Entwurfsdokument

Schaltsysteme-Arbeitsblätter im Netz

Softwareprojekt 2018

Dr.-Ing. Heinz-Dietrich Wuttke
René Hutschenreuter
David Sukiennik

Alexander Beyer
Lars Hinneburg
Marcel Burkhardt
Sarah Löcklin
Eike Krieg
Linda Oppermann
Peter Klein
Stephan Enseleit

Inhalt

1 Anforderungsanalyse

1.1 Analyse funktionaler Anforderungen

SANE_FA_01 **Wertetabelle**

Es gibt einen View Wertetabelle mit einer konfigurierbaren Anzahl von Eingangs- und Ausgangsvariablen. Die Ausgänge sollen zwischen 1, 0 und * wechselbar sein. Auf der linken Seite kann man die Zeilenindizes (numerischen Wert der Eingangsbelegung) ablesen.

SANE_FA_02 **Karnaugh-Veitch-Diagramm**

Die Zentrale Boolesche Funktion wird in diesem View in einem Karnaugh-Veitch-Diagramm dargestellt. Dabei werden maximal 6 Eingangsvariablen unterstützt. Sollten es mehr als 6 Eingangsvariablen sein, muss der Nutzer eine Auswahl treffen. Auch die Ausgänge müssen ausgewählt werden. Die einzelnen Belegungen sind dabei auch per Klick oder Tap zwischen 1, 0 und * umschaltbar. Per Drag and Drop, Shift-Klick und Long-Press Tab können die Bits in Blöcken gruppiert werden. Die Blöcke können auch automatisch gebildet werden. Grundlage der Minimierung ist der Quine-McCluskey-Algorithmus.

SANE_FA_03 **Boolesche Ausdrucksalgebra**

Die zentrale boolesche Funktion wird in den vier gängigen Formen KKNF, KNF, KDNF und DNF dargestellt. Erfolgt die Eingabe einer Funktion in KNF oder DNF, so ist die Rekonstruktion der entsprechenden kanonischen Funktion nicht möglich. Erfolgt die Eingabe als KKNF oder KDNF wird die minimale Form mittels QMC berechnet. Änderungen in einer Funktion ändern auch den zentralen Datensatz.

SANE_FA_04 **Quine-McCluskey View**

Die durch QMC berechneten und ausgewählten Primimplikanten und Kürzungstabellen werden in dieser View dargestellt.

SANE_FA_05 **Funktionsindizes**

Die Eingabe der zentralen booleschen Ausdrücke ist als Funktionsindizes möglich. Die Ausdrücke werden durch Hexadezimalzahlen bestimmt.

SANE_FA_06 **Mengendiagramm**

Die zentrale Funktion wird in einem Mengendiagramm dargestellt. die Anzahl der Kreise richtet sich dynamisch nach den Ausgangsvariablen (maximal drei). Die Anzahl der Elemente richtet sich nach den Eingangsvariablen. Um die Übersichtlichkeit zu wahren, wird die Anzahl der auszuwählenden Variablen auf vier beschränkt.

SANE_FA_07 **Funktions-Hasards**

Es werden Funktions-Hasards in einem KV-Diagramm ohne Funktion dargestellt.

SANE_FA_08 **Programmierbare Strukturen**

Die boolesche Funktion ist als ROM, PLA und GAL darstellbar und veränderbar sein, wobei ein Klick auf einen Knotenpunkt diesen verändert.

SANE_FA_09 **Struktur-Hasards**

Struktur-Hasards innerhalb der gefundenen Blöcke im KV-Diagramm werden gefunden und vermieden. Für die Anzeige und Simulation der Schaltung wird die Schnittstelle zu BEAST verwendet.

SANE_FA_10 **QMC-Algorithmus**

Der Algorithmus soll als Wiederverwendbares Modul geschrieben werden.

SANE_FA_11 **Settings**

Im Settings-Menü ist werden globale Einstellungen getroffen, die View übergreifend genutzt werden können. Weiterhin sind die Zeichen für Konjunktion, Disjunktion, Negation, Äquivalenz, Antivalenz und Implikation änderbar, außerdem wird durch Invertierung der Farben ein Nacht-Modus geschaffen.

SANE_FA_12 **Progressive Webapplikation**

Die Seite lässt sich als Single-Page-Applikation (SPA) auf dem Endgerät installieren.

SANE_FA_13 **Export und Import des Anwendungszustandes**

Die Zentrale boolesche Funktion wird im *localStorage* des Browsers gespeichert.

SANE_FA_14 **Übergabe einer Startkonfiguration**

SANE kann eine Startkonfiguration übergeben werden, die den initialen Zustand festlegt. Hierbei werden bestimmte Views zugelassen und andere ausgelassen. Die boolesche Funktion wird direkt auf die mitgegebenen Daten angepasst.
Die Übergabe der Startkonfiguration erfolgt so, dass SANE über einen Link oder iFrame eingebunden werden kann.

SANE_FA_15 **Teachermodus**

Der Teachermodus wird über die Startkonfiguration aktiviert. Mit ihm kann verhindert werden, dass bestimmte Views die zentrale boolesche Funktion ändern. Stattdessen wird die Änderung überprüft, ob der Ausdruck der View mit dem Zentralen übereinstimmt.

