Diplomarbeit

Entwicklung und Umsetzung der graphischen Nutzerschnittstelle für das Projekt SALT

Malte Wienecke
DLR Simulations- und Softwaretechnik, BA Mannheim

DLR

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Entwicklung und Umsetzung der grafischen Nutzerschnittstelle (GUI) für das Projekt SALT

DIPLOMARBEIT

für die Prüfung zum

Diplom-Ingenieur (Berufsakademie)

der Fachrichtung Informationstechnik
an der Berufsakademie Mannheim

von

Malte Wienecke

September 2006

Bearbeitungszeitraum	**3 Monate**
Kurs	**TIT03ANS**
Ausbildungsfirma	**Deutsches Zentrum für Luft- und Raumfahrt e.V. Köln-Porz**
Gutachter der Ausbildungsfirma	**Andreas Schreiber**
Gutachter der Studienakademie	**Prof. Dr. Rainer Colgen**

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich meine Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Malte Wienecke Köln, den 22. September 2006

Vorwort

Diese Diplomarbeit stellt den Abschluss meines Studiums zum Diplom-Ingenieur (BA) für Informationstechnik an der Berufsakademie Mannheim dar. Die Praxisphasen des dualen Studiums absolvierte ich in der Einrichtung für Simulations- und Softwaretechnik des Deutschen Zentrums für Luft- und Raumfahrt e.V. (DLR) in Köln-Porz.
Diese Arbeit beschäftigt sich mit der Erweiterung einer Software, die zur Betrachtung der Informationsflüsse innerhalb eines Systems verwendet wird. Während dieser Betrachtung werden die sicherheitsrelevanten Komponenten ermittelt. Bei der entwickelten Erweiterung handelt es sich um die Visualisierung der Elemente des Systems in Form eines gerichteten Graphen. Außerdem fällt unter dieser Visualisierung noch die Darstellung der Analyseergebnisse.
An dieser Stelle möchte ich einen besonderen Dank an meine Kommilitonen aussprechen, mit denen ich eine einmalige Studienzeit verbringen konnte und mir in jeder Situationslage zur Seite gestanden haben. Ebenso einen herzlichen Dank an die BA Mannheim, die mir ein gutes Studium geboten hat.
Zuletzt möchte ich noch meiner gesamten Abteilung danken, für eine tolle Zeit, die ich mit ihnen verbringen durfte und für die heiteren sowie die lehrreichen Momente während der Praxisphasen. Vielen Dank auch für das einzigartige Arbeitsklima, dass ich sehr vermissen werde.

„Entwicklung und Umsetzung der grafischen Nutzerschnittstelle (GUI) für das Projekt SALT“

Name: **Wienecke, Malte**
Matrikelnummer: **163616**
Kurs: **TIT03ANS**
Firma: **Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)**
Einrichtung für Simulations- und Softwaretechnik
Köln-Porz

Das Programm SALT wurde im Auftrag des Instituts für Verkehrsführung und Fahrzeugsteuerung entwickelt. Es wird verwendet um den Informationsfluss innerhalb eines geschlossenen Systems, zum Beispiel eines Fahrzeugs, zu betrachten und das Sicherheitsniveau der einzelnen Komponenten zu bestimmen.

Ziel dieser Arbeit ist das Programm entsprechend zu erweitern, so dass eine Visualisierung der Analyseergebnisse möglich ist. Außerdem sollen in dieser generierten Darstellung die kritischen Pfade erkennbar sein. Ein zusätzliches Ziel ist die Darstellung des Projektaufbaus mit den dazugehörigen Zusammenhängen, bereits während der Erstellung eines Projektes.

Da es sich um die Darstellung eines Informationsflusses handelt, wurde festgelegt, dass die Visualisierung in Form eines gerichteten Graphen erfolgen sollte. Nach der Analyse des Problems und der Aufstellung der Anforderung müssen die Klassen zur Generierung des Graphen erzeugt werden. Außerdem wird eine Methodik entwickelt, die nicht nur das Darstellen, sondern auch das Extrahieren von Informationen aus dem Graphen ermöglicht (s. Abbildung).

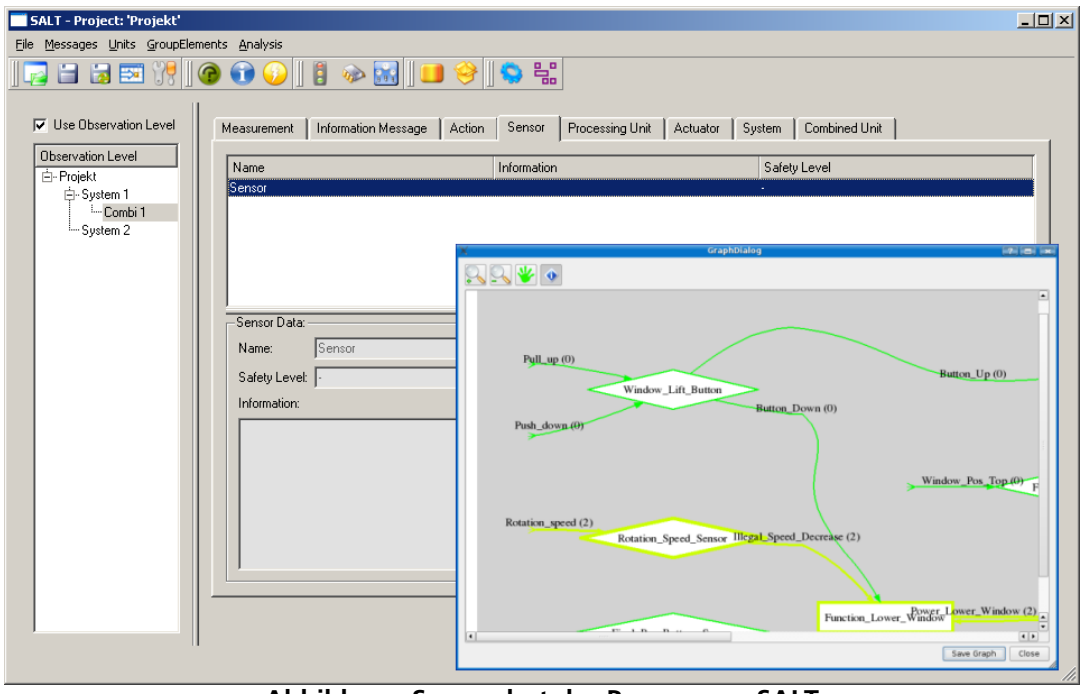


Abbildung: Screenshot des Programms SALT

„Entwicklung und Umsetzung der grafischen Nutzerschnittstelle (GUI) für das Projekt SALT“[1]

Name:	**Wienecke, Malte**
Matriculation number:	**163616**
Course:	**TIT03ANS**
Company:	**German Aerospace Center (DLR)** **Simulation and Software Technology** **Cologne**

The program SALT has been developed for the Institute of Traffic Management and Vehicle Control at the German Aerospace Center. It is used to analyse the flow of information inside closed systems, for example in vehicles, in order to identify the safety level of each component.

The aim of this thesis is to extend the functionality of the existing program with methods to visualise the results and to represent safety relevant components in a suitable way.

Based on the results of the analysis, the decision was made, that the flows of information should be represented by directed graphs. After analysing the complete problem and the different tasks, the classes for the generation of these directed graphs had to be implemented. A method has to be developed to display the generated graphs. In addition an access to detailed information about the items already shown in the graph should be possible.

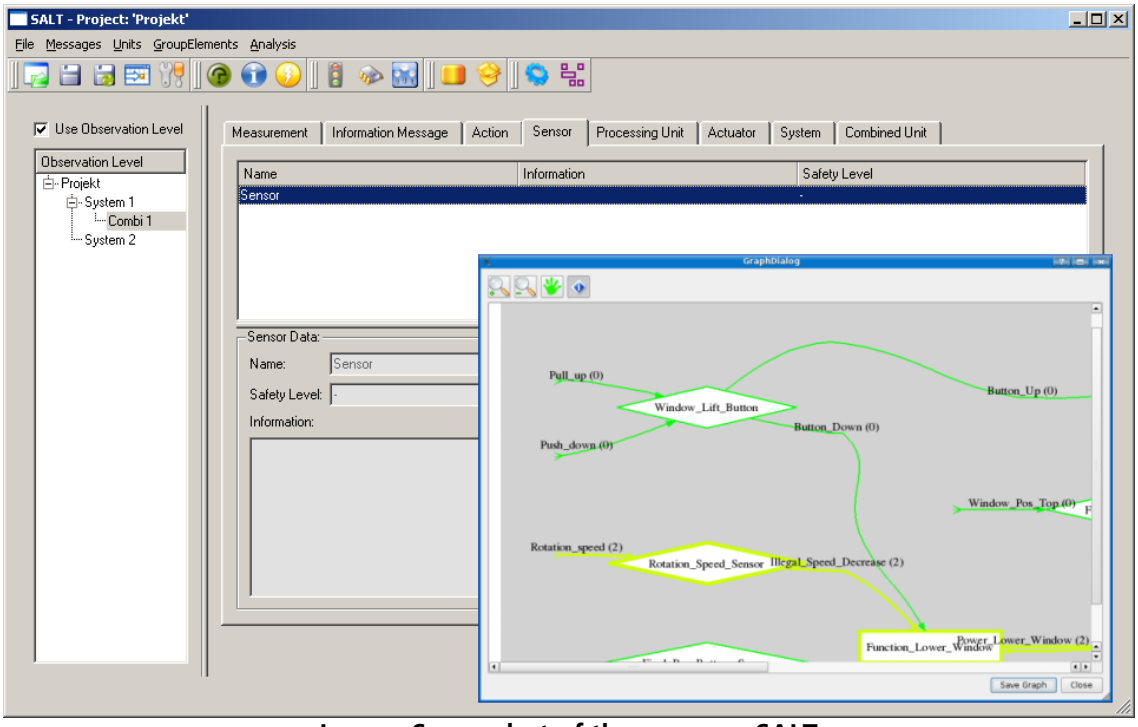


Image: Screenshot of the program SALT

[1] „Development and implementation of the graphical user interface (GUI) for the project SALT“

A. Inhaltsverzeichnis

B. Verzeichnisse

B.1. Abbildungsverzeichnis

B.2. Tabellenverzeichnis

B.3. Abkürzungsverzeichnis

Abkürzung	Erklärung
API	Application Programming Interface
DB	Datenbank
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
EN	Europäische Normung
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IEC	International Electrotechnical Commission
SALT	Safety Allovcation Tool
SIL	Safety Indication Level
XML	Extensible Markup Language

1. Einleitung

> *„Es ist Freitag. Im Zug stehen die Leute dicht aneinander gedrängt und warten, dass der Zug endlich abfährt. Das Pfeifen ertönt und die Türen schließen sich. Doch im letzten Moment springt noch ein Passant in die Tür. Das Bein hängt fest, doch die Tür öffnet sich wieder."*

Wer kennt diese Situation nicht? Nicht auszudenken, was passiert wäre wenn irgendein Mechanismus innerhalb der Tür nicht funktionieren und sie deshalb nicht öffnen würde. Aus diesem Grund ist es sehr wichtig die Elemente zu ermitteln, die diese Aktion auslöst. Versagt eines dieser Elemente, besteht die Möglichkeit, dass Personen zu Schaden kommen. Doch welche Elemente beeinflussen diese Aktion und sei es nur auf trivialste Weise?
Die Einrichtung für Simulations- und Softwaretechnik des Deutschen Zentrums für Luft- und Raumfahrt e.V. (DLR) hat im Auftrag des Instituts für Verkehrsführung und Fahrzeugsteuerung ein Programm namens SALT entwickelt. Es ist dafür konzipiert, sicherheitsrelevanten Elemente innerhalb eines Systems, zum Beispiel eines Fahrzeuges, zu ermitteln und ihnen ein entsprechendes Sicherheitslevel zuzuweisen. Mit Hilfe dieser Sicherheitslevel wird gekennzeichnet, wie fatal ein Versagen des entsprechenden Elementes wäre.
Die Aufgabenstellung dieser Diplomarbeit wurde grob unter dem Titel „Entwicklung und Umsetzung der grafischen Nutzerschnittstelle für das Projekt SALT" zusammengefasst.
Die Aufgabenstellung besteht hauptsächlich darin, die Analyseergebnisse und die daraus resultierenden Zusammenhänge in geeigneter Weise zu visualisieren.
Bei dieser Darstellung sollen vor allem die Informationsflüsse hervorgehoben werden, die das Sicherheitsniveau innerhalb eines Systems oder Subsystems bestimmen oder maßgeblich beeinflussen. Diese Informationsflüsse werden auch „kritische Pfade" genannt.
Eine weitere Teilaufgabe ist die Darstellung des Systemaufbaus bereits während der Dateneingabe, so dass Zusammenhänge schon während des Anlegens erkannt und bei Fehlern angepasst werden können.
In den folgenden Kapiteln werden die einzelnen Schritte dargestellt, die zur Entwicklung der Lösung dieser Aufgaben geführt haben.
In dem zweiten Kapitel wird das Programm SALT vorgestellt und die Methodik an Hand eines Beispieles erklärt. Außerdem wird auf verschiedene Programm-erweiterungen eingegangen, die im Laufe der Diplomarbeit zusätzlich bearbeitet werden mussten.
Danach wird das Vorgehensmodell vorgestellt, nach dem diese Aufgabenstellung bearbeitet wurde.
Das Kapitel der Analyse befasst sich kurz mit den Erkenntnissen, die während der Analysephase gewonnen wurden. Danach folgt die Auswahl des Toolkits, das für die Generierung der Graphen genutzt werden sollte.
Im Kapitel Design und Umsetzung werden, basierend auf den Ergebnissen der Analysephase, die Maßnahmen vorgestellt, die zur Lösung des Problems führen.
Die Aspekte der Qualitätssicherung, an denen sich diese Arbeit orientiert, sind in dem dafür vorgesehenen Kapitel festgehalten.
Schließlich erfolgt eine kurze Zusammenfassung der erreichten Ziele und ein Ausblick auf die Weiterentwicklung des Programms SALT.

2. SALT

In diesem Kapitel wird kurz erklärt, was SALT ist und wozu es gebraucht wird. Außerdem wird die Methodik, mit der die Applikation ihre Analyse durchführt, erläutert und an einem Beispiel angewandt. Schließlich wird noch der Entwicklungsstand zu Beginn dieser Arbeit festgehalten und die nötigen Programmerweiterungen, die im Vorfeld noch bearbeitet werden mussten, beschrieben.

2.1. Einführung in SALT

SALT oder „**S**afety **AL**location **T**ool" ist ein Programm, das im Auftrag des Instituts für Verkehrsführung und Fahrzeugsteuerung des Deutschen Zentrums für Luft- und Raumfahrt entwickelt wurde. Es wird genutzt, um den Informationsfluss innerhalb eines Projektes zu betrachten und das Sicherheitslevel oder den SIL-Wert (SIL = „Safety Indication Level") der einzelnen Elemente zu bestimmen. Als Projekt versteht man in diesem Fall ein Gesamtsystem, wie zum Beispiel ein Fahrzeug.
Dieser SIL-Wert wurde in der Norm IEC/EN 61508 [7] eingeführt und dient als Kenngröße der funktionalen Sicherheit. Ein Wert von 4 stellt dabei den höchsten Wert dar. Ein wert von SIL 1 entsprechend den niedrigsten. Ein Wert von 0 bedeutet, dass keine Sicherheitsrelevanz vorliegt.

Sicherheits-Integritätslevel	Ausfälle pro Stunde
4	10^{-9} - 10^{-8}
3	10^{-8} - 10^{-7}
2	10^{-7} - 10^{-6}
1	10^{-6} - 10^{-5}

Tabelle 1: SIL-Zuordnung bei hoher oder kontinuierlicher Anforderungsrate [22]

In Tabelle 1 ist ein Zuordnungsschema für die Zuweisung des SIL Wertes dargestellt. Hierbei untersucht man eine Komponente, die unter hoher oder kontinuierlicher Anforderung steht, und betrachtet die Ausfallrate pro Stunde. Je geringer diese Rate ist, umso höher ist das Sicherheitslevel, welches der Komponente zugeordnet werden kann. So können zum Beispiel Komponenten, mit einer hohen Ausfallrate nicht für Hochsicherheitssysteme eingesetzt werden.
Um die verschiedenen Komponenten des Projektes aus der Realität, zum Beispiel die Motoren zum Öffnen einer Tür, in das Programm zu übertragen, werden sie vereinfacht und in drei Arten unterteilt, die Nachrichten (Messages), die Einheiten (Units) und die Gruppenelemente (Group Elements).

