Brandenburgische Technische Universität Cottbus – Senftenberg
Lehrstuhl Programmiersprachen und Compilerbau

Bachelorarbeit

Entwicklung einer interaktiven, graphischen Benutzeroberfläche für den constraint-basierten Konfigurator RangeConfig

Andreas Fehn

6. Mai 2014

Gutachter: Prof. Dr. rer. nat. habil. Petra Hofstedt
Betreuer: Peter Sauer, M. Sc.

Inhaltsverzeichnis

1 Einleitung

Die heutige Welt ist stark vernetzt und globalisiert. Die Mehrzahl der über sieben Milliarden Menschen auf der Erde will konsumieren und benötigt Güter aller Art. Die Industrie kann diese große Nachfrage nur durch Massenfertigung decken. Allerdings muss die Industrie nicht nur große Stückzahlen von Produkten zur Verfügung stellen, sondern auch auf individuelle Wünsche der Kunden eingehen, um wettbewerbsfähig bleiben zu können. Dies klingt zunächst nach zwei unvereinbaren Größen.

Das Dilemma lässt sich aber mit dem Konzept der *kundenindividuellen Massenproduktion* lösen. Wie der Name erahnen lässt, bedeutet dies, Güter mit einer Effizienz zu produzieren, welche an die der Massenfertigung herankommt, und mit diesen Gütern trotzdem individuelle Bedürfnisse verschiedener Kunden zu erfüllen [vgl. TJ01]. Eine hohe Effizienz wird erreicht, indem einzelne Teile eines Produkts mittels Massenproduktion gefertigt werden und durch eine individuelle Kombination der Einzelteile das gewünschte Produkt entsteht. Somit bleiben die Produktionskosten gering, obwohl individuelle Produkte angefertigt werden [vgl. Pil00].

Kundenindividuelle Massenproduktion kann durch die Verwendung von Produktkonfiguratoren erreicht werden. Ein Konfigurator ist ein Werkzeug, mit dessen Hilfe Produkte oder Dienstleistungen individuell zusammengestellt werden können. Danach kann die persönliche Kombination bei einem Hersteller in Auftrag gegeben werden.

Es existieren Konfiguratoren für verschiedene Anwendungsgebiete. Darunter fällt zum Beispiel die individuelle Ausstattung eines Pkw oder eines PCs. Konfiguratoren dafür können zum Beispiel mit *Feature-Modellen* arbeiten. Diese Modelle bilden alle Merkmale eines Produktes oder einer ganzen Reihe von Produkten in einer Baumstruktur ab. Für jedes Merkmal gibt es nur endlich viele verschiedene Auswahlmöglichkeiten. Ein Beispiel für so einen Konfigurator ist *FdConfig*, welcher an der Brandenburgischen Technischen Universität Cottbus (BTU) entwickelt wurde [vgl. SG11].

Diese Art von Konfiguratoren haben aber einen entscheidenden Nachteil. Weil sie nur auf endlichen Wertemengen arbeiten können, ist die Handhabung von Intervallen sehr umständlich. Jeder mögliche Wert in einem Intervall muss einzeln in einer Liste gespeichert werden, was bei großen Intervallen sehr ineffizient ist. Die Verarbeitung von reellen Intervallen wird dadurch sogar unmöglich, da eine unendliche Liste nicht in endlicher Zeit erzeugt werden kann und deshalb nicht alle reellen Werte zwischen zwei Zahlen betrachtet werden können. Eine weitere Schwachstelle besteht

in der Darstellung. Diejenigen Teile der graphischen Oberfläche, welche Intervalle darstellen oder es dem Benutzer ermöglichen Werte aus einem Intervall auszuwählen, können durch große Intervalle mit vielen Werten sehr unübersichtlich werden.

Aus diesen und anderen Gründen soll ein neuer Konfigurator entwickelt werden, mit welchem allgemeine Konfigurationsprobleme gelöst werden können. Dabei sollen sowohl beschränkte als auch unbeschränkte Intervalle unterstützt werden, die nicht nur ganze Zahlen sondern sogar reelle Zahlen enthalten können und effizient verarbeitet und dargestellt werden.

1.1 Aufgabenstellung

Ziel dieser Bachelorarbeit ist die Entwicklung einer graphischen Benutzeroberfläche für den constraint-basierten Konfigurator RangeConfig. Die Funktionalität des Konfigurators, das *Backend*, wird im Rahmen einer weiteren Bachelorarbeit von *Andreas Lindner* am Lehrstuhl Programmiersprachen und Compilerbau an der BTU entwickelt. Die beiden Arbeiten sind aufeinander abgestimmt und laufen zeitlich parallel ab.

Die Idee für RangeConfig entstammt der Konfiguration von Cyber Physical Systems (CPS). Daher ist dies auch der erste konkrete Anwendungsbereich für das Programm. Eine eingehende Einführung in das Thema CPS wird in Abschnitt 2.2 gegeben. Die graphische Oberfläche soll den Benutzer beim Konfigurationsprozess unterstützen, weshalb besondere Anforderungen zu beachten sind. Sie muss es dem Benutzer erlauben, verschiedene Komponenten zu einem komplexen System zusammenzufügen, welche verschiedene Modi und Eigenschaften besitzen. **Anmerkung:** Mehrere Modi werden derzeit vom Backend nicht unterstützt. In der Benutzeroberfläche sind alle Voraussetzungen vorhanden, aber deaktiviert. Deshalb kann in Abschnitt 5.2 keine Abbildung mit mehreren Modi gezeigt werden.

Weitere Anmerkung: Da das erste konkrete Anwendungsgebiet von RangeConfig die Konfiguration von CPS war, ist die komplette Anwendung an die Terminologie der CPS angelehnt. Beispielsweise macht der Begriff „Komponente“ nicht bei allen Konfigurationsproblemen Sinn. Besser wäre eine Bezeichnung für eine Sammlung von Eigenschaften. Trotzdem werden in dieser Bachelorarbeit die CPS-Begriffe verwendet, da sie sich im Laufe des Entwicklungsprozesses durchgesetzt haben.

RangeConfig verwendet im Hintergrund ECL^{i}PS^e Prolog, welches eine Implementierung der logischen Programmiersprache *Prolog* mit einem integrierten System zur Lösung von Constraints ist. Es werden Constraints verwendet, um den Wertebereich von Variablen einzuschränken, wie später gezeigt wird. ECL^{i}PS^e bietet verschiedene Bibliotheken an, die unterschiedlichen Zwecken dienen. Die IC-Bibliothek unterstützt die Verarbeitung von Constraints auf diskreten und kontinuierlichen Intervallen und eignet sich deshalb perfekt für die Anwendung auf CPS [vgl. Hö+13].

1.2 Übersicht über den Aufbau der Arbeit

Wie im vorhergegangenen Abschnitt dargelegt, ist die Hauptaufgabe dieser Bachelorarbeit die Erstellung einer graphischen Benutzeroberfläche für den Produktkonfigurator RangeConfig. In Kapitel 2 wird eine Einführung in die interaktive Produktkonfiguration gegeben (Abschnitt 2.1) und das zugrundeliegende mathematische Modell erläutert (Unterabschnitt 2.1.1). In Unterabschnitt 2.1.2 werden einige wichtige Eigenschaften von interaktiven Produktkonfiguratoren herausgearbeitet. Außerdem wird in Abschnitt 2.2 erklärt, was genau unter einem CPS zu verstehen ist.

Kapitel 3 dient dem Verständnis für graphische Benutzeroberflächen. Dazu werden zunächst einige generelle Anforderungen an graphische Oberflächen gestellt (Abschnitt 3.1). Dabei werden Maßstäbe (Unterabschnitt 3.1.1) und Richtlinien für Benutzeroberflächen (Unterabschnitt 3.1.2) von Shneiderman und Plaisant [SP10] thematisiert. Im Anschluss wird in Abschnitt 3.2 auf das wichtige Gebiet der Nebenläufigkeit eingegangen. Beendet wird das Kapitel mit einer Aufschlüsselung der Anforderungen an die graphische Oberfläche von RangeConfig (Abschnitt 3.3).

In Kapitel 4 wird das verwendete Datenmodell (Abschnitt 4.1) vorgestellt und der Aufbau der Vorlagen für Komponenten erläutert (Abschnitt 4.2).

In Kapitel 5 wird die Entscheidung für die verwendete Bibliothek zur Erstellung graphischer Benutzeroberflächen begründet (Abschnitt 5.1). Im Zuge dessen wird der strukturellen Aufbau eines Programms mit dieser Bibliothek erläutert (Unterabschnitt 5.1.1). Die Entscheidung hat auch Einfluss auf die Implementierung der Kommunikation zwischen der Oberfläche und dem Backend, deren Details in Unterabschnitt 5.1.2 zu finden sind. Im weiteren Verlauf dieses Kapitels wird der Aufbau einiger wichtiger Fenster der Oberfläche gezeigt (Abschnitt 5.2) und schließlich die Einhaltung der in Abschnitt 3.3 aufgestellten Anforderungen an die graphische Oberfläche von RangeConfig analysiert (Abschnitt 5.3).

In Kapitel 6 wird die bewältigte Arbeit zusammengefasst. Außerdem wird ein Ausblick auf mögliche Verbesserungen und Arbeiten gegeben, welche die graphische Benutzeroberfläche weiter verfeinern können.

2 Interaktive Produktkonfiguration und Cyber Physical Systems

Dieses Kapitel gibt eine Einführung in das Thema der Produktkonfiguration. Dabei wird zunächst in Abschnitt 2.1 erklärt, was unter einem Produktkonfigurator zu verstehen ist und welchen Nutzen die Verwendung eines solchen hat. In Unterabschnitt 2.1.1 wird auf das Konfigurationsproblem eingegangen und in Unterabschnitt 2.1.2 werden einige Eigenschaften von Produktkonfiguratoren herausgearbeitet. Außerdem werden in Abschnitt 2.2 Cyber Physical Systems näher erläutert, deren Konfiguration eines der Anwendungsgebiete von RangeConfig darstellt.

2.1 Einführung in die Produktkonfiguration

Wie bereits in der Einleitung erwähnt, werden unter anderem von Unternehmen Produktkonfiguratoren eingesetzt, mit deren Hilfe Kunden individuelle Produkte gestalten können. Diese Produkte können dann entweder komplett im Rahmen der Massenfertigung produziert werden oder sie bestehen aus Einzelteilen, welche wiederum durch Massenfertigung hergestellt und individuell kombiniert werden können.

Im Laufe des Konfigurationsprozesses erwartet der Konfigurator vom Benutzer eine Reihe von Entscheidungen, die verschiedene Teile des Produktes betreffen. Beispielsweise können bei einigen Onlineshops Computer vom Kunden selbst zusammengestellt werden. Die Bauteile werden entweder einzeln an den Kunden gesendet oder sie werden bereits zusammengebaut verschickt. Ein PC besteht aus vielen Einzelteilen, die in verschiedenen Ausführungen von mehreren Herstellern angeboten werden. Der PC-Konfigurator des Onlineshops ALTERNATE umfasst zum Beispiel 22 verschiedene Komponenten für einen PC. Allein für die Komponente „Prozessor“ stehen 154 verschiedene Varianten zur Auswahl [vgl. Alt]. Der Konfigurator könnte nun alle möglichen Kombinationen aller Komponenten zur Auswahl stellen. Daraus würde jedoch eine sehr lange und unübersichtliche Liste entstehen, in welcher der Kunde seine Wunschkonfiguration suchen müsste. Stattdessen bietet der Konfigurator die Komponenten einzeln zur Auswahl an. So erhält der Benutzer einen schnellen Überblick über die möglichen Attribute des PCs und weiß, welche Entscheidungen von ihm erwartet werden [vgl. Sto07].

Ein Produktkonfigurator kann auch Regeln aufstellen, um bestimmte Kombinationen von Attributwerten zu verbieten oder zu erzwingen. Ein Mainboard kann beispielsweise nur Prozessoren aufnehmen, die einen passenden Sockel besitzen. Ein PC-Konfigurator kann also nach der Auswahl eines Mainboards mit einem FM2-Sockel die Auswahl auf diejenigen Prozessoren einschränken, welche mit dem FM2-Sockel kompatibel sind.

Natürlich entstehen durch die Verwendung eines Produktkonfigurators auch Kosten. Allerdings werden diese durch die Vorteile für das Unternehmen aufgewogen. Zum einen grenzt sich das individuelle Produkt gegen ähnliche Produkte von Konkurrenten ab und erhöht somit die Kundenbindung. Zum anderen wird der Kunde zufrieden gestellt, weil er „sein“ persönliches Produkt in einer einfachen Art und Weise entworfen hat. Dafür sorgt auch der Umstand, dass Produktkonfiguratoren meist darauf ausgelegt sind, dass keine ungültigen Konfigurationen erzeugt, das heißt keine Regeln verletzt werden können [vgl. Pil00]. Dies wird dadurch erreicht, dass der Konfigurator dem Benutzer nur diejenigen Attributwerte zur Auswahl anbietet, deren Kombination mit den bisher schon ausgewählten Werten keine Regeln verletzt. Ein solcher Konfigurator wird *Backtracking-frei* genannt, da der Benutzer niemals gezwungen wird eine Konfigurationsentscheidung zurückzunehmen (englisch: *to backtrack* – zurückziehen). Der PC-Konfigurator von oben, der die Auswahl des Prozessors einschränkt, ist Backtracking-frei.

2.1.1 Mathematische Beschreibung

Produktkonfiuratoren lösen eine bestimmte Klasse von Problemen, nämlich das Konfigurationsproblem. Dieses kann folgendermaßen mathematisch modelliert werden:

Definition 2.1 (Konfigurationsproblem)**.** Ein Konfigurationsproblem C lässt sich als 3-Tupel $C = (X, D, F)$ darstellen, wobei X der Menge der zu wählenden Attribute $\{x_1, \ldots, x_n\}$ entspricht. Die Menge D ist das kartesische Produkt der endlichen Mengen ihrer Attributwerte $D_1 \times D_2 \times \ldots D_n$. In der Menge $F = \{f_1, f_2, \ldots, f_m\}$ werden einschränkende prädikatenlogische Formeln abgelegt, die Bedingungen festlegen, welche die Attribute x_i erfüllen müssen. Eine gültige Konfiguration wählt für alle Attribute x_i genau einen Attributwert $d_j \in D_i$, so dass alle Regeln aus F erfüllt sind [vgl. Sto07; Sub+04].

Zur besseren Veranschaulichung folgt ein Beispiel für die Erstellung eines T-Shirts. Dazu muss die Farbe („rot“, „schwarz“ oder „weiß“), die Größe („klein“, „mittel“ oder „groß“) und der Aufdruck („Men In Black“ (MIB) oder „Rettet die Wale!“ (RDW)) gewählt werden. Bei all diesen Kombinationen gibt es zwei Einschränkungen zu beachten. Der Aufdruck MIB kann nur zusammen mit einem schwarzen T-Shirt gewählt werden. Sollte ein kleines T-Shirt gewählt werden, ist der RDW-Aufdruck

nicht möglich, da dazu die Abbildung eines Wales gehört, der nicht auf ein kleines T-Shirt passt [vgl. Sub+04].

