PSE - VINJAB: Resultate der Qualitätssicherung

Jonas Haas Nicolas Schreiber Valentin Springsklee
Yimeng Zhu David Grajzel

29. Februar 2016

Inhaltsverzeichnis

Kapitel 1

Einführung

Vorwort

Um die Testresultate auch später reproduzieren zu können, lohnt es sich die verwendete Umgebung zu dokumentieren.

1.1 Entwicklungsumgebung

- IDE: JetBrains WebStorm (11.0)

1.2 Ausführungsumgebung

1.2.1 Server

- Betriebssystem: Raspbian (4.1)
- HTTP Server: Node.js (5.7.0)

1.2.2 Client

- Webbrowser: Chrome (44.0) oder Firefox (42.0)

1.3 Werkzeuge fürs Testen

- Jasmine
  Um unsere Testfälle einfach und automatisiert durchlaufen lassen zu können, haben wir uns entschlossen eine bekannte Javascript-Testumgebung zu verwenden, die Jasmine heißt. Man schreibt die Testfälle in Typescript und danach führt man Jasmine aus: man soll in einer HTML Datei die Javascript Dateien von Jasmine, die Testdateien und die zu testende Dateien referenzieren. Wenn man die HTML Datei öffnet, werden die Tests durchgeführt und die Resultate automatisch angezeigt (Abb. 1.1).
- Karma
  Damit wir wissen, welche Code-Zeilen wir schon überprüft haben ist es

von Vorteil ein Werkzeug zu haben, das uns die Codeabdeckung anzeigt (Abb. 1.2) und uns beim Schreiben von Testfällen unterstützt (Abb. 1.3). Als Tool zur Messung der Codeabdeckung haben wir Karma (verwendet intern Istanbul) ausgewählt.

Jasmine

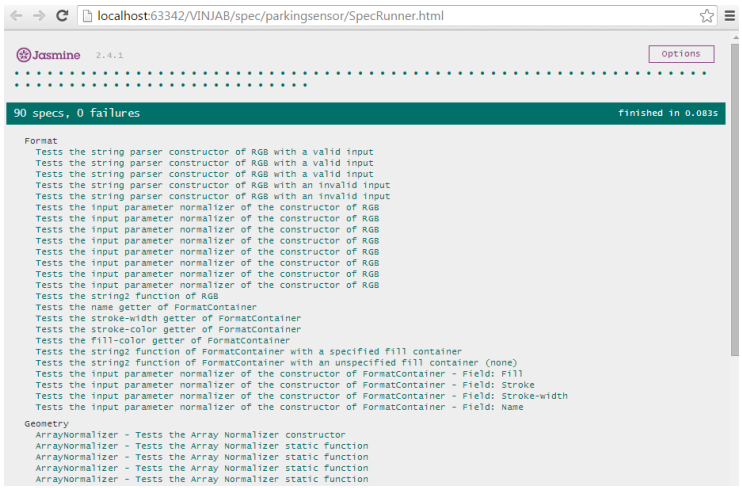

Abbildung 1.1: So werden die Testresultate in Chrome dargestellt

Karma-Übersicht

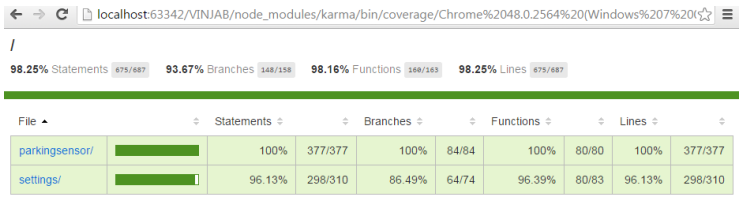

Abbildung 1.2: Codeabdeckung bei den einzelnen Modulen

Karma-Abdeckung

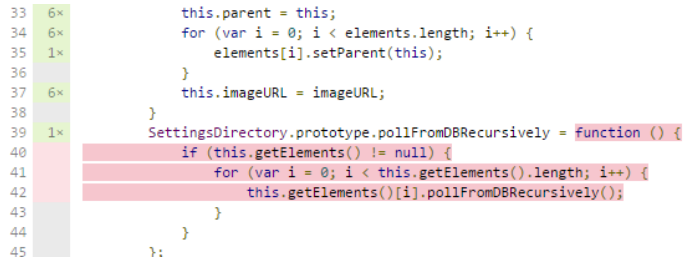

Abbildung 1.3: Die nicht abgedeckte Codezeilen werden rot markiert

Kapitel 2

Statische Code-Analyse

2.1 Ziele und Resultate

Wir haben die statische Code-Analyse durchgeführt und versucht dabei die Anzahl der Warnungen zu minimieren. Es ist aber zu beachten, dass wir die statische Code-Analyse nur als Hilfe betrachten und es gehört nicht zu unseren Zielen alle Warnungen zu unterdrücken. In vielen Fällen machen die Warnungen keinen Sinn und obwohl man die unterdrücken kann, wäre das Endergebnis schlechter. In solchen Situationen haben wir nichts unternommen.