Die **Nachrichten** bilden die Informationszufuhr der Einheiten. Dabei unterscheidet man drei verschiedene Nachrichtentypen:

- **Messwerte**:
  Dieser Nachrichtentyp spiegelt reelle Messdaten, zum Beispiel die Temperatur, in der Applikation wieder. Sie ermöglichen die Aufnahme von Informationen in das betrachtete System und besitzen aus diesem Grund kein Quellelement. Als Ziel kommen nur Sensoren in Frage, da die Informationen der Messwerte nur von ihnen verarbeitet werden können.
- **Informationsnachrichten**:
  Informationsnachrichten dienen zum Informationsaustausch zwischen verschiedenen Einheiten. Sie besitzen sowohl eine Quelle als auch mindestens ein Ziel. Außerdem kann dieser Nachrichtentyp von Verarbeitungseinheiten durchgeschleust werden, so genantes „Pipelining". Bei diesem „Pipelining" werden die Informationen, ohne bearbeitet zu werden, unter den gleichen Namen weitergeleitet. Die durchgeschleusten Informationen haben allerdings Auswirkung auf das Sicherheitslevel der entsprechenden Verarbeitungseinheit.
- **Aktionen**:
  Diese Nachrichtenart symbolisiert die vom System ausgelösten Aktionen, wie zum Beispiel das Schließen einer Tür oder das Aktivieren eines Signals. Sie können nur von bestimmten Einheiten ausgelöst werden, den so genannten Aktuatoren. Da dieser Nachrichtentyp nur Auswirkungen auf die Umwelt hat, ist es in SALT nicht möglich den Aktionen ein Ziel zu zuweisen.

Bei den **Einheiten** handelt es sich um Elemente, die eingehende Informationen verarbeiten und die Ergebnisse als ausgehende Nachrichten weitergeben. Diese Verarbeitung ist in SALT nur als Verknüpfung dargestellt. Es besteht die Möglichkeit diese Verknüpfungen mittels einer Limitierung einzuschränken. Das hat zur Folge, dass im Zuge einer Analyse nur Anfragen bearbeitet werden mit einem kleineren Sicherheitslevel als die Limitierung. Der Rest wird zunächst auf das Niveau der Beschränkung reduziert und dann verarbeitet.
Man unterscheidet zwischen drei verschiedenen Arten von Einheiten:

- **Sensor**:
  Diese Einheitsart besitzt die Fähigkeit Messwerte aufzunehmen und zu bearbeiten. Außerdem können auch Informationsnachrichten verarbeitet werden.
- **Verarbeitungseinheit**:
  Die Verarbeitungseinheiten werden verwendet, um Informationsnachrichten zu bearbeiten. Sie Besitzen außerdem die Fähigkeit des „Pipelining", die bereits oben bei den Informationsnachrichten beschrieben wurde.
- **Aktuator**:
  Bei den Aktuatoren handelt es sich um Einheiten, die Aktionen ausführen. Außerdem können auch Informationsnachrichten verarbeitet werden. Typische Aktuatoren sind zum Beispiel Motoren oder Signalgeber.

Die dritte Art von Elementen sind die **Gruppenelemente**. Sie können Einheiten und Nachrichten zusammenfassen. Die Gruppenelemente unterscheidet man in **Systeme** und **Vereinigungen**. Der Unterschied dieser beiden Typen ist, dass Systeme nur zur groben Strukturierung genutzt werden und Vereinigungen auch verschachtelt oder in Systemen vorkommen können.

2.2. Die Methodik von Salt

Das Verfahren, mit dem SALT die Sicherheitslevel den einzelnen Elementen zuordnet, ist eine vom Institut für Verkehrsführung und Fahrzeugsteuerung patentierte Methode. Sie betrachtet die einzelnen Informationsflüsse, die zur Auslösung der Aktionen innerhalb eines Projektes führen.
Grundvorrausetzung einer Analyse ist, dass allen Aktionen ein Sicherheitsniveau zugeordnet wurde. Außerdem müssen bei jeder Einheit innerhalb eines Projektes die Verknüpfungen für die ein- und ausgehenden Nachrichten gesetzt sein.
Die Betrachtung beginnt bei den Aktionen. Dafür werden sie nach dem Sicherheitslevel sortiert und iterativ abgearbeitet. Dabei wird der Nachrichtenfluss der einzelnen Aktion schrittweise zurückverfolgt.
Im ersten Schritt wird die Quelle oder Auslöser der Nachricht betrachtet. Bei dieser Einheit werden alle Verknüpfungen der jeweiligen Nachricht ermittelt. Für jede ermittelte Verknüpfung wird außerdem das Sicherheitslevel festgehalten, mit dem der Analysepfad weiter berechnet wird. Dabei werden die Limitierungen der einzelnen Verknüpfungen berücksichtigt.
Im nächsten Schritt betrachtet man die eingehenden Nachrichten der einzelnen Verknüpfungen. Dabei wird überprüft, ob die jeweilige Nachricht schon bearbeitet wurde und ein Sicherheitsniveau aufweist, das größer oder gleich dem gespeicherten Sicherheitswert ist. Wenn dieser Fall zutrifft, muss der entsprechende Analysepfad der Nachricht nicht weiter bearbeitet werden. Es erfolgt nur eine Überprüfung, ob sich die Nachricht bereits auf dem aktuellen Analysepfad befindet, also ob sich eine Schleife gebildet hat.
Wurde die Nachricht jedoch noch nicht bearbeitet oder ist das Sicherheitslevel kleiner als der gespeicherte Wert, so wird der gespeicherte Sicherheitswert ihr zugewiesen und die Analyse mit dieser Nachricht fortgesetzt.
Dieser Vorgang wiederholt sich für jede Nachricht bis zum Erreichen eines Messwertes. Wurden alle möglichen Pfade abgearbeitet, so kann das Sicherheitslevel auch für die Einheiten und Gruppenelemente berechnet werden. Bei Einheiten entspricht dieser Wert dem höchsten Sicherheitsniveau aller angrenzenden Nachrichten. Bei Verarbeitungseinheiten werden außerdem noch die durchgeschleusten Nachrichten mit einbezogen. Gruppenelemente erhalten den höchsten SIL-Wert ihrer untergeordneten Elemente.
Der Vorteil einer Analyse mit der hier beschriebenen Methodik besteht hauptsächlich in der enormen Verringerung des Zeitaufwandes im Gegensatz zu der herkömmlichen Analysemethodik. Der Zeitgewinn lässt sich auf die von hinten nach vorne ablaufende Betrachtung des Informationsflusses zurückzuführen.

2.3. Beispiel des Fensterhebers

Um die beschriebene Methodik besser zu verstehen wird der Ablauf an einem Beispiel näher erläutert. Ein einfaches Beispiel für SALT, auf das im Laufe dieser

Arbeit noch häufiger zurückgegriffen wird, ist ein einfacher elektronischer Fensterheber mit einer Einklemmsicherung. An dieser aus dem Leben gegriffenen Anwendung können die meisten Sachverhalte einfach erklärt und demonstriert werden.
Die Informationsflüsse dieses Beispiels sind in Abbildung 1 schematisch dargestellt. Zu erkennen sind außerdem die verschiedenen Einheiten, die im Folgenden näher erläutert werden.

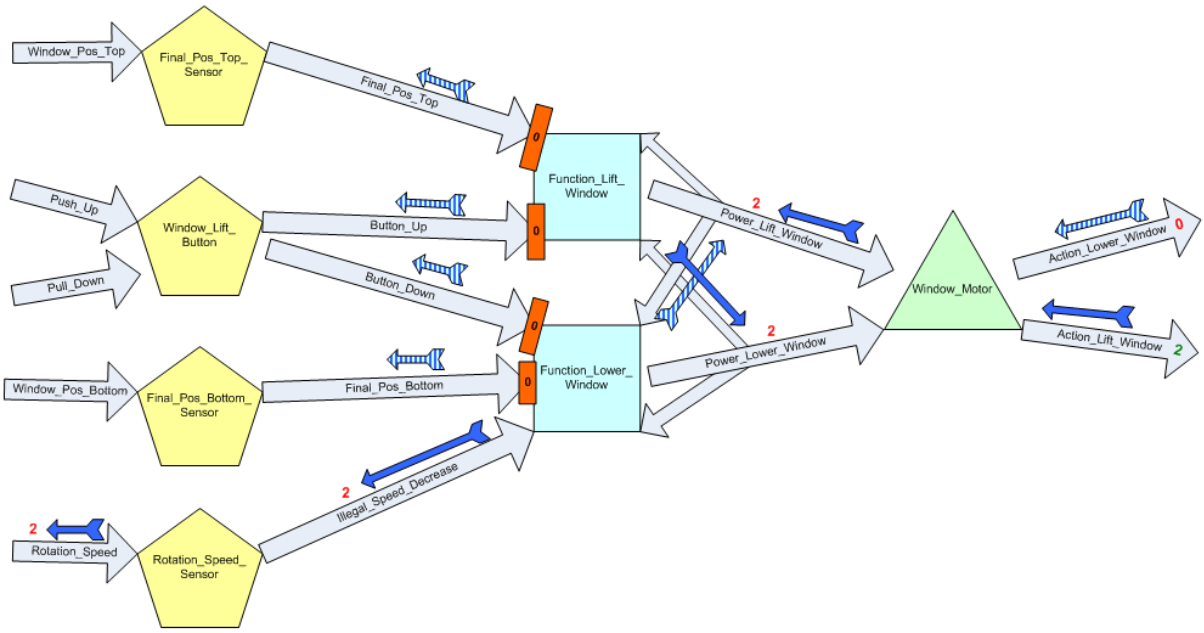


Abbildung 1: Darstellung der Informationsflüsse eines Fensterhebers

Der hier vorgestellte Fensterheber weist die verschiedenen Einheitsarten aus. Wobei der Aktuator als Dreieck, die Verarbeitungseinheiten als Vierecke und die Sensoren als Fünfecke dargestellt sind.
Die Messwerte werden von den vier verschiedenen Sensoren verarbeitet. Dabei handelt es sich um folgende Einheiten:

- **Final_Pos_Bottom_Sensor**:
  Dieser Sensor überprüft, ob die Fensterscheibe die untere Endlage erreicht hat. Wurde diese Lage erreicht, erhält der Sensor den Messwert „Window_Pos_Bottom“ und gibt mit der Informationsnachricht „Final_Pos_Bottom“ die entsprechende Information weiter.
- **Final_Pos_Top_Sensor**:
  Der „Final_Pos_Top_Sensor“ ermittelt, ob das Fenster vollständig geschlossen ist, also die obere Endlage erreicht wurde. Dabei verarbeitet er den Messwert „Window_Pos_Top“ und stößt, falls die Bedingung zutrifft, die Nachricht „Final_Pos_Top“ an.
- **Window_Lift_Button**:
  Bei diesem Sensor handelt es sich um den eigentlichen Knopf zum Heben und Senken des Fensters. Dieser Knopf kann entweder nach oben („Push_Up“) oder nach unten („Push_Down“) gedrückt werden. Durch das nach oben Drücken wird die Nachricht „Button_Up“ ausgelöst. Sie vermittelt, dass der Knopf betätigt wird und der Wunsch zum Schließen der

Scheibe besteht. Wird der Knopf wiederum nach untern geschoben, erfolgt die Weitergabe der Nachricht „Button_Down“.

- **Rotation_Speed_Sensor**:
  Dieser Sensor überprüft fortlaufend die Umdrehungsgeschwindigkeit („Rotation_Speed“) des Fensterhebermotors. Wird dieser Motor durch etwas abgebremst, verringert sich die Umdrehungsgeschwindigkeit und es erfolgt die Auslösung der Nachricht „Illegal_Speed_Decrease“. Dieser Mechanismus dient als Sicherung, damit festgestellt werden kann, ob etwas vom Fenster eingeklemmt wird.

Die Logik zum Heben und Senken der Scheibe beinhalten die beiden Verarbeitungseinheiten. Diese Logik wird, wie bereits erwähnt, innerhalb von SALT nur vereinfacht als Verknüpfungen dargestellt. Die beiden hier verwendeten Verarbeitungseinheiten sind:

- **Function_Lift_Window**:
  Diese Verarbeitungseinheit erhält die Logik, wann ein Scheibe gehoben wird und wann nicht. Vereinfacht dargestellt handelt es sich um die Verknüpfung der eingehenden Nachrichten „Final_Pos_Top“, „Button_Up“, „Power_Lift_Window“ und Power_Lower_Window“ und der ausgehenden Nachricht „Power_Lift_Window“. Zu erkennen ist, dass diese ausgehende Nachricht unter anderem auf ihre Quelle zurückweist und somit auch als eingehende Nachricht gilt. Außerdem sind die Verknüpfungen mit den Nachrichten „Final_Pos_Top“ und „Button_Up“ auf einem Sicherheitslevel von 0 eingeschränkt, da davon ausgegangen wird, dass die Elemente, die zum Auslösen dieser Nachrichten beitragen, nicht sicherheitsrelevant sind.
- **Function_Lower_Window**:
  Die zweite Verarbeitungseinheit befasst sich mit dem Senken der Scheibe. Die eingehenden Nachrichten „Button_Down“, „Final_Pos_Bottom“, „Illegal_Speed_Decrease“, „Power_Lift_Window“ und „Power_Lower_Window“ sind mit der ausgehenden Nachricht „Power_Lower_Window“ verknüpft, die, wie man erkennt auch auf ihre Quelle zurückweist. Es existieren wie in der anderen Verarbeitungseinheit Schranken für die Nachrichten „Button_Down“ und „Final_Pos_Bottom“.

Der einzige Aktuator in diesem Beispiel ist der „**Window_Motor**“. Er beschreibt den Motor, der das Fenster hebt („Action_Lift_Window“) und senkt („Action_Lower_Window“). Um das Öffnen und Schließen des Fensters einzuleiten, benötigt der Motor die Informationen „Power_Lower_Window“, bzw. „Power_Lift_Window“.
Um die Analyse durchführen zu können müssen den Aktionen Sicherheitslevel zugewiesen werden. Die Aktion „Action_Lift_Window“ bekommt ein SIL Wert von 2 zugeordnet, da Personen leicht bis schwer verletzt werden können, wenn das Schließen der Scheibe im falschen Moment stattfindet. Dieses falsche Schließen erfolgt etwa 10^{-6} bis 10^{-7} Mal in einer Stunde. Dieser Wert ergibt nach Tabelle 1ein SIL Wert von 2.

Das Öffnen des Fensters, also die Aktion „Action_Lower_Window“, wird als nicht sicherheitsrelevant eingestuft, da beim Ausführen dieser Aktion keine Person zuschaden kommen kann.
Das Anwenden der Methodik wird im Folgenden an Hand eines Analysepfades näher erläutert. Dieser Analysepfad ist in der Abbildung mit blauen Pfeilen gekennzeichnet.
Begonnen wird bei der Aktion mit dem höchsten SIL-Wert, also bei der Aktion „Action_Lift_Window“.
Im ersten Schritt wird der Auslöser dieser Aktion ermittelt, der „Window_Motor“. Er weist nur eine Verknüpfung für die jeweilige Nachricht auf, die mit der Informationsnachricht „Power_Lift_Window“. Da für die Verknüpfung keine Limitierung vorliegt, muss die Betrachtung mit einem Sicherheitswert von 2 weitergeführt werden.
Im nächsten Schritt wird überprüft, ob die Nachricht „Power_Lift_Window“ schon bearbeitet wurde. Da dies nicht der Fall ist, wird der Informationsnachricht das Sicherheitsniveau 2 zugewiesen und die Analyse mit dieser Nachricht fortgesetzt.
Als nächstes wird wieder der Auslöser der zu betrachteten Nachricht bestimmt. Dabei handelt es sich um die Verarbeitungseinheit „Function_Lift_Window“. Sie weist eine Vielzahl von Verknüpfungen für die entsprechende Nachricht auf. Es wird jedoch erst die Verknüpfung mit der Nachricht „Power_Lower_Window“ bearbeitet. Die anderen Verknüpfungen werden zur späteren Abarbeitung markiert. Das wird in der Abbildung mit den gestreiften Pfeilen dargestellt.
Da die Nachricht „Power_Lower_Window“ auch noch nicht bearbeitet wurde, wird ihr der SIL Wert 2 zugewiesen und mit ihr die Analyse weitergeführt.
Es erfolgt wieder die Bestimmung des Auslösers der Nachricht. Dabei handelt es sich um die Verarbeitungseinheit „Function_Lower_Window“. Sie weist verschiedene Verknüpfungen mit der Nachricht „Power_Lower_Window“ auf, doch nur die mit der Nachricht „Illegal_Speed_Decrease“ wird zur weiteren Analyse verwendet. Die anderen werden wiederum zur späteren Abarbeitung markiert.
Da auch hier keine Limitierung vorhanden ist und die Nachricht „Illegal_Speed_Decrease“ noch nicht bearbeitet wurde, wird ihr das Sicherheitslevel 2 zugewiesen. Die Quelleinheit dieser Nachricht ist der Sensor „Rotation_Speed_Sensor“. Er besitzt nur eine nicht eingeschränkte Verknüpfung der Nachricht mit den Messwert „Rotation_Speed“. Weil diese Nachricht noch nicht bearbeitet wurde wird ihr der entsprechende SIL Wert von 2 zugewiesen. Da es sich jedoch um einen Messwert handelt, endet hier die Betrachtung des Analysepfades.
Danach erfolgt die Abarbeitung der markierten Nachrichten. Falls keine ausstehenden Abarbeitungspfade mehr zur Verfügung stehen, wird die Analyse mit der Betrachtung der nächsten Aktion fortgesetzt. Diese Schritte werden jedoch an dieser Stelle nicht näher erläutert, da sie nach dem gleichen, vorgestellten Schema abgearbeitet werden.
Bei der vollständigen Analyse der Aktionen ist zu erkennen, dass nur der angezeigte Pfad mit einem Sicherheitslevel von 2 versehen ist. Die anderen Pfade sind durch Limitierungen an den Verknüpfungen auf einen Wert von 0 beschränkt.