Das Konfigurationsproblem für dieses Beispiel sieht folgendermaßen aus:

$$
\begin{aligned}
X &= \{x_1, x_2, x_3\} = \{\text{Farbe}, \text{Größe}, \text{Aufdruck}\},\\
D_1 &= \{\text{rot}, \text{schwarz}, \text{weiß}\},\\
D_2 &= \{\text{klein}, \text{mittel}, \text{groß}\},\\
D_3 &= \{\text{MIB}, \text{RDW}\},\\
F &= \{f_1, f_2\} \text{ mit}\\
f_1 &= (x_3 = \text{MIB}) \implies (x_1 = \text{schwarz}) \text{ und}\\
f_2 &= (x_3 = \text{RDW}) \implies (x_2 \neq \text{klein}).
\end{aligned}
$$

Die möglichen Attributwerte werden in den Mengen D_1, D_2 und D_3 angegeben. Die beiden Einschränkungen können durch f_1 und f_2 dargestellt werden. Die Anzahl aller möglichen Kombinationen ergibt sich aus $|D| = |D_1| \cdot |D_2| \cdot |D_3| = 18$, von denen aber aufgrund der Einschränkungen nur neun gültig sind. Die gültigen Kombinationen sind in der folgenden Abbildung 2.1 dargestellt [vgl. Sub+04].

(rot, mittel, RDW)	(schwarz, mittel, MIB)	(schwarz, groß, RDW)
(rot, groß, RDW)	(schwarz, mittel, RDW)	(weiß, mittel, RDW)
(schwarz, klein, MIB)	(schwarz, groß, MIB)	(weiß, groß, RDW)

Abbildung 2.1: Gültige Konfigurationen aus dem Beispiel zur Erstellung eines T-Shirts.

Bei komplexeren Modellen mit vielen Attributen entsteht sehr schnell eine hohe Anzahl von möglichen Kombinationen ($|D|$). Der naive Ansatz, bei jeder Entscheidung des Benutzers die Einhaltung aller Regeln sequentiell zu überprüfen, kann dabei sehr schnell an seine Grenzen stoßen. Man kann das Konfigurationsproblem auch als *Constraint-Satisfaction-Problem* (CSP) (deutsch: Bedingungserfüllungsproblem) betrachten. CSPs gehören zur Klasse der Suchprobleme, bei denen eine gültige Variablenbelegung innerhalb eines Suchraums gesucht wird. Dieser Suchraum wird von allen möglichen Belegungen der Variablen aufgespannt. Um alle gültigen Lösungen zu finden, muss der gesamte Raum abgesucht werden, was bei vielen Attributen einen sehr großen Aufwand darstellt. Bei CSPs kann die Menge der möglichen Kombinationen der Attributwerte durch Constraints (deutsch: Bedingungen) eingeschränkt werden [vgl. HA04].

Die Constraints entsprechen den prädikatenlogischen Formeln aus Definition 2.1. Sie sind grob ausgedrückt Funktionen mit einer bestimmten Anzahl von Argumenten, die je nach Belegung dieser Argumente erfüllt sind oder nicht. Ein Argument kann dabei nicht nur als Eingabe für ein Constraint dienen, sondern auch als Ausgabe. Das bedeutet, dass ein Constraint auch den Wert einer Variable ändern beziehungsweise ihren Wertebereich beschränken kann. Algorithmen zur effizienten Lösung von

CSPs nutzen diesen Umstand aus, um die Anzahl der möglichen Kombinationen zu reduzieren, dadurch den Suchraum einzuschränken und somit den nötigen Aufwand zu verringern [vgl. Her76; HW07; Nie14].

2.1.2 Eigenschaften von interaktiven Konfiguratoren

Konfiguratoren verfolgen einen von zwei Ansätzen. Der Prozess der Konfiguration ist entweder interaktiv oder nicht-interaktiv. Bei nicht-interaktiven Konfiguratoren muss der Benutzer alle Entscheidungen über gewünschte Attributwerte in einem Zug eingeben. Der Konfigurator versucht dann mit diesen Eingaben eine gültige Konfiguration zu finden.

Der nicht-interaktive T-Shirt-Konfigurator verlangt vom Benutzer die Auswahl der Farbe, der Größe und des Aufdrucks zur selben Zeit. Der Benutzer kann also ein „rotes“, „kleines“ T-Shirt mit der Aufschrift „Rettet die Wale!“ konfigurieren. Aus dem Beispiel und der Abbildung 2.1 ist schon bekannt, dass diese Konfiguration des T-Shirts nicht gültig ist, da das Constraint f_2 verletzt ist, welches ein kleines T-Shirt gemeinsam mit dem RDW-Aufdruck verbietet. Der Konfigurator erkennt diesen Konflikt aber nicht während der Eingabe, sondern erst danach bei der Verarbeitung. Zur Auflösung des Konflikts hat der Konfigurator zwei denkbare Möglichkeiten. Entweder wählt er automatisch einen geeigneten Attributwert, indem er die Größe auf „mittel“ oder „groß“ setzt, oder er gibt eine Fehlermeldung aus und fordert den Benutzer auf, die Eingaben zu korrigieren. Bei der ersten Alternative stellt aber die Auswahl eines besseren Attributwertes ein Problem dar. Woher weiß der Konfigurator, dass ein Attributwert besser als ein anderer ist? Natürlich kann man den Benutzer die einzelnen Attributwerte nach seinen Neigungen sortieren lassen, jedoch würde die Eingabe dadurch erheblich verkompliziert werden.

Eine Fehlermeldung zu lesen und die Eingaben zu korrigieren, bedeutet für den Benutzer in diesem Beispiel zwar nur relativ wenig Aufwand, weil er zur Berichtigung des obigen Fehlers nur eine andere Größe als „klein“ wählen muss. Trotzdem wird sein Arbeitsfluss gestört und er hat den Eindruck, einen Fehler begangen zu haben. Viel gravierender ist dies bei umfangreicheren Konfiguratoren mit vielen Attributen und komplexen Regeln. Dort könnten Änderungen von mehreren Attributwerten zur Lösung eines Konflikts nötig sein, was einen aufwendigen Prozess für den Benutzer bedeutet, wenn er viele Attributwerte einzeln korrigieren muss.

Ein interaktiver Konfigurator kann solche Konflikte von vornherein verhindern, indem er nach jeder Entscheidung des Benutzers diejenigen Attributwerte deaktiviert, die mit der aktuellen Belegung der Attribute kombiniert mindestens eine der Regeln verletzen würden. Dies bedeutet, dass der Benutzer nur über ein Attribut nach dem anderen entscheiden kann. Der Konfigurator muss also einen Zustand verwalten, in dem die bereits gewählten und die noch wählbaren Attributwerte gespeichert sind. Beim interaktiven T-Shirt-Konfigurator wählt der Benutzer zunächst die Farbe „rot“ aus. Der Konfigurator verarbeitet die Eingabe und schränkt wegen des Constraints

f_1 den zulässigen Wertebereich für den Aufdruck ein (der MIB-Aufdruck kann nur auf ein schwarzes T-Shirt aufgedruckt werden; er wird also entfernt). Danach wählt der Benutzer im zweiten Schritt den einzigen verbliebenen Aufdruck „Rettet die Wale!“, woraufhin der Konfigurator aufgrund des Constraints f_2 den zulässigen Wertebereich für die Größe einschränkt. Das Constraint schließt ein kleines T-Shirt mit dem RDW-Aufdruck aus, weshalb die Größe nur noch die Werte „mittel“ und „groß“ annehmen kann. Schlussendlich wählt der Benutzer die „mittlere“ Größe und hat den Konfigurationsprozess für sein T-Shirt ganz ohne frustrierende Fehlermeldungen und Korrekturen von Eingaben abgeschlossen. **Anmerkung:** Da jedes T-Shirt einen Aufdruck besitzen muss, sollte der Konfigurator den zweiten Schritt automatisiert als Konsequenzentscheidung ausführen. Dies ist nötig, damit immer eine gültige Konfiguration erreicht werden kann. Andernfalls könnte der Benutzer im zweiten Schritt statt dem Aufdruck die Größe „klein“ wählen, woraufhin kein Aufdruck mehr zur Auswahl stünde.

Wie man sieht, unterstützen Constraints die Interaktivität von Konfiguratoren. Sie erlauben es, Wertebereiche von Variablen einzuschränken und somit die Wahl von ungültigen Konfigurationen auszuschließen. Das wiederum bedeutet, dass der Benutzer nie zu einer vorherigen Konfiguration zurückkehren muss. Constraints helfen also auch bei der Umsetzung von Backtracking-freien Konfiguratoren. Die schrittweise Einschränkung von Wertebereichen durch Constraints muss dabei keine bestimmte Reihenfolge einhalten, sondern kann in beliebiger Abfolge durchgeführt werden.

2.2 Einführung in Cyber Physical Systems

Wie bereits erwähnt ist ein Anwendungsfall von RangeConfig die Konfiguration von Cyber Physical Systems (CPS). So ein System besteht aus verschiedenen Komponenten, wie CPUs, Sensoren, Aktoren und Bussen. Diese Komponenten sind miteinander verbunden und haben verschiedene Eigenschaften, die in mehreren Ausprägungen auftreten können. Ein Beispiel für ein CPS ist eine intelligente Weste, die in einem betreuten Seniorenwohnheim eingeführt werden soll, um gestürzten Bewohnern bessere und schnellere Hilfe zukommen zu lassen.

Die Weste muss erkennen können, wenn der Träger stürzt, was für die Weste eine Beschleunigung bedeutet. Zur Erkennung der Beschleunigung besitzt sie zwei Beschleunigungssensoren. Die Weste muss außerdem erkennen, ob sie getragen wird oder nicht. Andernfalls kann ein falscher Alarm ausgelöst werden, wenn die Weste beispielsweise in einer Tasche transportiert und dadurch „beschleunigt“ wird. Dazu besitzt sie einen Wärmesensor. Damit die Weste die Daten von den Sensoren verarbeiten kann, wird eine Recheneinheit benötigt und ein Bus, welcher die Daten transportiert. Um das Pflegepersonal im Notfall alarmieren zu können, wird ein Bluetooth-Sender eingebaut, der direkt an die Recheneinheit angeschlossen ist. Dazu

kommt noch eine Batterie, um das gesamte System mit Energie zu versorgen [vgl. Hö+13].

Die einzelnen Bestandteile der Weste besitzen verschiedene Eigenschaften. Die CPU benötigt die Versorgungsspannung V_{cc}, arbeitet unter der Frequenz f und hat eine bestimmte Stromaufnahme I für jeden Modus. Sie unterstützt mindestens zwei Modi, einen Arbeitsmodus, in dem die CPU aktiv ist und arbeitet, und einen Ruhemodus, in dem sie inaktiv ist und somit weniger Strom verbraucht. Die CPU besitzt also mindestens I_{aktiv} und I_{inaktiv}. Außerdem hat die CPU eine Reihe von Anschlüssen, von denen jeder bestimmte Protokolle unterstützt. Die Sensoren, der Bus und das Bluetooth-Modul benötigen genauso wie die CPU eine Versorgungsspannung V_{cc} und haben einen Arbeits- und einen Ruhemodus. Bei ihnen wird direkt der Stromverbrauch P_{aktiv} und P_{inaktiv} für jeden Modus gespeichert. Der Bus und die Sensoren besitzen jeweils noch eine bestimmte Datenübertragungs- beziehungsweise Abtastrate f [vgl. Hö+13].

Zwei sehr wichtige Anforderungen an die Weste sind Energiesparsamkeit, damit die Batterie nicht ständig ausgetauscht werden muss, und Zuverlässigkeit, damit Stürze sicher erkannt werden. Wenn die Sensoren eine relativ geringe Abtastrate haben und die CPU unter einer relativ geringen Frequenz arbeitet, kann zwar Energie gespart werden. Allerdings leidet darunter die Zuverlässigkeit, da Stürze nur noch selten erkannt werden können, wenn die beiden Beschleunigungssensoren zum Beispiel nur alle fünf Sekunden einen Wert messen. Es muss also eine Balance zwischen Energieverbrauch und Gesamtfrequenz des Systems hergestellt werden. Zur Lösung dieses Konflikts können Constraints eingesetzt werden, welche die verschiedenen Modi der Komponenten beachten.

Das Ziel ist eine Weste mit minimalem Energieverbrauch, die aber trotzdem zuverlässig arbeitet. Es müssen also erst unbedingt nötige Anforderungen erfüllt sein, bevor der Energieverbrauch minimiert werden kann. Zum Beispiel kann die Abtastrate des Wärmesensors auf einen Messwert pro Sekunde festgesetzt werden. Dies reicht aus, um relativ schnell erkennen zu können, dass die Weste angezogen wurde. Die Abtastrate der Beschleunigungssensoren kann zum Beispiel von 6,25 Hz bis zu 3200 Hz reichen, wodurch auch der Stromverbrauch beeinflusst wird. Die Frequenz des Datenbusses muss hoch genug sein, um alle Daten von den Sensoren zur CPU transportieren zu können. Die CPU wiederum muss in der Lage sein, alle Daten zu verarbeiten. All diese Bedingungen spielen eine Rolle für den Gesamtenergieverbrauch und können durch Verknüpfungen von Constraints dargestellt werden [vgl. Hö+13].

Von diesem Beispiel kann zu allgemeinen CPS abstrahiert werden. Grundlegende Bausteine eines Systems sind Komponenten, die Eigenschaften in verschiedenen Modi besitzen. Die Komponenten sind durch eine bestimmte Assoziation verknüpft (im Beispiel ist der Datenbus ebenfalls eine Komponente, die mit den Sensoren und der Recheneinheit verbunden ist). Zusätzlich existieren Constraints, die entweder auf dem gesamten System oder auf einzelnen Komponenten liegen.

RangeConfig eignet sich aber nicht nur für die Planung von CPS, sondern auch für die Lösung von allgemeinen Konfigurationsproblemen. Dazu müssen die entsprechenden Regeln oder Constraints, die Bestandteil des Konfigurationsproblems sind, selbst festgelegt werden. Es kann also zwischen zwei Gruppen von Benutzern der Anwendung unterschieden werden. Einerseits wird von einem „Entwickler“ eine Problemkategorie modelliert, wobei auch entsprechende Constraints hinzugefügt werden. Bei CPS kann zum Beispiel eine Bedingung sein, dass eine CPU höchstens mit so vielen anderen Komponenten verknüpft wird, wie sie Anschlüsse besitzt (die Anzahl der Anschlüsse kann als Eigenschaft der CPU modelliert werden). Andererseits kann ein „Anwender“ ein konkretes Problem aus einer Kategorie mit RangeConfig lösen, wobei er sich an die gesetzten Constraints halten muss. **Anmerkung:** Derzeit wird in der Anwendung keine Unterscheidung dieser beiden Rollen vorgenommen.

3 Graphische Benutzeroberflächen

Für Computerprogramme ist die Benutzeroberfläche sehr wichtig, da sie die Schnittstelle zwischen dem Programm und dem Benutzer bildet. Interaktive Programme sind noch mehr von der Benutzeroberfläche abhängig, da sie darüber vom Benutzer bedient werden. Die Oberfläche bietet Möglichkeiten zur Eingabe von Daten und Ausgabe von Ergebnissen oder Fehlermeldungen. Eine gut gestaltete Benutzeroberfläche erhöht die Produktivität und die Zufriedenheit des Benutzers bei der Verwendung.

Neben textbasierten Oberflächen existieren auch graphische Benutzeroberflächen (englisch: Graphical User Interfaces (GUIs)), die in der heutigen Zeit wohl jeder kennt. Sie sprechen eine größere Bandbreite von Benutzern an, da sich nicht nur Experten darin zurecht finden, sondern auch Einsteigern Hilfestellungen geboten werden, um komplizierte Funktionen mit wenigen Mausklicks auszuführen. Komplexe Zusammenhänge können graphisch aufbereitet und so dem Benutzer leichter verständlich gemacht werden.