2.2 Resultate

Hier sieht man eine kurze Zusammenfassung, welche und wie viele Fehler wir behoben haben.

2.2.1 Einparkhilfe

Die häufigste Fehler waren:

- Variable x redundant in solchen Zeilen: var x = func(); return x; (statt return func();)
- Typo, myvariable wurde statt myVariable verwendet.

Modul	Warnungen	Davon behoben	LOC	Hinweis
Communicator	2	1	29	Verzeichnisname nicht geändert
Format	4	4	160	-
Geometry	12	12	397	-
ParkingSensorStarter	28	2	166	XML lowercase Namen ignoriert
Primitive	1	1	107	-
PSGUI	20	16	165	-
Visualization	11	9	300	-

2.2.2 Settings

Die häufigste Fehler waren:

- Typo, myvariable wurde statt myVariable verwendet.
- Typos in String Konstanten und in HTML-Parametern (z.B. tbody in document.createElement('tbody');)

Modul	Warnungen	Davon behoben	LOC	Hinweis
CompositeStructure	41	38	618	Alte Kommentare, 25x ruid Typo.
Renderer	17	10	404	HTML String Typos nicht geändert.
SettingsData	4	1	188	No Typo error in word splitted.
TSettings	55	40	383	Typos in Strings nicht geändert.

2.2.3 BluetoothOBD

Die häufigste Fehler waren:

- Der Inspektor kann viele Befehle nicht erkennen. Deswegen sind keine Warnungen behoben worden.

Modul	Warnungen	Davon behoben	LOC	Hinweis
BluetoothOBD	11	0	141	-

2.2.4 Proxy

Die häufigste Fehler waren:

- Die Namen der Funktionen werden mit Abkürzung geschrieben.

Modul	Warnungen	Davon behoben	LOC	Hinweis
Proxy	2	0	76	-

2.2.5 Terminal

Die häufigste Fehler waren:

- Die Namen der Funktionen werden mit Abkürzung geschrieben.

Modul	Warnungen	Davon behoben	LOC	Hinweis
Terminal	12	0	95	-

2.2.6 Server

Die häufigste Fehler waren:

- Der Code Inspektor kennt kein Deutsch. Die deutsche Wörte werden als "falsch buchstabiert"gezeichnet.

Modul	Warnungen	Davon behoben	LOC	Hinweis
Server	3	0	101	-

2.2.7 Server Starter

Die häufigste Fehler waren:

- Der Name der Datenbank LevelUp wird vom Code Inspektor nicht erkannt.

Modul	Warnungen	Davon behoben	LOC	Hinweis
ServerStarter	5	0	50	-

Kapitel 3

Unit Tests

3.1 Aggregierte Funktionen

Allgemeine Schwierigkeiten

Von jedem Topic kann der Durchschnittswert mit der Klasse berechnet werden. Der Durchschnittswert wird zeitgewichtet berechnet. Zu Testzwecken können Dummy-Werte durch synchrone oder asynchrone Methodenaufrufe auf dem Server-Side Bus verteilt werden. Allerdings ist im Allgemeinen nicht möglich, dem Durchschnittswert vorherzusehen, weil die Laufzeiten der Funktionen nicht bekannt sind.

3.1.1 Abweichungen beim Durchschnitt

- Symptom: Als Durchschnitt der beiden Werte 22 und 24 wird 23.0000848 berechnet.
- Grund: Der Durchschnitt wird zeitgewichtet berechnet und die Laufzeit der Funktion ist nicht vorhersehbar.
- Behebung: Fehler tritt auf schnellen Maschinen nicht auf (Referenz: i7 4790)

3.2 Bluetooth OBD

3.2.1 NO DATA Object empfangen

- Symptom: Bluetooth antwortet mit einem Object, das nur NO DATA Information enthält.
- Grund: Die Befehle der Anforderung der Daten sind zu schnell nach OBD geschickt.
- Behebung: Ein Befehl wird erst nach OBD geschickt, wenn die Antwort der letzten Anfrage ankommt.