2.4. Programmumgebung von SALT

Für die Entwicklung von SALT wurde Python [21] als Programmiersprache ausgewählt, da Python nicht nur eine einfache Syntax bietet, sondern auch sehr übersichtlich ist. Aus der Erfahrung innerhalb der Abteilung für Simulations- und Softwaretechnik ist festzustellen, dass Python als leistungsstark gilt und als Standartsprache für neue Projekte genutzt wird, wenn nicht anderweitige Anforderungen dem entgegensetzen. Außerdem bietet Python die Vorteile einer modernen objektorientierten Programmiersprache, wie zum Beispiel das Zusammenschließen mehrer Klassen zu Paketen, so dass eine deutliche Abgrenzung ermöglicht wird.
SALT weist die in Abbildung 2 dargestellte 3-Schichten-Architektur auf. Bei diesem Modell kann jede Schicht nur auf die jeweils unter ihr liegende Schicht zugreifen. Das hat zur Folge, dass das Programm eine hohe Flexibilität und Wartbarkeit aufweist. Außerdem besteht durch diese Kapselung die Möglichkeit, einzelne Schichten einfach auszutauschen oder zu erweitern.

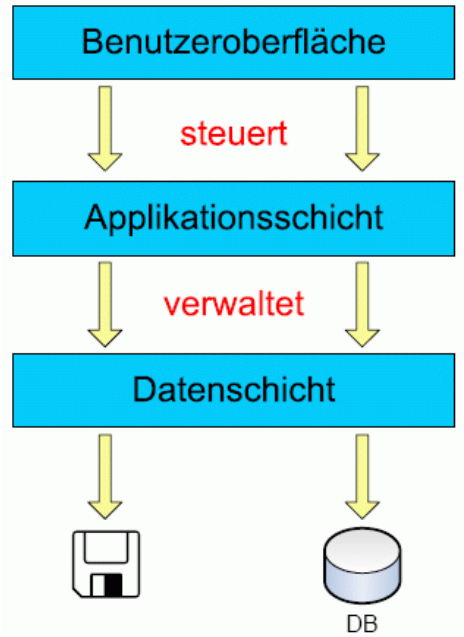


Abbildung 2: Die 3-Schichten-Architektur von SALT

Die Schicht der grafischen Benutzeroberfläche dient als Schnittstelle zwischen dem Benutzer und der eigentlichen Applikation. Sie wurde mit Hilfe des Qt-Toolkits [23] erstellt, das mit „pyQt“ [20] eine gute Verknüpfung an Python bietet.
Die mittlere Schicht beherbergt die gesamte Applikationslogik. Hier sind alle entscheidenden Algorithmen hinterlegt, wie zum Beispiel der Analysealgorithmus. Außerdem erfolgt in dieser Schicht die Verwaltung der Datenstrukturen.
In der untersten Schicht sind die Datenstrukturen angesiedelt. Außerdem befinden sich hier die Klassen, die als Schnittstelle zum eigentlichen System dienen. Sie ermöglichen den Zugriff auf die Datenbank, in denen die Projekte hinterlegt sind, und das Schreiben und Lesen von Dateien auf der Festplatte.
Bei der darunter liegenden Datenbank handelt es sich um eine MySQL Datenbank [15], die jedoch ohne weiteres ausgetauscht werden kann. Es ist lediglich eine Anpassung der Klasse notwendig, die auf die Datenbank zugreift.

2.5. Entwicklungsstand zu Beginn der Arbeit

Der Entwicklungstand zum Ausgangszeitpunkt dieser Arbeit beschränkt sich auf die minimale Funktionalität. Es besteht die Möglichkeit Projekte anzulegen, sie mit den dazugehörigen Elementen zu füllen und basierend auf diesen Eingaben die Analyse des Projektes durchzuführen. Die daraus resultierenden Ergebnisse werden jedoch nur in einer sehr simplen Form, einer Auflistung der Analysepfade, angezeigt.
Der in Abbildung 3 dargestellte Screenshot stellt für das bereits beschriebene Beispiel des Fensterhebers das Hauptfenster von SALT dar. Im unteren Bereich des Fensters ist die einfache Darstellung der Analyseergebnisse erkennbar. Die kritischen Pfade werden mit Hilfe einer farblichen Markierung neben den Analysepfaden kenntlich gemacht.
Der mittlere Bereich des Hauptfensters dient der Visualisierung der Elementdaten. Diese Darstellung ist abhängig vom jeweiligen Elementtyp. Im oberen Bereich wird das Element ausgewählt, dessen Informationen im mittleren Bereich dargestellt werden. Über die Toolbar oder das Menü können die einzelnen Funktionen aufgerufen werden, wie zum Beispiel das Erstellen von Elementen oder das Starten der Analyse.

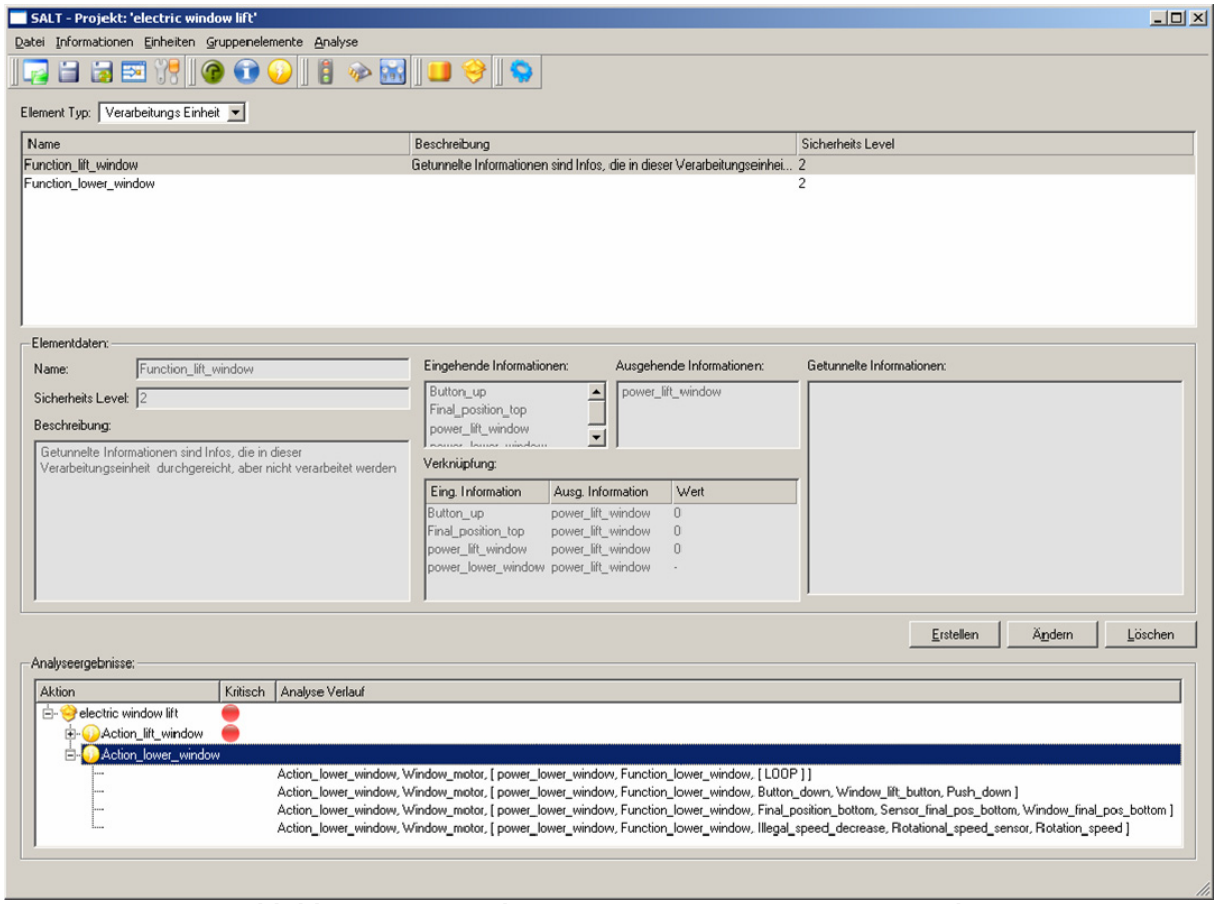


Abbildung 3: Screenshot von SALT zum Ausgangszeitpunkt

2.6. Erweiterungen zum Bearbeiten der Thematik

In diesem Abschnitt wird kurz auf die verschiedenen Erweiterungen und Änderungen eingegangen, die im Hinblick auf die eigentliche Bearbeitung dieser Arbeit vollzogen werden mussten.

2.6.1. Einführung der Observationsschichten

Eine grundlegende Erweiterung ist die Einführung von isolierten Sichtebenen, den so genannten Observationsschichten. Mit Hilfe dieser Erweiterung ist es möglich Gruppenelemente unabhängig vom Rest des Projektes zu betrachten und zu bearbeiten. Das verbessert nicht nur die Übersichtlichkeit, sondern ermöglicht vor allem eine bessere Unterteilung in Unterprobleme, die separat voneinander bearbeitet und anschließend zusammengefügt werden können.
Um die verschiedenen Schwierigkeiten und Eigenschaften der Observationsschichten zu klären, wurden verschiedene Vorüberlegungen gemacht, die in einfachen Skizzen festgehalten wurden. Diese Vorüberlegungen behandeln zuerst die einfachen Sachverhalte und erhöhen schrittweise die Komplexität.
Als erstes wird davon ausgegangen, wie in Abbildung 4 dargestellt, dass Einheiten (in der Zeichnung als Punkt dargestellt) nur einem Gruppenelement (als Rechteck dargestellt) zugewiesen werden können und diese Gruppenelemente anderen Gruppenelementen unterstellt sind. Das übergeordnete Element wird auch als „Superior“ und das jeweilige untergeordnete als „Subordinate“ bezeichnet.

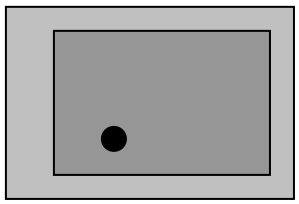


Abbildung 4: Ein Element, ein Superior

Es ist jedoch möglich, dass Einheiten wie in Abbildung 5 mehreren Gruppenelementen unterstellt sind. Hierbei ist jedoch zu beachten, dass diese Gruppenelemente den, bzw. die gleichen Superiors haben.

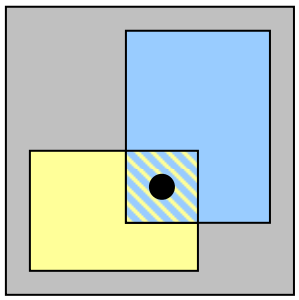


Abbildung 5: Eine Element, mehrere Superiors

Da die Observationsschichen vom Rest des Projektes isoliert betrachtet werden, können die Einheiten mit Hilfe der Nachrichten nur mit Einheiten kommunizieren, die denselben Gruppenelementen untergeordnet sind. Damit ein

gruppenelementübergreifender Informationsfluss gewährleistet ist, müssen den Gruppenelementen ein- bzw. ausgehende Nachrichten zugeordnet werden können. Diese Nachrichten sind in Abbildung 6 grün bzw. blau dargestellt.

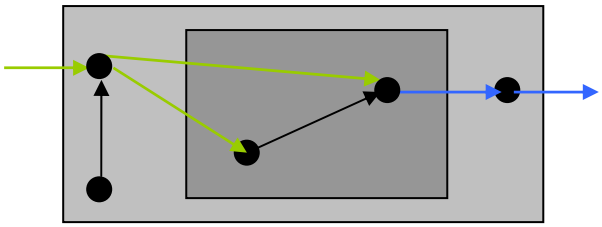


Abbildung 6: Skizze für ein- und ausgehende Nachrichten

Eine weitere Überlegung betrifft den Fall, wenn ein Gruppenelement ausgeweitet bzw. verkleinert wird und wie solche Änderungen auf die ein- oder ausgehenden Nachrichten wirken. In Abbildung 7 sind solche Änderungen beispielhaft dargestellt. Zu erkennen ist, dass bei einer Änderung des originalen Gruppenelements mit der durchgezogenen Umrandung, hin zum neuen Gruppenelement mit der gepunkteten Umrandung, sowohl ein-, also auch ausgehende Nachrichten angepasst werden müssen.

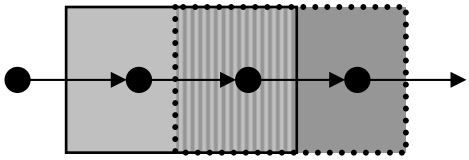


Abbildung 7: Erweitern und Verkleinern eines Gruppenelements

Neben diesen Vorüberlegungen gibt es verschiedene weitere Fragestellungen, zum Beispiel was geschieht, wenn ein Gruppenelement die Superiors ändert oder austauscht. Da die Komplexität dieses aufgezeigten Problems verhältnismäßig groß ist, wird auf eine nähere Erläuterung verzichtet.
Auf der Basis dieser Vorüberlegungen konnte die eigentliche Erweiterung beginnen. Zuerst musste sichergestellt werden, dass die ein- und ausgehenden Nachrichten für jeden Gruppenelementtyp deklariert werden können. Aus diesem Grund musste zuerst die Datenbankschicht angepasst werden, bevor die Änderung der Datenstrukturen möglich war. Durch den objektorientierten, modularen Aufbau der Datenstrukturen in SALT, wie in Abbildung 8 dargestellt, ist eine Erweiterung oder Anpassung sehr einfach. Die Vererbungshierarchie ermöglicht, dass zum Beispiel Funktionalität, die von allen Datenstrukturklassen verwendet wird, an einem zentralen Punkt, in der „Element“-Klasse, abgelegt werden kann.

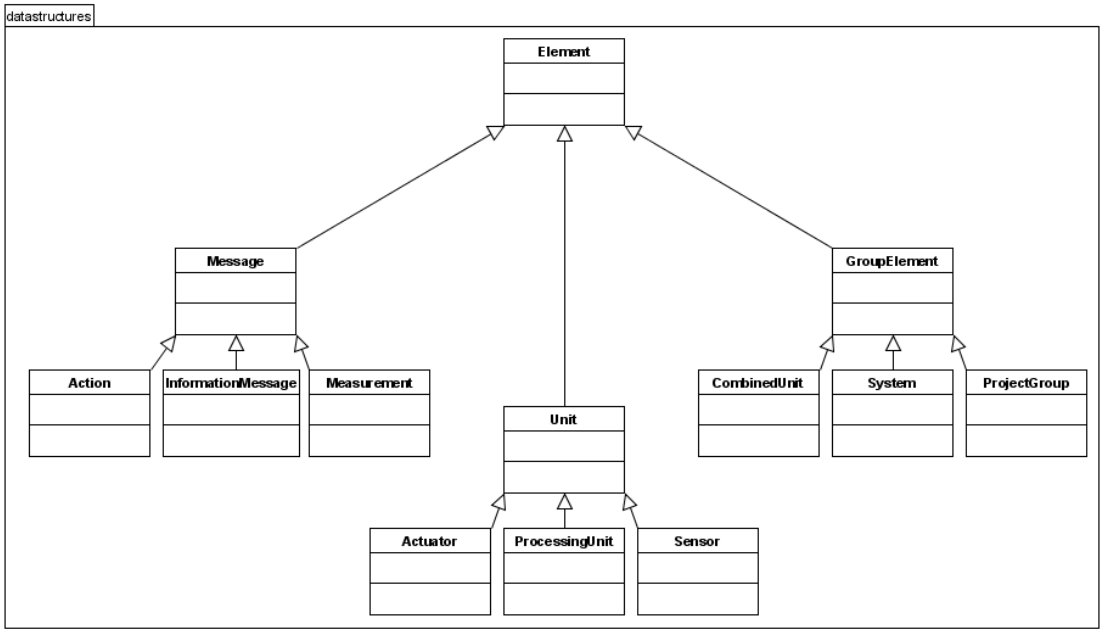


Abbildung 8:Ableitungshierarchie der Datenstrukturen

Anlässlich der Einführung der Observationsschichten musste sowohl für die Gruppenelemente, als auch für die Elemente im Allgemeinen, die Funktionalität erweitert werden. Für die Elemente handelt es sich dabei unter anderem um eine Methode, die die Superiors des Elementes ermittelt. Bei den Gruppenelementen musste vor allem die Funktionalität für die ein- und ausgehenden Nachrichten erweitert werden.
Außerdem wurde die so genannte „ProjectGroup" oder Projektgruppe (s. Abbildung 8) als neue Datenstruktur angelegt. Bei diesem Gruppenelement handelt es sich um die oberste Observationsschicht, die direkt dem Projekt zugeordnet ist. Da, wie bereits erwähnt, der Aufbau objektorientiert gehalten ist, erweist sich eine solche Erweiterung als sehr einfach. Wie in Abbildung 9 dargestellt, muss hierfür lediglich eine Klasse formuliert werden, die von der jeweiligen Gruppenklasse, in diesem Fall der „GroupElement" Klasse, abgeleitet wird. Die klassenspezifische Funktionalität, falls vorhanden, muss auch in diese Klasse eingefügt werden. Auf der beigelegten CD dieser Arbeit ist der Quellcode für die Projektgruppe hinterlegt.
Um den Vorteil der Observationsschichten nutzen zu können, musste auch die grafische Benutzeroberfläche angepasst werden. In Abbildung 9 ist daher als Beispiel das neu gestaltete Hauptfenster dargestellt.

