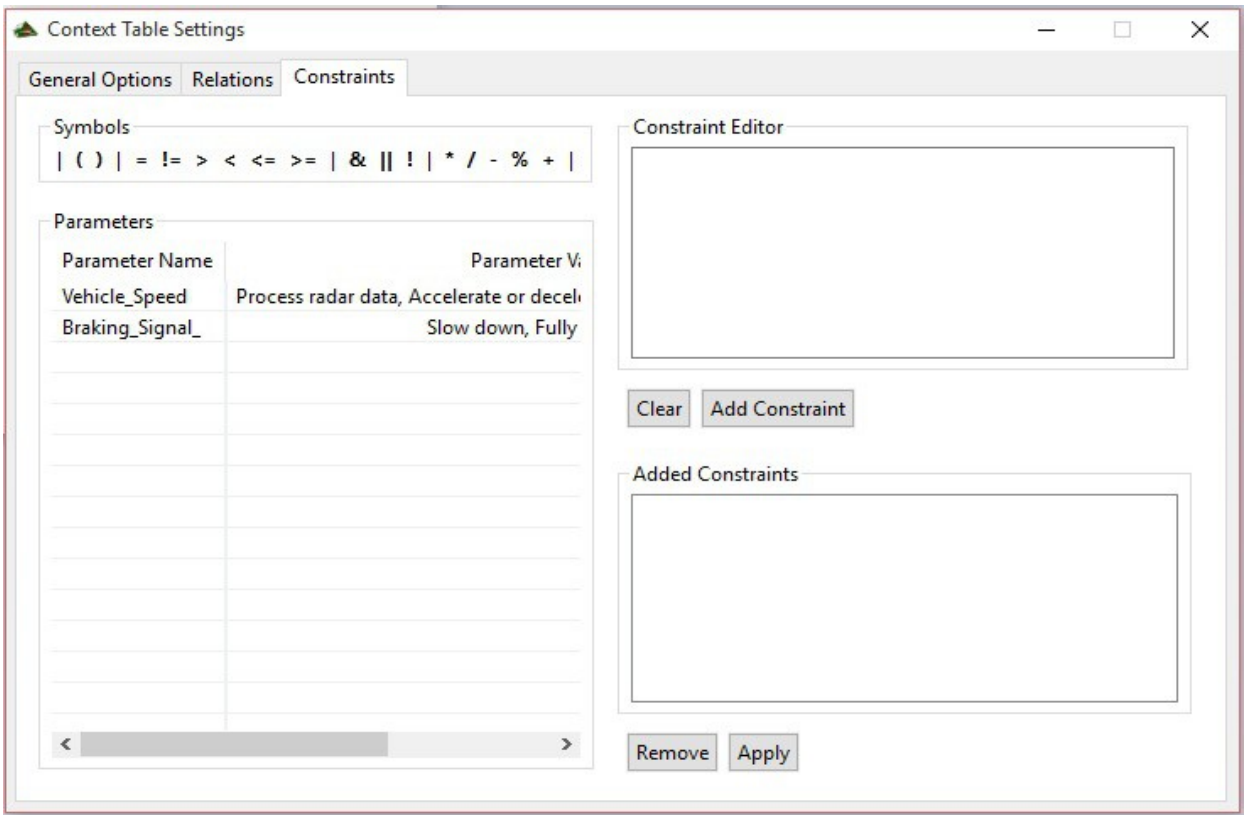

Abbildung 5.1.7: Die "Constraints" Ansicht - Es können Boolsche Ausdrücke über den Editor erstellt werden