3.2.2 Falsche Daten empfangen

- Symptom: Die Drehzahl ist nicht der Realität entspricht.
- Grund: Die Umrechnung der empfangenen Daten ist nicht korrekt.
- Behebung: Die Umrechnung der entsprechenden Daten ist nach richtiger Formel korrigiert.

3.3 UI

Allgemeine Schwierigkeiten

Mehrere der genutzten Bibliotheken stellen eine Erweiterung von JQuery dar. Diese Erweiterung scheint jedoch nicht mit Karma zu funktionieren. Ohne Karma treten diese Probleme nicht auf.

3.3.1 Datamodel Werte nicht definiert

- Symptom: Error: "topic is undefined"beim Erstellen eines Datenmodells
- Grund: Bei einem undefinierten Signalname versucht der PostalBus auf null zu subscriben. Dies warf natürlich einen Error.
- Behebung: Überprüfe vor subscribe auf 'undefined'

3.3.2 Grid nicht zerstörbar

- Symptom: Beim Löschen des HTML-Elements und erneuten erzeugen treten Fehler im Hinzufügen von Widgets auf.
- Grund: Das Grid selbst wird nicht gelöscht. Nur die Stelle an der es angezeigt wird, wird gelöscht.
- Behebung: Zerstöre auch das Grid

3.3.3 Gauge Widget highlights fehlen

- Symptom: Beim Erzeugen von verschiedenen Gauges gibt es einen Fehler, dass "high"nicht definiert sei.
- Grund: Manchmal fehlen die highlights in der signals.xml. Dort fehlen dann auch die ticks.
- Behebung: Überprüfe die Werte und setze sie zur Not auf Standartwerte. Diese Standartwerte sind in der Datei einstellbar.

3.3.4 signals.xml finden

- Symptom: In den Tests wird die signals.xml Datei nicht gefunden.
- Grund: Der Pfad der signals.xml war relativ zu dashboard.js.
- Behebung: Der Pfad wurde auf relativ zum root-Verzeichnis des Projektes gewählt.

3.3.5 Erster Wert in Gauges

- Symptom: Die Gauges zeigen bis zum ersten Update eines Wertes die falschen Werte an.
- Grund: Beim erstellen des Widgets wird der aktuelle Wert nicht in das Gauge geschrieben.
- Behebung: Schreibe den aktuellen Wert bei der Initialisierung in das Gauge.

3.3.6 Chrome leert Array

- Symptom: Beim Erstellen eines Widgets kann die Konfiguration davon nicht finden. Dieses Problem tritt nur mit Chrome auf.
- Grund: Unbekannt | Das Array wird nach dem erstellen irgendwie geleert.
- Behebung: Das Array mit den Konfigurationen ist jetzt statisch. Das bedeutet jedoch auch, dass Signalkonfigurationen in allen WidgetFactories gleich sind.

3.3.7 Leeres Grid nicht serializable

- Symptom: Ein leeres Grid hat nur "]äls Rückgabewert bei der serialize-Funktion.
- Grund: Die For-Schleife berücksichte keine leeren Arrays.
- Behebung: Bei einem leeren Array "[]ßurückgeben.

Resultate / Statistiken

Hier sind die Testresultate der UI-Elemente zu sehen. Man muss beachten, dass dies wirklich nur die Unit-Tests beinhaltet. Die Coverage von Manuellen Tests ist dabei nicht mitinbegriffen. Diese sind vor allem bei den UserInterface-Tests jedoch ein maßgeblicher Teil. Ein weiteres Problem ist, dass durch das Compilen von Typescript-Code in Javascript-Code am Anfang der Datei nötige jedoch nicht von Jasmine genutzte Zeilen stehen die im Coverage auch nicht berücksichtigt werden.

Dateiname	Abdeckung
Map	60%
dashboard	50%
dataCollection	74%
dataModel	79%
grid	47%
widget	81%
widgetFactory	93%
lineChartWidget	89%
percentGaugeWidget	90%
speedGaugeWidget	90%
textWidget	100%

3.4 Rückfahrkamera

3.4.1 RGB Objekte dürfen nur mit drei bytes initialisiert werden.

- Symptom: Es war möglich RGB Objekte mit r/g/b Werten außerhalb [0,255] zu initialisieren.
- Grund: Es gab keinen min/max Limiter im Konstruktor.
- Behebung: Konstruktor angepasst, jetzt werden alle Werte auf [0,255] abgebildet (truncated).

3.4.2 RGB Objekte sind doch keine RGG Objekte.

- Symptom: Man initialisierte RGB mit r/g/b und bei Nachfrage bekam die Werte r/g/g zurück.
- Grund: Konstruktor hat this.blue auf Math.max(0, this.green) gesetzt.
- Behebung: Tippfehler korrigiert.