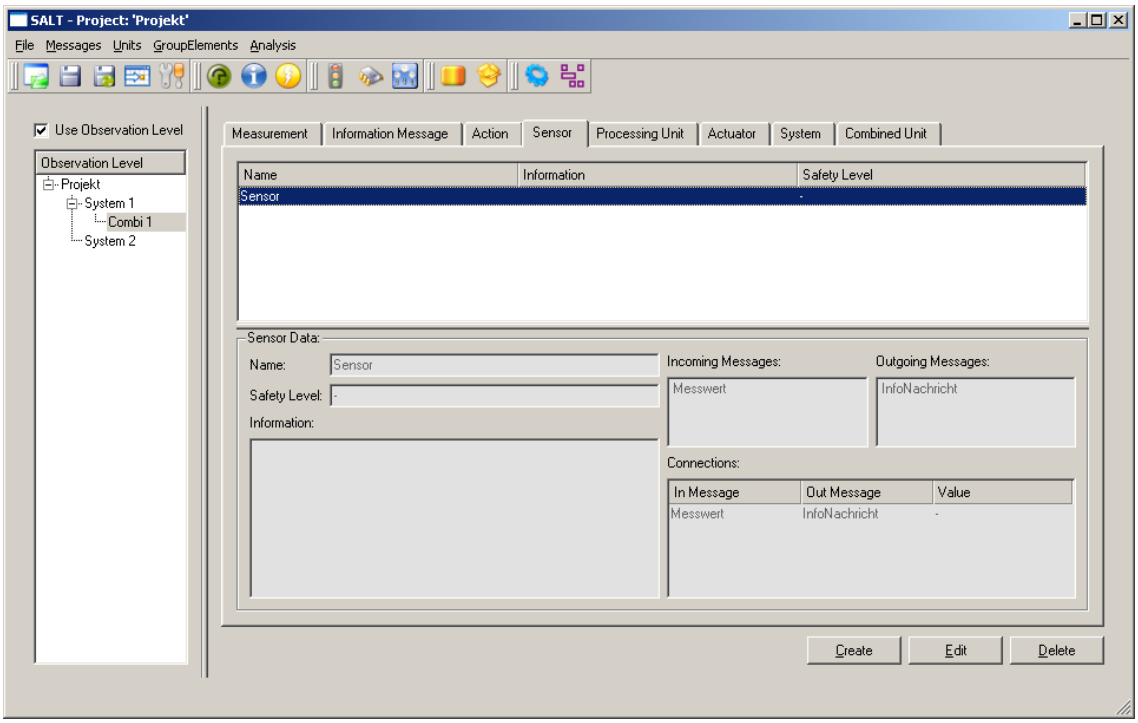


Abbildung 9: Hauptfenster nach Einführung der Observationsschichten

Am linken Rand des Fensters ist der Projektbaum dargestellt, der die Verschachtelung der Gruppenelemente anzeigt. Hier erfolgt die Auswahl des jeweiligen Gruppenelements, das isoliert betrachtet werden soll. Über den dargestellten Projektbau kann die Funktionalität der Observationsschichten deaktiviert werden. Dadurch werden nicht nur die Elemente der Observationsschicht im Fenster angezeigt sondern die alle Elemente des Projektes.
Außerdem wurde die Auswahl der Elementtypen angepasst (vergleiche Abbildung 3). Mit Hilfe des Tab-Widgets wird eine offensichtliche Möglichkeit geboten zwischen den verschiedenen Ansichten der einzelnen Elemente zu wechseln. Außerdem fördert es die Erweiterbarkeit der Applikation, da bei Hinzufügen einer Datenstruktur nur eine neue Seite dem Tab-Widget hinzugefügt werden muss.
Nach dem hier beschriebenen Vorgehen können auch andere Datenstrukturen erweitert oder neue erstellt werden. Dadurch kann SALT auf eine einfache Weise modifiziert werden, wobei die Anpassung der grafischen Benutzeroberfläche oft die meiste Zeit in Anspruch nimmt.

2.6.2. Umstellung des SALT Algorithmus

Eine weitere Änderung ist die Umstellung des rekursiven Analysealgorithmus auf ein iteratives Verfahren. Diese Umstellung war notwendig, da die mögliche Rekursionstiefe in Python limitiert ist und es bei großen Projekten durchaus vorkommen kann, dass während der Analyse eine Vielzahl von Elementen durchlaufen werden muss. Bei dieser Neugestaltung des Algorithmus wurde der eigentliche Programmstack nachgebildet. Außerdem konnte im Zuge dieser Umstellung, mit Hilfe einer verbesserten Datenbankabfrage und einer überarbeiteten Datensatzmodifikation die Performance des Analyseablaufs verbessert werden.

3. Vorgehensweise

Die Entwicklung von SALT und somit auch die Bearbeitung dieser Diplomarbeit liegen den Vorgehensweisen des Software Engineerings zu Grunde. Unter Software Engineering versteht man *„die zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Herstellung und Anwendung von umfangreichen Software-Systemen."* [1]
Die Umsetzung des Projektes SALT unterliegt einem von der Abteilung vorgegebenen Vorgehensmodell. Es ist in Abbildung 10 schematisch dargestellt.

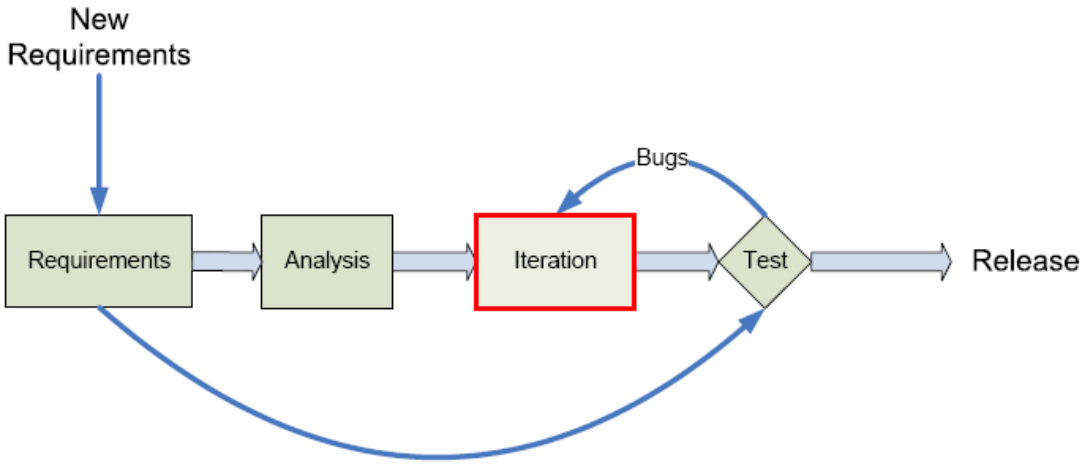


Abbildung 10: Vorgehensmodell des Projektes SALT

Dieses Modell ist für Anwendungen spezialisiert, dessen Anforderungen variabel und erweiterbar sind. Zu Beginn der Entwicklung werden die Anforderungen definiert, die an das Programm gestellt werden. Diese werden in einer Analysephase untersucht und bewertet. Basierend auf den Ergebnissen erfolgt in der Iteration die Umsetzung der Anforderungen. Anschließend muss die entwickelte Problemlösung getestet werden. Die Testfälle dafür resultieren aus der Anforderungsdefinition. Werden Fehler oder Mängel entdeckt, wird eine weitere Iteration durchlaufen. Dieser Vorgang wiederholt sich bis alle Tests erfolgreich durchgeführt wurden und keine Fehler mehr auftreten. Es existiert also ein funktionstüchtiges Programm, dass veröffentlicht wird. Das Besondere an diesem Vorgehen ist jedoch, dass während der Entwicklung neue Anforderungen definiert werden können. In diesem Fall werden die neuen Anforderungen analysiert und basierend auf dem bestehenden Programm die Erweiterungen umgesetzt. Dadurch wird eine flexible Entwicklung sichergestellt, die schnell auf mögliche Änderungen reagieren kann. Aus diesem Grund wurde auch kein klassisches Vorgehensmodell gewählt, wie zum Beispiel das „Wasserfallmodell", da auf Änderungen innerhalb der Anforderungen schlecht reagiert werden kann und somit die Dynamik und die Flexibilität während der Entwicklung zu stark eingeschränkt sind.

Im Folgenden sind die einzelnen Phasen des Vorgehensmodells näher erläutert:

- **Anforderungsdefinition:**
  In dieser Phase werden die Anforderungen definiert, die das Projekt erfüllen muss. Diese Anforderungen werden mit dem Projektpartner aufgestellt. Dabei werden die verschiedenen Anwendungsfälle besprochen und festgehalten. Basierend auf den hier gewonnenen Ergebnissen kann in der Testphase verifiziert werden, ob die gewünschte Funktionalität umgesetzt wurde.
- **Anfordeungsanalyse:**
  Während der Anforderungsanalyse werden die Ergebnisse der Anforderungsdefinition näher betrachtet und bewertet. Es erfolgt eine detaillierte Beschreibung der Usecases und der Zeitplan für die Entwicklung wird aufgestellt.
- **Iterationsphase:**
  Resultierend aus dem Zeitplan und der detaillierten Beschreibung der Anforderungen erfolgt in der Iteration die eigentliche Umsetzung der Erweiterung. Dabei erfolgt das Design und die Implementierung des zu lösendem Problems. Das hier entwickelte Programm dient als Input für die Testphase.
- **Testphase:**
  In der Testphase erfolgen die Akzeptanztests der entwickelten Software. Dabei wird das Programm mit Hilfe der Ergebnisse der Anforderungs-definition verifiziert. Werden Fehler oder Mängel festgestellt, so wird die Iterationsphase erneut durchlaufen. Ist dies nicht der Fall, so kann eine neue Version der Software veröffentlicht werden.

Die wichtigste Phase ist hierbei die Iterationsphase. In Abbildung 11 ist eine detaillierte Darstellung dieser Phase abgebildet.

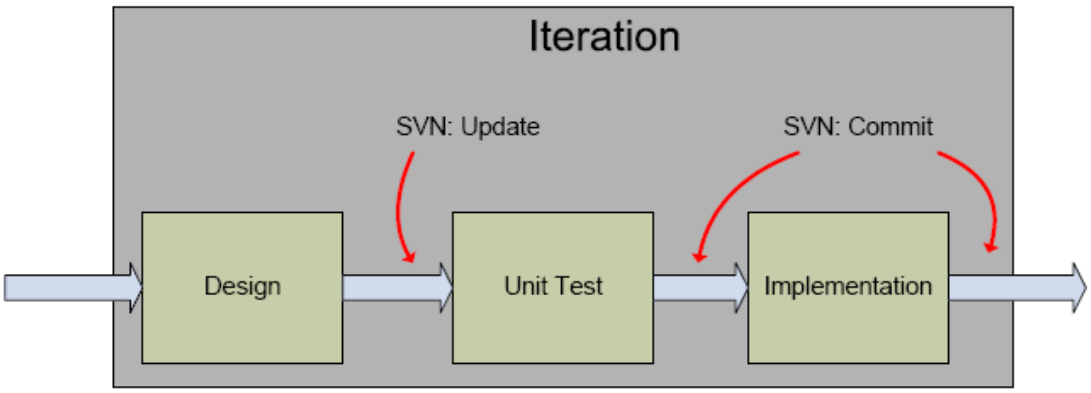


Abbildung 11: Detaillierte Darstellung der Iterationsphase

Die Iterationsphase ist in drei Abschnitte unterteilt, die bei jeder Iteration durchlaufen werden:

- **Designphase**:
  Mit der Phase beginnt jeder Iterationsschritt. Hier erfolgt, basierend auf den Ergebnissen der Analysephase, das Design des zu lösenden Problems. Dabei werden die notwendigen Klassen- und Sequenzdiagramme erstellt, die den Lösungsweg beschreiben.
- **Modultest-Definition**:
  Während der Modultest-Definition werden die Komponententests (s. Kapitel 7.1) für die entsprechende Problemstellung definiert. Dabei werden für jede zu entwickelnde Methode ein oder mehrere Unit-Tests geschrieben, die die Funktionalität der Methode verifizieren. Somit kann die Korrektheit und der Stand während der Implementierung überprüft werden.
- **Implementierung**:
  Die Implementierung beschreibt die eigentliche Umsetzung. Hierbei wird fortlaufend überprüft, ob die Funktionalität der Entwicklung mit den definierten Unit-Tests übereinstimmt. Wurde die gesamte Funktionalität umgesetzt, endet der Iterationsschritt mit dem entwickelten Programm.

Neben diesen drei Phasen ist in der Abbildung auch noch gekennzeichnet, wann die verschiedenen Interaktionen mit den Versionsverwaltungssystem (s. Kapitel 7.5) erfolgen. Dabei handelt es sich vor der Definition der Komponententests, um eine Aktualisierung des Quellcodes, auf dem die Erweiterung basiert. Nach Festlegung der Unit-Tests und nach Beendigung der Implementierung werden die Änderungen des Quelltextes mit Hilfe eines „Commits“ (to commit, eng. = „anvertrauen“) in das Versionsverwaltungssystem übertragen.

Nach dem beschriebenen Vorgehensmodell wird die Aufgabenstellung im Rahmen dieser Arbeit bearbeitet. Hierbei sind vor allem die Analyse mit der Auswahl der Graphentoolkits und das Design sowie die Umsetzung näher beschrieben.

4. Anforderungsanalyse

Vor der eigentlichen Umsetzung der Aufgabenstellung, erfolgt die Analyse der Problemstellung. Dieser Problemstellung ist die Erweiterung des bereits existierenden Programms SALT. Dabei sollen die Möglichkeit entwickelt werden die Analyseergebnisse grafisch darzustellen. Dabei soll der Informationsfluss auf einer geeigneten Weise dargestellt werden. Außerdem soll die Möglichkeit bestehen den Projektaufbau mit den dazugehörigen Zusammenhängen darzustellen.
Dabei wird eine Entscheidung berücksichtigt, die bereits im Vorfeld der Analyse getroffen wurde und zwar dass die Darstellungsweise der Informationsflüsse, am besten als gerichtete Graphen umzusetzen ist. Grund für diese Entscheidung ist die gute Darstellbarkeit der Flussrichtung und die Gewichtung der einzelnen Kanten innerhalb eines gerichteten Graphen.
Basierend auf dieser Entscheidung werden in diesem Kapitel zum einen die Usecases aufgestellt. Sie klären die einzelnen Sachverhalte, die zur Generierung und Visualisierung eines Graphen führen.
Zum anderen erfolgt die Aufstellung der Anforderungen an die jeweiligen Graphen. Sie beschreiben die Bedingungen, die der Graph erfüllen muss. Zusätzlich werden noch Anforderungen erarbeitet, die an die Erweiterung von SALT gestellt werden. Bei dieser Erweiterung handelt es sich um die Funktionalität zur Erzeugung und Darstellung der Graphen innerhalb der bereits vorhandenen Umgebung von SALT, die im Rahmen dieser Arbeit zu entwickeln war.

4.1. Usecases

Die Usecases oder Anwendungsfälle identifizieren die verschiedenen Möglichkeiten, die der Benutzer auslösen kann, um einen Graphen zu erzeugen und darzustellen.
Die drei erkannten Anwendungsfälle sind im Folgenden kurz erläutert.