SANE_FA_16 **Export im BEAST-Format**

Die zentrale boolesche Funktion wird ins BEAST-Format konvertiert und in einer von BEAST importierbaren Datei gespeichert.

SANE_FA_17 **Support von g-Parametern**

G-Parameter an sich nehmen sehr starken Einfluss auf alle Views, die zugrundeliegende globale Datenstruktur und die allgemeine Funktion des SANE-Projektes. Als Ausblick soll in der aktuellen Implementierung eine spätere Erweiterung um g-Parameter als umschaltbare Option berücksichtigt werden. Tabellen sollen eine Eingabe von mehreren möglichen Ausgängen und deren direkte Auswahl als aktiver Ausgang, sowie Karnaugh-Veitch-Diagramme die Eingabe eines logische g-Ausdruckes in eines der Felder etc. ermöglichen. Dieses Feature soll in diesem Softwareprojekt an sich noch nicht umgesetzt, aber vorbereitet und bedacht werden.

SANE_FA_18 **Spracheinstellungen**

Im gesamten Programm ist die Sprache zwischen Deutsch und Englisch mittels eines Schalters in den Settings umschaltbar. Die Spracheinstellungen können später noch durch weitere Sprachen ergänzt werden.

SANE_FA_19 **Dual-View-Mode**

Ist das Browserfenster hinreichend groß, werden zwei Views nebeneinander angezeigt. Änderungen in einer View werden sofort in der parallel angezeigten View dargestellt. Ist das Fenster nicht groß genug, wird automatisch zur Single-Page-Darstellung gewechselt.

1.2 Analyse nichtfunktionaler Anforderungen

SANE_NFA_01 **Polymer**

Um SANE erweiterbar zu halten, werden die verschiedenen Elemente als Web Components modularisiert. Die Komposition der Module entscheidet über die spätere Funktionalität. Die spätere Erweiterung mit weiteren Modulen ist möglich.

SANE_NFA_02 **Material Design**

Das Design folgt den durch Polymer bereitgestellten Elementen. Es wird auf das Material Design zurückgegriffen.

SANE_NFA_03 **Responsives Design**

Alle Erweiterungen sollen ein responsives Design umsetzen. Die Anwendung soll sich auf einem Desktop-Rechner, Tablet und Smartphone komfortabel bedienen lassen. Hierbei ist sowohl auf das Design, als auch auf die Bedienung mit Maus/ Tastatur oder Touchscreens zu achten.

SANE_NFA_04 **Datenfluss**

Der Datenfluss ist strikt unidirektional. Das bestehende Datenmodell wird genutzt und ggf. erweitert. Sämtliche Kommunikation findet von den Submodulen mittels Events statt. Änderungen des Anwendungszustandes werden mittels das Data-Bindings aus Polymer an die Submodule übergeben.
Dies garantiert eine mögliche spätere Erweiterung des Systems mit weiteren Funktionen.

SANE_NFA_05 **Separierung einer Event-Klasse**

Basierend auf dem Datenfluss des Programms werden Änderungen im Anwendungszustand (sog. Actions) von den einzelnen Modulen an das oberste Modul weitergegeben. Diese Actions sollen in eine separierte Klasse ausgelagert werden, um eine einfache Wiederverwendung zu ermöglichen.

SANE_NFA_06 **Modul für boolesche Operationen**

Alle mehrfach benötigten booleschen Funktionen sind möglichst atomar auszulagern, um eine Wiederverwendung zu vereinfachen.

SANE_NFA_07 **Nutzung von importHelpers**

Um duplizierten Code der von TypeScript benötigten Funktionen zu vermeiden, wird TypeScript mit der Compiler-Option *importHelpers* ausgeführt und die Laufzeitbibliothek *tslib* einmalig geladen werden. Dies benötigt eine geringere Prozessorleistung, dadurch ist SANE auch auf schwächeren Systemen performanter.

SANE_NFA_08 **TypeScript**

Der JavaScript-Code ist in TypeScript zu entwickeln. TypeScript ist eine typisierende Obermenge von JavaScript und kann sämtlichen JavaScript-Code lesen und TypeScript-Code in JavaScript-Code verschiedener Versionen kompilieren, um ihn für viele Endgeräte lesbar zu machen.
Es sollen mittels TypeScript alle Variablen, Klassen und Methoden typisiert sein, um das Risiko von Fehlern zu reduzieren.

SANE_NFA_09 **Moderne Plattformunabhängigkeit**

Der Quelltext soll in modernem JavaScript und mit aktuellem CSS entwickelt werden. Das entspricht ECMAScript 6 (auch ECMAScript 2015 oder ES6 genannt) und CSS 3. Mittels TypeScript können und sollen auch neuere JavaScript-Funktionen genutzt werden, da sie browser-kompatibel kompiliert werden. Als Referenzplattform fungiert die aktuelle Version des Browsers Chrome. Die Verwendung von JQuery wird explizit vermieden.