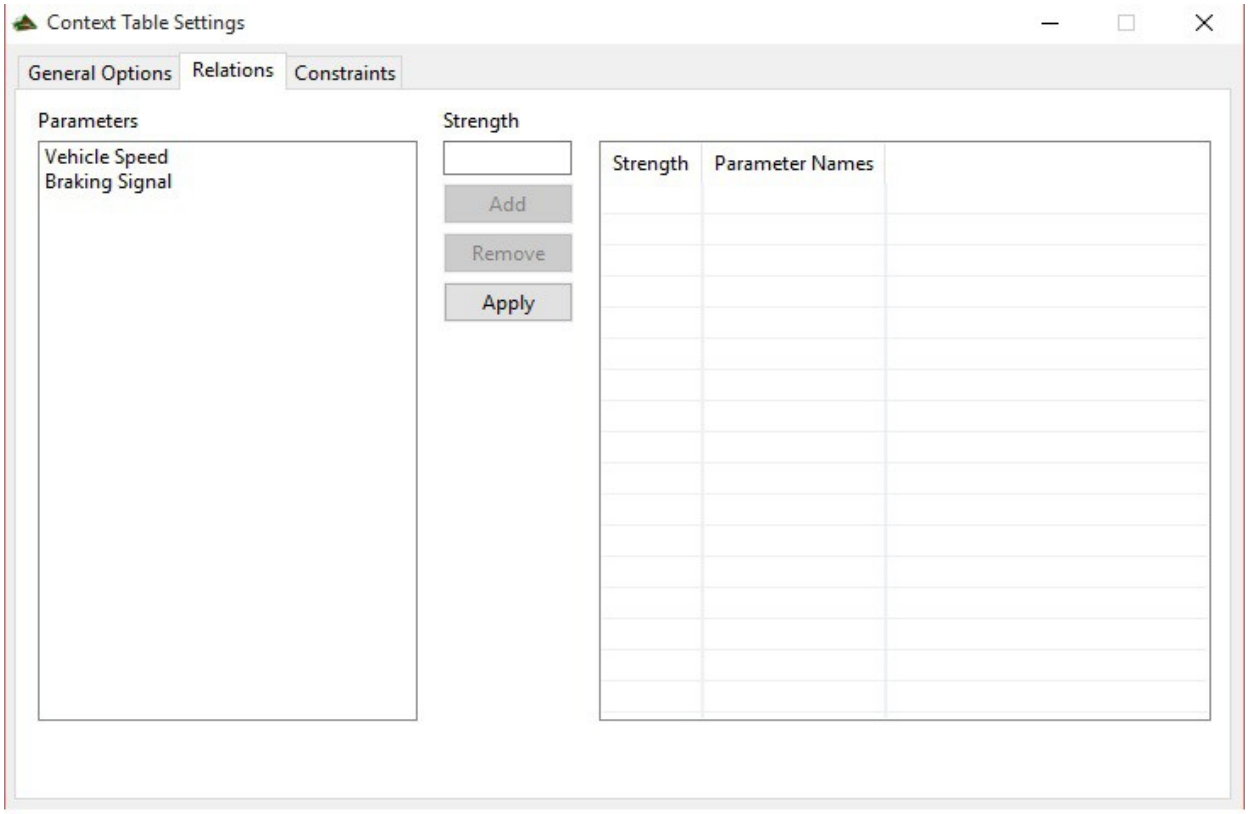

Abbildung 5.1.8: Die "Relations" Ansicht - hier können unterschiedliche Teststärken für die Parameter definiert werden

Hat man nun eine Tabelle generiert, so kann man alle Szenarien durchgehen und entscheiden, ob diese im jeweiligen Kontext gefährlich sind oder nicht. Befindet man sich im Reiter „Control Action Not Provided“ so gibt es nur eine „Checkbox“ pro Zeile, da man dort keine zeitliche Unterscheidung machen kann. Möchte der Anwender nun ein Szenario hinzufügen, so kann er auf den „Add Button“ klicken. Dies generiert eine leere Zeile, welche er mit den entsprechenden Informationen befüllen kann. Möchte der Nutzer einen Eintrag löschen, so muss er nur die Zeile selektieren und auf den „Remove Button“ klicken.

Eine weitere Funktion der Tabelle ist die Verifizierung von Einträgen (Abbildung 5.1.9). Falls ein Eintrag in „Provided“ einem Eintrag in „Not Provided“ identisch ist (inklusive der Hazardous-Spalte), so werden beide Einträge rot hinterlegt und der Schriftzug am unteren Ende der Tabelle zeigt an, wieviele Konflikte in der Tabelle vorhanden sind. Dies ist besonders bei großen Tabellen mit sehr vielen Einträgen hilfreich, da sonst sehr schnell logische Fehler entstehen können. Auf diese Weise können solche Fehler leicht vermieden werden. Diese Funktion wird nicht automatisch aufgerufen, sie muss über den „Verify Button“ ausgelöst werden.

Wenn die Anzahl der Einträge sehr groß ist, dann kann es helfen sie über den eingebauten Filter in der Tabelle darstellen zu lassen, welcher in der Grafik 5.1.9 zu sehen ist. Momentan unterstützt der Filter folgende Funktionen: „Alle Einträge anzeigen“, „Nur Einträge anzeigen, welche als Hazardous markiert sind“ und „Nur Einträge anzeigen, welche nicht als Hazardous markiert sind“.