3.4.3 Geometry.Vec4 Initialisierung mit null war fehlerhaft

- Symptom: Man initialisierte Geometry.Vec4 mit NULL und getValues gab NULL zurück, statt (0,0,0,0).
- Grund: Reihenfolge der Code-Zeilen im Konstruktor vertauscht.
- Behebung: Reihenfolge der Zeilen korrigiert.

3.4.4 Geometry.Vec4 Zugriff auf null trotz Gegenmaßnahmen

- Symptom: Zugriff auf Attributwert null, obwohl if (!null) ... .
- Grund: Zwei if() statements nacheinander sahen folgt aus: if(null), if(...). Man sollte aber an dieser Stelle if(null), else if(...) verwenden.
- Behebung: If auf else if geändert.

3.4.5 ArrayNormalizer verkürzt Arrays nicht

- Symptom: Array der Länge 3 wird eingegeben, ArrayNormalizer soll dies auf Länge 2 normalisieren, die Ausgabe hat trotzdem die Länge 3.
- Grund: Fallunterscheidung fehlt, Fall 1: Ziellänge $<$ Eingabelänge (in diesem Fall soll der Array verlängert werden), Fall 2: Ziellänge $>$ Eingabelänge (hier soll der Array gekürzt werden).
- Behebung: Fallunterscheidung hinzugefügt.

Resultate / Statistiken

Hier sind die Testresultate zu sehen. Es ist dabei zu beachten, dass GUI-Klassen und solche Adapter-Klassen, die eine komplexe Umgebung/Initialisierung brauchen evtl. nicht (vollständig) abgedeckt sind.

Dateiname	Abdeckung	Jasmine Expects	Code LOC	Spec LOC
Format	100.00%	24	160	162
Geometry	100.00%	265	397	612
Primitive	100.00%	61	107	137

3.5 Settings

3.5.1 Settings directory append child/set parent Problem

- Symptom: The set parent Funktion arbeitete nicht wie gewünscht, beim Elternknoten wurde das Kind nicht hinzugefügt, wenn die Funktion set parent beim Kindknoten aufgerufen wurde.
- Grund: Fehlerhafte Implementierung, kein Aufruf der Funktion append child des Elternknotens.
- Behebung: Jetztige auf eine funktionierende Implementierung geändert.

3.5.2 Settings (L/N) Parameter mit Client side buffer == null

- Symptom: Settings List und Numeric (L/N) Parameter SetValue Funktion stürzt ab, wenn die Klasse SettingsLParameter mit ClientSideBuffer == null initialisiert wurde.
- Grund: Es wird nicht im Konstruktor und auch nicht in der Funktion geprüft, ob ClientSideBuffer undeclared oder null ist.
- Behebung: Codezeile, die prüft ob ClientSideBuffer null ist wurde hinzugefügt.

3.5.3 Settings List Parameter MVC View Aktualisierung

- Symptom: Wenn man setValue() in Settings List Parameter (Controller) aufruft, dann soll der Wert in View und Model entsprechend geändert werden. Im Model erfolgt die Änderung, die Anzeige ändert sich jedoch nicht.
- Grund: Code-Zeile, die bei einer Wertänderung das Modell auch ändert ist nicht vorhanden.
- Behebung: Fehlende Code-Zeile wurde hinzugefügt.

3.5.4 Settings Parameter MVC View Index

- Symptom: Wenn man beim Controller das letzte Element auswählt, dann soll es in View auch ausgewählt werden. Liest man jedoch selectedIndex zurück, bekommt man den Wert -1, statt zum Beispiel 5, wenn die Liste 5 Elemente hat.
- Grund: Index fängt in View bei 0 an.
- Behebung: Die Zeile, die den Wert von selectedIndex auf n setzt wurde geändert, so wird jetzt den Wert auf n-1 gesetzt. Hinweis: die Liste der Elemente fängt in XML Dateien weiterhin bei 1 an.

3.5.5 Settings Parameter get Full Uid wenn Parent == null

- Symptom: Funktion getFullUid() in SettingsParameter stürzt beim Aufruf ab, wenn SettingsParameter mit Parent == null initialisiert wurde, also wenn setParent() nicht früher aufgerufen wird.
- Grund: Die Funktion greift auf this.parent zu, was in diesem Fall noch nicht auf einen Wert != null gesetzt wurde, weil man früher die Funktion setParent() nicht aufgerufen hat.
- Behebung: Es wird in getFullUid() geprüft ob man den Elternknoten schon auf einen gültigen Wert gesetzt hat. Wenn nicht dann wird getRUid() zurückgegeben.