Die Entwicklung von benutzerfreundlichen Oberflächen wird immer wichtiger. Für viele Anwendungen ist ihre Benutzeroberfläche das Alleinstellungsmerkmal und ihre intuitive und leichte Bedienbarkeit hebt sie von Anwendungen mit ähnlicher oder gleicher Funktion ab. Deshalb binden Entwickler von graphischen Benutzeroberflächen viele Forschungsergebnisse aus der Wahrnehmungspsychologie ein. Aber auch kulturspezifische Einflüsse sind zu beobachten. Beispielsweise werden Benutzeroberflächen für Sprachräume, in denen von rechts nach links gelesen wird, anders entworfen als für den westlichen Raum [vgl. SP10].

Dieses Kapitel gibt in Abschnitt 3.1 eine Einführung in allgemeine Anforderungen an graphische Benutzeroberflächen. Dazu werden in Unterabschnitt 3.1.1 einige Maßstäbe für die Entwicklung einer Benutzeroberfläche erläutert. In Unterabschnitt 3.1.2 werden die acht goldene Regeln des Interface Design von Shneiderman und Plaisant [SP10] vorgestellt. Danach wird in Abschnitt 3.2 eine kurze Einführung in das Thema der Nebenläufigkeit gegeben und das Kapitel in Abschnitt 3.3 mit den Anforderungen an die Oberfläche von RangeConfig abgeschlossen.

3.1 Anforderungen an eine graphische Benutzeroberfläche

Das Ziel jedes Designers ist es, eine Oberfläche zu erschaffen, die sowohl funktionalen als auch ästhetischen Anforderungen gerecht wird. Allerdings ist es ein steiniger Weg bis dahin und beinhaltet vorausschauende Planung, Untersuchungen der Bedürfnisse der zukünftigen Benutzer und wiederholtes und intensives Testen. Das Aufstellen von Anforderungen umfasst weit mehr als nur das generelle Fordern nach „Benutzerfreundlichkeit“. Die konkreten Anforderungen an die Benutzeroberfläche hängen stark von der jeweiligen Anwendung und der angesprochenen Zielgruppe ab [vgl. SP10]. Deshalb gilt es zu klären, was unter „Benutzerfreundlichkeit“ verstanden werden kann und welche Schlussfolgerungen für die graphische Oberfläche von RangeConfig gezogen werden können. In den folgenden Unterabschnitten sollen allgemeine Anforderungen und Leitlinien erörtert werden, die beim Entwurf von graphischen Benutzeroberflächen hilfreich sein können.

3.1.1 Maßstäbe für Benutzeroberflächen

Nach Shneiderman und Plaisant [SP10] gibt es einige Maßstäbe, nach denen alle Benutzeroberflächen bewertet werden können. Teilweise fließen sie in die Anforderungen an die GUI von RangeConfig ein:

- *Lernzeit.* Wie lange dauert es, bis ein durchschnittlicher Benutzer der Zielgruppe die Ausführung von bestimmten, vorgegebenen Aktionen erlernt?

- *Ausführungsgeschwindigkeit.* Wie lange braucht ein durchschnittlicher Benutzer, um bestimmte, vorgegebene Aufgaben zu erledigen?

- *Fehlerrate.* Wie viele und vor allem welche Fehler machen Benutzer bei der Ausführung von vorgegebenen Aufgaben? Die Zeit, die benötigt wird, um Fehler wieder zu korrigieren, beeinflusst die Ausführungsgeschwindigkeit und die subjektive Befriedigung, weshalb der Behandlung von Fehlern große Bedeutung beigemessen wird.

- *Erinnergungsvermögen.* Wie lange kann sich ein Benutzer merken, wie bestimmte Aufgaben ausgeführt werden? Hierbei spielen die Lernzeit und die Nutzungshäufigkeit der Anwendung eine große Rolle.

- *Subjektive Befriedung.* Wie angenehm war das Benutzen von verschiedenen Teilen der Oberfläche? Dieser Maßstab ist schwer zu messen, weshalb bei mündlichen oder schriftlichen Befragungen oft eine frei formulierte Antwort erwartet wird.

Das Ziel ist es natürlich, bei jedem dieser Maßstäbe das Ideal zu erreichen. Allerdings müssen sehr oft Kompromisse eingegangen werden, was unter anderem auch am (zumeist) beschränkten Budget beziehungsweise an einer beschränkten Arbeitszeit liegen kann. Beispielsweise kann die Ausführungsgeschwindigkeit durch die Verwendung von komplizierten Tastenkombinationen erhöht werden, wodurch jedoch die Lernzeit verlängert wird. Es kann aber auch ein intelligentes System implementiert werden, welches den Lernfortschritt des Benutzers aufzeichnet und ab einem gewissen Kenntnisstand zusätzlich zur Steuerung mit der Maus Tastenkombinationen anbietet. Das System könnte noch erweitert werden, so dass für Experten eine Makrofunktionalität angeboten wird, mit der viele Aktionen mit nur einer Eingabe ausgeführt werden können. So werden sowohl Einsteiger und fortgeschrittene Benutzer als auch Experten angesprochen [vgl. SP10].

3.1.2 Die acht goldenen Regeln des Interface Design

Shneiderman und Plaisant [SP10] stellen acht Regeln vor, die sich über Jahrzehnte herauskristallisiert haben und eine Grundlage bieten, nach der eine graphische Benutzeroberfläche entworfen werden kann. Diese acht Prinzipien sollen auf die meisten interaktiven Systeme anwendbar sein, wobei sie aber zu prüfen und an das konkrete System anzupassen sind. Bei der Umsetzung der Oberfläche von RangeConfig wurde der Versuch unternommen, die Regeln einzuhalten. Dazu fließen sie vor allem in die Aufstellung der Anforderungen an die GUI ein.

Im Folgenden werden die acht Regeln beschrieben:

1. *Bemühe Dich um Konsistenz.* Sich ähnelnde Situationen sollen ähnliche Abläufe von Handlungen erfordern. In der kompletten Oberfläche sollen dieselben Begriffe ohne Abwandlungen fallen. Das graphische Erscheinungsbild, also Farben, die Schriftart und -größe und so weiter, soll in der kompletten Anwendung gleich sein.

2. *Sorge für umfassende Benutzbarkeit.* Die Bedürfnisse der Benutzer können sich stark unterscheiden. Sie hängen unter anderem von Alter, Kompetenzstufe und eventuellen körperlichen Beeinträchtigungen ab. Eine Oberfläche, die „mitlernt“ und beispielsweise Hilfestellungen für Einsteiger und Tastenkombinationen für Experten anbietet, wird von den Benutzern als qualitativ hochwertig empfunden. Deshalb soll eine Benutzeroberfläche veränderlich gestaltet sein und sich an die individuellen Bedürfnisse der Benutzer anpassen lassen.

3. *Gib aufschlussreiche Rückmeldungen.* Jede Benutzeraktion soll eine Rückmeldung des Systems verursachen, die zu der Art der Eingabe passt. Häufig ausgeführte Aktionen können dabei kleinere Reaktionen auslösen, wohingegen seltenere und größere Aktionen auch größere Reaktionen nach sich ziehen sollen. Bei

Änderungen an graphischen Objekten bietet es sich an, diese Änderung durch eine veränderte Darstellung direkt anzuzeigen.

4. *Entwirf Dialoge für abgeschlossene Handlungen.* Handlungen, die in mehrere einzelne Aktionen geteilt sind, sollen in Gruppen gegliedert werden, die einen Anfang, ein Mittelteil und ein Ende aufweisen. Durch eine aufschlussreiche Rückmeldung am Ende der Handlung weiß der Benutzer, dass die Handlung abgeschlossen ist, und kann sich der nächsten Aufgabe widmen.

5. *Verhindere Fehler.* Soweit möglich soll eine Oberfläche so entworfen werden, dass schlimme Fehler bereits im Vorfeld ausgeschlossen werden. Beispielsweise können Schaltflächen deaktiviert werden, wenn die entsprechende Aktion in der aktuellen Situation nicht ausführbar ist. Sollte der Benutzer einen Fehler machen, soll die Oberfläche diesen identifizieren und einfache und klare Anweisungen geben, um den Fehler zu beheben.

6. *Erlaube das Rückgängigmachen von Aktionen.* Soweit möglich sollen alle Aktionen einfach umkehrbar sein. Mit dem Wissen, dass der Benutzer seine Aktionen wieder rückgängig machen kann, ist er mutiger und wagt es eher unbekannte Menüpunkte zu erforschen.

7. *Überlasse dem Benutzer die Kontrolle.* Vor allem erfahrene Benutzer wollen die Kontrolle über die Oberfläche haben. Sie möchten, dass die Oberfläche ihnen gehorcht und auf ihre Eingabe reagiert.

8. *Entlaste das Kurzzeitgedächtnis.* Menschen können nur eine beschränkte Menge von Informationen im Kurzzeitgedächtnis behalten. Deshalb ist es wichtig, dass sich Benutzer nicht viele Informationen merken müssen, während sie von einem Teil der Benutzerberfläche zu einem anderen wechseln.

Eine Analyse, inwieweit diese Regeln eingehalten wurden, ist in Abschnitt 5.3 zu finden.

3.2 Nebenläufigkeit

Moderne Betriebssysteme erwecken beim Benutzer den Eindruck, dass viele Programme gleichzeitig und parallel ausgeführt werden können. Ein PC hat aber nur eine beschränkte Anzahl von Prozessorkernen zur Verfügung, auf denen jeweils nur ein Programm (ein Prozess) zur selben Zeit abgearbeitet werden kann. Um viele Prozesse „quasi-parallel“ auszuführen, wird jedem Prozess eine gewisse Arbeitszeit auf dem Prozessorkern gewährt. Nach dieser Zeit wird der aktuelle Zustand des Prozesses gespeichert und der nächste Prozess darf eine bestimmte Zeit lang auf den Kern zugreifen. Um innerhalb eines Prozesses mehrere Aktionen „quasi-parallel“ auszuführen, kann jeder Prozess mehrere leichtgewichtige Prozesse

(Threads) beherbergen. Die Threads teilen sich alle Ressourcen des dazugehörigen Prozesses.

Eine nebenläufige Umsetzung von Programmen wird verwendet, um mehrere unabhängige Teile eines Programms unabhängig voneinander abarbeiten zu können. Als Beispiel sei ein Programm zur Simulation physikalischer Gegebenheiten genannt. Dieses Tool speichert während der Simulation Zwischenergebnisse in eine Datei auf der Festplatte. Das Schreiben auf eine herkömmliche Festplatte ist sehr langsam im Vergleich zur Arbeitsgeschwindigkeit eines Prozessors. Das Programm muss also während der Simulation regelmäßig warten bis das Schreiben abgeschlossen ist. Um eine schnellere Ausführung zu ermöglichen, können die beiden Operationen *Simulation* und *Datenspeicherung* nebenläufig implementiert werden, so dass sie „quasi-parallel“ geschehen. Dann kann die Simulation schon fortgeführt werden, während die letzten Zwischenergebnisse noch gespeichert werden.

Dieses Beispiel lässt sich auch auf die Programmierung eines graphischen Programms übertragen. Das Beispielprogramm führt im Hintergrund Berechnungen aus und lädt Bilder aus dem Internet herunter. Während diesen Operationen muss das Programm regelmäßig auf Benutzereingaben warten. Die Eingaben vom Benutzer kommen im Vergleich zur Arbeitsgeschwindigkeit des Prozessors aber extrem langsam, weshalb die Berechnungen und der Download erheblich ausgebremst werden. In einer nebenläufigen Implementierung kann das Programm während der Wartezeit auf Benutzereingaben die Berechnungen durchführen beziehungsweise Bilder herunterladen.

Ein anderes Beispielprogramm erwartet vom Benutzer die Eingabe einer beliebigen Zahl und zerlegt diese Zahl auf Knopfdruck in ihre Primfaktoren. Dieser Prozess nimmt bei sehr großen Zahlen viel Zeit in Anspruch. Während dieser Zeit „hängt“ die Oberfläche des Programms und reagiert nicht auf Benutzereingaben. In diesem Beispiel ist das kein großes Problem, da der Benutzer keine anderen Eingaben tätigen kann als die Angabe einer Zahl und die Beauftragung der Zerlegung. Bei komplexeren Oberflächen ist es aber wünschenswert, dass alle Teile der Benutzerberfläche immer verwendet werden können. Auch dies kann durch eine nebenläufige Implementierung erreicht werden, indem die Primfaktorzerlegung in einem Hintergrundthread ausgeführt wird.

Auch die graphische Benutzeroberfläche von RangeConfig ist nebenläufig umgesetzt, um diese positiven Effekte für die Anwendung zu erzielen.

3.3 Anforderungen an die Oberfläche von RangeConfig

Um Anforderungen an die GUI von RangeConfig aufzustellen, wird das Konzept der User-Stories eingesetzt. Eine User-Story ist ein natürlichsprachlicher Satz eines Benutzers, der Anforderungen an ein Software-Produkt enthält. Im Folgenden wird

zunächst jede User-Story angegeben und dann erläutert, welche Bedeutung der Satz für die graphische Benutzeroberfläche hat.

„Ich möchte das entworfene CPS in einer graphischen Darstellung sehen."
Die einzelnen Komponenten des Systems müssen eine graphische Repräsentation besitzen. Außerdem sollen Verbindungen zwischen den Komponenten angezeigt werden. Weiterhin muss die Position der Komponenten veränderlich sein, damit dem Benutzer die Freiheit zur eigenen Anordnung gewährt wird. Es wäre hilfreich eine Funktion zur automatischen Anordnung der Komponenten zu haben, bei der sich die graphischen Elemente nicht überschneiden.

Eigenschaften müssen leicht lesbar sein und ihre Zugehörigkeit zu einer Komponente muss verdeutlicht werden. Es müssen hierbei die verschiedenen Modi beachtet werden, in denen sich die Komponenten befinden können. Auch vorhandene Constraints müssen dargestellt und eventuelle Verletzungen markiert werden.

„Ich möchte ein System aus mehreren Teilsystemen zusammenstellen."
Zusätzlich zu den vorgegebenen Komponententypen muss ein Typ „Subsystem" vorhanden sein, der andere Komponenten aufnehmen kann. Dies ist eher eine Anforderung an den Konfigurator an sich. Für die Oberfläche bedeutet dies, dass auch eine Ansicht so eines Subsystems geöffnet werden kann. Somit kann das gesamte System dargestellt werden oder nur ein Ausschnitt davon.

„Während Berechnungen möchte ich weiterhin die GUI benutzen können."
Die Oberfläche muss interaktiv und reaktionsfähig bleiben, während das Backend im Hintergrund Berechnungen durchführt. Dies wird über die nebenläufige Implementierung der Benutzeroberfläche erreicht.

„Ich möchte beliebige Systeme erstellen können."
Der Konfigurator darf nicht nur die Komponententypen von CPS unterstützen. Er muss vielmehr beliebige Systeme unterstützen, deren Komponententypen einfach ausgetauscht werden können. Die Bedeutung eines bestimmten Typs hängt dann allein von der Art des Systems ab. Auch diese Anforderung wird eher an den eigentlichen Konfigurator gestellt. Allerdings wird ein Konzept benötigt, mit dessen Hilfe jeder Komponententyp eines Systems dargestellt werden kann. Dies fällt in den Verantwortungsbereich der GUI.

„Ich will nicht immer wieder ähnliche Komponenten neu erstellen müssen."
Komponenten müssen als Vorlage abspeicherbar sein. Zusätzlich können vorgefertigte Komponenten angeboten werden, um den Konfigurationsprozess zu beschleunigen und um dem Benutzer Beispiele zur Erstellung eigener Komponenten zu liefern.