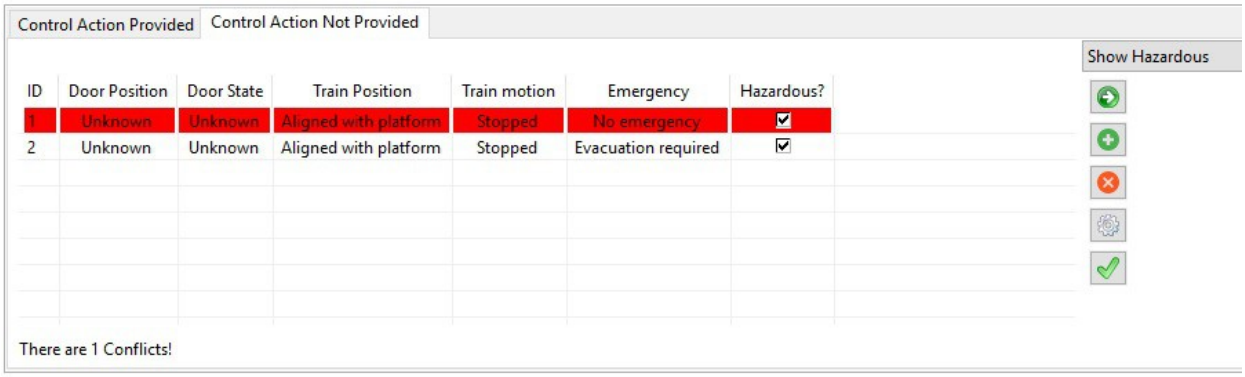

Abbildung 5.1.9: Logischer Konflikt in einer Kontext Tabelle, Filtereinstellung: Hazardous Only

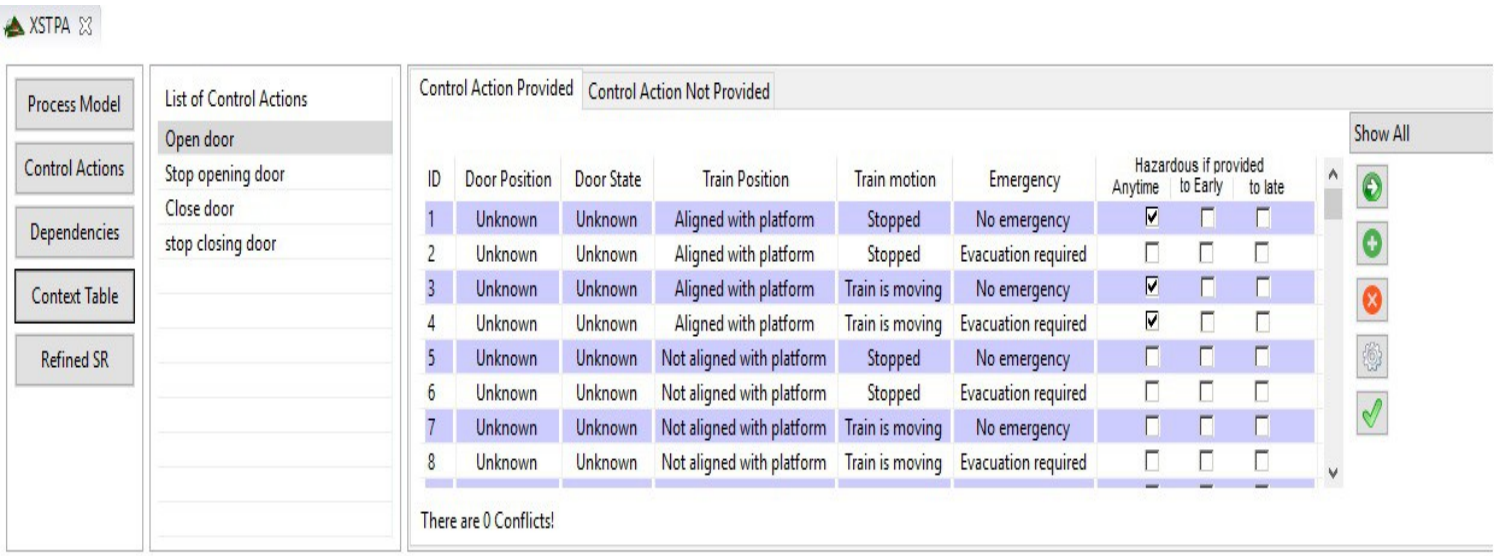

Abbildung 5.1.10: "Context Table" Ansicht im “Provided” Kontext

Wie man erkennen kann, weicht diese Ansicht am stärksten von der geplanten Ansicht des Prototyps ab. Dies hatte hauptsächlich den Ursprung, dass bei der Planung des Prototyps noch zu wenig über den Arbeitsablauf bekannt war. Hauptsächlich wurden die Tabellenspalten geändert, sowie einiges an Funktionalität hinzugefügt.