3.5.6 Settings Parameter Elternknoten ist nicht definiert

- Symptom: Funktion getParent() wird aufgerufen und man bekommt undefined statt null zurück, wenn man früher den Elternknoten nicht explizit definiert hat.
- Grund: Es gibt keinen Initialwert von this.parent, solange es nicht explizit auf einen Wert (auch null) gesetzt wird, ist es undefined.
- Behebung: This.parent wird immer mit null initialisiert.

3.5.7 Settings Numeric Parameter Min/Max Limiter

- Symptom: Wenn der Initialwert außerhalb von [Min, Max] liegt, wird es auf [Min, Max] abgebildet. Setzt man jedoch mit setValue() den Wert, werden die Mindestwert/Maximalwert Attribute nicht berücksichtigt und man kann den Attribut Wert beliebig setzen.
- Grund: Es wird in setValue() nicht geprüft, ob der Wert, den man jetzt eingibt gültig ist (ob es innerhalb [Min, Max] liegt.
- Behebung: Entsprechende Code-Zeilen addiert, wenn der Eingabewert ausserhalb [Min, Max] liegt, wird es auf Min (falls Wert $<$ Min) oder Max (falls Wert $>$ Max) gesetzt.

Resultate / Statistiken

Hier sind die Testresultate zu sehen. Es ist dabei zu beachten, dass GUI-Klassen und solche Adapter-Klassen, die eine komplexe Umgebung/Initialisierung brauchen evtl. nicht (vollständig) abgedeckt sind.

Dateiname	Abdeckung	Jasmine Expects	Code LOC	Spec LOC
CompositeStructure	94.58%	58	612	165
SettingsData	99.07%	58	188	252

Kapitel 4

Manuelle Tests

4.1 UI

4.1.1 Allgemeine Widget-Funktion

- Teste das Erzeugen von verschiedenen Widgets auf dem Dashboard. Der Test wird über den localhost also nicht über den Raspberry Pi durchgeführt, da der komplette Test nur auf dem Client läuft und der Pi dem Browser dieselben Dateien anbietet.
- Vorbedingung: Der Server muss laufen.(Optional: Internetverbingung für GoogleMaps)
- Nachbedingung (Erfolg): Alle Widgets lassen sich mit verschiedenen Signalen erzeugen. Ihre Größe ist änderbar. Sie lassen sich verschieben und auch löschen.
- Nachbedingung (Fehlschlag): Ein Widget lässt sich nicht erzeugen, verschieben, oder transformieren. Oder eine Fehlermeldung wird geworfen.
- Beschreibung/Testschritte:
  - Öffne im Browser 10.10.0.1:8000
  - Wechsle in den Edit-Modus über den passenden Button.
  - Wähle zufällig ein Signal über das obere Dropdown Menü
  - Wähle zufällig ein Widget im unteren Dropdown Menü
  - Klicke auf den ÄddButton
  - Verschiebe das Widget an eine zufällige Stelle
  - Wiederhole die letzten 4 Schritte einige male
  - Zuletzt aktiviere den Löschen-Modus über die Checkbox
  - Lösche alle erzeugten Widgets

4.1.2 Dashboard-Konfiguration

- Man testet ob die Dashboard-Konfiguration gespeichert wird. Und ob man diese wieder aufrufen kann.
- Vorbedingung: Der Server muss laufen (Optional: Internetverbindung für Google Maps)
- Nachbedingung (Erfolg): Die gewählte Konfiguration wird wieder angezeigt.
- Nachbedingung (Fehlschlag): Eine andere oder keine Konfiguration wird angezeigt
- Beschreibung/Testschritte:
  - Öffne im Browser 10.10.0.1:8000
  - Wechsle in den Edit-Modus über den passenden Button.
  - Wähle zufällig ein Signal über das obere Dropdown Menü
  - Wähle zufällig ein Widget im unteren Dropdown Menü
  - Klicke auf den ÄddButton
  - Verschiebe das Widget an eine zufällige Stelle
  - Wiederhole die letzten 4 Schritte beliebige male
  - Wechsle wieder in den Dashboard-Modus
  - Lade die Homepage neu

