UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

An Algorithm for Dependency-Preserving Smart Home Updates

Koray Uzun
Matrikelnummer:3081690
Angewandte Informatik (Bachelor)

Fachgebiet Verteilte Systeme, Abteilung Informatik
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

2. September 2021

Erstgutachter: Prof. Dr-Ing. Torben Weis
Zweitgutachter: Peter Zdankin
Zeitraum: 1. September 2042 - 1. Januar 2043

Abstract

Smart Home Systeme sind Netzwerke von elektronischen Geräten innerhalb eines Wohnraumes. Diese Systeme unterliegen, wie jede andere Technologie auch, einem ständigen Wandel. Während andere Technologien, wie zum Beispiel Smartphones meist nach einigen wenigen Jahren ausgetauscht, erwartet man von Smart Home Systemen eine lange Lebensdauer. Die meist komplexen und teuren Installationen sind nicht dazu ausgelegt alle paar Jahre ausgetauscht zu werden. Dennoch erwartet man, wie von anderen Technologien auch, dass die Systeme sich den Gegebenheiten der Zeit anpassen. Neue Funktionen oder oft auch Sicherheitslücken erfordern ein ständiges updaten der Systeme oder einzelner Geräte. Mit diesen Updates geht jedoch ein großes Risiko einher. Aufgrund der riesen Diversität verschiedener Smart Home Systeme kann vom Hersteller meist nicht garantiert werden, dass ein Update keine Schäden anrichtet. Lediglich das Update im Einzelnen wird auf mögliche Fehlerquellen überprüft, nicht jedoch das Zusammenspiel des Updates mit anderen Geräten. Dies wäre aufgrund der riesen Diversität an Smart Home Systemen für den Hersteller auch gar nicht umsetzbar. Dennoch benötig man eine Möglichkeit Updates präventiv zu überprüfen, um so größere Schäden zu meiden und die Langlebigkeit von Smart Home Systemen zu gewährleisten. Diese Abschlussarbeit beschäftigt sich mit der Implementation und Evaluation eines Algrorithmus, welcher Abhängigkeiten zwischen Geräten untersucht und basierend auf diesen Abhängigkeiten Updatekonfigurationen findet, die keine Probleme verursachen. Der Algorithmus basiert auf dem Paper "An Algorithm for Dependency-Preserving Smart Home Updates" von Peter Zdanking, Matthias Schaffeld, Marian Waltereit, Oskar Carl und Torben Weis.

[1]

[1] Wikipedia: `https://en.wikipedia.org/wiki/Abstract_(summary)`

Contents

Chapter 1

Einführung

In den letzten Jahren stieg die Anzahl der Smart Home Geräte stark. Laut Verified Market Research wird der Smart Home Markt im Jahr 2019 auf bereits 80 Milliarden Dollar geschätzt. Prognostiziert werden 207.88 Milliarden Dollar im Jahr 2027. Dies entspricht einer jährlichen Wachstumsrate von 13.52 Prozent [1]. Zudem ist es so, dass innerhalb von Smart Home Systemen die Anzahl der Geräte ebenfalls stetig steigt. Während 2016 durchschnittlich 5.8 Geräte über ein Smart Home System vernetzt waren, waren es 2018 schon bereits 8.1 Geräte [2]. Es ist zu vermuten, dass sich dieser Trend vorerst so weiterentwickeln wird und somit Smart Home Systeme immer größer werden. Wie jede andere Technologie müssen Smart Home Systeme regelmäßig geupdatet werden, um zum Beispiel Sicherheitslücken zu schließen oder neue Funktionalitäten hinzuzufügen. Dies geschieht nicht zentral für das gesamte Netzwerk, sondern meist wird jedes Gerät unabhängig von den anderen geupdatet. Mit der steigenden Anzahl an Geräten, steigt in einem Smart Home System auch logischerweise die Anzahl der Updates und somit insgesamt die Komplexität von Smart Home Systemen. Das Problem hierbei ist, dass mit steigender Komplexität zum einen automatisch eine höhere Fehleranfälligkeit bezüglich Updates einhergeht, zum anderen wird es schwieriger Fehler zu beheben, die durch Updates verursacht werden. Updates im Einzelnen besitzen bereits viele Fehlerquellen, wie zum Beispiel Sicherheitslücken oder generell fehlerhafte Implementationen. Dies sind jedoch Aspekte die bereits vom Entwickler weitestgehend vermieden werden. Viel Problematischer ist das Zusammspiel verschiedener Geräte und den dazugehörigen Updates. Ein an sich fehlerfreies Update kann bei der Installtion in einem Smart Home System, trotz "fehlerfreier" Programmierung große Probleme verursachen. Was ist wenn zum Beispiel ein Update eine Funktion eines Geräts so verändert, dass ein anderes Gerät nicht mehr darauf zugreifen kann. Durch solche kleine Änderungen können Abhängigkeiten zwischen Geräten zerstört werden, wodurch ganze Teile eines Smart Home Systems ausfallen können. Solch ein Ausfall bedeutet für den Nutzer meist ein großer finanzieller Schaden, da Updates oft nicht rückgängig gemacht werden können und somit Ersatz beschafft werden muss. Aus diesen Gründen sind präventive Maßnahmen notwendig, um die Langlebigkeit von Smart Home Systemen gewährleisten zu können. Langlebigkeit wird nämlich von den meisten Usern erwartet, wie man an der Resonanz zum Abschalten der HueBridge V1 von Philips sehen kann [3].