SANE_NFA_10 **Performanz**

Die Anwendung soll performant laufen, um auf allen verbreiteten Smartphones schnell ausführbar zu sein. Dies geschieht durch das Zwischenspeichern berechneter Werte, da dadurch unnötige Rechenoperationen vermieden werden.

2 Meilensteine

Phasen	**Deadline**
Planung	
Freigabe Lastenheft	03.04.2018
Freigabe Pflichtenheft	02.05.2018
Entwurf	
Festlegung des Systemaufbaus	02.05.2018
Bestimmung des Aufbaus der Komponenten und deren Beziehungen	02.05.2018
Festlegung des Datenflusses	02.05.2018
Realisierung	
Das System für den Datenfluss wurde implementiert.	10.05.2018

Das Datensegment der zentralen booleschen Funktion und die Wertetabelle-View wurden als Grundlage für die weitere Entwicklung implementiert.	10.05.2018
Der QMC-Algorithmus und die Realisierung dessen in den Views wurden implementiert.	13.05.2018
Alle Views wurden erstellt und sind funktionsfähig und fehlerfrei.	03.06.2018
Die Schnittstellen für Import/Export, BEAST und die Startkonfiguration wurden eingefügt und getestet.	03.06.2018
Alle Elemente der Anwendung wurden an ein einheitliches Erscheinungsbild angeglichen.	10.06.2018
Alle Musskriterien wurden bearbeitet.	10.06.2018
Je nach verbleibender Zeit wurden weitere Wunschkriterien realisiert	24.06.2018
Nach einem abschließenden Test wurden letzte Korrekturen vorgenommen.	01.07.2018
Auslieferung	
Veröffentlichung der Anwendung über die Server der Auftraggeber	03.07.2018
Veröffentlichung des Benutzerhandbuchs und weiterer notwendiger Dokumente	03.07.2018

3 Risikoanalyse

Da das Unified Process Model risikoorientiert[1] arbeitet, müssen wir uns auch mit möglichen kritischen Problemen auseinandersetzen. Wenn wir zum Beispiel bei der Implementierung von Kernfeatures unserer Webseite auf große oder uns noch unbekannte Probleme stoßen, können diese mehr Zeit beanspruchen als vorgesehen. Es besteht somit über die gesamte Entwicklungszeit des Projekts die Möglichkeit, dass wir unsere Aufgaben nicht rechtzeitig erfüllen können und wir hinter unserem Zeitplan liegen. Die verschiedenen Meilensteine sind wichtig für unsere Planung und sind dadurch auch für eine Risikoanalyse und -abschätzung entscheidend. Die grundlegenden Klassen, welche alle Views brauchen, sind entscheidend für die spätere Entwicklung. Sie sind der riskanteste Teil der Realisierung und können die weitere Planung und im schlimmsten Fall das gesamte Projekt gefährden. Daher ist es wichtig, zuerst diese grundlegenden Features fehlerfrei zu implementieren. Nichtsdestotrotz haben auch alle anderen

[1] siehe A. Zimmermann / Softwaretechnik

Aufgaben in der Realisierung das Risiko, anspruchsvoller zu sein, als zunächst erwartet. Aufgrund der genannten Risiken wird bei jeder Aufgabe 50 % der Zeit, die voraussichtlich benötigt wird, auf die erwartete Zeit aufgeschlagen, um sicherzustellen, dass die Aufgabe in der gegebenen Zeit bearbeitet werden kann.

4 Entscheidung für das Vorgehensmodell

Wir haben uns für das iterative Vorgehensmodell Unified Process entschieden. Es bietet sich aus mehreren Gründen für unsere Arbeit an. Da das Modell eine iterative Arbeitsweise vorsieht, werden wir in der Implementierungsphase dementsprechend vorgehen und einzelne Elemente getrennt voneinander implementieren und auch testen. Eine weitere Eigenschaft des Unified Process Modells ist die Anwendungsgetriebenheit. Unser Vorgehen wird ebenfalls darauf beruhen, dass wir als ersten Schritt eine Grundstruktur unseres Programms aufbauen, welche wir nach und nach mit neuen Features erweitern werden, sobald sie entworfen und implementiert wurden. Diese Arbeitsweise bezeichnet man als evolutionär, was auch eine Eigenschaft des Vorgehensmodells ist. Durch das Unified Process Modell sind wir auch in der Lage, die einzelnen Phasen der Entwicklung zeitlich anzupassen, falls wir in einer Phase zeitliche Probleme bekommen sollten.