4.1.1. UC010: Analysepfad darstellen

UC010: Analysepfad darstellen

Beschreibung: Basierend auf den Daten des ausgewählten Analysepfades soll ein Graph erzeugt und Dargestellt werden.

Vorbedingung: Ein Analysepfad wurde ausgewählt.

Nachbedingung: Der erzeugte Graph ist dargestellt.

Ablauf:
1. Analysepfad wird bestimmt
2. Elemente des Pfades werden ermittelt
3. Benachbarte Elemente werden bestimmt
4. Graph wird mit den bekannten Elementen erzeugt
5. Graph wird in der GUI angezeigt

Kommentar: Zu 3.:
Bei den benachbarten Elementen handelt es sicht um die Einheiten, die zwar nicht direkt auf dem Pfad liegen, aber mit beeinflusst werden, wie zum Beispiel weitere Ziele der Nachrichten.

Der erste Usecase beschreibt, wie das Visualisieren eines Analysepfades zustande kommt. Grundlegend dafür ist, dass eine Analyse durchgeführt wurde und der Benutzer einen Pfad zum Darstellen ausgewählt hat. Basierend auf diesen Daten werden die Elemente ermittelt, die angezeigt werden sollen. Außerdem werden noch die Nachbarelemente bestimmt, die davon beeinflusst werden. Danach erfolgen das Generieren und die Darstellung des Graphen.

4.1.2. UC020: Aktuelle Observationsschicht darstellen

UC020: Aktuelle Observationsschicht darstellen

Beschreibung: Ein Graph, der nur die Elemente der derzeitigen Observationsschicht enthält, soll erzeugt werden.

Vorbedingung: Es ist ein Observationslevel ausgewählt.

Nachbedingung: Der erzeugte Graph ist dargestellt.

Ablauf:
1. Observationsschicht bestimmen
2. Elemente innerhalb des Gruppenelements werden bestimmt
3. Ein- und Ausgehende Nachrichten werden betrachtet und Quellen, bzw. Ziele ermittelt.
4. Nachrichten die in oder aus einem untergeordnetem Gruppenelement weisen, werden festgehalten.
5. Graph mit den gewonnen Erkenntnissen wird erzeugt
6. Graph wird in der GUI angezeigt

Kommentar: Zu 4.:
Da keine in die Tiefe gehende Betrachtung vorgenommen wird, müssen untergeordnete Gruppenelemente wie Einheiten dargestellt werden.

Der zweite Anwendungsfall dient der Darstellung eines einzelnen Gruppenelementes, bzw. einer Observationsschicht. Dafür muss der Benutzer eine Observationsschicht auswählen, die dargestellt werden soll. Danach erfolgt die Identifikation der Elemente innerhalb des Gruppenelements sowie die Quellen und Ziele der ein- bzw. ausgehenden Nachrichten. Da es eine Visualisierung einer einzelnen Observationsschicht ist, die nicht in die Tiefe geht, müssen alle Nachrichten erkannt werden die zu oder von Elementen aus untergeordneten Gruppenelementen kommen. Für diese Nachrichten gilt das Gruppenelement als Auslöser bzw. Ziel in der Darstellung. Anschließend kann der Graph erzeugt und angezeigt werden.

4.1.3. UC030: Gesamtes Projekt darstellen

UC030: Gesamtes Projekt darstellen

Beschreibung: Ein Graph soll erzeugt werden, der das gesamte Projekt mit allen Elementen enthält.
Vorbedingung: Das Projekt muss vorhanden sein.
Nachbedingung: Der erzeugte Graph ist dargestellt.
Ablauf:
1. Alle Elemente des Projektes werden ermittelt
2. Einordnung der einzelnen Elemente in die dazugehörigen Gruppenelemente
3. Erzeugung des Graphen
4. Graph wird in der GUI angezeigt

Der dritte und letzte Usecase befasst sich mit der Darstellung eines kompletten Projektes. Dabei werden alle Elemente dargestellt. Sie werden den jeweiligen Gruppenelementen zugeordnet und dann in einem Graphen festgehalten, der anschließend angezeigt wird.

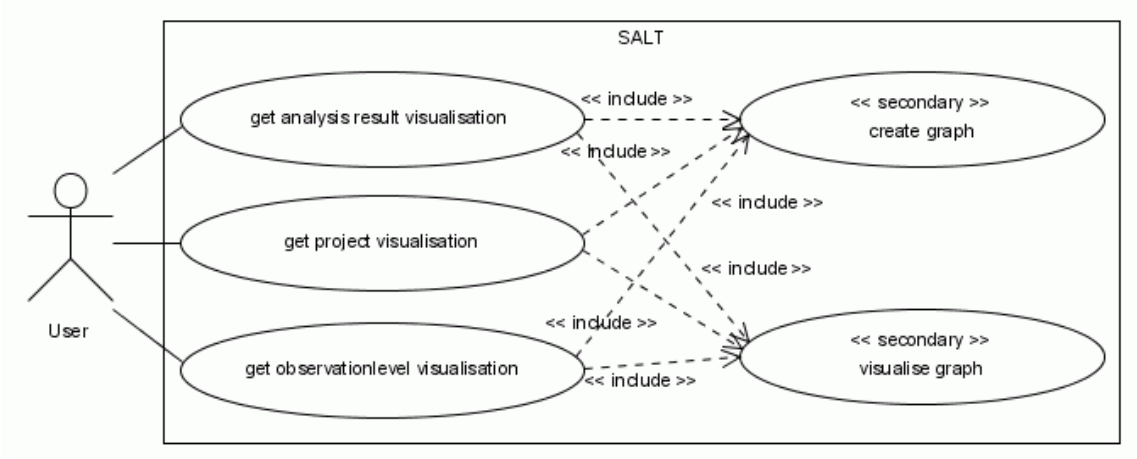


Abbildung 12: Usecasediagramm

Bei näherer Betrachtung dieser drei Anwendungsfälle, erkennt man, dass die Generierung und Visualisierung der Graphen bei allen Fällen vorkommen. Aus diesem Grund sind die Erzeugung und Darstellung als Secondary-Usecases, wie in Abbildung 12 gezeigt, extrahiert worden. Die anderen Anwendungsfälle greifen auf sie zurück.

Somit wurden fünf Anwendungsfälle identifiziert, welche für die zu entwickelnde Erweiterung notwendig sind, wobei die Secondary-Usecases hier nicht weiter erläutert werden.

4.2. Anforderungen an den Graphen

In der Analysephase muss geklärt werden, was für grundsätzliche Anforderungen an die Graphen gestellt werden. Das Durchsprechen und Durchspielen der verschiedenen Anwendungsfälle ermöglichte eine Konkretisierung der in Tabelle 2 aufgelisteten Anforderungen. Hierbei ist zu beachten, dass die Nachrichten als Kanten und die Einheiten als Knoten angesehen werden.

Nr.	Anforderungsbeschreibung	Verbindlichkeit
G010	Der darzustellende Graph muss gerichtet sein, damit der Nachrichtenfluss deutlich wird.	Pflicht
G020	Die Knoten des Graphen müssen beschriftbar sein.	Pflicht
G030	Die Kanten des Graphen müssen beschriftbar sein.	Pflicht
G040	Es muss verschiedene Arten (Rechteck, Ellipse, …) von Knoten geben, die die einzelnen Einheitstypen visualisieren.	Pflicht
G050	Es müssen Kanten mit mehreren Zielen darstellbar sein.	Pflicht
G060	Es müssen Kanten darstellbar sein, die keinen Knoten als Quelle haben. (Messwerte)	Pflicht
G070	Es müssen Kanten darstellbar sein, die als Ziel keinen Knoten aufweisen. (Aktionen)	Pflicht
G080	Es muss dargestellt werden können, dass Informationsnachrichten durch Verarbeitungseinheiten durchgeschleust werden.	Pflicht
G090	Zum Verdeutlichen des kritischen Pfades müssen Kanten hervorgehoben werden können.	Pflicht
G100	Es muss im Graphen die Zugehörigkeit der einzelnen Elemente zu den Gruppenelementen erkennbar sein.	Pflicht

Tabelle 2: Anforderungen an den Graphen

Neben den Anforderungen an die Graphen müssen noch zusätzliche Forderungen an die entsprechende Erweiterung gestellt werden. Diese Anforderungen basieren auf der in Kapitel 2.4 beschriebene Programmumgebung und müssen erfüllt werden, sodass eine Einbettung in das bestehende SALT Programm möglich ist. In Tabelle 3 sind die ermittelten Systemanforderungen beschrieben.

Nr.	Anforderungsbeschreibung	Verbindlichkeit
S010	Die Erweiterung muss Python 2.3 kompatibel sein	Pflicht
S020	Es muss möglich sein die Graphen mit Hilfe des GUI-Toolkits „pyQt 3“ anzuzeigen.	Pflicht
S030	Die Erweiterung muss Windows XP lauffähig sein.	Pflicht
S040	Das Erstellen und Anzeigen der Graphen sollte nicht länger als eine Sekunde beanspruchen.	Wunsch

Tabelle 3: Systemanforderungen

5. Auswahl des Graphen-Toolkits

In diesem Kapitel wird anhand der gestellten Anforderungen aus Kapitel 4.2 ein Toolkit zur grafischen Darstellung von Graphen ausgewählt. Basierend auf einer Internetrecherche konnten zwar zahlreiche Software Tools ermittelt werden, die Python mit einer Datenstruktur für Graphen erweitern, diese jedoch nicht Visualisieren können. Der eigentliche Auswahlpool umschloss mehrere Toolkits, die jedoch durch eine Vorauswahl weiter dezimiert wurden. Bei diesem Verfahren fielen vor allem veraltete Tools aus dem Pool, sowie Tools, die offensichtlich einige der gewünschten Anforderungen, zum Beispiel das Darstellen gerichteter Graphen, nicht unterstützen. In Tabelle 4 sind die Graphen-Toolkits angezeigt, die während der Vorauswahl ausgeschlossen wurden. Außerdem sind die Gründe des Ausschlusses dargestellt.

Tool	Ausschlusskriterium
Pygraphlib [19]	Veraltet
Gato [11]	Keine gerichteten Graphen
Pygraph [18]	Keine Visualisierung
Boost.Python [9]	Keine Visualisierung

Tabelle 4: Ausgeschlossene Graphen-Toolkits

Nach dieser Vorauswahl konnte das detaillierte Auswahlverfahren mit zwei Toolkits durchgeführt werden. Zu den Favoriten zählte zum einen ein Pythontool namens „NetworkX“ [16]. Dabei handelt es sich um ein Softwarepaket zur Erstellung, Manipulation und Studie komplexer Netzwerke [5]. Es stellt verschiedene Algorithmen zur Verfügung, um verschiedenste Aspekte des Graphen zu analysieren und diese anzuordnen. Für die eigentliche grafische Darstellung greift „NetworkX“ auf externe Tools zurück. In Abbildung 13 ist als Beispiel ein Graph ausgewählt, der mit Hilfe der Python 2D-Plotting Bibliothek „matplotlib“ [14][6] erstellt worden ist.