4.1.3 Replay Modus

- Man testet ob der Replay-Modus funktioniert. Also ob man auf Befehl eine vergangene Fahrt anzeigen lassen kann.
- Vorbedingung: Der Server muss laufen (Optional: Internetverbindung für Google Maps) und er muss bereits mindestens eine Fahrt hinter sich haben.
- Nachbedingung (Erfolg): Nach spätestens ein paar Sekunden werden Daten einer vergangenen Fahrt angezeigt.
- Nachbedingung (Fehlschlag): Es werden entweder keine neuen Daten mehr angezeigt oder weiterhin die aktuellen Live-Daten.
- Beschreibung/Testschritte:
  - Öffne im Browser 10.10.0.1:8000
  - Wechsle in den Replay-Modus über den passenden Button.
  - Wähle zufällig eine Fahrt über das Dropdown Menü
  - Klicke auf den SStartButton

4.2 Terminal und Proxy testen

4.2.1 Server und Client verbinden

- Es ist die Verbindungaufbau zwischen den Server und Endgerät zu testen, indem man durch ein Browser den Server anfragt und schaut, ob die Startkonfiguration erfolgreich angezeigt werden kann.
- Vorbedingung: Der Server ist gestartet. Ein Endgerät ist durch WLAN mit dem Server verbunden.
- Nachbedingung (Erfolg): Die Startkonfiguration der Dashboards werden im Browser angezeigt.
- Nachbedingung (Fehlschlag): Im Browser wird die Startkonfiguration nicht richtig angezeigt.
- Beschreibung/Testschritte:
  - Schritt eins: Der Benutzer startet den Server im Raspberry-PI.
  - Schritt zwei: Der Benutzer verbindet sein Endgerät durch WLAN mit dem Server.
  - Schritt drei: Der Benutzer öffnet den Browser und geht zur Adresse des Servers.

4.2.2 Multibenutzer

- Es ist die Fähigkeit, dass die Software mit mehreren Benutzern laufen kann.
- Vorbedingung: Der Server ist gestartet und kann die Daten über ein Auto einsammeln durch OBD-Bluetooth-Adapter. Mehrere Endgeräte sind durch WLAN mit dem Server verbunden. Der Fahrer fährt mit dem Auto.
- Nachbedingung (Erfolg): Die Endgeräte können die Echtzeitdaten über das Auto nach Wunsch anzeigen.
- Nachbedingung (Fehlschlag): Die Endgeräte können die Daten nicht richtig anzeigen.
- Beschreibung/Testschritte:
  - Schritt eins: Der Fahrer steckt das OBD-Bluetooth-Adapter in das Auto ein und lässt das Auto anzünden.
  - Schritt zwei: Der Fahrer startet den Server auf dem Raspberry PI.
  - Schritt drei: Der Benutzer verbindet seine Endgeräte mit dem Server.
  - Schritt vier: Der Benutzer öffnet den Browser auf verschiedenen Geräten und geht zur Adresse des Servers.
  - Schritt fünf: Der Benutzer fordert unterschiedliche Widgets auf den Endgeräten und überprüft, ob die angekommene Daten mit der Realität übereinstimmen.

4.2.3 Client Value Subscriptions

- Lässt sich der User einen bislang nicht gezeigten Wert anzeigen, wird dieser auch auf Server Seite (vom Proxy des entsprechenden Clients) subscribed.
- Wenn der Client die Verbindung trennt, soll der Proxy auf Server-Seite gelöscht werden. Davor sollen aber alle Subscriptions des Proxy gelöscht werden. Die Methode `unsubscribeAll(): void` der Klasse Bus wird in den Unit-Tests abgedeckt. **??**
- Vorbedingung: Es muss mindestens ein Client mit dem Server verbunden sein und mindestens einen Wert anzeigen.
- Nachbedingung (Erfolg): Für kein `Topic topic` ist Der Proxy im Array `Broker.get().subscribers[topic]` enthalten
- Nachbedingung (Fehlschlag): in der Hashmap `Broker.get().subscribers` taucht eine Referenz auf einen Proxy auf, der keine Kommunikation zu einem Client hat.

Kapitel 5

Testszenarien

5.1 Der Test von BluetoothOBD

1. Der Fahrer steckt den Bluetooth-OBD-Adapter in die Schnittstelle vom OBD im Auto ein. Das rote Licht auf dem Adapter soll leuchten.
2. Der Fahrer startet den Server auf dem Raspberry-PI.
3. Der Fahrer lässt das Auto anzünden.
4. Der Benutzer verbindet sein Endgerät mit dem Server durch WLAN.
5. Der Benutzer öffnet einen Browser und geht zur Adresse von 10.10.0.1: 8000.
6. Der Benutzer soll die Dashboards mit Startkonfiguration sehen.
7. Der Fahrer fährt mit dem Auto.
8. Der Benutzer soll die Echtzeitdaten des Autos sehen.
9. Der Benutzer fordert andere Informationen über das Auto an und im Browser werden verschiedene Dashboards angezeigt.
10. Die Aggregierte Funktionenen sollen mit der Realität übereinstimmen.