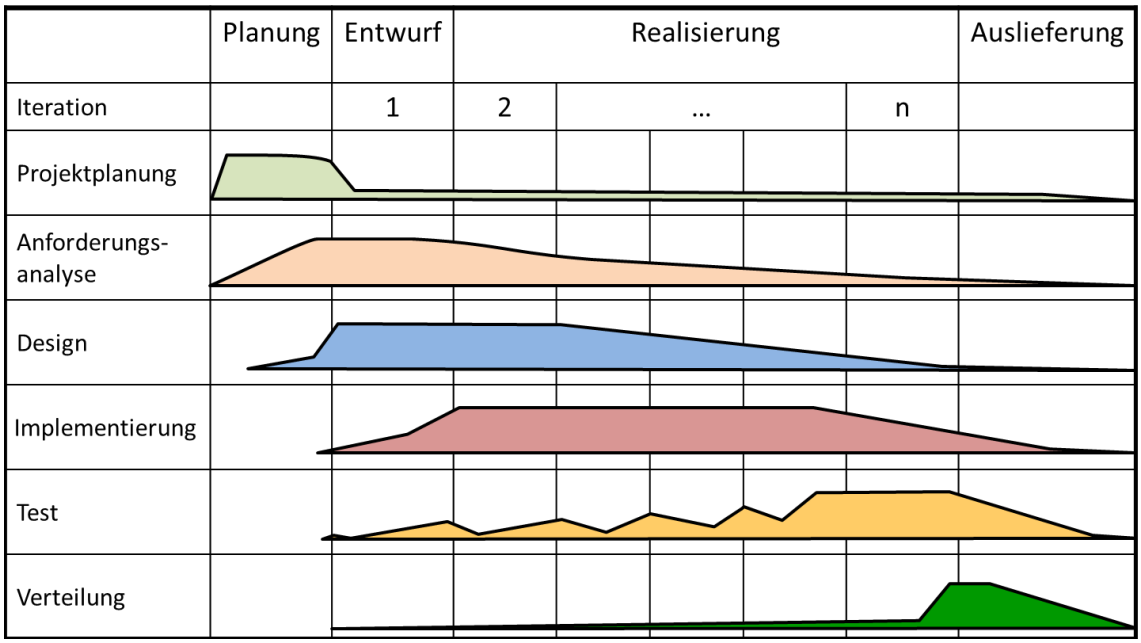

Abbildung 4-1: Unified Process-Phasenablaufdiagramm[2]

[2] https://www.tu-ilmenau.de/sse/lehre/softwareprojekt/projektablauf/

Die parallel ablaufenden Kernprozesse müssen nicht strikt wie in Abbildung 4-1 eingehalten werden, sondern können an das jeweilige Projekt angepasst werden. Auch ist es möglich, auf Änderungen im Verlauf des Projekts zu reagieren, wenn zum Beispiel die Implementierungsphase länger dauert als geplant. Die Projektplanung zu Beginn des Projekts ist nicht so hoch wie beispielsweise beim Wasserfallmodell, jedoch werden wir erneute Planungen für spätere Iterationen vornehmen müssen. Ein Nachteil des Unified Process Modells ist die nicht integrierte Qualitätssicherung, welchen wir aber durch periodische Tests ausgleichen werden[3]. Durch die Definition von Meilensteinen sind wir in der Lage, unseren Fortschritt im Auge zu behalten und mögliche Änderungen in der Planung entsprechend durchzuführen.
Wir werden uns an den Phasen und ihrer Dauer aus Abbildung 4-1 orientieren, jedoch halten wir es für möglich, dass wir die Projektplanung eventuell zu einem späteren Zeitpunkt erneut benötigen könnten, falls dies erforderlich wird. Wie in der Abbildung zu sehen, werden wir uns frühzeitig auf ein einheitliches Design festlegen und immer wieder Tests durchführen.

5 Entwurfsalternativen

Wir haben uns nicht für das Wasserfallmodell entschieden, da es unserer Meinung nach zu starr ist. Es ist zu unflexibel für eventuelle Änderungen in der Planung. Das Wasserfallmodell geht von einem Schritt direkt in den Nächsten. Wir möchten jedoch zum Beispiel Teile des fertigen Programms bereits während der Implementierung testen und so Fehlern, welche später erkannt werden, entgegenwirken.
Das agile Vorgehen wiederum ist uns zu ungenau. Die einzelnen Phasen sind nicht klar genug strukturiert und sind zu schnell. Die vorgegebenen Programmiersprachen und Tools in unserem Projekt waren uns komplett neu, weswegen wir einen Teil der ersten Phase des Projekts ausschließlich zur Einarbeitung in die Sprachen und Konzepte benötigt haben. Dieses Vorgehen wäre in einem agilen Vorgehensmodell nicht möglich gewesen, da man in so einem Modell schon in der ersten Phase grobe Züge des fertigen Programms implementiert. Wir haben uns für den Unified Process entschieden, weil er einen guten Kompromiss zwischen dem konservativen und starren Modell des Wasserfalls und dem sehr schnellen und ereignisgetriebenen agilen Vorgehensmodell darstellt. Wir haben somit eine gewisse Struktur über den Verlauf des Projektes hinweg und können aber auch auf Veränderungen reagieren und unsere Planung anpassen.

[3] vgl. A. Zimmermann / Softwaretechnik

6 Entwurfsziele

1. **Korrektheit:** Alle Funktionen sollen ihre Aufgabe wie vorgesehen erfüllen.
2. **Zuverlässigkeit**: Nach dem Aufrufen der Webseite soll das System problemlos laufen. Eventuell auftretende Fehler sollen die Funktionsfähigkeit des Systems nicht beeinflussen.
3. **Erweiterbarkeit**: Das System soll im Zwecke einer zukünftigen Entwicklung erweiterbar sein. Mögliche Erweiterungen sind:
   - weitere Views,
   - weitere Minimierungsverfahren,
   - g-Parameter und
   - Anbindung an andere Systeme wie Moodle und BEAST.