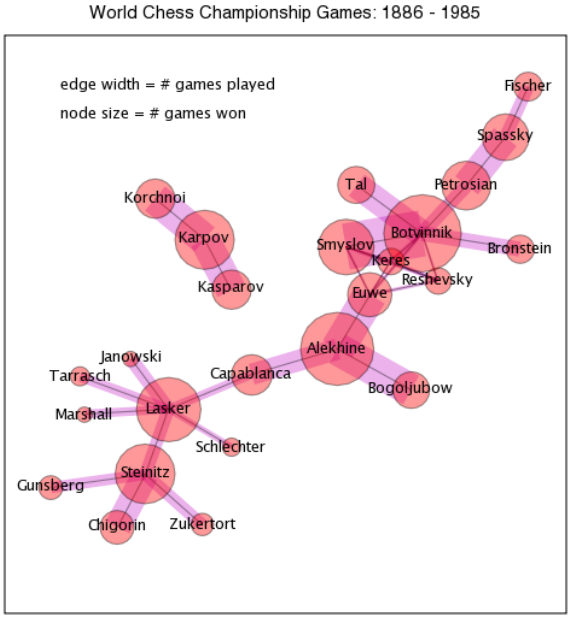


Abbildung 13: Beispiel für NetworkX

Zum anderen fiel ein Programm namens „Graphviz“[12] auf. Hierbei handelt es sich um eine auf die Visualisierung von Graphen spezialisierte Software, die mit verschiedenen Layout Programmen Graphen auf verschiedene Weisen darstellen kann. Für die Generierung werden die Graphen, wie in Abbildung 14 dargestellt, in der so genannten DOT Sprache beschrieben.[3][4] Außerdem besteht die Möglichkeit mit Python direkt auf die Fähigkeiten von „Graphviz“ zuzugreifen, zum Beispiel mit dem Modul „pyDot“ [17]. Anzumerken ist, dass „Graphviz“ auch zur grafischen Darstellung der Graphen bei „NetworkX“ verwendet werden kann, jedoch werden nicht alle Möglichkeiten, die „Graphviz“ bietet, ausgeschöpft.

```
digraph G {

        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```

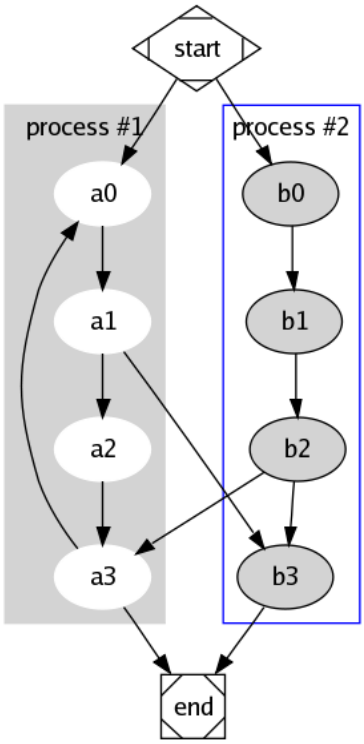


Abbildung 14: Beispiel für Graphviz mit dem entsprechenden DOT Code

Um die eigentliche Bewertung durchführen zu können, wurden die Anforderungen aus Kapitel 4.2 ihrer Bedeutung entsprechend gewichtet. Außerdem wurde das 5-stufige Bewertungsschema der Tabelle 5 angelegt, mit dem die Unterstützung der einzelnen Anforderungen des Toolkits bewertet wurde.

Bewertung	Punkte
volle Unterstützung	100
gute Unterstützung	75
zufrieden stellende Unterstützung	50
schlechte Unterstützung	25
Keine Unterstützung	0

Tabelle 5: Bewertungsschema

Mit Hilfe dieses Schemas wurden im Zuge des Designs die einzelnen Tools nacheinander auf die aus der Analyse resultierenden Anforderungen überprüft und validiert. Nachdem die beiden Toolkits beurteilt wurden, mussten die Ergebnisse ausgewertet werden. Dazu wurde die prozentuale Gewichtung der Anforderung

mit den dazugehörigen Bewertungen multipliziert und die daraus resultierenden Werte am Ende aufsummiert. In Tabelle 6 sind die Auswertungsergebnisse für NetworkX“ mit „matplotlib“ als grafisches Frontend und „Graphviz“ mit „pyDot“ als Schnittstelle zu Python dargestellt. An diesen Ergebnissen ist zu erkennen, bis zu welchem Grad die einzelnen Tools die gegebenen Anforderungen erfüllen.

Anforderungen	Gewichtung	NetworkX	Graphviz
G010	100%	50	100
G020	100%	100	100
G030	100%	25	100
G040	100%	75	100
G050	100%	75	75
G060	100%	75	75
G070	100%	75	75
G080	100%	0	75
G090	100%	100	100
G100	100%	25	100
S010	100%	100	100
S020	100%	75	50
S030	100%	100	100
S040	50%	*(Zu diesem Zeitpunkt nicht bestimmbar)*	
Ergebnis:		875	1150

Tabelle 6: Auswertungsergebnisse von NetworkX und Graphviz

Basierend auf dieser Bewertung stellte sich heraus, dass bei einem maximalen Punktewert von 1350 das Tool „NetworkX“ die Anforderungen zu fast 65% erfüllt. „Graphviz“ ist jedoch mit etwa 85% deutlich besser geeignet ist. Anzumerken ist allerdings, dass die Anforderung S040 nicht überprüft werden konnte, da aus Zeitmangel keine authentischen Graphen erzeugt wurden. Weil es sich bei S040 nicht um eine verpflichtende Anforderung handelte, kann das Fehlen dieses Wertes im Rahmen dieser Arbeit hingenommen werden. Ohne Berücksichtigung der letzten Anforderung würde die maximale Punktzahl auf 1300 sinken. Das würde bewirken, dass „Graphviz“ zu etwa 88% und „NetworkX“ zu etwa 67% die Anforderungen erfüllt.
Aus diesen gewonnenen Erkenntnissen konnte eindeutig die Entscheidung getroffen werden, dass das Tool „Graphviz“ als Graphentoolkit für die Erweiterung in SALT eingesetzt werden sollte, da es den größten Teil der Anforderungen sehr gut unterstützt.

6. Design und Umsetzung

Nach der Auswahl des Toolkit zur Erstellung der Graphen, konnte mit dem eigentlichen Design und der Umsetzung der Erweiterung begonnen werden. Dabei wurde zuerst das Aussehen des Graphen festgelegt. Anschließend musste die Klasse entwickelt werden, die zur Generierung der Graphen verwendet werden soll. Danach wurde das Verfahren entworfen und umgesetzt, das die erzeugten Graphen in der grafischen Benutzerschnittstelle darstellen sollte.

6.1. Design der Graphenelemente

Nachdem das Tool zur Erstellung der Graphen ermittelt worden war, konnte mit dem Design der Graphenelemente begonnen werden. Hierbei wurde festgelegt, auf welche Art und Weise die verschiedenen Anforderungen, die während der Analysephase an den Graphen gestellt wurden (siehe Kapitel 4.2), umgesetzt werden sollten. Für eine nähere Betrachtung eignen sich jedoch nur die Anforderungen G040 bis G100, da die anderen grundlegend sind und keine nähere Erläuterung benötigen. Im Folgenden sind die Ergebnisse der Betrachtung der einzelnen Anforderungen beschrieben.

- G040:
  Für die in Anforderung G040 geforderte Darstellung der verschiedenen Einheitstypen, stellt „Graphviz" eine Vielzahl unterschiedlicher Knotenformen zur Verfügung, sodass jeder Einheitsart eine Form zugewiesen werden kann, wie in Abbildung 15 dargestellt wird.

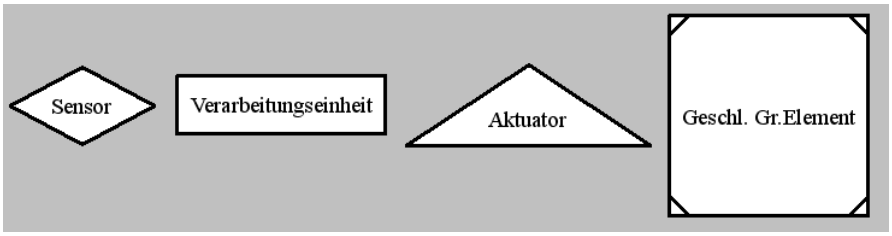


Abbildung 15: Darstellung der verschiedenen Knotentypen

- G050:
  Schwieriger war die Umsetzung der Anforderung G050, die das Darstellen von Kanten mit mehreren Zielen verlangt. Dazu wurde ein „Zwischenknoten" eingeführt, von dem aus das Ansprechen der einzelnen Zielknoten erfolgt. In Abbildung 16 ist ein Beispiel für eine Kante mit zwei Zielen dargestellt.

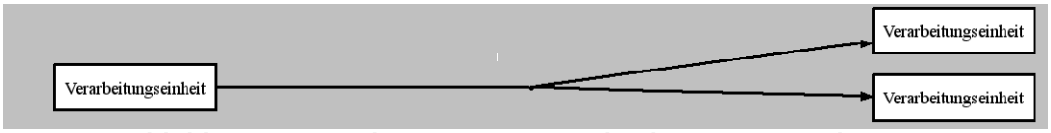


Abbildung 16:Visualisierung einer Nachricht mit zwei Zielen

- G060 und G070:
  Die Anforderungen G060 und G070 konnten zusammen betrachtet werden, da bei beiden Kanten dargestellt werden sollen, die entweder keinen Ziel- oder Quellknoten besitzen. Weil Kanten sowohl ein Ziel, als auch eine Quelle haben müssen, konnte dieser Konflikt nur mit Hilfe eines nicht sichtbaren Knotens gelöst werden. Dieser Knoten ersetzt den fehlenden Quell-, bzw. Zielknoten. Zusätzlich wurden die Kantenenden angepasst, um so eine Verdeutlichung der Nachrichten hervorzurufen. Das damit erzielte Ergebnis ist in Abbildung 17 zu betrachten.

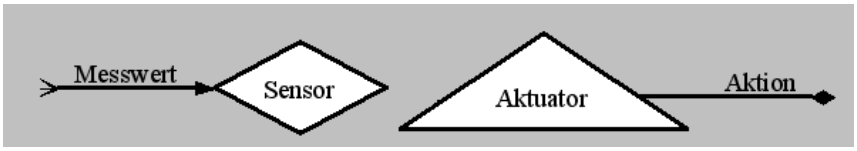


Abbildung 17: Darstellung der Messwerte und Aktionen

- G080:
  Ein weiterer Punkt, der berücksichtigt werden musste, war die von Anforderung G080 verlangte Darstellung des „Pipelinings“. Um in Graphen das Durchschleusen von Informationsnachrichten durch Verarbeitungseinheiten zu verdeutlichen, griff man auf anders geformte Kantenenden zurück (s. Abbildung 18).

Abbildung 18: Darstellung von "Pipelines"

- G090:
  Zur Visualisierung der verschiedenen Sicherheitslevel wurde während des Designs die Einfärbung der Kanten und Knotenränder festgelegt. Dabei sollen die Farben von Grün, für nicht sicherheitsrelevant, bis hin zu Rot, für sicherheitsrelevant, verwendet werden. Um die Anforderung G090 zu erfüllen, sollen alle Kanten des kritischen Pfades außerdem noch verdickt dargestellt werden.

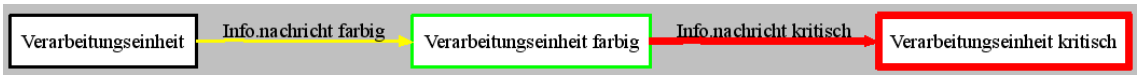


Abbildung 19: Hervorhebung der Sicherheitslevel und des kritischen Pfades

- G100:
  Um die von der letzten Anforderung verlangte Visualisierung der Zugehörigkeit zu verdeutlichen, wurden so genannte „Cluster“ eingeführt. Dadurch wird die Erstellung eines Subgraphen ermöglicht, dessen Hintergrundfarbe angepasst werden kann, so dass die Zugehörigkeit verdeutlicht wird.

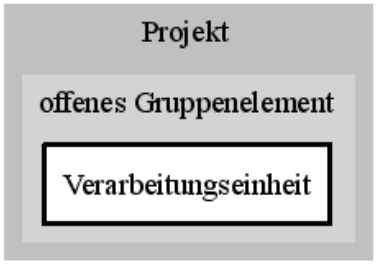


Abbildung 20: Darstellung offener Gruppenelemente als "Cluster"

Außerdem wurde während des Designs festgehalten, dass das eigentliche Layout der Graphenelemente jederzeit angepasst werden kann. Aus diesem Grund soll das Layout in einer XML Datei abgelegt werden, um so das Anpassen zu erleichtern. Dieses Vorgehen wurde schon an anderen Stellen in SALT erfolgreich umgesetzt.

6.2. Entwicklung des GraphHelpers

Nach der Aufstellung des Graphendesigns wurde die Klasse des GraphHelpers entworfen. Bei diesem Helper handelt es sich um eine Klasse, die zur Erzeugung des Graphen, jedoch nicht zur Visualisierung, verwendet wird. Um die einzelnen Aufgaben der Klasse zu ermitteln, wurden die Usecases aus Kapitel 4.1 herangezogen. Darauf basierend konnten folgende Methoden abgeleitet werden, die zur Erzeugung der Graphen benötigt werden:

- **createNode:**
  Diese Methode erzeugt einen Knoten des Graphen entsprechend der mitgegebenen Informationen. Darunter fallen zum einen der Name und die Beschriftung des Knotens sowie zum anderen der Sicherheitslevel und der darzustellende Knotentyp.
- **createEdge:**
  Mit Hilfe dieser Methode wird eine Kante des Graphen von dem mitgegebenen Quellknoten bis zum Zielknoten erzeugt. Sie erhält außerdem noch die Informationen über den Kantentyp und Beschriftung sowie die entsprechende Farbe, in der die Kante angezeigt werden muss.
- **createMessageEdges:**
  Um die Kanten einer kompletten Nachricht zu erzeugen wird diese Methode der Klasse aufgerufen. Sie berücksichtigt den Nachrichtentyp (Messwert, Aktion, Informationsnachricht), überprüft ob die Nachricht durchgeschleust wird und ob mehre Ziele vorhanden sind. Im letzten Fall wird ein Hilfsknoten erzeugt, damit mehrere Ziele angesprochen werden können.

- **createClusterOfGroupElement:**
  Gruppenelemente, deren Inhalt ersichtlich sein soll, auch „offene Gruppenelemente“ genannt, werden mit Hilfe dieser Methode erstellt. Dabei wird ein Subgraph erzeugt, auch „Cluster“ genannt, der sich farblich abhebt. In diesem „Cluster“ werden die Elemente eingefügt, die zu dem jeweiligen Gruppenelement gehören.
- **createGraph:**
  Der komplette Graph wird mit dieser Methode erzeugt. Sie greift auf die oben genannten Funktionen zurück, um den gewünschten Graph zu erstellen.
- **getAnalysisPathGraph:**
  Der Graph eines einzelnen Analysepfades wird mit dieser Funktion erzeugt. Sie benötigt dazu nur die Informationen des Pfades und erzeugt mit Hilfe der genannten Funktionen den entsprechenden Graphen.
- **getObservationLevelGraph:**
  Dadurch wird der Graph mit Hilfe des übergebenen Observationslevels, also eines Gruppenelements, erzeugt. Dabei ist zu beachten, dass in der Darstellung sowohl die eingehenden, als auch die ausgehenden Nachrichten berücksichtigt sind. Außerdem werden bei dieser Betrachtung die untergeordneten Gruppenelemente des mitgegebenen Observationslevels als „geschlossene Gruppenelemente“ betrachtet. Das bedeutet, dass keine Elemente innerhalb der untergeordneten Gruppenelemente dargestellt werden.
- **getProjectGraph:**
  Der Graph, der mit dieser Methode erzeugt wird, beinhaltet alle Elemente innerhalb des Projektes. Alle Gruppenelemente werden als „offene Gruppenelemente“ dargestellt.

Das Zusammenspiel der einzelnen Funktionen ist in Abbildung 21 für das Beispiel eines Observationslevelgraphen in Form eines Sequenzdiagramms dargestellt. Abgebildet sind dabei die einzelnen Methodenaufrufe des GraphHelpers. Die Attribute bzw. Rückgabewerte der Funktionen sind zusätzlich an den Aufrufen angezeichnet. Außerdem beinhaltet das Diagramm noch in rot gekennzeichnete Erklärungen, die Zusatzinformationen zur eigentlichen Funktionalität geben. Darüber hinaus ist zu erkennen, dass die Methode „createClusterOfGroupElement“ sich rekursiv aufruft. Dieses Vorgehen dient der Darstellung von verschachtelten Gruppenelementen. Sie werden jedoch nur angezeigt, wenn die entsprechenden Gruppenelemente weitere Elemente beinhalten, die noch angezeigt werden müssen.

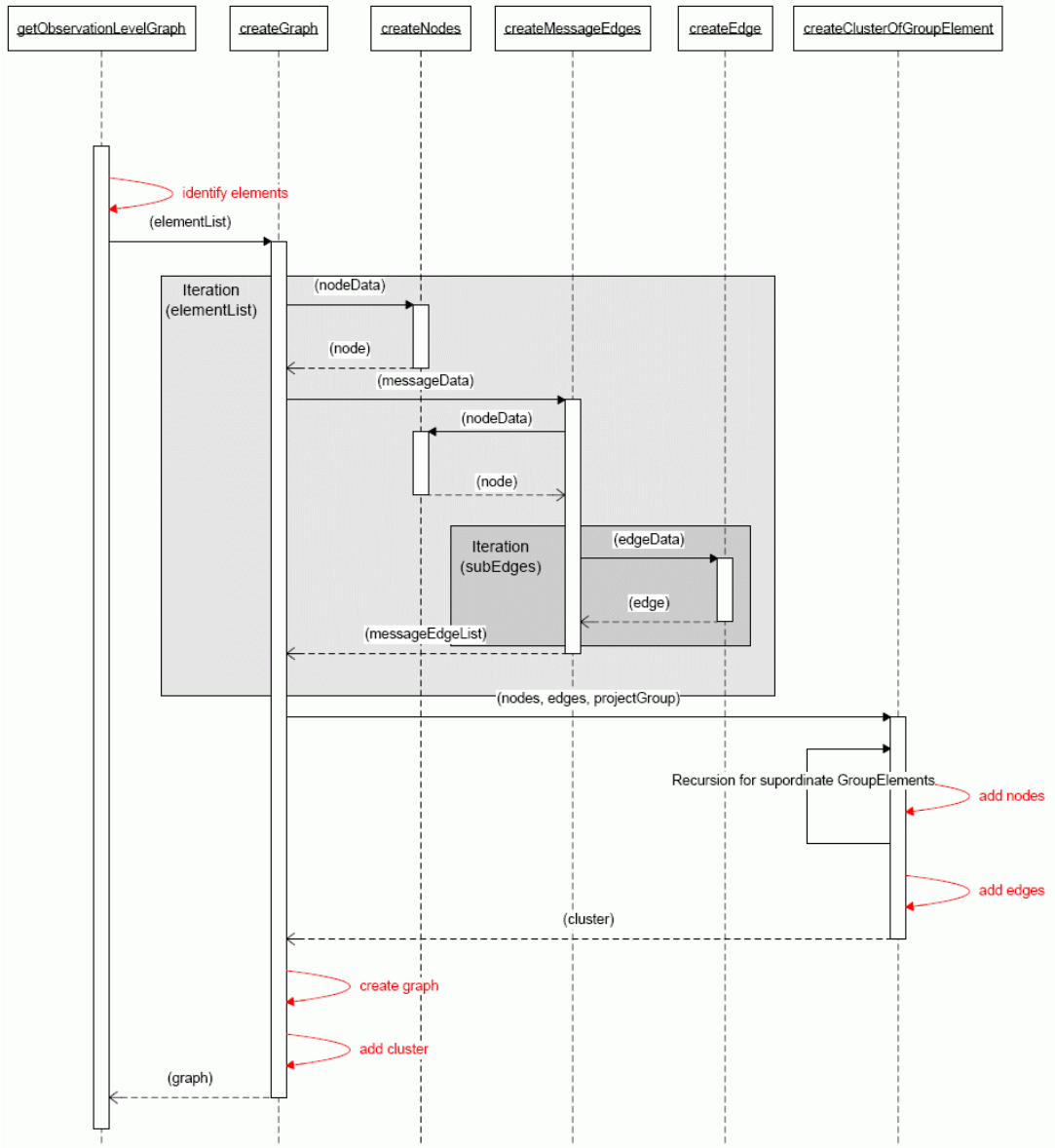


Abbildung 21: Sequenzdiagramm zur Erstellung eines Observationslevelgraphen

Das hier beschriebene Vorgehen wird auch zum Erstellen der Graphen der Analysepfade oder des Projektes verwendet. In diesen Fällen werden jedoch nur die Elemente angezeigt, die für den entsprechenden Fall notwendig sind.
Ein generierter Beispielgraph ist in Abbildung 22 zu erkennen. Er zeigt den Aufbau des bereits vorgestellten Fensterhebers. Die zur Erzeugung benötigten Daten bekommt der „GraphHelper“ aus den erstellten Elementen des Projektes. Gut zu erkennen sind die verschiedenen Einheiten mit den jeweiligen Nachrichten, die von dem Element bearbeitet werden. Zu erkennen ist außerdem der kritische Pfad, der verdickt dargestellt ist.

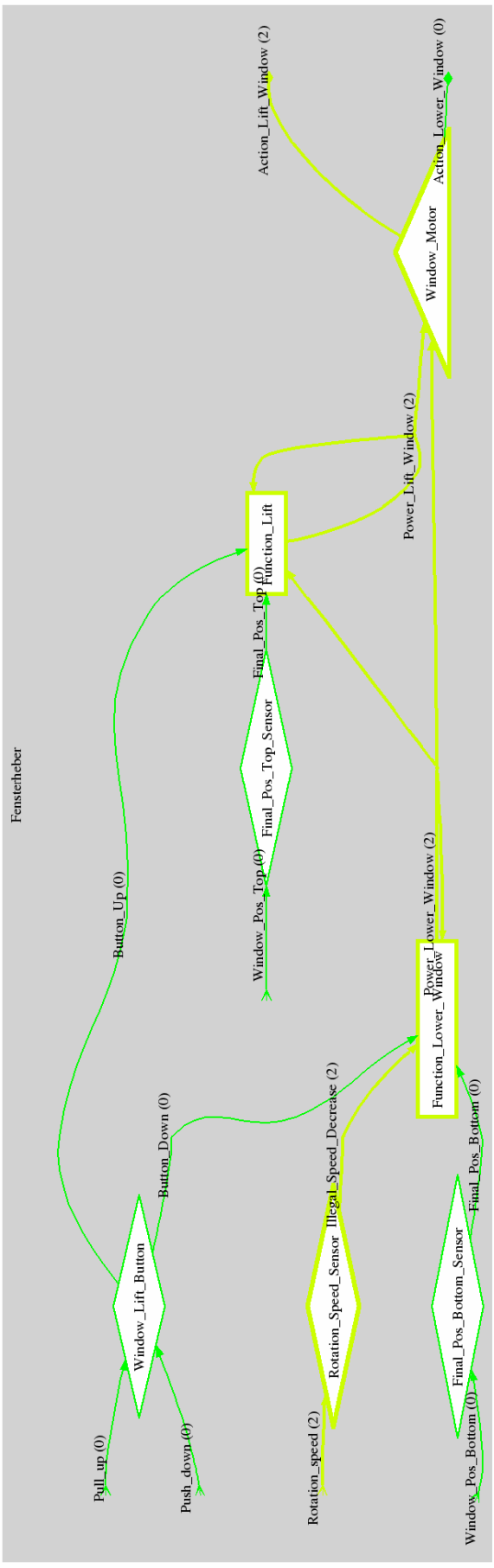


Abbildung 22: Generierter Graph des Fensterhebers

6.3. Design und Umsetzung des GraphDialogs

Die generierten Graphen des „GraphHelpers“ werden im „GraphDialog“ dargestellt. Dabei handelt es sich um einen Qt-Dialog, der in SALT eingebettet ist. Dieser Dialog soll aufgerufen werden, sobald der Benutzer einen Graphen anzeigen möchte. Im Fall der Analysegraphen muss zuerst eine Analyse des Projektes durchgeführt werden. Anschließend kann der Benutzer zwischen den einzelnen Analysepfaden einen auswählen, den er darstellen möchte. Erst danach wird der Graph generiert und im „GraphDialog“ angezeigt.
Im Zuge des Designs und der Umsetzung wurde neben dem grafischen Aufbau der Benutzeroberfläche auch die darunter liegende Funktionalität entwickelt. Der Graph sollte nämlich nicht nur dargestellt werden, sondern der Dialog sollte auch die Möglichkeit des Vergrößern und Verkleinern bieten. Zusätzlich sollte der Benutzer die Fähigkeit haben, Informationen über die einzelnen Elemente des Dialoges zu ermitteln.

6.3.1. Design der grafischen Benutzeroberfläche

Die grafische Oberfläche wurde mit Hilfe des Designers der Firma Trolltech [23] entworfen. Dabei wurde darauf geachtet, dass der Dialog möglichst einfach ist, aber dem Benutzer trotzdem die gewünschte Funktionalität bietet.
Aus diesem Grund ist der Dialog, wie in Abbildung 23 dargestellt, wie folgt aufgeteilt: Oben befinden sich die verschiedenen Schaltflächen, die der Auswahl der Funktionalität dienen. Dabei handelt es sich, von links nach rechts betrachtet, um das Vergrößern, Verkleinern, Verschieben und das Erhalten der Informationen aus dem dargestellten Graphen.
Der Graph wird im Bereich unter der Funktionsauswahl dargestellt. Dieser Bereich nimmt den größten Platz ein. Mittels der Scrollbalken ist es möglich, auch große Graphen darzustellen und zu manövrieren.
Am unteren Rand des Dialoges sind die Schaltflächen zum Speichern des Graphen und zum Schließen des Dialoges angeordnet.

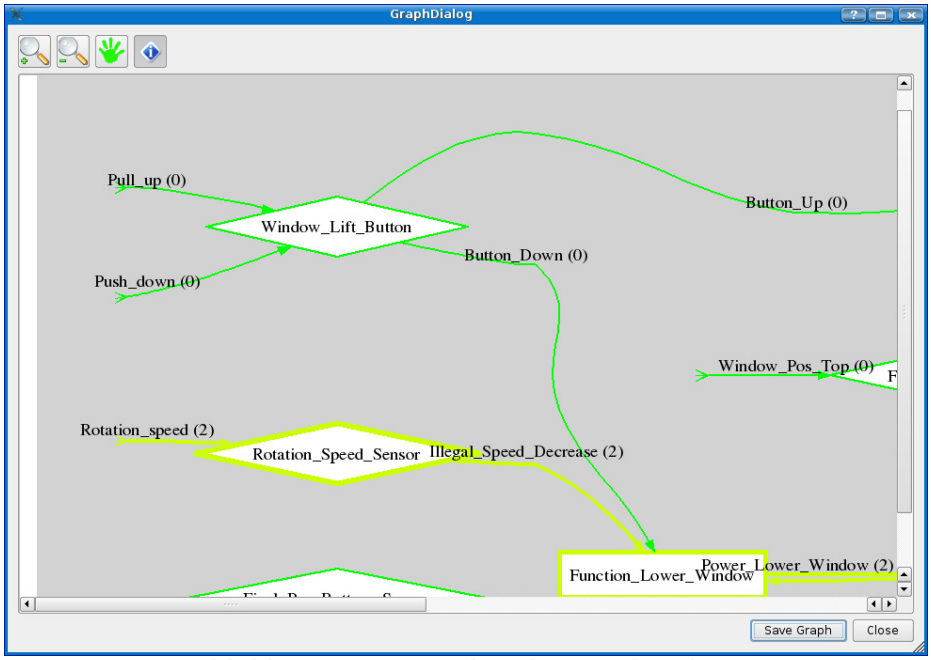


Abbildung 23: Screenshot des GraphDialogs

6.3.2. Design und Umsetzung der Funktionalität

Um die Funktionalität des Dialoges umzusetzen, wurden zuerst die Klassenbeziehungen des Dialoges entworfen. Die Ergebnisse davon sind im Klassendiagramm in Abbildung 24 dargestellt. Zu Erkennen ist hierbei, dass die Klasse „GraphDialog" von der Klasse „QDialog" des Qt Toolkits abgeleitet ist, welches zur Gestaltung der grafischen Benutzeroberfläche verwendet wird. Außerdem beinhaltet der „GraphDialog" auch den beschriebenen „GraphHelper".

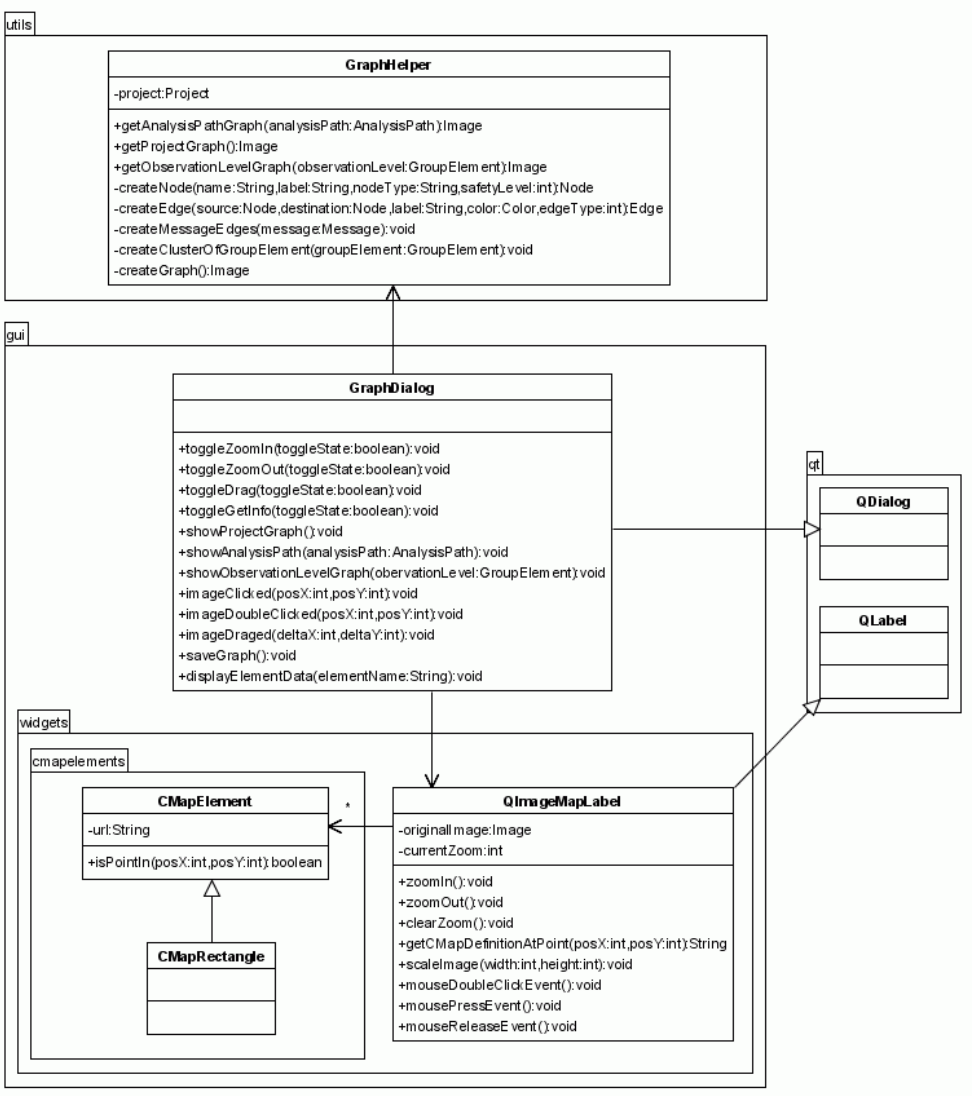
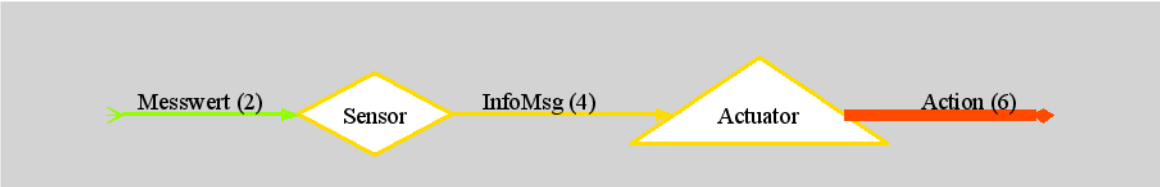


Abbildung 24: Klassendiagramm des GraphDialogs

Kernstück des Dialogs ist jedoch die Klasse „QImageMapLabel". Diese Klasse dient der eigentlichen Darstellung des Graphen. In dieser Klasse ist außerdem die Funktionalität hinterlegt, die zum Vergrößern und Verkleinern genutzt wird. Dabei wird das originale Bild mit einem von der Vergrößerungsstufe abhängigen Faktor skaliert.
Die größte Schwierigkeit bestand jedoch in der Extraktion von Informationen über die einzelnen Elemente des Graphen. Üblicherweise werden Graphen aus verschiedenen Teilen zu einem Graphen aufgebaut, so dass es möglich ist, die Informationen der einzelnen Bestandteile zu erfragen. In diesem Fall liegt der

Graph allerdings als ein komplettes Bild vor. Das Bild alleine bietet jedoch keine Möglichkeit die Informationen über die darauf abgebildeten Graphenelemente zu erlangen.
Eine mögliche Lösung wäre die Implementierung eines eigenen Parsers der DOT-Sprache. Dies ist jedoch zu aufwändig, da die gesamte Funktionalität von „Graphviz“ neu zu implementieren wäre, wie die Anordnungsalgorithmen der einzelnen Knoten.
Zur Lösung dieses Problems wurden so genannte „Image-Maps“ verwendet. Diese Image-Maps sind in den HTML Spezifikationen [8] definiert und dienen zur Erstellung von verweissensitiven Grafiken. Mit Hilfe dieses, aus dem Umfeld des Internet bekannten Vorgehens, können verschiedene Bereiche eines Bildes auf unterschiedliche Internetseiten verweisen. Außerdem ist es möglich die Form des verweissensitiven Bereichs festzulegen. Zur Auswahl stehen Rechtecke, Ellipsen und Polygone. Die Informationen über die Position, die Form und das Ziel der Verankerung, auch „Link“ genannt, werden in der Image-Map hinterlegt.
Das Toolkit „Graphviz“ bietet die Möglichkeit, neben dem Bild des erstellten Graphen, auch die entsprechende Image-Map zu generieren. Dafür müssen jedoch während der Definition des Graphen die entsprechenden Verlinkungsziele angegeben werden.
In Abbildung 25 ist ein Beispielgraph mit der dazugehörigen Image-Map zu erkennen. Die Besonderheit hierbei ist, dass nicht auf Internetseiten verwiesen wird, sondern ausschließlich der Name des entsprechenden Graphelements als Verankerung angegeben ist. Außerdem werden pro Nachricht drei Bereiche in der Image-Map deklariert. Dabei handelt es sich um die Beschriftung, den Kantenursprung und den Kantenkopf.