Kapitel 6

Benchmarks

6.1 Webbrowser - Zeichnen von SVG-Grafiken

6.1.1 Einleitung

Bei der Einparkhilfe verwenden wir SVG-Grafiken, um den Pfad des Autos zu zeichnen. Um es gut abschätzen zu können, wie schnell die verschiedene Webbrowser SVG-Grafiken zeichnen können ist es Sinnvoll eine einfache HTML-Datei zu schreiben, die sich immer wieder neu lädt. Die Frage ist, wie lange dauert es, bis alles gezeichnet wird?

6.1.2 Ablauf

- Da wir nicht genau wissen, wie gut die Darstellung intern parallelisiert ist, weisen wir Firefox einen einzigen Prozessorkern zu (64-bit, 2.5 GHz). Wir schalten alle Puffern aus, damit es immer wieder neu geladen und gezeichnet wird. Windows 7 puffert jedoch das Bild im Hintergrund, das heißt, dass wir nicht auf Festplattenzugriffe achten müssen.
- Wir öffnen eine HTML-Datei, die lediglich ein SVG Bild enthält und stellen sicher, dass es immer wieder neu geladen wird. Zum Beispiel, wenn wir die Zeile <META HTTP-EQUIV="refresh"CONTENT="1» in die HTML-Datei einfügen, dann wird sie alle Sekunden neu geladen.
- Wir öffnen mehr und mehr Tabs und beobachten, wie stark der Prozessorkern belastet wird (Abb. 6.1).
- In unserem Fall können wir 20 Tabs öffnen und die Belastung beträgt 61%. Dies bedeutet, dass es 0.61 x 1000 ms / 20 = 30.5 ms dauert das Bild in Firefox zu zeichnen.
- Hinweis: Wir haben nur bewiesen, dass Firefox maximal 31 ms braucht um das Bild zu zeichnen, es ist möglich, dass der Wert weniger ist.

Firefox mit 20 SVG Tabs

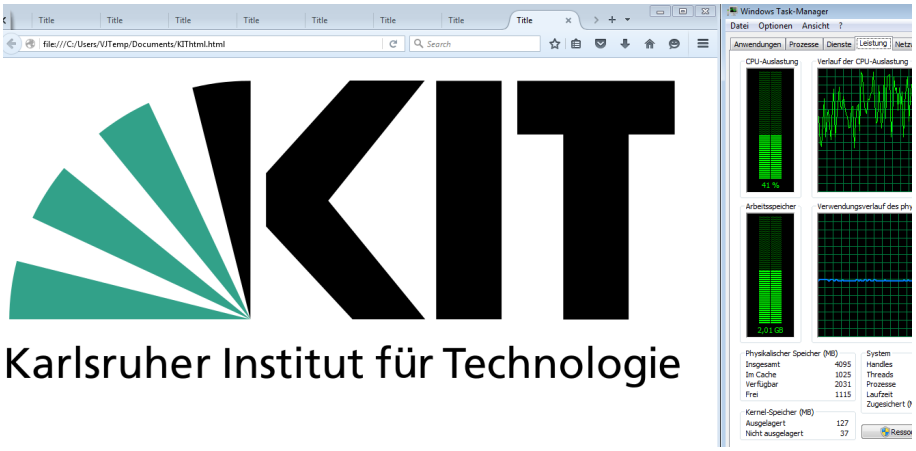

Abbildung 6.1: Durchschnittliche Auslastung des CPU-Kerns: 61%

6.2 Chrome - Darstellungs von Settings

6.2.1 Einleitung

Um zu wissen, wie schnell wir die Settings darstellen können und mit wieviel Aufwand die Darstellung verbunden ist, können wir einfach versuchen ziemlich schnell auf verschiedene GUI-Elemente zu klicken und beobachten, wie lange es dauert, bis alles neu gezeichnet wird. Selbstverständlich klicken wir (hoffentlich) langsamer, als unser Rechner die klicks verarbeitet. Um alles zumindest teilweise zu automatisieren (eine vollständige Automatisierung ist auch möglich, in diesem Fall macht aber wenig Sinn), verwenden wir das Programm Autohotkey. Autohotkey kann verschiedene Skripte öffnen, die Mausbewegungen, Klicks, Tastatureingaben... usw. beschreiben.