„Ich möchte zu einem gespeicherten System ein Constraint hinzufügen."
Diese User-Story umfasst gleich zwei Anforderungen. Zum einen muss die

Oberfläche eine Möglichkeit anbieten, ein System abzuspeichern beziehungsweise ein gespeichertes System wieder zu laden. Hierbei ist es wichtig, auf ein plattformunabhängiges Speicherverfahren zu achten, damit diese Funktionalität von RangeConfig auf allen Betriebssystemen verwendet werden kann. Zum anderen muss eine Möglichkeit vorhanden sein, Constraints zu einer Komponente oder dem gesamten System hinzuzufügen.

„Ich will das Programm möglichst einfach bedienen können."
Die Oberfläche muss kompliziertere Aktionen einfach verpacken, so dass sie zum Beispiel mit nur einem Mausklick ausgeführt werden können. Dies betrifft aber auch die Eingabe von Daten, wie das Hinzufügen einer Komponente oder das Verbinden von zwei Komponenten. Auch die Eingabemaske zum Hinzufügen von Eigenschaften oder Constraints muss einfach gestaltet sein und soll dem Benutzer Hilfestellungen geben.

Im Folgenden werden die Anforderungen zusammenfassend aufgelistet und in *Muss-* und *Kannkriterien* eingeteilt. Die Anforderungen wurden hauptsächlich nach ihrer Notwendigkeit für die Funktionalität der graphischen Benutzeroberfläche eingeteilt. Allerdings ist auch der voraussichtliche Zeitaufwand für die Umsetzung in die Einteilung eingeflossen.

Musskriterien Dies sind Anforderungen, die unbedingt erfüllt werden müssen, damit die GUI bestimmungsgemäß verwendet werden kann.

- Graphische Darstellung des gesamten Systems mit Eigenschaften in mehreren Modi und Constraints
- Anzeige eines Teils des modellierten Systems
- Interaktivität der GUI
- Austauschbare und damit beliebige Komponententypen
- Speichern von Komponenten als Vorlage für ähnliche oder gleiche Komponenten
- Speichern und Laden von Systemen
- Eingabe von Constraints
- Einfache Bedienung

Kannkriterien Diese Anforderungen müssen nicht unbedingt erfüllt werden, bieten aber einen Mehrwert bei Erfüllung und können dem Benutzer die Arbeit erleichtern.

- Automatische Anordnung der Komponenten
- Darstellung von verletzten Constraints
- Vorgefertigte Komponenten (je nach Typ des Systems)
- Exportieren und Importieren von Subsystemen

4 Datenmodell und Templatebibliothek

Im Zuge der Entwicklung von RangeConfig wurde ein Datenmodell für die Speicherung der modellierten Systeme entworfen. In Abschnitt 4.1 wird das Datenmodell im Detail beschrieben. Zur Wiederverwendbarkeit einzelner Komponenten wurde eine Templatebibliothek implementiert, deren Umsetzung in Abschnitt 4.2 dargestellt wird.

4.1 Das Datenmodell

Das Datenmodell wurde in Zusammenarbeit mit *Andreas Lindner* erarbeitet und erlaubt die Modellierung rekursiver Systeme in einer Baumstruktur [vgl. Lin14]. In Abbildung 4.1 sind die verwendeten Java-Klassen und ihre Abhängigkeiten dargestellt.

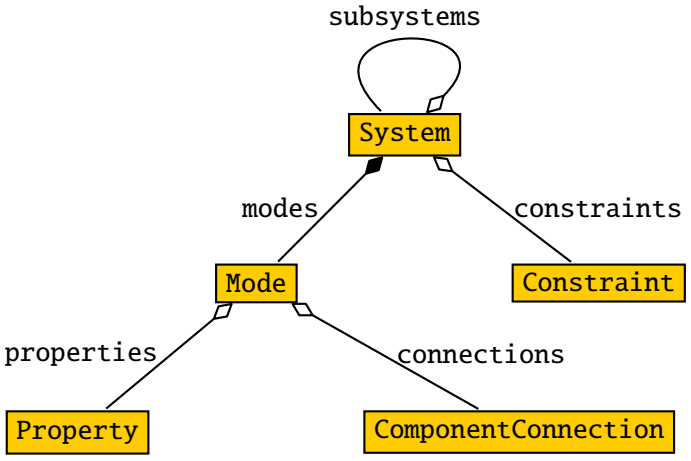

Abbildung 4.1: Vereinfachte Darstellung des Datenmodells.

In diesem Modell dient ein System als Container für weitere Systeme. Dadurch wird eine hierarchische Gruppierung von Komponenten ermöglicht. Eine Komponente ist technisch gesehen auch ein System mit einem besonderen Typ (dieses Attribut ist nicht in der Abbildung enthalten), enthält aber keine Subsysteme. Dies ist keine Einschränkung des Modells, von der GUI wird aber verhindert, dass der Benutzer zu einer Komponente eine Subkomponente hinzufügt.

Der Typ eines Systems wird als Attribut vom Typ `String` umgesetzt. Dadurch ist es möglich, beliebige Typen festzulegen und zu unterscheiden. Festgelegt wird der Typ indirekt vom Benutzer, indem er über die Templatebibliothek eine Komponente mit einem dort definierten Typ zum System hinzufügt. Eine andere Möglichkeit wäre, für jeden möglichen Komponententyp eine eigene Java-Klasse anzulegen, in welcher die notwendigen Daten gespeichert werden. Dies ist allerdings sehr unflexibel und gestattet nicht die Planung von beliebigen Systemen.

Jedes System besitzt einen Namen zur besseren Identifikation durch den Benutzer (dieses Attribut ist ebenfalls nicht in der Abbildung enthalten). Außerdem gehören zu jedem System mindestens ein Modus und beliebig viele Constraints. Die Modellierung der Modi und Constraints wird in den nächsten Unterabschnitten beschrieben.

4.1.1 Eigenschaften und Modi

Die Modellierung erlaubt es für jede Komponente und jedes System Eigenschaften anzulegen. Diese Eigenschaften setzen sich aus einem Namen, einem Eigenschaftstyp, dem Wert und einer optionalen Einheit zusammen. Jede Eigenschaft kann entweder einen Einzelwert, ein Intervall oder eine Menge von gleichartigen Werten darstellen. Je nach Eigenschaftstyp sind verschiedene Typen der Werte erlaubt.

- Einzelwerte: beliebige Zahlen, Zeichenketten oder Wahrheitswerte
- Intervalle: diskret oder kontinuierlich (also entweder ganze Zahlen oder Dezimalzahlen)
- Mengen: beliebige Zahlen oder Zeichenketten

Das Modell unterstützt zusätzlich die Angabe von alternativen Belegungen der Eigenschaftswerte. Dies wird durch verschiedene Modi ermöglicht. Im Modell ist festgelegt, dass jedes System immer mindestens einen Modus, den Basismodus, enthalten muss. Dieser Basismodus enthält alle Eigenschaften des Systems mit den Basiswerten. In jedem weiteren Modus können die Werte von Eigenschaften aus dem Basismodus überschrieben werden, aber keine neuen Eigenschaften hinzugefügt werden.

Jeder Modus besitzt eine Liste von allen verbundenen Komponenten, die in dem zugehörigen System enthalten sind. So wird es ermöglicht, Komponenten nur in bestimmten Systemmodi zu verbinden.

Mit verschiedenen Modi können mehrere Zustände einer Komponente modelliert werden. Zum Beispiel kann damit eine CPU zwei Zustände haben, von denen einer ein „Arbeitszustand“ mit hohem Stromverbrauch und der andere ein „Ruhezustand“ mit geringem Stromverbrauch ist.

Die drei Eigenschaftstypen Einzelwert, Intervall und Menge sind hierarchisch miteinander verknüpft. Der Einzelwert ist ein Subtyp der Menge und die Menge ist wiederum ein Subtyp des Intervalls. Eine Eigenschaft mit einem bestimmten Typ kann auch einen Wert annehmen, welcher ein Subtyp dieses Typs ist. Davon ausgehend wäre es denkbar, eine Erweiterung zu implementieren, die es erlaubt, in einem Modus nicht nur den Wert der Basis-Eigenschaft zu ändern, sondern auch ihren Typ mit einem seiner Subtypen zu ersetzen.

4.1.2 Constraints

Während der Modellierung eines Systems mit RangeConfig können Einschränkungen und Regeln zwischen Eigenschaften hinzugefügt werden. Diese Regeln werden als Constraints modelliert und automatisch auf Einhaltung überprüft.

Die Constraints bestehen ihrer prädikatenlogischen Natur nach aus einem Prädikat und einer Reihe von Argumenten. Jedes Argument kann entweder einen festen Wert annehmen oder eine Referenz auf eine Eigenschaft eines Systems sein. Die Prädikate können den Wert von referenzierten Eigenschaften nicht nur lesen, sondern auch ändern. Zwischen den Referenzen wurde deshalb eine zusätzliche Unterscheidung eingeführt, um Änderungen des Wertes der jeweiligen Eigenschaft explizit verbieten zu können. Falls ein Prädikat einen Eigenschaftswert verändern möchte, dies aber von der Art der Referenz verboten wird, ist das Constraint nicht erfüllbar und eine Fehlermeldung wird ausgegeben.

4.2 Komponentenvorlagen

Mit RangeConfig können Komponenten mir sehr vielen Eigenschaften und verschiedenen Modi modelliert werden. Ein Sensor kann zum Beispiel viele verschiedene Arbeitsmodi besitzen, von denen jeder eine bestimmte Abtastrate mit einer bestimmten Stromaufnahme verbindet. Die Erstellung nur eines solchen Sensors ist sehr aufwändig. Falls nun ein identischer oder sehr ähnlicher Sensor hinzugefügt werden soll, wird dieser Aufwand verdoppelt. Um Komponenten wiederverwendbar zu machen, wurde die Unterstützung von Komponentenvorlagen implementiert.

Diese Vorlagen werden in einer Templatebibliothek gesammelt. Eine Bibliothek besteht aus zwei Teilen. Die Basisbibliothek gibt die möglichen Komponententypen an und bietet eine Liste von vorgefertigten Komponenten. Zu jeder Basisbibliothek können mehrere Benutzerbibliotheken geladen werden, welche ebenfalls vorgefertigte Komponenten enthalten. Jede Templatebibliothek bezieht sich auf einen speziellen Systemtyp beziehungsweise ein spezielles Konfigurationsproblem, beispielsweise ist für die Modellierung von CPS eine eigene Bibliothek vorhanden. Der Benutzer kann erstellte Komponenten zur Wiederverwendung in einer Benutzerbibliothek speichern, die zur entsprechenden Basisbibliothek passt.

Als Format der Dateien für die Templatebibliothek wurde XML gewählt, da es sich um einen plattformunabhängigen Standard handelt, die Dateien auch per Hand bearbeitet werden können und das Lesen und Schreiben durch Standardbibliotheken von Java unterstützt wird. Um die XML-Dateien zu verarbeiten, wird die Streaming API for XML (StAX) benutzt, welche in Java ab Version 6 implementiert ist.

Der Aufbau der XML-Dateien ist relativ simpel und wird nur kurz erklärt. Die kompletten vorgefertigten XML-Dateien für CPS können dem Anhang (Listing 1 und Listing 2) entnommen werden.

```
1 <library name="Cyber Physical System" abbreviation="CPS"
     version="1" subsystem-color="#c4c4c4">
2    <component-types>
3       <component-type name="CPU" color="#ff7fe9" />
4       <component-type name="Actuator" color="#30ff8d" />
```

Listing 4.1: Auszug aus der Basisbibliothek für CPS.

Listing 4.1 zeigt einen Auszug aus der Templatedatei für die CPS-Basisbibliothek. Die Basisbibliothek enthält einen Namen und eine Abkürzung. Die beiden Attribute werden genutzt, um festzustellen, ob eine Benutzerbibliothek mit der Basisbibliothek kompatibel ist. **Anmerkung:** Dies ist ein relativ schwacher Test. Für vollständige Fehlerfreiheit müsste überprüft werden, ob jeder benutzte Komponententyp in der Basisbibliothek definiert wird.

Ab Zeile 3 folgen die grundlegenden Komponententypen, die von der Bibliothek unterstützt werden. Die beiden Attribute `name` und `color` sind nötig, um die Typen identifizieren und mit einer eigenen Farbe darstellen zu können. Da der konkrete Typ einer Komponente nur als Zeichenkette angegeben wird, können mit der entsprechenden Bibliothek beliebige Systeme angelegt werden.

Listing 4.2 zeigt die Definition von vorgefertigte Komponenten. Sie besitzen einen Typ sowie einen Namen und eine Reihe von Eigenschaften. Diese Eigenschaften werden implizit dem Basismodus hinzugefügt. Die einzelnen Attribute der Eigenschaften entsprechen den Attributen des Datenmodells und bedürfen keiner weiteren Erklärung. Zu einer Komponente können verschiedene Modi vordefiniert werden, welche Eigenschaften mit ihren überschriebenen Werten enthalten.

Die XML-Dateien für Benutzerbibliotheken besitzen denselben Aufbau, wie in Listing 4.2 gezeigt.

```
1  <templates>
2     <component type="CPU" name="TI MSP430">
3        <properties>
4           <property id="1" type="SINGLE_VALUE" name="Ports"
               value="7" unit="" />
5           <property id="2" type="INTERVAL" name="Frequency"
               value="[32768000 .. 16000000]" unit="Hz" />
6           <property id="3" type="SET" name="Protocols" value=
               "{UART , SPI , I2C}" unit="" />
7        </properties>
8        <mode id="1">
9           <property id="1" value="18" />
10          <property id="2" value="[1 .. 10]" />
11          <property id="3" value="SPI" />
12       </mode>
13    </component>
```

Listing 4.2: Auszug aus der Basisbibliothek für CPS.

5 Umsetzung und Aufbau der graphischen Oberfläche

Dieses Kapitel beschreibt die Umsetzung und den Aufbau der GUI. In Abschnitt 5.1 wird erklärt, wie die graphische Benutzeroberfläche umgesetzt und welche Bibliothek dazu verwendet wurde. In Abschnitt 5.2 wird der Aufbau der GUI von RangeConfig gezeigt. Außerdem werden dort einige Fenster vorgestellt. Am Ende des Kapitels, in Abschnitt 5.3, werden die Anforderungen an die GUI nochmals aufgegriffen und im Bezug auf ihre Einhaltung analysiert.

5.1 Die Entscheidung für *JavaFX*

Wie effektiv eine graphische Benutzeroberfläche ist, hängt sehr stark von der verwendeten Technologie und den verfügbaren graphischen Elementen ab. Zu Beginn des Entwicklungsprozesses von RangeConfig fiel die Entscheidung, die Anwendung in der Sprache *Java* zu programmieren. Aufgrund dessen wurden die GUI-Frameworks *SWT*, *AWT*, *Swing* und *JavaFX* zur näheren Betrachtung herangezogen. All diese Frameworks sind plattformunabhängig.

Das *Standard Widget Toolkit* (SWT) stellt graphische Elemente, wie Schaltflächen oder Textfelder, plattformspezifisch dar. Programme, die SWT verwenden, sehen also wie native Programme des jeweiligen Betriebssystems aus. Allerdings werden zusätzliche Bibliotheken benötigt, die mit der Anwendung ausgeliefert werden müssen, da SWT kein Bestandteil der Java-Umgebung ist. Die Auslieferung des Programms für mehrere Arbeitsplätze mit verschiedenen Betriebssystemen wird dadurch erschwert. Deshalb wurde SWT als Erstes ausgeschlossen.