4. **Veränderbarkeit**: Es soll möglich sein das bestehende System zu verändern. Mögliche Änderungen sind:
   - Aktualisierung der genutzten Bibliotheken,
   - Erhöhung der Performanz und
   - Modernisierung des GUI.
5. **Verständlichkeit**: Laufende Kommentare im Code und eine ausführliche Dokumentation sollen das Verständnis des Systems für Entwickler ermöglichen.
6. **Performanz**: Die Anwendung soll auf allen Endgeräten leistungsfähig und zügig laufen.
7. **Nutzerfreundlichkeit**: Das GUI muss intuitiv und flüssig bedienbar sein. Auf auftretende Fehler und ungültige Eingaben wird der Nutzer hingewiesen.
8. **Portierbarkeit**: Solange entsprechende Browser unterstützt werden, ist SANE unabhängig vom Betriebssystem und der Hardware nutzbar.

7 Architekturmuster

Die Systemarchitektur von SANE ist stark an das MVC-Modell angelehnt. Dies ergibt sich durch den redux-ähnlichen Datenfluss. Redux sieht den Anwendungszustand in einem zentralen Element gespeichert vor, eine Änderung von diesem ist nur durch Funktionen des Datenobjektes selber möglich.
Die einzelnen Views kommunizieren als Submodule mit dem Hauptmodul des Systems. Dieses beherbergt alle nötigen Event-Handler, welche je nach Bedarf in der Lage sind, Funktionen des Datenelements aufzurufen.

Daraus ergibt sich die SANE-View als View, die übergeordnete SANE-App mit den Event-Handlern als Control und das SANE-Data-Element als Model.

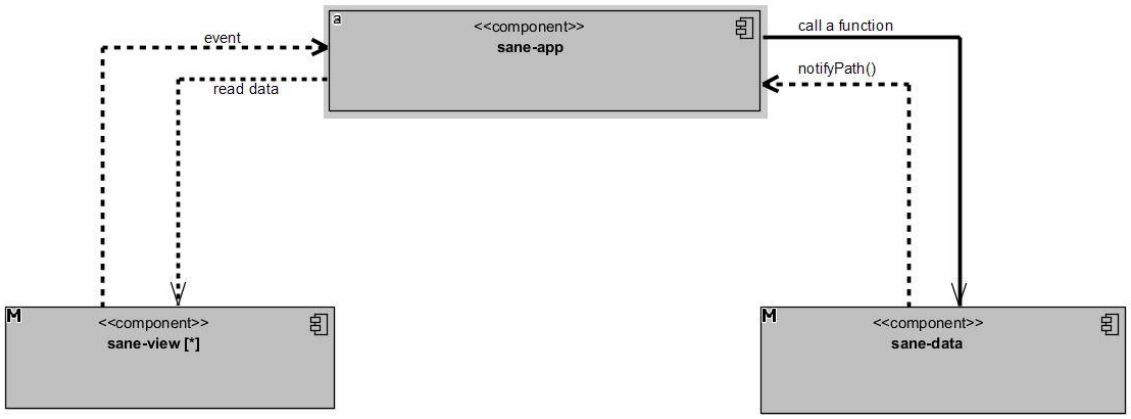

Abbildung 7-1: SANE-MVC-Modell

8 Systemzerlegung

8.1 SANE

Wenn ein Objekt einer SANE-View ein Event aussendet, löst dieses den zugehörigen Event-Listener in der SANE-App aus. Dieser wiederum kann eine Funktion des SANE-Data-Objektes aufrufen. Wenn in diesem Fall Daten geändert werden und dies auch durch das Datenobjekt bekannt gegeben wird, werden alle Objekte, welche ein Data-Binding auf die geänderten Daten haben, benachrichtigt und erhalten die neuen Daten.