6.2.2 Ablauf

- Wir schreiben ein Skript und stellen es ein, dass wir alle 30 Millisekunden einmal klicken wollen.
- Wir öffnen Chrome und stellen sicher, dass Settings angezeigt wird (Abb. 6.2).
- Wir lassen das Skript laufen und beobachten, was passiert.
- Wenn wir mit dem Cursor die numerische Eingaben (HTML Input/number) verändern, deren Darstellung tief in Chrome verankert ist und wir annehmen können, dass es so schnell wie möglich (oder sinnvoll) erfolgt, stellen wir fest, dass Chrome auf Windows etwa 10 Klicks pro Sekunde verarbeitet und es nicht an der CPU-Auslastung liegt. Wir haben versucht 33-mal pro Sekunde zu klicken. Was bedeutet dies für uns? Wenn wir 10 fps erreichen ohne dabei den Prozessor erheblich zu belasten, dann sind wir erfolgreich.

- Hinweis: Eine Videoaufnahme vom Benchmark ist vorhanden.
- Wie wir auf dem Video auch sehen können, erreichen wir 10 fps und verursachen dabei 16% CPU-Auslastung. Wir haben drei Kerne, die alle mit einer Geschwindigkeit von 2.4 GHz laufen.
- Fazit: Das Ergebnis wäre bei einem Programm, das nativ ausgeführt wird und in C geschrieben ist als schlecht zu beurteilen, in unserem Fall aber nicht, wir haben die Ziele erreicht, die Geschwindigkeit, die wir erreicht haben genügt.

Settings geöffnet in Chrome

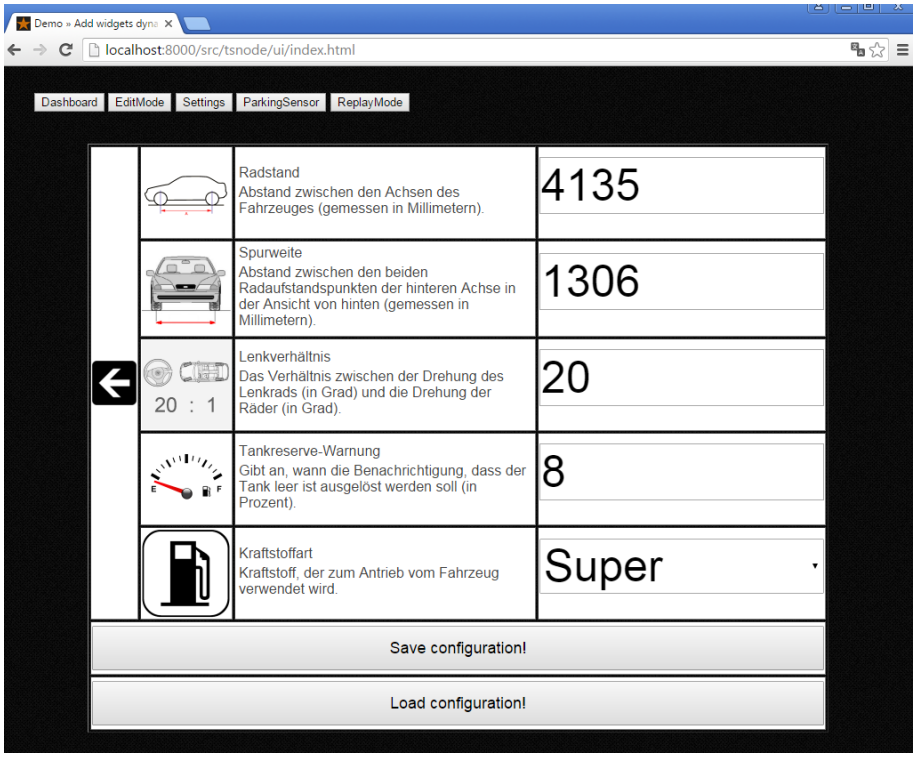

Abbildung 6.2: Durchschnittliche Auslastung eines CPU-Kerns: 50%

6.3 Data Source

Die Geschwindigkeit der Aktualisierung der Daten über Auto hängt von dem CAN-Bus im Auto, dem Bluetooth-Adapter, und der Frequenz des Abschicken der Befehle ab. Um alle Befehle eine sinnvolle Antwort zu haben, hat das Programm ein Mechanismus genutzt, dass jeder Befehl erst dann abgeschickt werden kann, wenn der Antwort des letzten Befehles ankommt. Durchschnittlich ist es 80ms zwischen zwei Befehle. Das heißt, pro Sekunde werden durchschnittlich 12.5 Befehle erforgreich nach OBD geschickt.

Abbildungsverzeichnis