Auch das *Abstract Window Toolkit* (AWT) benutzt wie SWT native graphische Elemente. Allerdings unterstützt AWT keine komplexeren Elemente wie beispielsweise Tabellen. Für die Darstellung von tabellarischen Inhalten, wie zum Beispiel die Auflistung von Eigenschaften in verschiedenen Modi, bietet sich aber natürlich eine Tabelle an. Tabellen können nur aus einfacheren GUI-Elementen nachgebaut werden. Dies ist aber mit einem unverhältnismäßig hohen Aufwand verbunden, vor allem wenn noch andere Frameworks zur Auswahl stehen. Dies war der Grund für das Ausscheiden von AWT.

Swing und *JavaFX* sind im Gegensatz zu SWT und AWT plattformunspezifisch, das heißt, eine Anwendung wird unabhängig vom Betriebssystem dargestellt. Das kann als Vorteil gesehen werden, da so den Benutzern aller Betriebssysteme eine gleiche oder sehr ähnliche Erfahrung geboten wird. Beide bieten komplexe graphische Elemente an, unterstützen Multithreading und lassen es relativ leicht zu, eigene Elemente zu erstellen. Die Wahl fiel auf *JavaFX*, weil dort das Aussehen einzelner Elemente oder auch der gesamten Anwendung mittels *Cascading Style Sheets* (CSS) angepasst werden kann. Außerdem bietet das *JavaFX SDK* ein einfaches konsolenbasiertes Werkzeug, um eine Anwendung mit allen benötigten Dateien in ein ausführbares Archiv zu packen und einfach auszuliefern.

5.1.1 Der Aufbau einer Anwendung mit *JavaFX*

Mit *JavaFX* wird die graphische Oberfläche einer Anwendung in verschiedene Szenen unterteilt. Jede Szene kann in einem eigenen Fenster angezeigt werden. Zu jeder dieser Szenen gehört ein *Scene graph*, der die graphischen Elemente einer Szene in einer baumähnlichen Datenstruktur hält. Je nach Art des Elements können andere Elemente darin aufgenommen werden. Die Elemente, die dies ermöglichen, erben von der gemeinsamen Java-Klasse `Parent` und entsprechen inneren Knoten im *Scene graph*. Andere Elemente, die keine Kindelemente aufnehmen, wie beispielsweise ein Textfeld oder eine Schaltfläche entsprechen Blättern im *Scene graph*.

Die graphische Oberfläche von *JavaFX* wird in einem eigenen Thread ausgeführt, dem `JavaFX Application Thread`. Änderungen am *Scene graph*, wie das Hinzufügen oder Entfernen eines Elements oder einem Baum von Elementen, sind nur in diesem Thread erlaubt. Wird dies in einem anderen Thread versucht, wird ein Fehler erzeugt.

Im `JavaFX Application Thread` können zwar auch lang andauernde Operationen ausgeführt werden, jedoch „hängt“ die Oberfläche dann und reagiert auf keine Benutzereingaben mehr bis die Operation abgeschlossen ist. Solche Operationen sollten deshalb in einem Hintergrundthread ausgeführt werden. Da die GUI von einem anderen Thread aus jedoch nicht manipuliert werden kann, wird eine andere Methode benötigt, um die Oberfläche über Fortschritte der Hintergrundoperation zu informieren. *JavaFX* bietet unter anderem dafür die Methode `Platform.runLater(Runnable)` an, welche den Code des `Runnable`-Objekts im `JavaFX Application Thread` ausführt. Somit kann eine Rückmeldung über den Fortschritt oder die Beendigung einer Hintergrundoperation an die GUI gemeldet werden.

5.1.2 Kommunikation zwischen der Oberfläche und dem Backend

Die Kommunikation zwischen den beiden Programmteilen GUI und Backend soll mittels MVC-Pattern erfolgen. Bei diesem Entwurfsmuster wird das komplette Pro-

gramm in die Teile **M**odel, **V**iew und **C**ontroller (deutsch: Modell, Präsentation und Steuerung) geteilt. Jedem Programmteil wird eine bestimmte Aufgabe zugewiesen. Das Modell enthält die Daten, welche von der Anwendung dargestellt werden sollen. Es ist unabhängig von Präsentation und Steuerung, aber kann mit diesen kommunizieren. Die Präsentation stellt die Daten aus dem Modell dar und wird über Änderungen informiert, um immer die aktuellen Daten anzuzeigen. Zusätzlich nimmt sie Benutzereingaben an und leitet sie an die Steuerung weiter. Die Steuerung muss die von der Präsentation erhaltenen Benutzereingaben verarbeiten und entsprechend das Modell und/oder die Präsentation verändern.

Die GUI übernimmt die Rolle der Präsentation, während das Backend sowohl das Modell als auch die Steuerung umfasst. Die GUI besitzt keine Methoden, um das Modell direkt zu verändern. Stattdessen werden alle Anfragen an die Steuerung innerhalb des Backends weitergeleitet. Dort wird je nach Anfrage und Zustand des Modells entschieden, ob beispielsweise eine Änderung erfolgen darf oder nicht. Die Steuerung ändert bei Bedarf das Modell, welches wiederum die GUI über die Änderung informiert.

Zur Modellierung dieser Zusammenhänge wurden gemeinsam mit *Andreas Lindner* die beiden Schnittstellen `Frontend` und `Backend` erstellt [vgl. Lin14]. Abbildung 5.1 zeigt einen Ausschnitt der Kommunikation zwischen dem `Frontend` und dem `Backend`. Das `Frontend` enthält nur Methoden, die vom `Backend` aufgerufen werden, um auf bestimmte Ereignisse hinzuweisen, beispielsweise, dass eine Komponente hinzugefügt wurde oder sich der Wert einer Eigenschaft geändert hat. Das `Backend` enthält einerseits Änderungsmethoden, mit denen eine gewünschte Änderung des Modells auf Gültigkeit überprüft werden kann. So eine Änderung kann das Hinzufügen einer Komponente oder das Ändern eines Wertes einer Eigenschaft sein. Andererseits enthält es Abfragemethoden, um Informationen über den Zustand des Modells einzuholen. Dazu gehört beispielsweise eine Methode, um alle Verbindungen zwischen den Komponenten eines Systems zu erhalten. Zusätzlich besitzt das `Backend` noch Methoden, um eine Implementierung der Schnittstelle `Frontend` als Listener, der dann über Ereignisse informiert wird, hinzuzufügen und zu entfernen.

Die in Abbildung 5.1 dargestellte Kommunikation zwischen `Frontend` und `Backend` läuft direkt und im gleichen Thread ab. Bei komplexen Systemen ist es möglich, dass das Backend so lange für die nötigen Berechnungen braucht, dass der Benutzer diese Verzögerung in der GUI bemerken kann. Deshalb bietet es sich an, diese Berechnungen in einem Hintergrundthread auszuführen.

Um dies zu erreichen, wurde die Klasse `BackgroundBridge` erstellt, welche die beiden Schnittstellen `Frontend` und `Backend` implementiert und bei der Erstellung je ein `Frontend`- und ein `Backend`-Objekt erhält. Die Idee dabei ist, dass diese Klasse alle Methodenaufrufe an das jeweilige Objekt weiterleitet, wobei die Aufrufe zum `Backend` in einem Hintergrundthread ausgeführt werden und die Aufrufe zum `Frontend` über die angesprochene Methode `Platform.runLater(Runnable)`

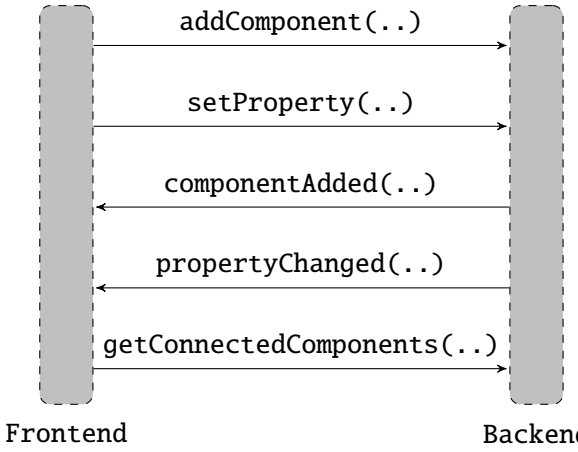

Abbildung 5.1: Auszug der Kommunikation zwischen `Frontend` und `Backend`.

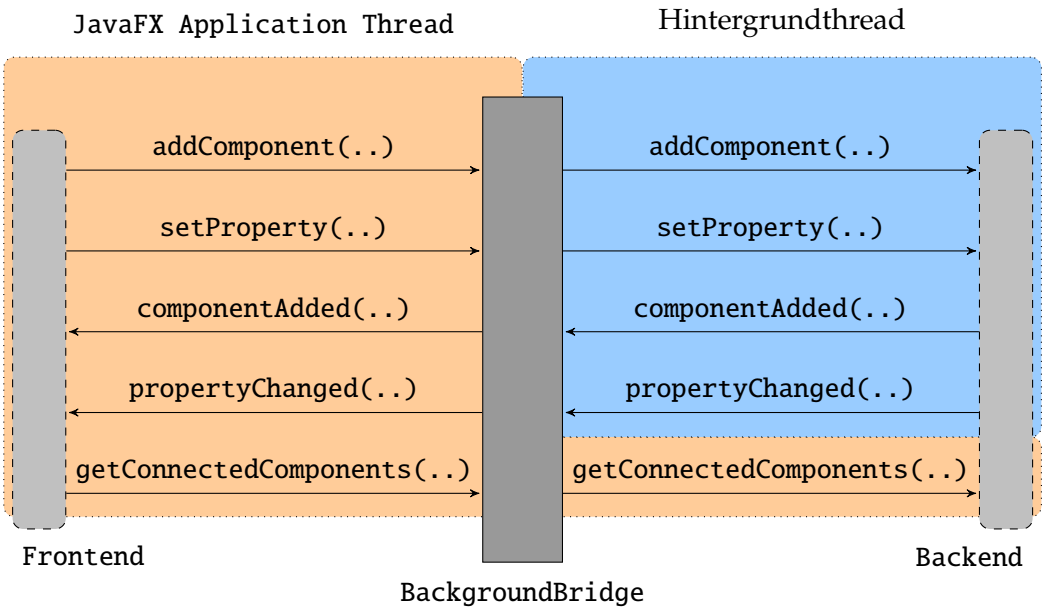

Abbildung 5.2: Auszug der Kommunikation zwischen `Frontend` und `Backend`. Hier mit der weiterleitenden Klasse `BackgroundBridge` und verschiedenen Threads.

im `JavaFX Application Thread` ausgeführt werden. Dazu wird eine Instanz der Klasse `BackgroundBridge` bei der Erstellung als Listener beim „echten“ `Backend` hinzugefügt. Sie leitet die Ereignisse an das gegebene `Frontend` weiter und stellt dadurch eine Brücke zwischen den beiden Programmteilen und verschiedenen Threads dar, wie in Abbildung 5.2 angedeutet. **Anmerkung:** Eine Ausnahme stellen dabei die Methoden dar, die den Zustand des Modells abfragen. Da bei diesen keine Hintergrundberechnungen nötig und damit keine Verzögerungen zu erwarten sind, können sie direkt im aufrufenden Thread, also dem `JavaFX Application Thread`, ausgeführt werden.

5.2 Aufbau der graphischen Oberfläche

In diesem Abschnitt wird die graphische Oberfläche von RangeConfig beschrieben. Die Oberfläche ist in englischer Sprache gehalten. Übersetzungen können aber leicht eingepflegt werden, da zur Internationalisierung der GUI die Klasse `java.util.ResourceBundle` mit einer entsprechenden **.properties*-Datei verwendet wurde. In dieser Datei sind die zu übersetzenden Texte in Schlüssel-Wert-Paaren abgelegt.

In Unterabschnitt 5.2.1 wird der Aufbau des Hauptfensters der Anwendung beschrieben. In Unterabschnitt 5.2.2 beziehungsweise Unterabschnitt 5.2.3 werden die Fenster gezeigt, welche die Bearbeitung einer Eigenschaft beziehungsweise eines Constraints erlauben. In Unterabschnitt 5.2.4 werden weitere kleinere Fenster erläutert.

5.2.1 Das Hauptfenster der graphischen Oberfläche

Der Benutzer soll sich in der graphischen Benutzeroberfläche sofort zurecht finden. Deshalb wurde das Hauptfenster klar und einfach aufgebaut. Das Hauptfenster dient zur Bearbeitung eines einzelnen Systems. Für die Bearbeitung eines Subsystems wird ein neues Fenster geöffnet.

Abbildung 5.3 zeigt eine verkleinerte Darstellung des Hauptfensters mit geladener CPS-Bibliothek. In der Titelleiste des Fensters wird der Pfad zur bearbeiteten Datei angezeigt oder ein Platzhalter, wie in diesem Fall. Der Umstand, dass im Dokument nach dem letzten Speichervorgang Änderungen vorgenommen wurden, wird durch ein Sternchen (*) vor dem Dateipfad angezeigt. Bei der Bearbeitung eines Subsystems wird nach dem Dateipfad der Name des Subsystems angezeigt. Die beiden Menüpunkte *File* und *Edit* erlauben das Speichern des gesamten Systems, das Hinzufügen und Entfernen einer Benutzerbibliothek, das Schließen der Anwendung und das Hinzufügen einer Eigenschaft oder eines Constraints zum aktuell betrachteten System. Der eigentliche Arbeitsbereich kann in drei Regionen aufgeteilt werden.

Abbildung 5.3: Verkleinerte Darstellung des Hauptfensters von RangeConfig.

In Region (1) wird die aktuell geladene Komponentenbibliothek angezeigt. Sie enthält eine Liste von vorgefertigten Komponenten, die per Drag-and-Drop zum System hinzugefügt werden können. Die ersten Komponenten (in der Kategorie *Component types*) sind „leer“, das heißt, sie besitzen keine Eigenschaften oder Modi und dienen dazu eine neue Komponente des entsprechenden Typs anzulegen. Darunter folgt die Kategorie *Base library* mit den Komponenten, die von der Basisbibliothek (in diesem Fall CPS) vordefiniert wurden. Danach folgt für jede geladene Benutzerbibliothek eine Kategorie, welche die Komponenten aus der jeweiligen Bibliothek enthält. Die geladenen Benutzerbibliotheken können über den Menüpunkt *File* > *Manage libraries...* verwaltet werden (siehe Unterabschnitt 5.2.4, Punkt „Auswahl der Komponentenbibliothek“).

In Region (2) werden alle Komponenten und Subsysteme des aktuell betrachteten Systems dargestellt. Sie dient als Ziel für die Drag-and-Drop-Geste, um Komponenten hinzuzufügen. Subsysteme werden über das Kontextmenü hinzugefügt. Jede Komponente (und jedes Subsystem) wird durch ein Rechteck dargestellt, welches mit der Farbe gefärbt ist, die durch die Komponentenbibliothek für den entsprechenden Komponententyp festgelegt wurde. In Abbildung 5.4a ist die Recheneinheit für die intelligente Weste zu sehen. Dieses Rechteck enthält den Namen und alle Eigenschaften der Komponente sowie zwei Links, welche die Anzahl der Modi beziehungsweise der Constraints dieser Komponente anzeigen (die Links sind nicht in der Abbildung enthalten). Die Links werden nur angezeigt, falls die Komponente auch tatsächlich Modi beziehungsweise Constraints besitzt. Durch einen Klick auf einen der beiden Links gelangt der Benutzer zu einem Fenster, welches die zur Komponente zugehörigen Modi oder Constraints auflistet. Optional kann das Rechteck durch einen Mausklick auf die Titelleiste verkleinert werden, um sichtbaren Platz zu sparen (siehe Abbildung 5.4b). Der Benutzer kann jedes Rechteck verschieben, in dem er auf das kleine Kreuz neben dem Komponentennamen klickt und die Maus zieht. Die Region hat Bildlaufleisten, die eingeblendet werden, sobald eine Komponente außerhalb des sichtbaren Bereichs verschoben wird.