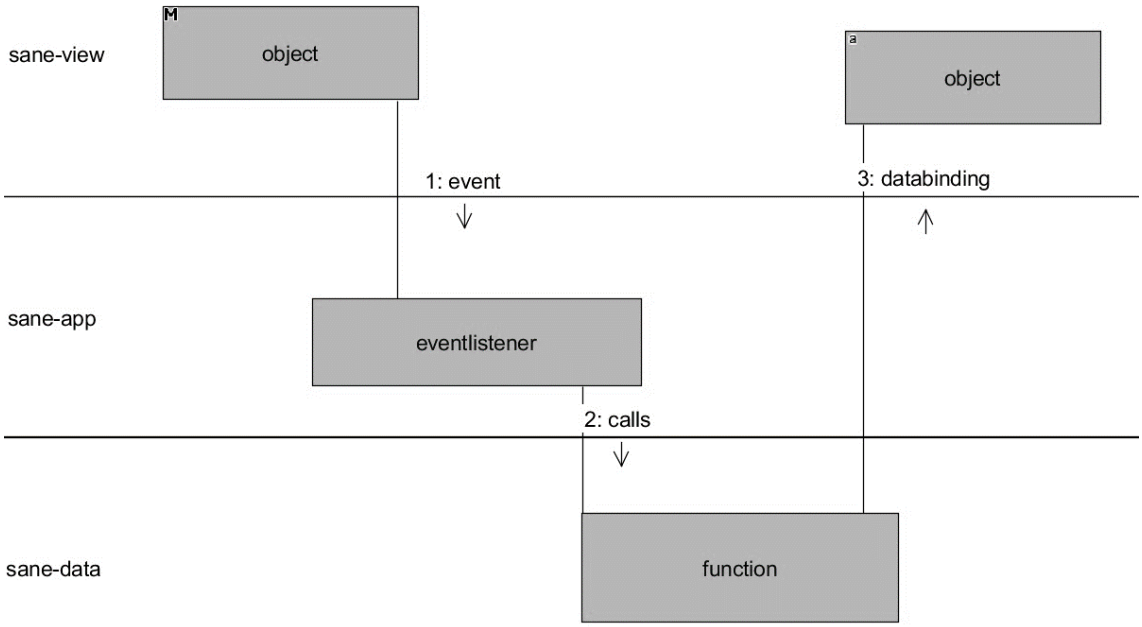

Abbildung 8-1: SANE-Communication

Dem Hauptmodul der SANE-App wird jederzeit genau ein SANE-Data Objekt zugeordnet, zudem kann es beliebig viele SANE-Views zugeordnet bekommen.

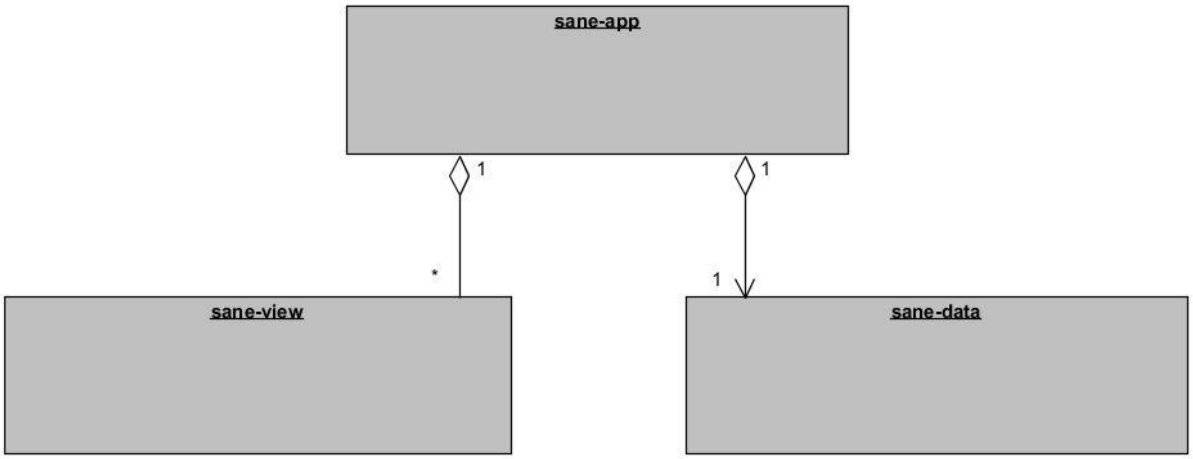

Abbildung 8-2: SANE-Composit

8.2 Entwurf der funktionalen Anforderungen

Datenobjekt

Das SaneData-Objekt lässt sich sehr gut mit einer Wertetabelle vergleichen, welche zusätzlich zu der Anzahl der Eingangs- und Ausgangsvariablen auch über eine Spalte für h* und eine Spalte für die Ausgangsbelegungen als Index verfügt. Über ein Interface sollten andere Klassen auf Variablen wie Input-Spalten, Output-Spalten und die einzelnen Zeileninhalte zugreifen können. Die einzelnen Zeilen sind als Number-Array zu implementieren. Benötigte Konstanten im SaneData-Objekt sind unter anderem Limits für die Anzahl der Ein- und Ausgangsvariablen. Weitere benötigte Variablen sind die Anzahl der Input- und Output-Spalten, sowie die Zeilenanzahl.

Prinzipiell notwendig sind Funktionen wie *setMask()* und *toggleMask()* für h*, Methoden zum Inkrementieren und Dekrementieren der Ein- und Ausgangsvariablen, *toggleBit(),* um einzelne Bits zu ändern, und *computeNumRows(),* damit beim Inkrementieren und Dekrementieren der Eingangsvariablen die Zeilenzahl berechnet werden kann.

Des Weiteren sind das Mitgeben einer Startbelegung und einer reset-Funktion notwendig.

Wertetabelle

Für die Umsetzung der Wertetabelle müssen folgende Use-Cases bedacht werden: das Hinzufügen und Entfernen von Eingangs- und Ausgangsvariablen, die Eingabe eines gewünschten Ausgabewerts, die damit verbundene Umsetzung von nicht-

deterministischen Ausgabebelegungen und das Umschalten von einem bestimmten Ausgangsbit und h*.
Dafür sind unter anderem folgende Events von der View auszulösen: *incOutputVariable(), decOutputVariable(), incInputVariable(), decInputVariable(), setOutput(), toggleBit()* und *toggleMask().*
Die Events müssen an die entsprechenden HTML-Elemente gebunden werden.
Im Hauptmodul *sane-app* sind Event-Listener für die Events zu implementieren. Die Funktionen, die daraufhin im SaneData-Objekt benötigt werden - zusätzlich zu den für die Events impliziten Funktionen, sind unter anderem *computeNOutputColumns()* und das dafür benötigte *getBitWidth().*
HTML-seitig wird die Wertetabelle in einer dynamischen HTML-Tabelle als Polymer-Element umgesetzt.