```
<area shape="rect" href="Actuator" title="Actuator" alt="" coords="507,59,709,150">
<area shape="rect" href="Sensor" title="Sensor" alt="" coords="248,72,368,136">
<area shape="rect" href="Action" title="Action" alt="" coords="734,83,813,104">
<area shape="rect" href="Action" title="Action" alt="" coords="672,100,680,108">
<area shape="rect" href="Action" title="Action" alt="" coords="833,100,841,108">
<area shape="rect" href="InfoMsg" title="InfoMsg" alt="" coords="392,83,483,104">
<area shape="rect" href="InfoMsg" title="InfoMsg" alt="" coords="364,100,372,108">
<area shape="rect" href="InfoMsg" title="InfoMsg" alt="" coords="536,100,544,108">
<area shape="rect" href="Messwert" title="Messwert" alt="" coords="123,83,224,104">
<area shape="rect" href="Messwert" title="Messwert" alt="" coords="95,100,103,108">
<area shape="rect" href="Messwert" title="Messwert" alt="" coords="244,100,252,108">
```

Abbildung 25: Beispielgraph mit dazugehöriger Image-Map

Um die Funktionalität des „QImageMapLabels“ in Bezug auf die Image-Maps zu erweitern, wurde der Klasse eine Liste hinzugefügt, die für jeden Eintrag in der Image-Map, eine Instanz der „CMapElement“ Klasse aufweisen sollte. Von dieser Klasse leiten sich die verschiedenen Image-Map-Klassen ab, die jeweils eine der Umrissarten wieder spiegeln, zum Beispiel für die Form des Rechtecks die Klasse „CMapRectangle“. Sie enthalten zum einen das Verankerungsziel, das in der Image-Map deklariert wurde und zum anderen eine Methode, die ermittelt, ob sich ein bestimmter Punkt innerhalb des definierten Bereichs befindet.

Mit dieser Lösung besteht die Möglichkeit, vom Benutzer ausgewählte Elemente aus einem als Bild vorliegenden Graphen zu bestimmen. Wird jedoch das Bild vergrößert oder verkleinert, können die Elemente nicht mehr eindeutig zugeordnet werden, da die definierten Bereiche der Image-Map nicht mehr mit denen des Bildes übereinstimmen.
Dieses Problem wurde behoben, indem die ausgewählte Position des Benutzers erst auf die Koordinaten des Originalbildes umgerechnet wird, bevor die Weiterleitung der Anfragen an die „CMapElemente“ erfolgt.

7. Qualitätssicherung

In diesem Kapitel wird kurz auf die verschiedenen Maßnahmen der Qualitätssicherung eingegangen, die während der Entwicklung eine Rolle gespielt haben. Darunter werden der Modultest, der Codingstyle, die Versionierung und das Bugtracking behandelt.

7.1. Modultest

Eine Möglichkeit, Fehler in einem frühen Stand der Entwicklung zu erkennen und zu verhindern, bieten Unit-Tests, auch Komponenten oder Modultests genannt. Mit Hilfe dieser Tests wird die Software nicht in ihrer Gesamtheit getestet, sondern auf Basis ihrer einzelnen Komponenten, zum Beispiel Funktionen oder Methoden einer Klasse. Dafür werden diese Komponenten in einer künstlichen Umgebung mit verschiedenen Testdaten konfrontiert, um an Hand der daraus folgenden Reaktion ihre Korrektheit zu prüfen. Somit ist beim Zusammenfügen zu einer Applikation gewährleistet, das jede einzelne Komponente korrekt arbeitet. Bei diesen Tests wird kein Wert auf Performance gelegt. Es wird nur überprüft, ob diese einzelne, vom Rest der Software isolierte Komponente, die Funktionalität aufweist, die ihr zugeschrieben ist. Außerdem sollten auch Ausnahmeerscheinungen und Abbruchbedingungen durch die Tests abgedeckt werden.
Im Zuge der Entwicklung von SALT wurden für die wichtigsten Klassen der Anwendung diese Modultests geschrieben. Darunter fallen vor allem die Klassen der Datenstrukturen und die Klasse, die zur eigentlichen Betrachtung des Nachrichtenflusses verwendet wird. Jedoch nicht alle Klassen eignen sich für diese Tests gleichermaßen. So wird zum Beispiel die Klasse des GraphHelpers nicht mit Komponententests abgedeckt, da die Verifikation mit einem erzeugten, als Bild vorliegenden Graphen als nicht praktikabel empfunden wurde.
Des Weiteren wurden auch die Klassen der grafischen Benutzeroberfläche nicht mit Hilfe der Unit-Test überprüft, da die GUI nur als Schnittstelle zwischen dem Benutzer und der Applikation dienen soll und nur auf die bereits getestete Funktionalität aufsetzt.

7.2. Automatische Code-Dokumentation

Um eine Dokumentation des Quellcodes zu erstellen, wurde das Tool „Epydoc“[10] verwendet. Es ermöglicht eine automatische Generierung einer API Dokumentation (API = „application programming interface“, deutsch: „Schnittstelle zur Anwendungsprogrammierung“). Um eine Dokumentation zu ermöglichen, muss der Quellcode in einer vordefinierten Weise kommentiert sein. Das Tool durchsucht den Quellcode nach diesen vorgegebenen Schlüsselwörtern und generiert daraus die Dokumentation, die in verschiedenen Formaten abgespeichert werden kann. In Abbildung 26 ist eine generierte Dokumentation in HTML-Format zu erkennen.

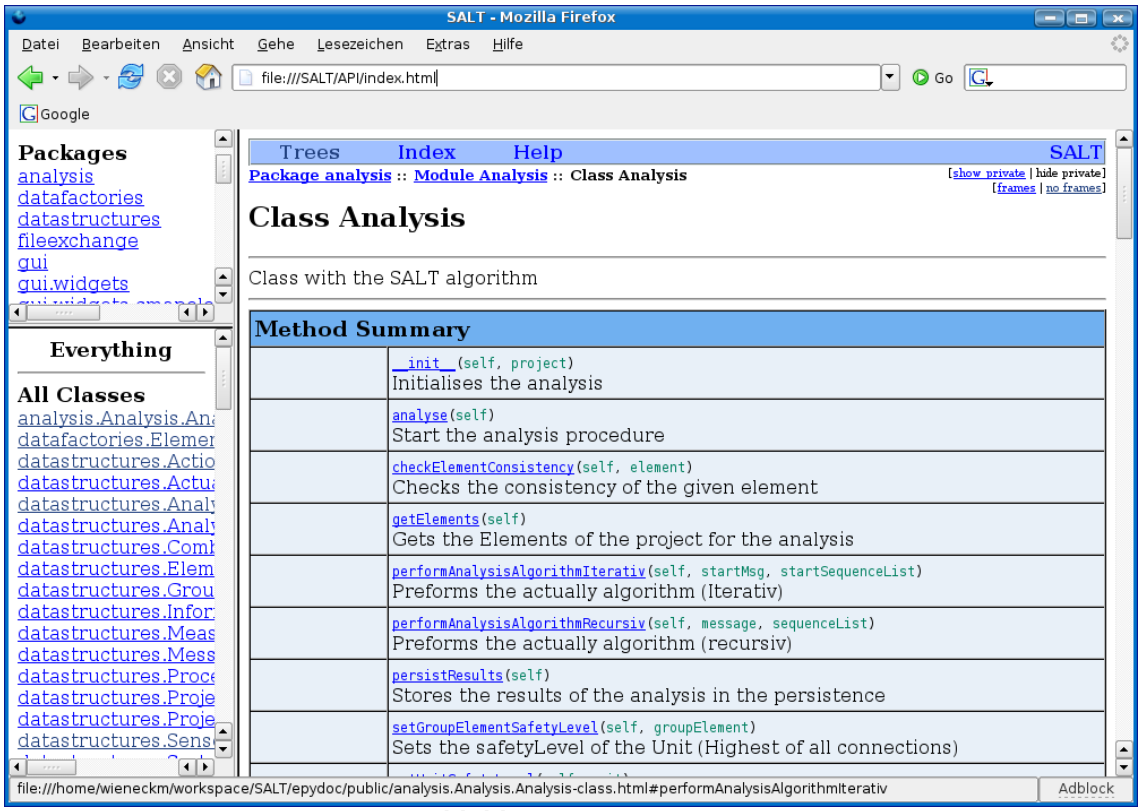


Abbildung 26: SALT-API

7.3. Coding-Style

Der Coding-Style beschreibt alle Richtlinien, die während der Implementierung an den Quellcode gestellt werden. Das beinhaltet alles, von der Namensgebung der Variablen bis zur maximalen Zeilenanzahl innerhalb einer Klasse. Diese Kodierrichtlinien sind für die jeweilige Programmiersprache festgelegt. In diesem Kapitel wird jedoch nicht auf die einzelnen Richtlinien eingegangen, sondern nur eine Abweichung beschrieben, die während der Entwicklung von SALT gemacht wurde.
Bei dieser Abweichung von den Richtlinien handelt es sich um die Namensgebung privater bzw. versteckter Methoden. Der Coding-Style besagte, dass, wie in Python üblich, der Namen der Methoden mit jeweils zwei Unterstrichen beginnen und enden muss, der Name der dazwischen liegt jedoch nur aus Kleinbuchstaben bestehen darf. Ein Beispielname, der an diese Konvention anlehnt, ist zum Beispiel:

```
__testmethode__
```

Die Änderungen, die für SALT vorgenommen wurden besagen, dass der eigentliche Methodenname nicht nur aus Kleinbuchstaben bestehen muss, sondern auch Großbuchstaben und Unterstriche erlaubt sind. Mit dieser Erweiterung sind daher auch folgende Methoden Namen möglich:

```
__testMethode__
__test_methode__
```

Diese Erweiterung wurde als sinnvoll erachtet, da somit die Lesbarkeit der Methodennamen verbessert wird. Ein weiterer Vorteil ist, dass die Richtlinie so der Namensgebung für öffentliche Methoden angenähert wird, die die gleichen Namenskonventionen beinhaltet, mit Ausnahme der Unterstriche zu Beginn und am Ende des Methodennamens.

7.4. Bug-Tracking

Unter einem Bug-Tracking-System verbirgt sich eine Applikation, mit deren Hilfe man aufgetretene Fehler und Erweiterungsvorschläge melden und ihren Entwicklungsstand verfolgen kann. Dafür muss, sobald ein Fehler erkannt wird, ein Eintrag erfolgen, dem eine eindeutige Identifikationsnummer zugewiesen wird. Unter dieser Nummer können weitere Kommentare, Erläuterungen, Lösungsvorschläge und ähnliches eingetragen und verwaltet werden.
Im Zuge der Entwicklung von SALT wurde ein Bug-Tracking-System namens „Mantis“ [13] verwendet. Es bietet die Möglichkeit die Verwaltung der Fehlereinträge online durchzuführen (siehe Abbildung 27) und ist somit standortunabhänig erreichbar. Außerdem wird damit ein einfacher Weg für Projektpartner und Anwender von SALT angeboten, neue Fehler oder Verbesserungen zu melden.

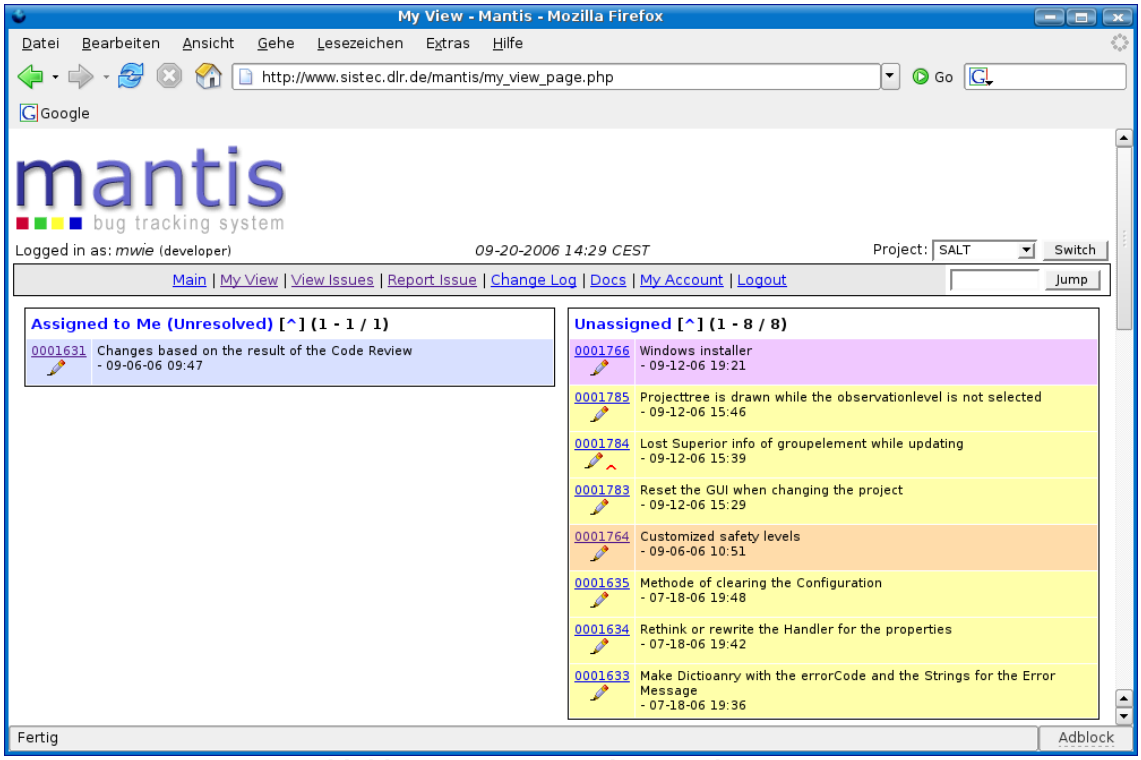


Abbildung 27: MANTIS des Projektes SALT

7.5. Versionierung

Als Versionsverwaltungssystem wurde bei der Entwicklung von SALT „Subversion“ genutzt. Es ermöglicht, neben der Versionierung, den parallelen Zugriff mehrere Entwickler, die gleichzeitig an einem Projekt arbeiten, auf den Quellcode, ohne dass Konflikte bei der Verwaltung der Quelltexte entstehen. Außerdem wird durch eine Historienverwaltung ermöglicht, dass zu jeder Zeit der bisherige Stand des Quellcodes mit einer älteren Version verglichen oder ausgetauscht werden kann. Dadurch kann unter anderem nachvollzogen werden, wann ein Fehler in der Anwendung zum ersten Mal aufgetreten ist. [2]
Zusätzlich sind noch verschiedene Test in die Versionsverwaltung integriert, so dass der Quellcode bei Übermittlung auf die Kodierrichtlinien hin überprüft wird. Zusätzlich wird bei jeder Modifizierung der Quelldateien überprüft, ob die Änderung einem Eintrag des Bug-Tracking-Systems zugeordnet werden kann. Ist dies nicht der Fall wird die Übertragung nach Subversion nicht gestattet.

8. Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde das Programm SALT in Bezug auf die Aufgabenstellung bearbeitet. Ziel war es, die bestehende Applikation so zu erweitern, dass die Analyseergebnisse in geeigneter Weise, mit Berücksichtigung der kritischen Pfade, visualisiert werden konnten. Außerdem sollte bereits während der Dateieingabe das Systemdesign grafisch darstellbar sein.
Zur Bewältigung dieser Zielsetzungen, wurde im Laufe dieser Arbeit das Programm SALT um die Klasse des „GraphHelpers" erweitert. Sie verarbeitet die Elementinformationen des jeweiligen Projektes und generiert daraus den entsprechenden Graphen. In diesem Graphen werden die Zusammenhänge und Verknüpfungen grafisch aufgearbeitet und dargestellt.
Um diese generierten Graphen anzeigen zu können, wurde der „GraphDialog" entwickelt. Diese Klasse erweitert die grafische Benutzeroberfläche und ermöglicht die Darstellung des Graphen. Neben der Funktionalität des Vergrößerns und Verkleinerns, bietet er vor allem die Möglichkeit, Informationen einzelner Elemente aus dem als Bild vorliegendem Graphen zu erlangen. Diese Extraktion der Informationen wird mittels der Technologie der Image-Maps ermöglicht.
Darüber hinaus wurden im Zuge der Diplomarbeit noch einige Erweiterungen an SALT durchgeführt, so dass zum Beispiel eine isolierte Betrachtung einzelner Gruppenelemente in Form der Observationslevel möglich ist. Außerdem wurde der rekursive Analysealgorithmus durch ein iteratives Verfahren ersetzt. Zusätzlich besteht die Möglichkeit die Bezeichnung der Sicherheitslevel den verschiedenen Anforderungen anzupassen.

Für die weiterführende Entwicklung für SALT wären noch verschiedene Erweiterungen denkbar. Vorstellbar wäre ein rein grafisches Anlegen von Projekten und den einzelnen Elementen.

Zum Zeitpunkt der Beendigung dieser Diplomarbeit, befand sich die Applikation SALT in einer Testphase bei den Projektpartnern des Instituts für Verkehrsführung und Fahrzeugsteuerung. Dabei soll das Programm auf Alltagstauglichkeit und unter realitätsnahen Bedingungen getestet werden, um eventuelle Schwachstellen zu erkennen.

Im Allgemeinen kann jedoch gesagt werden, dass die Projektpartner mit dem Programm zufrieden sind und die verwendete Methodik und die Applikation im Rahmen einer Konferenz vorgestellt wurde.

C. Anhang

C.1. Inhalt der beigelegten CD

- Schriftliche Ausarbeitung der Diplomarbeit (PDF)
- Verwendete Quellen
- SALT-API (HTML)
- Quellcode:
    - „GraphHelper“-Klasse
    - „GraphDialog“-Klasse
    - „QImageMapLabel“-Klasse
    - „CMapElement“-Klasse
    - „CMapRectangle“-Klasse
    - „GroupElement“-Klasse
    - „ProjectGroup“-Klasse

D. Literaturverzeichnis

[1] **Balzert, Helmut (2001):** *Lehrbuch der Software-Technik - Band 1*, 2. Auflage, Spektrum Akademischer Verlag, Heidelberg, ISBN 3-8274-0480-0

[2] **Collins-Sussman, Ben - Fitzpatrick, Brian W. - Pilato, C. Michael (o.J.):** *Version Control with Subversion - For Subversion 1.2*

[3] **Ellson,. John u.a (2003):** *Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools,* AT&T Labs, Florham Park/USA

[4] **Gansner, Emden R. - North, Stephen C. (1999):** *An open graph visualization system and its applications to software engineering,* AT&T Labs, Florham Park/USA

[5] **Hagberg, Aric - Schult, Dan - Swart, Pieter (o.J.):** *NetworkX: Python Software for the Analysis of Networks,* Los Alamos National Laboratory, Los Alamos/USA

[6] **Hunter, John (2006):** *The Matplotlib User's Guide*

[7] **International Electrotechnical Commission (IEC) (2002):** *Functional safety and IEC 61508 - A basic guide,* IEC

[8] **Raggett, Dave - Le Hors, Arnaud – Jacobs, Ian (1999):** *HTML 4.01 Specification*, W3C

Internetquellen:

[9] **Boost.Python:** URL: http://www.boost.org/libs/python/doc/index.html

[10] **Epydoc:** *Epydoc Homepage,* http://epydoc.sourceforge.net/

[11] **Gato:** Graph Animation Toolkit, URL: http://gato.sourceforge.net/

[12] **Graphviz:** *Graphviz Homepage,* URL: http://www.graphviz.org/

[13] **Mantis:** *Mantis Bug Tracker,* http://www.mantisbt.org/

[14] **Matplot:**
Matplotlib / pylab - matlab style python plotting (plots, graphs, charts),
URL: http://matplotlib.sourceforge.net/

[15] **MySQL:** *MySQL AB :: Die populärste Open-Source-Datenbank der Welt, URL:* http://www.mysql.de/

[16] **NetworkX:** *NetworkX trac,* URL: https://networkx.lanl.gov/

[17] **PyDot:** *pyDot Homepage,*URL: http://dkbza.org/pydot.html

[18] **Pygraph:** URL: http://sourceforge.net/projects/pynetwork/

[19] **Pygraphlib**: URL: http://pygraphlib.sourceforge.net/

[20] **PyQt:** *Riverbank : PyQt : Overview,*
URL: http://www.riverbankcomputing.co.uk/pyqt

[21] **Python:** *Python Programming Language -- Official Website,*
URL: http://python.org/

[22] **RAMS Wolfgang H Baumann (2006):**
Was ist ein Funktionssicherheitssystem, Online in Internet: URL: http://www.rams.de/beratung/safety/61508/ [Stand:14.09.2006]

[23] **Trolltech:** *Trolltech Homepage,* URL: http://www.trolltech.com/