(a) ausgeklappter Zustand (b) eingeklappter Zustand

Abbildung 5.4: Die Recheneinheit der intelligenten Weste.

Jede Komponente wird nur in einem Modus angezeigt. Welcher Modus angezeigt wird, kann vom Benutzer gewählt werden. Die Eigenschaften ändern ihren Wert je nach Belegung in den verschiedenen Modi. Ist eine Eigenschaft in einem Modus nicht gesetzt, wird ihr Wert aus dem Basismodus angezeigt und mit einem vorangestellten Sternchen (*) versehen. Über das Kontextmenü können die Eigenschaften bearbeitet und gelöscht beziehungsweise zurückgesetzt werden, falls ein anderer als der Basismodus angezeigt wird.

Verbindungen zwischen Komponenten werden mit einer Linie zwischen den dazugehörigen Rechtecken angedeutet. Beim Verschieben einer Komponente folgt die Verbindung. Verbindungen wurden als *Java-Interface* umgesetzt. Somit kann ihre Darstellung einfach ausgetauscht werden. Derzeit sind zwei Implementierungen vorhanden (eine einfache Linie und eine orthogonale Verbindung), von denen aber nur eine verwendet wird. Die Verbindungen wurden abstrakt gehalten, damit spezielle Verbindungstypen anders dargestellt werden können. Beispielsweise wären bei CPS neben kabelbasierten Verbindungen auch drahtlose Verbindungen denkbar, die durch gestrichelte Linien dargestellt werden könnten. Komponenten können auf zwei Wegen verbunden werden. Einerseits über eine Drag-and-Drop-Geste von einer Komponente auf die andere. Und andererseits über das Kontextmenü jeder Komponente und das Wählen des Menüpunktes *Connect with component...*, welches ein Fenster zur Auswahl der zweiten Komponente öffnet. Verbundene Komponenten werden über den Menüpunkt *Disconnect from component...* des Kontextmenüs getrennt.

Subsysteme können nicht mit anderen Komponenten verbunden werden. Ist ein Subsystem das Ziel einer Drag-and-Drop-Geste, wird ein Fenster zur Auswahl einer Komponente in diesem Subsystem geöffnet. Die Verbindungslinie führt trotzdem in das Rechteck des Subsystems, um die Verbindung anzudeuten. Sonst verhalten sich Subsysteme nicht anders als normale Komponenten.

In Region (3) werden die Eigenschaften des aktuell betrachteten Systems angezeigt. Die Darstellung gleicht der Darstellung der Eigenschaften von Komponenten. Auch hier werden zusätzlich zwei Links für die Modi und Constraints des Systems angezeigt. Wie bei den Komponenten können die Eigenschaften über das Kontextmenü verwaltet werden.

5.2.2 Bearbeiten einer Eigenschaft

Das Fenster zum Bearbeiten einer Eigenschaft ist relativ komplex aufgebaut, da alle drei Eigenschaftstypen (*Einzelwert*, *Intervall* und *Wertemenge*) unterstützt werden müssen. Es dient sowohl dem Anlegen als auch dem Bearbeiten einer Eigenschaft. Deshalb sind Eingabefelder für alle benötigten Informationen vorhanden:

- Den Namen der Eigenschaft,
- den Typ der Eigenschaft,
- eine optionale Einheit und
- den eigentlichen Wert der Eigenschaft.

Je nachdem, welcher Eigenschaftstyp ausgewählt ist, ändert sich der untere Bereich des Fensters. Eine Vorschau für die Darstellung der Eigenschaft ist immer vorhanden.

Die Eingabemaske für die Bearbeitung eines Einzelwertes besitzt nur ein einfaches Textfeld, in das der Wert eingegeben wird. In Abbildung 5.5 ist die Eingabe der Frequenz für die Recheneinheit der intelligenten Weste dargestellt. Während der Eingabe wird der Typ des Wertes überprüft und in der Vorschau angezeigt.

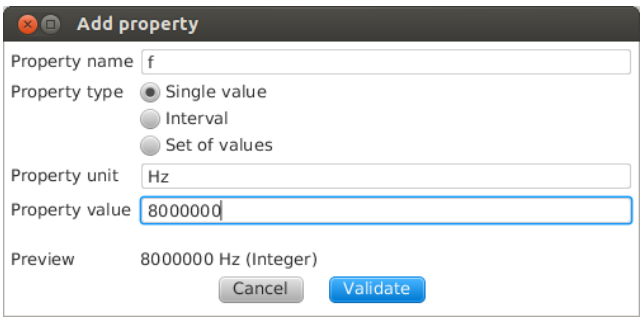

Abbildung 5.5: Anlegen des Einzelwerts für die Frequenz der Recheneinheit der intelligenten Weste.

Die Eingabemaske für die Bearbeitung eines Intervalls enthält einige zusätzliche graphische Objekte. Unter dem Eingabefeld für den Wert wird ein Hilfetext angezeigt, welcher den Benutzer bei der Erstellung eines Intervalls in korrekter Schreibweise unterstützt (siehe Abbildung 5.6). Ein Intervall kann entweder aus ganzen Zahlen oder Dezimalzahlen bestehen. Welche der beiden Möglichkeiten zutrifft, wird automatisch bei der Eingabe erkannt und durch das Kontrollkästchen *Discrete values* angezeigt. Durch das Ändern seines Zustandes kann der Benutzer seine Eingabe zwischen den beiden Formen konvertieren. Zusätzlich zu dem Vorschautext wird beim Bearbeiten eines Intervalls ein Zahlenstrahl angezeigt, welcher das Intervall graphisch darstellt. Die Schaltflächen darunter dienen zur Navigation im Zahlenstrahl und haben von links nach rechts folgende Bedeutungen: nach links

scrollen, heranzoomen, das komplette Intervall anzeigen, herauszoomen und nach rechts scrollen. Neben diesen Schaltflächen kann auch durch die Benutzung des Mausrads heran- und herausgezoomt und in Verbindung mit einer gehaltenen Steuerungstaste nach links beziehungsweise rechts gescrollt werden. Die Maske unterstützt die Eingabe von Intervallen mit Löchern, wie in Abbildung 5.7 dargestellt ist. In der Abbildung ist auch eine Möglichkeit zu sehen, eine unendliche Grenze anzugeben.

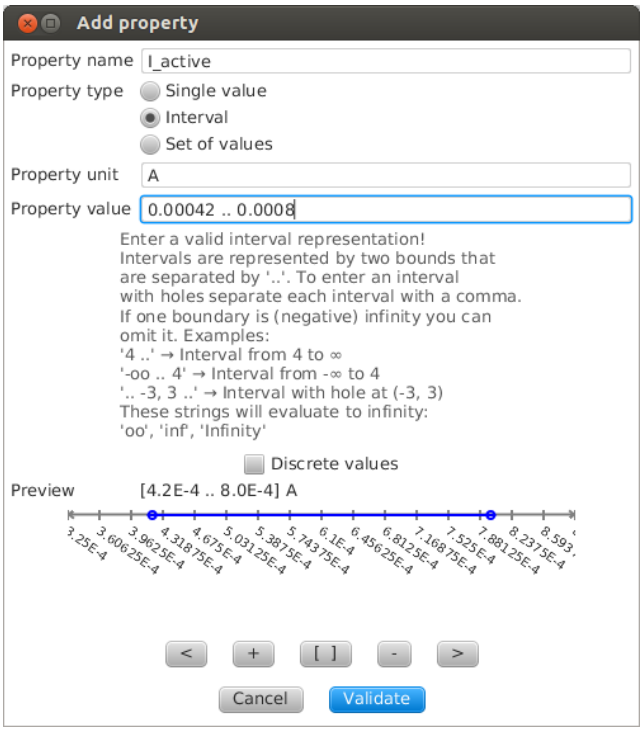

Abbildung 5.6: Anlegen des Intervalls für den benötigten Strom im aktiven Zustand der Recheneinheit der intelligenten Weste.

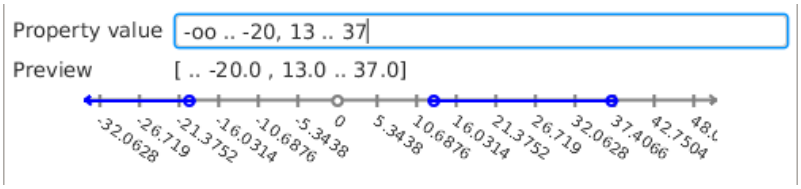

Abbildung 5.7: Eingabe eines Intervalls mit Loch und die daraus resultierende Vorschau.

Auch die Eingabemaske für Mengen von Werten enthält einen Hilfetext, um den Benutzer zu unterstützen (siehe Abbildung 5.8). Im Eingabefeld werden die Werte der Menge durch ein Komma getrennt eingegeben. Während der Eingabe wird auch hier automatisch der Typ der einzelnen Werte erkannt und in der Vorschau mit ausgegeben.

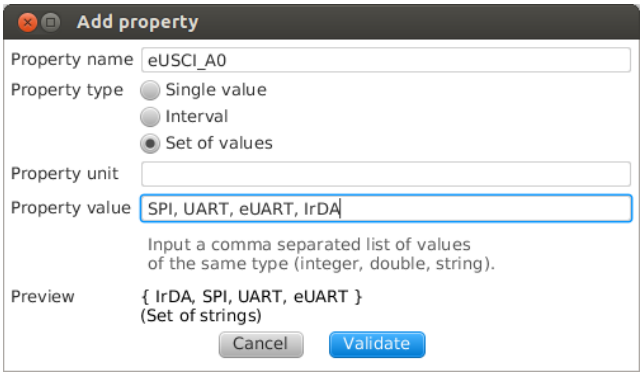

Abbildung 5.8: Anlegen der Menge der unterstützten Protokolle eines der Anschlüsse der Recheneinheit der intelligenten Weste.

Über die beiden Schaltflächen am unteren Ende des Fensters kann der Erstellungs-/Bearbeitungsprozess abgebrochen beziehungsweise die Eingabe validiert werden. Validieren heißt hier, dass eine Anfrage zur Erstellung/Änderung einer Eigenschaft an das Backend gestellt wird. Bei einer erfolgreichen Rückmeldung wird das Fenster geschlossen und der neue Wert dargestellt. Im Fall eines Fehlers bleibt das Fenster geöffnet und die Fehlermeldung wird angezeigt.

5.2.3 Bearbeiten eines Constraints

Bei der Bearbeitung eines Constraints muss der Benutzer das Prädikat und seine Argumente in Präfixform angeben. Das heißt, dass das Prädikat vor den Argumenten steht. Die Liste von Argumenten muss von runden Klammern eingeschlossen sein. Bei Prädikaten ohne Argument können die Klammern auch ausgelassen werden. Abbildung 5.9 zeigt die Eingabemaske während der Eingabe der Argumente. Ein Argument, das eine Eigenschaft referenziert, wird durch deren Namen repräsentiert. Damit der Benutzer sich nicht die Namen aller Komponenten und Eigenschaften merken muss, wurde ein Fenster entworfen, welches den Benutzer eine Eigenschaft auswählen lässt und über die Schaltfläche *Insert property...* erreichbar ist.

Die Eingabemaske für Constraints enthält einen Link, über den sich der Benutzer alle unterstützten Prädikate anzeigen lassen kann. Im Moment werden alle Prädikate des ECL^{i}PS^e Prolog-Solvers unterstützt und der Link öffnet die URL `http://eclipseclp.org/doc/bips/fullindex.html` in einem Webbrowser. Auf dieser Website werden alle Prädikate des Solvers aufgelistet.

Abbildung 5.10 zeigt das Fenster zur Auswahl einer Eigenschaft. Es enthält alle Eigenschaften eines gegebenen Systems und alle Komponenten innerhalb dieses Systems mit ihren Eigenschaften. Die Komponenten haben zur besseren Identifizierung eine farbige Markierung, welche mit der Farbe aus der Komponentenbibliothek

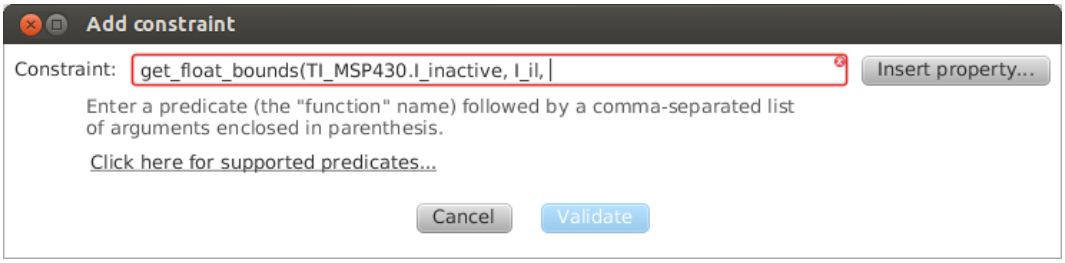

Abbildung 5.9: Eingabe eines Constraints, welches die Grenzen eines Intervalls auf zwei Eigenschaften abbildet.

übereinstimmt. Die Eigenschaften sind mit den Buchstaben „V“, „I“ oder „S“ markiert und deuten den Typ der Eigenschaft an. „V“ steht für einen einzelnen Wert, „I“ für ein Intervall und „S“ für eine Menge. Aus der Abbildung wird ersichtlich, dass die Komponenten und ihre Eigenschaften in einer Baumstruktur dargestellt werden. Somit bleiben auch komplexe Systeme übersichtlich. **Anmerkung:** Die Buchstaben „V“, „I“ und „S“ sind internationalisiert und können übersetzt werden.

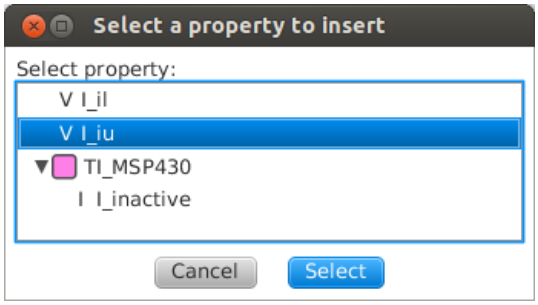

Abbildung 5.10: Fenster zur Auswahl einer Eigenschaft einer Komponente.

Nach einem Mausklick auf die Schaltfläche *Validate* wird überprüft, ob mindestens ein Argument des Prädikats eine Referenz auf eine Eigenschaft darstellt. Wenn dies der Fall ist, muss der Benutzer für jedes solches Argument bestimmen, ob das Prädikat den Wert der referenzierten Eigenschaft ändern darf oder nicht. Danach wird beim Backend angefragt, ob das Constraint erzeugt werden kann und gültig ist. Bei einem Fehler wird die Fehlermeldung angezeigt. Im Erfolgsfall wird das Fenster geschlossen und das Constraint hinzugefügt.

5.2.4 Sonstige Fenster

Die graphische Oberfläche besitzt selbstverständlich noch mehr Fenster. Einige von ihnen werden in diesem Unterabschnitt kurz erläutert.

Start der Anwendung Beim Start der Anwendung wird der Benutzer von einem Fenster begrüßt, welches ihm die Konfiguration eines neuen oder das Laden eines

gespeicherten Systems erlaubt (siehe Abbildung 5.11). Außerdem ist eine Schaltfläche enthalten, welche die Konfiguration eines CPS beschleunigt, indem automatisch die richtige Komponentenbibliothek gewählt wird. Falls die entsprechende Datei nicht gefunden werden kann, wird die Schaltfläche ausgeblendet.