Karnaugh-Veitch-Diagramm

Das KV-Diagramm weist in der Implementierung große Ähnlichkeit zu der Wertetabelle auf. HTML-seitig kann es ebenfalls als Tabelle umgesetzt werden. Es muss jedoch um ein Element erweitert werden, welches ein Bit zwischen 1,0 und * umschalten lässt. Das Event *toggleField()* lässt sich dafür z. B. mit der Funktion *rotate()* verbinden, welche zwischen den notwendigen Werten rotiert. Um das Diagramm richtig zu initialisieren, muss mittels *getBit()* die Belegungen der Indizes vom SaneData-Objekt ermittelt und diese in das entsprechende Tabellenfeld geschrieben werden.
Die Minimierung findet mit Hilfe der QMC-Klasse statt. Zusätzlich wird jedoch noch eine Funktion *computeCluster()* zur Ermittlung der Blöcke benötigt.
Es kann immer nur eine Ausgangsvariable gleichzeitig angezeigt werden. Die Auswahl erfolgt mittels Checkbox oder Dropdown-Menü.

QMC-View

Die QMC-View stellt die durch den QMC-Algorithmus berechneten Kürzungstabellen für den Nutzer übersichtlich dar. Die Daten, die zum Füllen der Tabelle nötig sind, können mit *getTable()* abgerufen werden. Die darzustellende Ausgangsvariable wird durch ein Dropdown Menü oder eine Checkbox ausgewählt.
Die Tabellen können, wie schon die Wertetabelle, als dynamische HTML-Tabelle in einem Polymer Element verwendet werden.

QMC-Algorithmus (Klasse)

Der QMC-Algorithmus berechnet auf Grundlage des SaneData-Objektes einen minimalen Ausdruck für die Ausgangsvariablen. Da der QMC-Algorithmus für eine große Anzahl von Eingangsvariablen im schlechtesten Fall eine exponentielle Laufzeit benötigt,

ist es notwendig, nur die relevanten Änderungen zu lokalisieren und zu berechnen. Dies ist erforderlich, damit die Anwendung auch auf schwacher Hardware performant laufen kann.
Die Klasse soll außerdem eine Funktion *getTable()* implementieren, welche die Kürzungstabellen für die QMC-View ausgibt. Nach der Terminierung ist der Event-Handler zu informieren und die minimale Funktion an *sane-expression* zu übergeben.

Settings

Die Settings können als Side-Navigation, Toolbar-Menü oder Kontextmenü dargestellt werden.
Die Definition der in der booleschen Algebra verwendeten Symbole für die Operatoren erfolgt in Textboxen. Diese Symbole werden auf ein Zeichen beschränkt, weiterhin sollen View-übergreifende Einstellungen getroffen werden können. Hierzu gehört unter anderem die Sprache der Anwendung. Zur Änderung dieser wird aktuell ein paper-toggle-Button verwendet. Für die Unterstützung weiterer Sprachen wäre die Verwendung einer Auswahlliste nötig.
Änderungen in den Einstellungen werden entsprechend durch den Event-Handler an die betroffenen Stellen weitergegeben.

Spracheinstellung

Aktuell kann zwischen Deutsch und Englisch gewechselt werden. Um die Programmierung in den einzelnen Script-Dateien nicht unnötig zu verkomplizieren, werden mithilfe einer *locales.json* die verwendeten Begriffe für alle gewünschten Sprachen hinterlegt.
In den Views wird der Begriff in der *locales.json* mit der Funktion "*{{localize('Begriff')}}*" abgefragt und entsprechend auf der Webseite dargestellt.

Funktions-Hasards

In dieser View werden Funktions-Hasards in einem Karnaugh-Veitch-Diagramm dargestellt. Zur Darstellung der Diagramme kann die gleiche Funktion wie in der View des KV-Diagramms verwendet werden, allerdings müssen die Funktionen zur Änderung von Werten nicht übernommen werden.
Die Hasards werden von einer Methode *computeHasard()* berechnet, welche ein Array mit den Eingangsbelegungen zurückgibt. Zu diesen Belegungen lassen sich die Koordinaten im Diagramm mittels *computeCoordinate()* berechnen. Diese werden von einer weiteren Methode *computePath()* zur Berechnung der Linienverläufe verwendet und von einer weiteren Methode *drawPath()* gezeichnet.

Übergabe einer Startkonfiguration

Zum externen Einbinden von SANE soll es möglich sein, die Software mit Startparametern zu versehen. Für diese Verwendung ist der Einsatz von URL-Parametern sinnvoll. Um eine zielgerichtete Manipulation der Startparameter zu erschweren, werden die Parameter verschlüsselt.