Der letzte Punkt in der Navigationsleiste nennt sich „Refined SR“ was für „Refined Safety Requirements“ steht. In dieser Ansicht werden alle Einträge aller erstellten Kontext-Tabellen aufgelistet, welche als „Hazardous“ gekennzeichnet sind (Abbildung 5.1.11). Diese Ansicht ist während der Entwicklung des Plugins entstanden, als Ersatz für den Wegfall der „And/Or Table“.

Die gezeigte Tabelle enthält die verlinkte „Control Action“ und den Kontext des jeweiligen Eintrags. Außerdem ist der Eintrag in einer verkürzten Form nochmals in der Spalte „Critical Combinations“ zu finden. Die Spalten „Related Unsafe CA“ und „Linked Hazards“ greifen auf Informationen aus dem „Unsafe Control Action Table“ von „*XSTAMPP*“ zurück. Es können mit jedem Eintrag die unsicheren „Control Actions“ verknüpft werden, welche in der „Unsafe Control Action Table“ erstellt wurden. In die letzte Spalte können weitere Constraints eingetragen werden, sodass das gefundene kritische Szenario bestmöglich reguliert ist.

Des weiteren befinden sich zwei Buttons in der „Refined SR“ Ansicht, einer ruft die Exportfunktion von „*XSTPA*“ auf (welche auch über den „*XSTAMPP*“ Export aufgerufen werden kann), der zweite generiert eine „*LTL*“-Tabelle aus den Einträgen in der „Refined SR“ Tabelle. Die „*LTL*“-Tabelle kann in Abbildung 5.1.12 betrachtet werden, diese bietet dem Nutzer zum jetzigen Zeitpunkt jedoch noch keinerlei Funktionalität, abgesehen davon, dass sie exportiert werden kann, um eine Weiterverarbeitung zu gewährleisten.

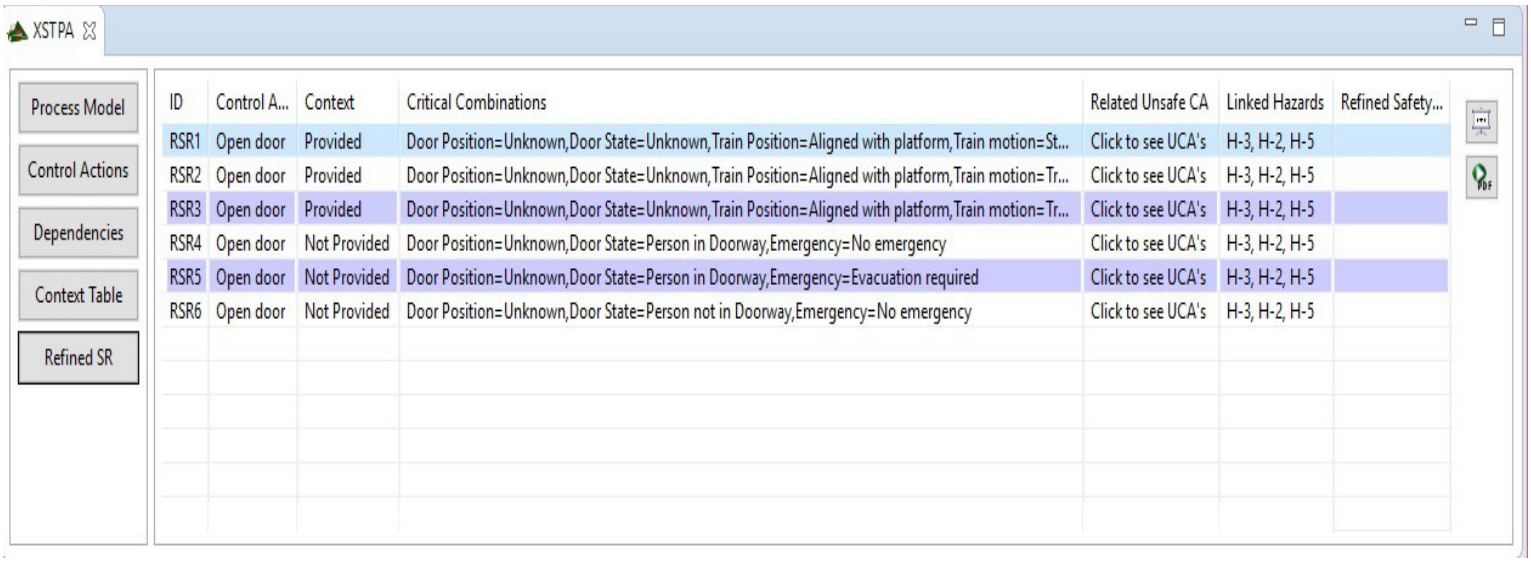

Abbildung 5.1.11: Refined Safety Requirements Tabelle - bietet Funktionen zum verlinken der "Unsafe Control Actions" aus "*XSTAMPP*"