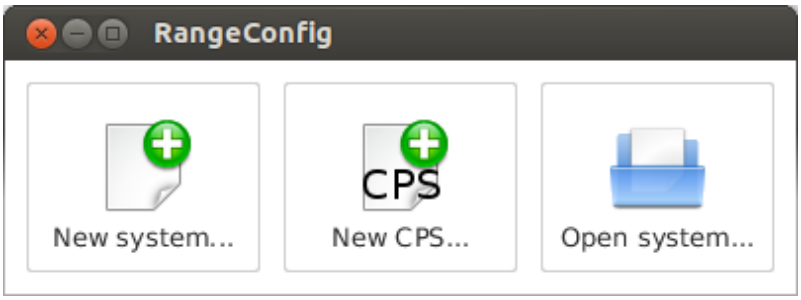

Abbildung 5.11: Begrüßungsfenster der GUI von RangeConfig.

Anzeigen der Constraints und Modi einer Komponente Der Benutzer kann alle Constraints und Modi einer Komponente in gesonderten Fenstern betrachten. Diese werden geöffnet, wenn er auf einen der entsprechenden Links in der Auflistung der Eigenschaften klickt. Die Constraints werden in Präfixform mit ihren Prädikaten und Argumenten aufgelistet, wie in Abbildung 5.12 dargestellt. In dem Fenster kann der Benutzer über die Schaltflächen unter der Liste ein neues Constraint zu der entsprechenden Komponente hinzufügen oder ein bestehendes bearbeiten oder löschen.

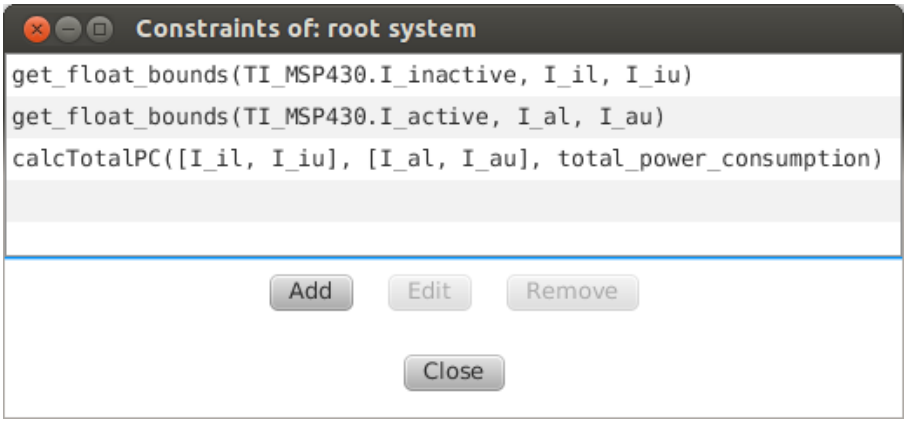

Abbildung 5.12: Fenster zur Auflistung aller Constraints einer Komponente (die dargestellten Constraints sind fiktiv).

Die verschiedenen Modi einer Komponente werden in tabellarischer Form dargestellt. Jede Spalte in der Tabelle entspricht einer Eigenschaft, jede Zeile entspricht einem Modus der Komponente. Die Zeile des aktiven Modus, welcher in der Übersicht angezeigt wird, ist mit fetter Schrift dargestellt. Wenn eine Eigenschaft in einem Modus nicht gesetzt ist, wird stattdessen der Wert aus dem Basismodus mit einem

vorangestellten Sternchen (*) angezeigt. Dies ist dieselbe Darstellungsweise wie in der Übersicht. Es stehen Schaltflächen zur Verfügung, welche das Erstellen und Löschen eines Modus erlauben, sowie das Auswählen des aktiven Modus.

Auswahl der Komponentenbibliothek Zur Auswahl der benutzten Komponentenbibliothek wurde ein Fenster erstellt, in welchem der Benutzer eine Basisbibliothek und eine beliebige Anzahl von Benutzerbibliotheken laden kann. In Abbildung 5.13 ist das Fenster mit ausgewählter CPS-Basis- und CPS-Benutzerbibliothek dargestellt. Es stehen Schaltflächen zur Verfügung, über die neue Benutzerbibliotheken hinzugefügt oder bereits hinzugefügte wieder entfernt werden können. Die Fläche auf der rechten Seite enthält eine Vorschau, welche genauso gestaltet ist, wie die Anzeige der Komponentenbibliothek im Hauptfenster der Anwendung.

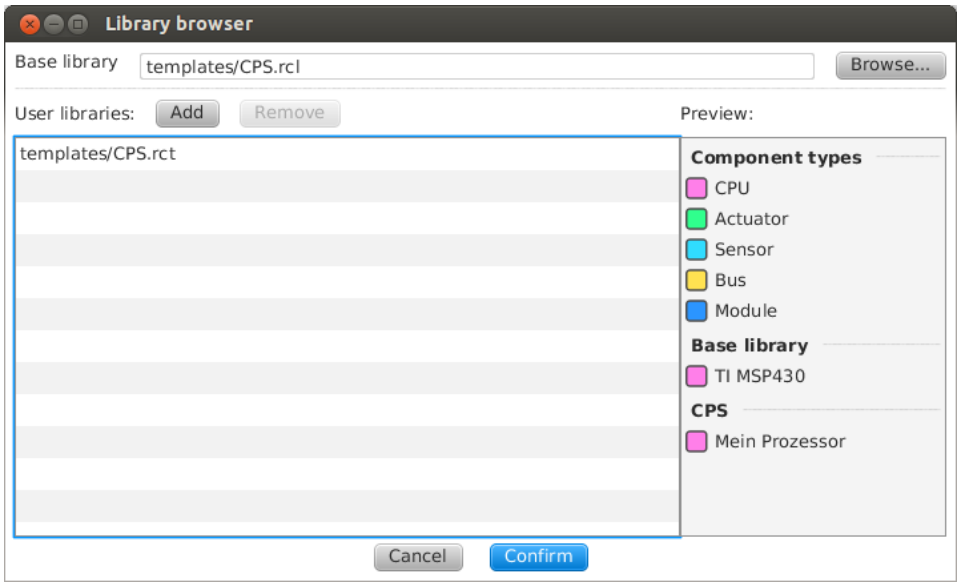

Abbildung 5.13: Fenster zur Zusammenstellung der Komponentenbibliothek (hier für CPS).

5.3 Bewertung der graphischen Oberfläche

In diesem Abschnitt wird nochmals auf die definierten Anforderungen an die GUI von RangeConfig eingegangen und analysiert, inwieweit sie bei der Entwicklung eingehalten werden konnten. Dabei werden vor allem die Anforderungen aus den User-Stories aus Abschnitt 3.3 betrachtet. Danach werden kurz die acht goldenen Regeln des Interface Design aus Unterabschnitt 3.1.2 thematisiert.

5.3.1 Musskriterien der graphischen Oberfläche

Graphische Darstellung des gesamten Systems mit Eigenschaften in mehreren Modi und Constraints Wie bereits in Unterabschnitt 5.2.1 angesprochen, wird jede Komponente durch ein farbiges Rechteck dargestellt und hat somit eine graphische Repräsentation. Die Rechtecke können bewegt werden, wodurch der Benutzer sie nach eigenem Belieben anordnen und sich so eine Übersicht schaffen kann. Damit er dabei noch mehr Freiheit hat, kann der Inhalt der Arbeitsfläche (Region (2) im Hauptfenster) verschoben werden. Wenn der Benutzer eine Komponente aus dem sichtbaren Bereich „herauszieht“, werden Bildlaufleisten eingeblendet, über welche er auf die gesamte Fläche zugreifen kann. Das dargestellte System kann nicht vergrößert oder verkleinert dargestellt werden. Deshalb könnte es bei komplexen Systemen, welche nicht komplett in den sichtbaren Bereich passen, zu Einschränkungen der Übersichtlichkeit kommen.

Von jeder Komponente wird nur ein Modus zur selben Zeit angezeigt. Der Benutzer kann selbst entscheiden, welcher Modus dargestellt werden soll. Dieser Kompromiss wurde eingegangen, da die graphischen Objekte für die Komponenten so relativ klein gehalten werden können. Über einige zusätzliche Mausklicks können trotzdem die Werte der Eigenschaften in anderen Modi angezeigt werden (im Fenster, welches alle Eigenschaften und Modi tabellarisch auflistet, siehe Unterabschnitt 5.2.4, Punkt „Anzeigen der Constraints und Modi einer Komponente“).

In der Übersicht wird nur die Anzahl der Constraints auf einer bestimmten Komponente angezeigt. Dadurch wird innerhalb der Darstellung einer Komponente viel Platz gespart, da die Systeme voraussichtlich sehr viele Constraints enthalten werden. Trotzdem können die Constraints durch nur einen Mausklick betrachtet und im dadurch geöffneten Fenster bearbeitet werden (siehe Unterabschnitt 5.2.4, Punkt „Anzeigen der Constraints und Modi einer Komponente“).

Anzeige eines Teils des modellierten Systems Das Hauptfenster stellt genau ein System mit seinen enthaltenen Komponenten dar. Jedes Subsystem kann in einem neuen Fenster geöffnet und dort betrachtet und bearbeitet werden. Die Darstellung einer benutzerdefinierten Auswahl von Komponenten ist nicht möglich.

Interaktivität der GUI Die GUI bleibt während den Berechnungen des Backends interaktiv und kann weiter benutzt werden. Dies wurde erreicht, indem die Oberfläche nebenläufig zum eigentlichen Konfigurator implementiert wurde. Trotzdem kann immer nur eine Anfrage an das Backend gestellt werden, sobald die letzte Berechnung beendet wurde. So wird verhindert, dass das Backend eine Berechnung auf der Basis von alten und eventuell bereits ungültigen Daten durchführt und somit eine ungültige Konfiguration entsteht.

Austauschbare und damit beliebige Komponententypen Das Backend behandelt jede Komponente gleichermaßen. Ihr Typ ist durch eine Zeichenkette festgelegt und dient nur zur graphischen Darstellung. Diese Tatsache wurde durch die Implementierung der Komponentenbibliotheken ausgenutzt. Durch verschiedene Bibliotheken kann jeder erdenkbare Komponententyp erzeugt und damit jeder mögliche Systemtyp konfiguriert beziehungsweise allgemeine Konfigurationsprobleme gelöst werden.

Speichern von Komponenten als Vorlage für ähnliche oder gleiche Komponenten Jede Komponente kann über ihr Kontextmenü als Vorlage gespeichert und wiederverwendet werden. Dazu muss der Benutzer eine Benutzerbibliothek auswählen, welche die Komponente aufnehmen soll. Es werden alle Eigenschaften und Modi der Komponente gespeichert.

Speichern und Laden von Systemen Ein modelliertes System kann in eine Datei abgespeichert werden. Dabei speichert die GUI einige zusätzliche Daten ab, die eigentlich nicht zur reinen Konfiguration nötig sind. Diese Informationen umfassen die verwendete Basisbibliothek und eventuelle Benutzerbibliotheken sowie die graphischen Positionen der Komponenten. Dies ist nötig, damit beim Laden derselbe Zustand und dieselbe Anordnung wiederhergestellt werden kann. Nachdem diese Daten gespeichert wurden, ist das Backend an der Reihe und speichert den internen Zustand des Systems in dieselbe Datei. Um ein gespeichertes System zu laden, werden die zusätzlichen Daten zunächst von der GUI ausgelesen, woraufhin das Backend den internen Zustand des Systems aus der Datei ließt.

Eingabe von Constraints Zu jeder Komponente und damit auch zu jedem System und Subsystem können Constraints hinzugefügt werden. Die Argumente ihrer Prädikate können sich auf Eigenschaften beziehen. Sie sind dabei nicht auf die Eigenschaften der Komponente des Constraints eingeschränkt, sondern können beliebige Eigenschaften im gesamten System referenzieren.

Einfache Bedienung Dem Benutzer wird die Verwendung der Oberfläche leicht gemacht. Um eine Komponente hinzuzufügen, muss er nur den entsprechenden Typ aus der Komponentenbibliothek in das Arbeitsfeld ziehen und einen Namen für die Komponente eingeben. An vielen Stellen im Hauptfenster werden Kontextmenüs angeboten, welche dem Benutzer den Zugriff auf weitere mögliche Aktionen erlauben.

Bei der Bearbeitung von Eigenschaften erhält der Benutzer je nach Eigenschaftstyp eine Hilfestellung. Seine Eingaben werden je nach Typ unterschiedlich interpretiert. Das Bearbeiten von Constraints geschieht durch eine simple Eingabemaske, die leider keine fortschrittlichen Hilfen, wie beispielsweise eine Autovervollständigung

von Prädikaten, anbietet. Das Verbinden von Komponenten geschieht graphisch und kann entweder intuitiv durch das Ziehen von einer Komponente auf eine andere oder durch ein eigenes Fenster erledigt werden. Das Lösen von Verbindungen geschieht über ein eigenes Fenster.

5.3.2 Kannkriterien der graphischen Oberfläche

Automatische Anordnung der Komponenten Der Benutzer ist selbst dafür verantwortlich, die Komponenten auf der Arbeitsfläche anzuordnen. Die Implementierung eines Layout-Managers war in der vorgegebenen Arbeitszeit nicht möglich. Stattdessen wurden die wichtigeren Musskriterien umgesetzt.

Darstellung von verletzten Constraints Es werden keine verletzten Constraints angezeigt. Dies ist auch gar nicht notwendig, da das System nicht in einen Zustand mit verletzten Constraints gebracht werden kann. Wird beispielsweise bei der Änderung eines Eigenschaftswertes ein Constraint verletzt, erhält der Benutzer eine Rückmeldung, dass die Änderung fehlschlug. Er kann dann entweder eine gültige Eingabe tätigen oder die Bearbeitung abbrechen.

Vorgefertigte Komponenten (je nach Typ des Systems) Es können vorgefertigte Komponenten verwendet werden. Sie sind in Komponentenbibliotheken definiert und gehören somit zu einem spezifischen Systemtyp. Mitgeliefert ist ein beispielhafter Mikrocontroller für die CPS-Bibliothek. Andere vorgefertigte Komponenten können entweder in der Basisbibliothek oder einer Benutzerbibliothek des jeweiligen Systemtyps angegeben werden.

Exportieren und Importieren von Subsystemen Der Benutzer kann kein einzelnes Subsystem exportieren. In der GUI ist die entsprechende Funktionalität zwar vorhanden, aber deaktiviert, da das Backend nur das gesamte System speichern kann. Dennoch kann der Benutzer ein vorhandenes und gespeichertes System als Subsystem importieren. Dabei werden alle Komponenten, Eigenschaften und Constraints des Systems in das neue Subsystem eingefügt.

5.3.3 Goldene Regeln des Interface Design

In diesem Unterabschnitt werden die in Unterabschnitt 3.1.2 angesprochenen goldenen Regeln des Interface Design von Shneiderman und Plaisant [SP10] auf die Benutzeroberfläche von RangeConfig angewendet. Dazu wird die Einhaltung jeder der acht Regeln bewertet.

Bemühe Dich um Konsistenz *JavaFX* bringt bereits einen vorgefertigten CSS-Stil mit. Dieser sorgt unter anderem dafür, dass alle graphischen Elemente dieselbe Farbpalette verwenden. Besonders sticht hierbei ein Blauton hervor, der allgemein verwendet wird, um anzuzeigen, welches Element aktuell den Eingabefokus besitzt. Außerdem haben alle Schaltflächen das gleiche Aussehen, wobei hier zwischen normalen Schaltflächen und solchen unterschieden wird, die ausgelöst werden, wenn die Eingabetaste betätigt wird (siehe hierzu unter anderem Abbildung 5.10). Durch diese Tatsache ist sichergestellt, dass die gesamte Oberfläche ein einheitliches Aussehen hat. Die Schaltflächen in Fenstern, in welchen ein Erstellungs- oder Bearbeitungsprozess stattfindet, sind stets nach demselben Schema benannt. Die Schaltfläche zur Bestätigung einer Aktion befindet sich stets auf der rechten Seite, wohingegen der Abbruchknopf links positioniert ist.