Nach dem Einlesen werden die Parameter mittels einer Methode dekodiert und ausgewertet. Dem Konstruktor des SaneData-Objektes werden diese mitgegeben. Die durch die Parameter gesetzten Einstellungen müssen ebenfalls zentral festgelegt werden. Für alle globalen Einstellungen bietet sich dabei eine eigene Klasse zur Verwaltung der Settings an.

Mengendiagramm

Das Mengendiagramm berechnet aus dem SaneData-Objekt die Bits der ausgewählten Ausgangsvariablen (maximal drei). Diese werden dann den Eingangsbelegungen (maximal 16, also vier Eingangsvariablen) zugeordnet. Mittels einer drawSet()-Methode werden schließlich die Mengendiagramme dargestellt. View-spezifisch sind dabei die verwendeten Ausgangsvariablen auszuwählen. Dies beeinflusst unter anderem auch die Anzahl der dargestellten Mengen.

Das Verschieben der Eingangsvariablen zwischen den Mengen wird über eine in die View eingebaute Methode realisiert, welche auch die Änderung in der Belegung für das SaneData-Objekt anpasst.

Boolesche Ausdrucksalgebra

Diese View benötigt einen Parser bzw. Interpreter, um die Eingaben des Nutzers in die Textboxen zunächst auf Korrektheit zu überprüfen und danach in ein für die Berechnungen genutzte Struktur zu übersetzen. Dazu kann unter anderem die Funktion *convertToExpression()* genutzt werden. Über die eingebundene QMC-Klasse kann dann aus der Struktur die DNF, KNF, KDNF und KKNF berechnet werden.

Bei Änderung des SaneData-Objektes in anderen Views wird die Übersetzung und Berechnung ebenfalls vorgenommen.

Es kann immer nur eine Ausgangsvariable für die Anzeige und Berechnung verwendet werden. Die Auswahl erfolgt mittels Checkbox oder Dropdown Menü.

Die berechneten Gleichungen werden dann in die übrigen Textboxen eingetragen. Das SaneData-Objekt wird ebenfalls aktualisiert.

Funktionsindizes

Die View zeigt beim Aufrufen die aktuellen Funktionsindizes für alle Ausgangsvariablen an - zum Umrechnen wird eine Funktion *binaryToHex()* benötigt. Die in jeweils einer

eigenen Textbox angezeigten Funktionsindizes können verändert werden und nach Bestätigung der Eingabe findet eine Berechnung mit *hexToBinary()* statt. Bei jeweiliger Interaktion mit den Elementen wird unter anderem das Event *submitFunctionIndex()* benötigt.

Progressive Webapplikation

Die Einbindung der Funktionalitäten für eine progressive Webapplikation sollte einfach sein, da bereits in Polymer die Polymer-App-Toolbox vorhanden ist, welche die Erstellung einer progressiven Webapplikation sehr vereinfacht. Die Komponenten von Polymer in dieser App-Toolbox sind bereits so entwickelt, dass sie eine Progressivität unterstützen.

Programmierbare Strukturen

Hier soll es möglich sein die Werte mittels eines PLA, GAL oder ROM darzustellen. Dafür ist eine Methode nötig, die die Belegung einer Ausgangsvariable auf den ausgewählten Logikbaustein abbildet. Für die Darstellung des Bausteins wird eine SVG verwendet. Es wird eine Funktion benötigt, die Änderungen in einer anderen View auf die Knotenpunkte anpasst. Außerdem wird eine Funktion benötigt, die die Änderungen, die durch einen Klick auf die Knotenpunkte entstehen, über ein Event und an den Listener und dann an das SaneData-Objekt überträgt.

9 Management persistenter Daten

SANE speichert den Anwendungszustand automatisch im *localStorage* des Browsers. Die Funktion ist in allen gängigen Browsern verfügbar. Dadurch wird das Speichern auf allen Endgeräten mit einem aktuellen Browser möglich sein.
Es können circa fünf Megabyte gespeichert werden. Zur Laufzeit der Anwendung kann davon ausgegangen werden, dass diese Grenze nicht überschritten wird, da nur einzelne Arrays mit der zentralen booleschen Funktion abgespeichert werden. Der *localStorage* ermöglicht durch das Abspeichern auch eine direkte Schnittstelle zu BEAST.
Dank des *localStorage* ist es möglich auf das Speichern von Cookies zu verzichten.

10 Verwendete Technologien und Entwicklungswerkzeuge

verwendete Programmiersprachen

- TypeScript, welches eine typisierte Obermenge von JavaScript ist

verwendete APIs

- verschiedenste Web APIs (z. B. DOM)

verwendete Bibliotheken

- Polymer
- Web Components (z. B. iron-icon, paper-button, app-drawer)

verwendete Tools

- Kommunikation: Slack
- Paketmanager: npm, bower
- Versionsverwaltung: SVN
- IDE: PHPStorm
- Bug-Tracking: Trello
- Datenaustausch: Google Drive
- Terminverwaltung: Trello

11 Quellenverzeichnis

A. Zimmermann/Softwaretechnik WS 17
Armin Zimmermann: Softwaretechnik-Vorlesungsskript
Ilmenau, WS 2017/18