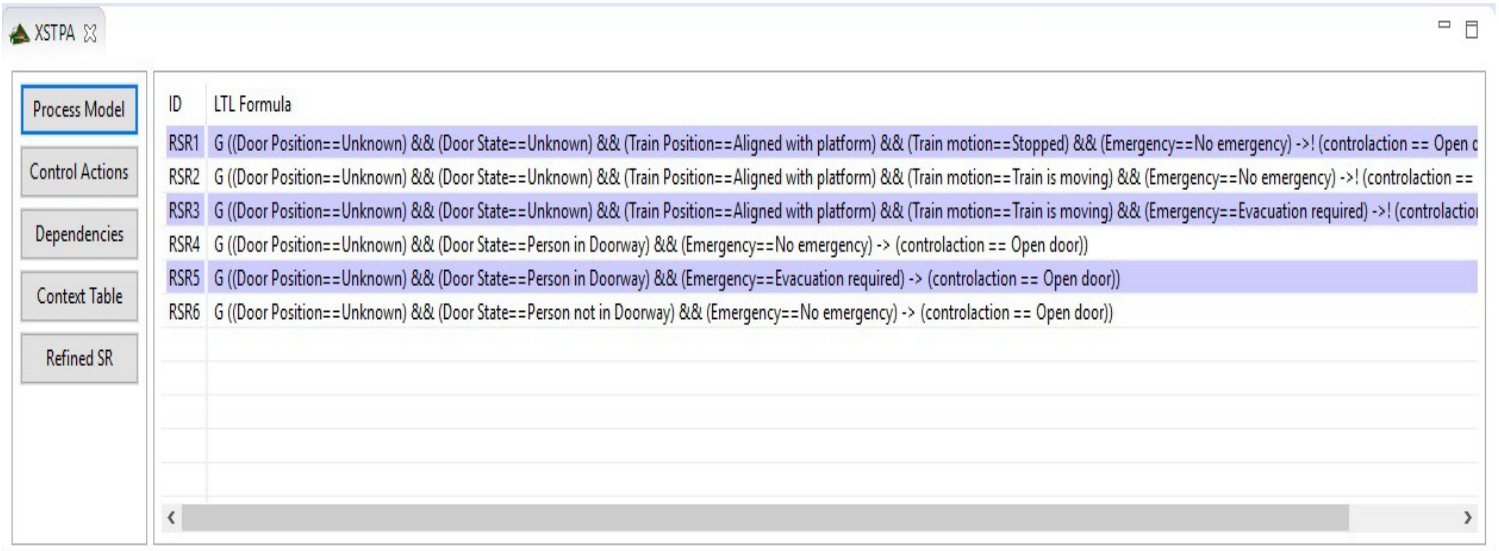

Abbildung 5.1.12: "*LTL*"-Tabelle - zeigt die als "Hazardous" markierten Einträge als boolsche Formel

5.2 Combinatorial Testing

In diesem Kapitel wird erläutert aus welchem Grund die Kontext-Tabellen auf ihre Größe hin modifizierbar gemacht wurden.Wie bereits erwähnt, können Kontext-Tabellen sehr schnell sehr groß werden. Für jede Prozessvariable die einer „Control Action“ hinzugefügt wird, erhöht sich die Anzahl der Gesamtkombinationen exponentiell.

Um ein Beispiel zu nennen: Zehn Variablen mit jeweils zwei möglichen Werten ergibt

$$2^{10} = 1024$$

Kombinationen. Und obwohl das noch ein relativ klein gewähltes Szenario ist, ist es für den Sicherheitsexperten sicherlich nicht angenehm 1024 Testfälle daraufhin zu überprüfen, ob diese gefährlich sind oder nicht. Aus diesem Grund wurde der Ansatz entwickelt, die Tabelleneinträge der Kontext-Tabelle mittels kombinatorischem Testen zu erstellen. Dieses Verfahren vergleicht nicht jeden einzelnen Wert mit einem anderen, sondern es vergleicht immer Wertepaare. Das führt zu einer deutlichen Reduktion der Testfälle. Um das Beispiel mit 2^{10} fortzuführen:

Wenn man bei zehn Variablen mit jeweils zwei Werten Paare bildet, so kommt man auf 45 mögliche Kombinationsmöglichkeiten, dies wird mit dem Binomialkoeffizient

$$\binom{10}{2} = 45$$

berechnet. Da jetzt aber mit Wertepaaren und nicht mehr mit einzelnen Werten gerechnet wird, muss die Anzahl aller möglichen Testfälle neu berechnet werden. Um diese zu erhalten müssen die Wertepaare mit den Kombinationsmöglichkeiten multipliziert werden:

$$2^2 * \binom{10}{2} = 180$$

Da man fünf Wertekombinationen in einen einzigen Testfall packen kann (vgl. Abbildung 5.2.1), benötigt man maximal

$$180/5 = 36$$

Testfälle um eine volle Abdeckung zu erreichen. Jedoch sind bei diesen 36 herausgearbeiteten Testfällen einige redundante Fälle dabei (vgl. Abbildung 5.2.1). Aus diesem Grund können diese 36 Testfälle auf acht Fälle reduziert werden (vgl. Abbildung 5.2.2), was eine Abdeckung von

$$2^2 * \binom{10}{2} = 180$$

Testfällen zur Folge hat.