Sorge für umfassende Benutzbarkeit Es wurden einige Tastenkombinationen integriert, zum Beispiel für das Schließen der Anwendung beziehungsweise eines Fensters und für das Speichern. Diese können immer verwendet werden und hängen nicht vom Kenntnisstand des Benutzers ab. Die Tastenkombinationen sind zwar nicht konfigurierbar, aber sie wurden an allgemein bekannte Kombinationen angelehnt (zum Beispiel veranlasst die Kombination `Strg + S` das Speichern des Systems).

Gib aufschlussreiche Rückmeldungen Jede Aktion in der GUI hat eine sichtbare Reaktion zur Folge. Dazu gehören das Hinzufügen und Entfernen von Komponenten, das Erstellen und Lösen von Verbindungen zwischen Komponenten, das Hinzufügen, Bearbeiten und Entfernen von Eigenschaften und Constraints und das Hinzufügen und Entfernen von Modi. Auch Änderungen an der verwendeten Komponentenbibliothek werden angezeigt.

Entwirf Dialoge für abgeschlossene Handlungen Handlungen, die mehrere Dialoge umfassen, werden abgeschlossen, indem der letzte dazugehörige Dialog geschlossen wird. Beispielsweise muss bei der Erstellung eines Constraints, welches eine Eigenschaft einschränkt, angegeben werden, ob das Prädikat die Eigenschaft ändern darf oder nicht. Nach der Überprüfung durch das Backend wird das Fenster geschlossen und der Benutzer landet entweder in der Übersicht über alle Constraints der zugehörigen Komponente oder auf dem Hauptfenster. Der Benutzer erkennt den Abschluss daran, dass keine Fenster geöffnet bleiben, welche die entsprechende Handlung betreffen.

Verhindere Fehler Sehr viele Fehler werden bereits im vornherein ausgeschlossen, indem die Oberfläche die Eingaben überprüft. Beispielsweise werden die Namen von Komponenten und Eigenschaften auf ungültige Zeichen und Einzigartigkeit überprüft. Außerdem werden Schaltflächen deaktiviert, wenn noch nicht alle benötigten

Eingabefelder korrekt ausgefüllt wurden. Fehler, die direkt in der Oberfläche abgefangen werden können, werden dem Benutzer durch aussagekräftige Meldungen mitgeteilt. Sollte im Backend ein Fehler entstehen, wird der GUI eine Fehlermeldung übergeben. Leider hat die Oberfläche keinen Einfluss auf den Inhalt der Meldung, weil keine umfassende Schnittstelle für Fehlermeldungen implementiert wurde. Unter Umständen kann es also vorkommen, dass eine Fehlermeldung angezeigt wird, mit welcher der Benutzer nichts anfangen kann.

Erlaube das Rückgängigmachen von Aktionen Die Oberfläche erlaubt leider nicht die Umkehrung von Aktionen, da das Backend keine Möglichkeit anbietet, einen vorherigen Zustand wiederherzustellen. Trotzdem können fast alle Aktionen rückgängig gemacht werden, indem zum Beispiel Komponenten wieder entfernt, Verbindungen wieder gelöst oder Constraint wieder gelöscht werden.

Überlasse dem Benutzer die Kontrolle Die graphische Oberfläche öffnet keine Fenster zu unerwarteten Zeiten, sondern nur, wenn es durch den Benutzer veranlasst wird. Die GUI könnte zum Beispiel direkt nach dem Hinzufügen eines Subsystems ein Fenster zur Bearbeitung öffnen. Dadurch könnten Benutzer jedoch verwirrt werden, da das geöffnete Fenster genau dieselbe Größe und einen sehr ähnlichen Aufbau hat wie das darunterliegende und somit kaum Unterschiede zu erkennen sind. Deshalb wird das Subsystem erst auf den Befehl des Benutzers hin geöffnet.

Entlaste das Kurzzeitgedächtnis Der Benutzer muss sich kaum Informationen merken, die er in einem anderen Teil der Oberfläche benötigt. Das wird vor allem dadurch erreicht, dass das Hauptfenster viele wichtige Informationen anzeigt, wie zum Beispiel alle vorhandenen Eigenschaften der Komponenten eines Systems. Auch beim Anlegen eines Constraints, dessen Argumente Eigenschaften referenzieren, kann der Benutzer die Eigenschaften über ein Fenster auswählen und muss sich nicht ihre Namen und genauen Schreibweisen merken.

Die acht goldenen Regeln von Shneiderman und Plaisant [SP10] geben nützliche Richtlinien für die Gestaltung einer graphischen Benutzeroberfläche vor. Bei der Programmierung ist es aufgrund der beschränkten Arbeitszeit nicht gelungen alle Regeln einzuhalten, wobei es aufgrund ihrer natürlichsprachlichen Formulierung schwierig ist, den genauen Grad der Einhaltung der einzelnen Regeln anzugeben.

6 Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde die Erstellung einer graphischen Benutzeroberfläche für den constraint-basierten Konfigurator RangeConfig behandelt. Für das Verständnis wurde ein Überblick über interaktive Produktkonfiguration, Constraints und CPS geschaffen. Außerdem wurden einige Grundlagen des großen GUI-Themenbereichs vermittelt und Anforderungen an die Oberfläche von RangeConfig aufgestellt.

Es wurde das verwendete Datenmodell erläutert. Da der eigentliche Konfigurator parallel zu dieser Arbeit im Rahmen einer weiteren Bachelorarbeit von *Andreas Lindner* entwickelt wurde, konnte eine gemeinsame Schnittstelle entwickelt werden, über welche die beiden Programmteile kommunizieren. Sowohl die Schnittstelle als auch das verwendete Datenmodell wurden mehrmals überarbeitet, um sie an neu entdeckte Anforderungen anzupassen. Ihr Aufbau lehnt sich sehr stark an die Konfiguration von CPS an, da dies die erste Anwendungsdomäne von RangeConfig ist.

Es wurden die Umsetzung und der Aufbau der graphischen Benutzeroberfläche erklärt. Dabei wurden einige Fenster mit ihren Funktionen vorgestellt. Außerdem wurde die Oberfläche bezüglich der Einhaltung einiger gestellter Anforderungen und Regeln bewertet. In der GUI wurden alle Anforderungen umgesetzt, die als Musskriterien festgelegt waren. Bei einigen Kannkriterien bestehen jedoch noch Möglichkeiten für weitere Verbesserungen. Beispielsweise ist kein Layoutmanager implementiert, welcher die graphischen Objekte der Komponenten nach bestimmten Vorgaben automatisch anordnen kann. Auch die acht goldenen Regeln wurden weitgehend eingehalten, wobei immer noch Möglichkeiten zur Verbesserung bestehen.

Es existieren aber auch einige andere Dinge, die noch nicht optimal implementiert sind. Die Darstellung von Constraints gibt dem Benutzer nur geringe Hinweise darauf, welche Eigenschaften in anderen Komponenten referenziert werden. Um die Referenzen besser zu verdeutlichen, könnten bei der Auswahl eines Constraints farbige Linien zu allen Komponenten führen, die referenzierte Eigenschaften enthalten, oder diese Eigenschaften könnten durch eine farbige Schrift hervorgehoben werden.

Einige Eingabemasken können überarbeitet werden. Beispielsweise wäre bei der Eingabe von Constraints eine automatische Vervollständigung der Prädikate mit Angabe der erwarteten Argumente denkbar. Ein anderes Beispiel ist die Bearbeitung einer Eigenschaft, welche eine Menge von Werten enthält. Hier könnte man durch

Anklicken für jedes Element der Menge bestimmen, ob es weiterhin in der Menge enthalten sein soll oder nicht.

Der Benutzer hat keine Möglichkeit Komponentenbasisbibliotheken über die graphische Oberfläche zu erstellen. Dazu muss er gegenwärtig XML-Dateien erstellen und richtig bearbeiten. Er kann zwar die CPS-Bibliothek als Vorlage verwenden, dies bedeutet aber trotzdem einigen Aufwand. Deshalb wäre es denkbar einen Editor für Komponentenbibliotheken zu realisieren.

Zum derzeitigen Stand gibt die Anwendung dem Benutzer keinerlei Hinweis darauf, ob eine Eigenschaft aufgrund eines Constraints geändert beziehungsweise eingeschränkt wurde. So eine Funktionalität wäre auf der GUI-Seite leicht zu implementieren, erfordert aber Änderungen an der Schnittstelle zum Backend. Die Rückmeldung `propertyChanged(..)` vom Backend zur GUI müsste zusätzlich den Initiator der Änderung übergeben, damit diese Information von der GUI interpretiert werden kann.

In der Anwendung gibt es für den Benutzer keine Möglichkeit neue benutzerdefinierte Prädikate zu erstellen. Dies steht der Verwendung von RangeConfig zur Lösung von speziellen Konfigurationsproblemen, wie CPS, entgegen. Wie bereits angedeutet, lehnt sich die komplette Anwendung noch sehr an die Planung von CPS an. Um allgemeinere Konfigurationsprobleme zu lösen, könnte das Konzept der Bibliotheken weitergeführt werden. In einer Bibliothek könnten nicht nur Komponententypen festgelegt werden, sondern auch welche Eigenschaftstypen, Prädikate und Verbindungen zwischen den Komponenten verfügbar sind. Diese Änderungen sind aber nicht nur in der GUI zu tätigen, sondern hätten auch Auswirkungen auf das Backend.

Noch tiefgreifender wäre die Umsetzung eines Konzepts, welches die Benutzer nach ihrer Rolle unterscheidet. Dies wurde bereits am Ende von Abschnitt 2.2 angeschnitten. Das Konzept könnte die Benutzer in zwei Gruppen einteilen. Der „Entwickler“ modelliert eine Klasse von Konfigurationsproblemen, wie beispielsweise CPS, legt dazu die verfügbaren Komponenten-, Eigenschaftstypen und Prädikate fest und erstellt vorgefertigte Komponenten. Der „Benutzer“ kann dann unter Einhaltung der Vorgaben ein konkretes Konfigurationsproblem der Klasse lösen. Die dafür notwendigen Änderungen sind vermutlich sehr komplex und zeitaufwändig.

Schlussendlich bleibt zu sagen, dass eine funktionsfähige Benutzeroberfläche für RangeConfig entstanden ist. Trotzdem bietet die bisherige Implementierung noch viel Freiraum für Verbesserungen. Das umfasst nicht nur die graphische Benutzeroberfläche, sondern auch den eigentlichen Konfigurator.

Literatur

[Alt] *PC-Konfigurator des Computeronlineshops ALTERNATE*. URL: `https://www.alternate.de/html/configurator/builder/pc/page.html` (besucht am 11. 04. 2014).

[HA04] Tarik Hadzic und Henrik Reif Andersen. *An introduction to solving interactive configuration problems*. Techn. Ber. TR-2004-49, The IT University of Copenhagen, 2004.

[Her76] Hans Hermes. *Einführung in die mathematische Logik: klassische Prädikatenlogik*. Teubner, 1976.

[HW07] Petra Hofstedt und Armin Wolf. *Einführung in die Constraint-Programmierung: Grundlagen, Methoden, Sprachen, Anwendungen*. Springer DE, 2007.

[Hö+13] Benny Höckner u. a. „Constraint-based Approach for an Early Inspection of the Feasibility of Cyber Physical Systems“. In: *Kiel Declarative Programming Days 2013* (2013), S. 183.

[Lin14] Andreas Lindner. „Entwicklung eines constraint-basierten Produktkonfigurators RangeConfig unter Nutzung von Finite-Domain- und Intervall-Constraints“. Bachelorarbeit. Brandenburgische Technische Universität Cottbus – Senftenberg, 2014.

[Nie14] Antoni Niederliński. *A Gentle Guide to Constraint Logic Programming via ECLiPSe*. Jacek Skalmierski Computer Studio, 2014.

[Pil00] Frank Thomas Piller. „Mass Customization“. In: *Handbuch Produktmanagement*. Hrsg. von Sönke Albers und Andreas Herrmann. Gabler Verlag, 2000, S. 883–907. ISBN: 978-3-663-05718-5.

[SG11] Denny Schneeweiß-Göritz. „Grafische, interaktive Produktkonfiguration mit Finite-Domain-Constraints“. Diplomarbeit. Brandenburgische Technische Universität Cottbus, 2011.

[SP10] Ben Shneiderman und Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 5. Aufl. Upper Saddle River, NJ: Pearson Addison-Wesley, 2010.

[Sto07] Henrik Stormer. „Kundenbasierte Produktkonfiguration“. In: *Informatik-Spektrum* 30.5 (2007), S. 322–326.

[Sub+04] Sathiamoorthy Subbarayan u. a. „Comparing two implementations of a complete and backtrack-free interactive configurator“. In: *Proceedings of the CP-04 Workshop on CSP Techniques with Immediate Application*. 2004, S. 97–111.

[TJ01] Mitchell M. Tseng und Jianxin Jiao. „Mass Customization“. In: *Handbook of Industrial Engineering – Technology and Operations Management*. New York: John Wiley & Sons, 2001.

Abbildungsverzeichnis

Anhang

Im folgenden Listing 1 ist die XML-Datei für die Basisbibliothek für CPS zu sehen. Listing 2 (auf Seite X) zeigt die XML-Datei einer Benutzerbibliothek für CPS. Die beiden Dateien werden von der Komponentenbibliothek verwendet. Für nähere Informationen siehe Abschnitt 4.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <library name="Cyber Physical System" abbreviation="CPS"
       version="1" subsystem-color="#c4c4c4">
3     <component-types>
4        <component-type name="CPU" color="#ff7fe9" />
5        <component-type name="Actuator" color="#30ff8d" />
6        <component-type name="Sensor" color="#30dcff" />
7        <component-type name="Bus" color="#ffe251" />
8        <component-type name="Module" color="#2b95ff" />
9     </component-types>
10    <templates>
11       <component type="CPU" name="TI MSP430">
12          <properties>
13             <property id="1" type="SINGLE_VALUE" name="Ports
                  " value="7" unit="" />
14             <property id="2" type="INTERVAL" name="Frequency
                  " value="[32768000 .. 16000000]" unit="Hz" />
15             <property id="3" type="SET" name="Protocols"
                  value="{UART , SPI , I2C}" unit="" />
16          </properties>
17          <mode id="1">
18             <property id="1" value="18" />
19             <property id="2" value="[1 .. 10]" />
20             <property id="3" value="SPI" />
21          </mode>
22       </component>
23    </templates>
24 </library>
```

Listing 1: Aufbau der Basisbiliotheksdatei für CPS.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <templates library="CPS" version="1">
3     <component type="CPU" name="Mein Prozessor">
4        <properties>
5           <property id="1" type="SINGLE_VALUE" name="Ports"
               value="7" unit="" />
6           <property id="2" type="INTERVAL" name="Frequency"
               value="[32768000 .. 16000000]" unit="Hz" />
7           <property id="3" type="SET" name="Protocols" value=
               "{UART , SPI , I2C}" unit="" />
8        </properties>
9        <mode id="1">
10          <property id="1" value="1" />
11          <property id="2" value="[327680 .. 160000]" />
12          <property id="3" value="UART" />
13       </mode>
14    </component>
15 </templates>
```

Listing 2: Aufbau einer Benutzerbibliotheksdatei für CPS.

Selbstständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Cottbus, den 6. Mai 2014

Andreas Fehn