Da es aber wie oben berechnet insgesamt 1024 mögliche Kombinationen gibt, ist dies unzureichend, da nur 180 von 1024 Fällen abgedeckt sind. Allerdings kann die Größe der Wertepaare erhöht werden, man kann statt einem Paar einfach eine Gruppe von drei Variablen zusammennehmen. Es ergeben sich für eine Dreiergruppe von Variablen die folgenden Werte:

- Anzahl aller möglichen Testfälle: $960 = 120 * 2^3$
- Anzahl Testfälle für volle Abdeckung (mit redundanten Fällen): 960/3=320
- Anzahl Testfälle für volle Abdeckung (ohne redundante Fälle): 13

Man erkennt schnell, dass es ein sehr gutes Ergebnis ist, wenn man mit 13 Testfällen ganze 960 Wertekombinationen abdecken kann. Es fehlen nun nur noch

$$64 = 1024 – 960$$

Testfälle um die volle Abdeckung aller Wertekombinationen zu erreichen. Es hat sich gezeigt, dass eine Gruppe von sechs Wertepaaren genügt, um mit einer sehr hohen Erfolgsquote alle sicherheitskritischen Szenarien zu ermitteln. Die Wahrscheinlichkeit, dass ein sehr wichtiges sicherheitsrelevantes Szenario bei den wenigen Testfällen, welche nicht abgedeckt werden, dabei ist, ist verschwindend gering.[6] [7]

Aus diesem Grund verwendet der in „*XSTPA*“ verwendete Algorithmus welcher vom „National Institute of Standards and Technology“ (NIST) entwickelt wurde, eine Teststärke zwischen 1 und 6. So ist sichergestellt, dass der Anwender eine überschaubare Anzahl an Szenarien einstufen muss und trotzdem die bestmögliche Sicherheit gewährleistet wird.

Der Algorithmus selbst wird über die Kommandozeile aufgerufen, mit den in „*XSTPA*“ gemachten Einstellungen, als Parameter. Der Algorithmus liest eine durch „*XSTPA*“ erstellte Textdatei aus, welche alle vom Algorithmus benötigten Informationen enthält. Der Algorithmus schreibt selbst eine Textdatei, welche die Ausgabe enthält. Diese wird in optisch aufbereiteter Form von „*XSTPA*“ in der dafür vorgesehenen Kontext-Tabelle wiedergegeben.

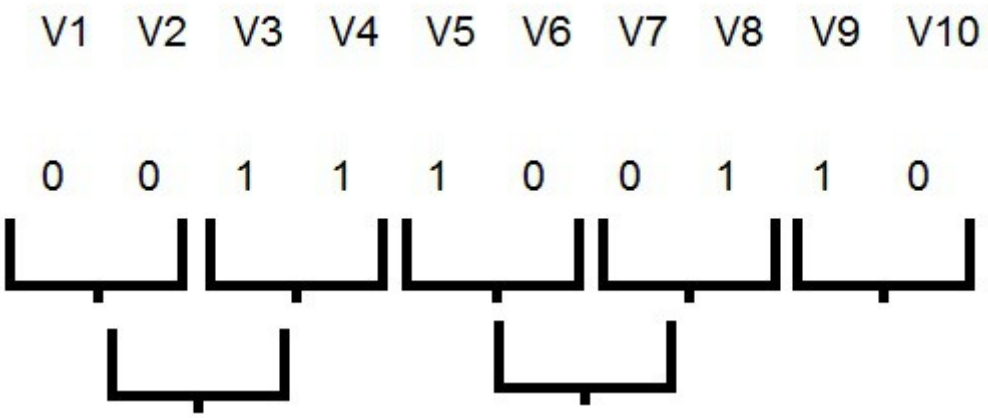

Abbildung 5.2.1: Aufteilung von Wertepaaren - Es passen fünf Wertepaare in einen Testfall, jedoch deckt jeder Test implizit noch weitere Paare ab.

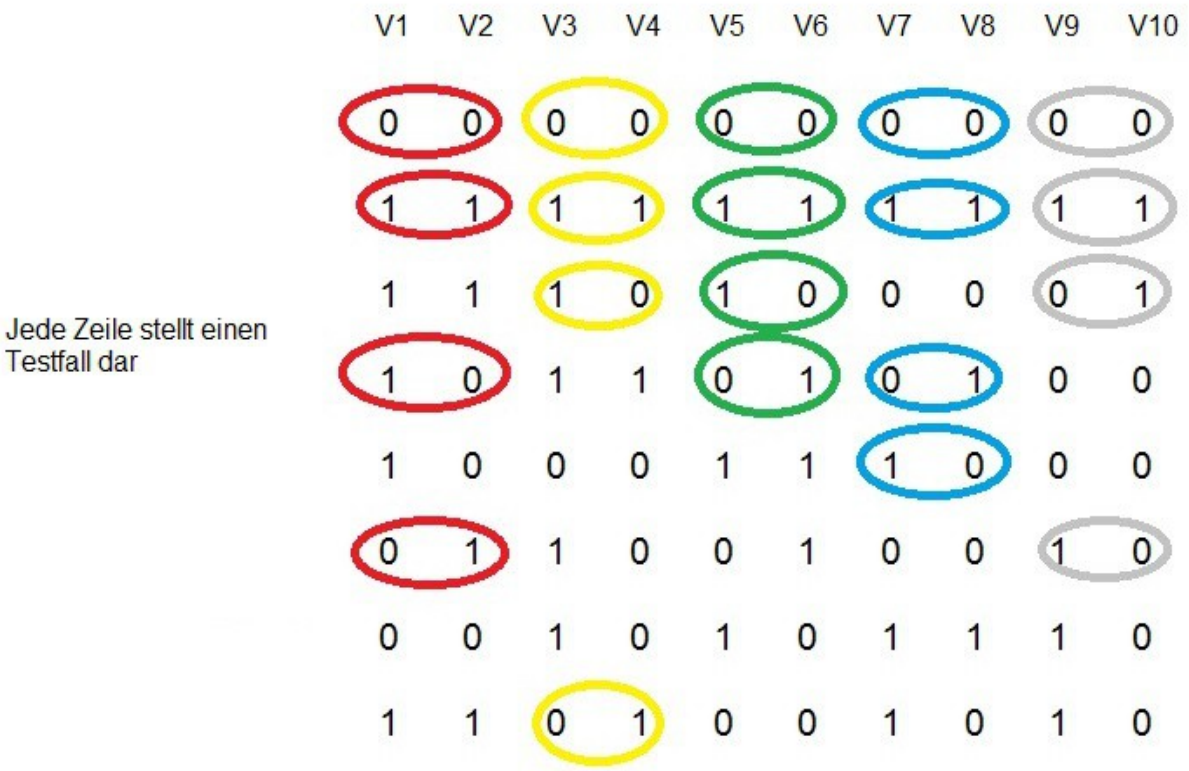

Abbildung 1: Covering Array - Ein Array, welches 180 Wertekombinationen mit nur 8 Tests überdeckt

6 Ergebnisse

Während des Projekts wurde ein Tool geschaffen, welches den „*STPA*“ Analyseprozess durch automatisierte Abläufe und simple Kontrollmöglichkeiten deutlich erleichtert. Das komplette Softwareprojekt wurde in enger Zusammenarbeit mit Asim Abdulkhaleq durchgeführt um ein Werkzeug zu erstellen, welches möglichst nah an der Praxis liegt.

„*XSTPA*“ ist ein Plugin welches für die von der Universität Stuttgart entwickelte „*XSTAMPP*“ Plattform entwickelt wurde. Es implementiert die Ansätze welche von John Thomas und Asim Abdulkhaleq theoretisch erarbeitet worden sind. Somit ist es möglich eine noch bessere Gefahrenanalyse mittels „*XSTAMPP*“ durchzuführen. Außerdem legt es den Grundstein für viele weitere Plugins durch die Aufbereitung der als gefährlich eingestuften Szenarien als „*LTL*“ Formel. Das Plugin bietet einen umfangreichen Export aller Daten als „CSV“, „PNG“ und „PDF“. Es wurde mit dieser Arbeit eine ausführliche Dokumentation aller Arbeitsschritte erstellt, sowie eine Erklärung für alle Funktionen innerhalb des Programms.

7 Zusammenfassung und Ausblick

Anlass zu dieser Arbeit war die Arbeit von John Thomas [4]. Er entwickelte eine mathematische Grundlage für „*STPA*“ und entwickelte eine Methode unsichere „Control Actions“ aus dem gegebenen Prozessmodell abzuleiten. Des weiteren bot die erweiterbare und sich immer noch in der Entwicklung befindliche „*XSTAMPP*“ Plattform eine gute Grundlage, diese Ansätze, sowie die Idee von Asim Abdulkhaleq die Kontext-Tabelle mittels kombinatorischem Testen zu verkleinern, in einem Werkzeug zu vereinen. Aus diesem Grund wurde das Plugin „Extended *STPA*“ entwickelt, sodass die „*XSTAMPP*“ Plattform zu einem mächtigen Werkzeug für jeden Sicherheitsanalysten werden kann.

In “*XSTPA*” wurden viele der geplanten Features umgesetzt, dennoch gibt es noch sehr viele Möglichkeiten das Plugin zu verbessern.

Einer der größten und wichtigsten Punkte ist die Speicherfunktion für die Tabellendaten von “*XSTPA*”. Momentan werden die Daten über ein XML-File abgelegt, welches zum Start von “*XSTAMPP*” ausgelesen wird. Problematisch ist dabei, dass es kein Versionierungssystem gibt. So kann für jede “Control Action” nur eine einzige Kontext-Tabelle abgespeichert werden. Es wäre deutlich besser, wenn für jede “Control Action” beliebig viele Tabellen abgespeichert werden könnten. Außerdem gibt es durch das Plugin ein neues Dateiformat. Die alte Projektdatei in welcher dass Plugin noch nicht enthalten war, wurde als „.Haz“ abgespeichert. Projekte in denen „*XSTPA*“ aktiv ist, werden unter der Dateiendung „.Hazx“ abgelegt. Dies führt dazu, dass alte Projekte unter Umständen nur schwer das neue Plugin nutzen können.

Ein weiterer wichtiger Punkt wäre die logische Verknüpfung des “Controllers” mit den jeweiligen “Control Actions”. Wenn diese Verbindung bestehen würde, könnte die “Usability” für den Nutzer deutlich erhöht werden, da so eine große Fehlerquelle deutlich eingedämmt werden würde.

Die Einbindung von „*ACTS*“, der Jar-Datei, welche den Algorithmus zum kombinatorischem Testen enthält, ist leider auch nicht optimal. Es wäre besser diese als Bibliothek in „*XSTPA“* einzubinden. Denn momentan wird sie als externe Datei aufgerufen. Dafür muss zwingend ein Pfad gesetzt werden, was für den Benutzer sehr verwirrend und umständlich sein kann.

Da die Zeit leider nicht mehr reichte, steht die „*LTL*“-Tabelle auch nur zum Export zur Verfügung. Es bestehen sehr viele Möglichkeiten diese booleschen Formeln auszuwerten, sodass auf eine einfache Art und Weise „constraints“ für die sicherheitskritischen Szenarien erstellt werden können. Diese könnten dann maschinell ausgelesen und weiterverarbeitet werden, was den Analyseprozess automatisierter ablaufen lassen würde.

Es gibt sicher noch einige Funktionen die in „*XSTPA*“ mit integriert werden könnten, jedoch sind die oben genannten jene, welche die größte Relevanz besitzen.

8 Literaturverzeichnis

[1] A. Abdulkhaleq, S. Wagner. Open Tool Support for System-Theoretic Process Analysis. In Proc. 2014 STAMP Conference, at Massachusetts Institute of Technology (MIT), USA, 2014.

[2] Asim Abdulkhaleq, Stefan Wagner: XSTAMPP: An Extensible STAMP Platform As Tool Support for Safety Engineering, In Proc. 2015 STAMP Conference at Massachusetts Institute of Technology (MIT).

[3] Nancy G. Leveson, Engineering a Safer World, Systems Thinking Applied to Safety, MIT press, 2012.

[4] J. Thomas, Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis. Technical Report SAND2012-4080, Sandia National Laboratories, 2012.

[5] R. Kuhn, ACTS Users Guide, 2009.

[6] R. Kuhn, Combinatorial Coverage Measurement, 2010.

[7] R. Kuhn, Introduction Combinatorial Testing, 2011.

[8] S. Wagner, Vorlesungsfolien "Sichere und zuverlässige Softwaresysteme", Wintersemester 2014/15

[9] B. Daum, Rich-Client-Entwicklung mit Eclipse 3.3, Anwendungen entwickeln mit Eclipse RCP, SWT, Forms, GEF, BIRT, JPA u.a.m.
, 2007

[10] R, Ebert, Eclipse RCP, Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client Plattform 3.7, 2011

[11] A, Abdulkhaleq, S. Wagner, Integrated STPA and Software Modeling, Software Engineering Group, Institute of Software Technology University of Stuttgart, Germany March, 2015

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Ort, Datum, Unterschrift)