1.1 Ziel der Arbeit

Eine Steigerung der Lebensqualität ist das Kernziel von Smart Home Systemen. Dies erreicht man zum Beispiel durch Konfiguration von automatisierten Prozessen. Dadurch können Nutzer nämlich Zeit, Energie und in gewissen Belangen auch Geld sparen. Laut Umfragen geben 57 Prozent der Nutzer von Smart Home Systemen in Amerika an jeden Tag durchschnittlich 30 Minuten Zeit zu sparen (https://policyadvice.net/insurance/insights/smart-home-statistics/). Zusätzlich lassen sich nach Berechnungen des Fraunhofer Instituts für Bauphysik durch Automatisierungen bis zu ca. 40 Prozent Heizkosten sparen. (https://www.ibp.fraunhofer.de/content/dam/ibp/ibp-neu/de/dokumente/sonderdrucke/bauphysik-gertis/6-einsparpotenziale-intelligente-heizungsregelung.pdf). Solche Vorteile zeigen, dass sich die anfangs teure Anschaffung eines Smart Home Systemes auf lange Zeit sogar rentieren kann. Sicherlich ist dies der Idealfall, aber dennoch zeigen diese Zahlen, dass Smart Home Systeme viele Potenziale beinhalten. Es hängt jedoch von vielen Faktoren ab, ob sich die Anschaffung eines Smart Homes lohnt und objektiv ist dies eigentlich gar nicht zu beurteilen. Was man jedoch objektiv sagen kann, ist dass Nutzer aufgrund der teuren Anschaffungskosten eines Smart Homes mit Sicherheit davon ausgehen, dass ihr System viele Jahre erhalten bleibt. Zum Beispiel gibt es Smart Home Systeme, welche beim Bau eines Hauses direkt mit geplant werden, sodass man, wie zum Beispiel bei KNX alle Geräte über ein permanent installiertes Bus System verbinden kann. (Paper). Natürlich gibt es auch günstigere Varianten, aber dennoch erwartet man als Nutzer generell, dass wenn man ein funktionierendes System hat, dieses auch über viele Jahre hinweg weiter funktionieren wird. Daher is das Ziel dieser Arbeit die Implementation und Evaluation eines Algorithmus, welcher Abhängigkeiten zwischen Geräten untersucht und basierend auf diesen Abhängigkeiten Updatekonfigurationen findet, die keine Probleme verursachen. Mit diesem Algorithmus wäre es für Smart Home Nutzer möglich Updates bereits vor der Installation zu überprüfen, um so sicherzugehen, dass sie keine Ausfälle verursachen. Die Anwendung dieses Algorithmus sollte für den Nutzer möglichst einfach sein. So könnte man zum Beispiel ein Smartphone nutzen, welches als zentrales Gerät im System dient. Auf diesem zentralen Gerät wäre es dann möglich den Algorithmus laufen zu lassen und sich über das Smartphone dann für die passende Updatekonfiguration zu entscheiden. Smartphones bieten sich besonders an, da sie eine intuitive Bedienung für den Nutzer ermöglichen und im Normalfall jeder Nutzer eines Smart Homes ein Smartphone besitzt. So umgeht man das Problem, dass es, wie bereits erwähnt, etliche Smart Home Anbieter mit verschiedener Software und Geräten gibt.

Chapter 2

Related Work

In diesem Kapitel werden Smart Home Systeme genaue beleuchtet und das Thema Dependency Management aufgegriffen, wobei bereits bestehende Ansätze/Lösungen diesbezüglich genauer betrachtet. Zusätzlich wird der aktuelle Stand des Dependency Managements im Bereich Smart Home untersucht.

2.1 Smart Homes

[**Paper [12]**]

Im vorherigen Kapitel wurde bereits erwähnt, dass es viele Arten von Smart Home Systemen gibt. Im Rahmen dieser Arbeit ist es jedoch nicht notwendig die verschiedenen Smart Home Architekturen genauer zu beleuchten. Dennoch schadet es nicht sich einen groben Überblick über die aktuellen Technologien zu verschaffen. Grob kann man die verschiedenen Architekturen darin unterscheiden, ob sie eine Verknüpfung zu einer Cloud besitzen. Cloud gebundene Architekturen bieten dem Nutzer meist eine einfachere Installation und Wartung, indem zum Beispiel Updates automatisch installiert werden.Gleichzeitig bringen sie aber auch potenzielle Gefahren mit sich. Die Abhängigkeit von einer Cloud macht Smart Home Systeme sehr verwundbar. Wird zum Beispiel ein externer Service, wie zum Beispiel die Cloud ausgeschaltet, kann es zu starken Ausfällen im System führen. [5](Towards Longevity of Smart Homes). Außerdem sind Geräte, die mit einer Cloud verbunden sind anfälliger gegenüber Angriffe von Außenstehen, die sich Zugriff auf das System beschaffen wollen. Nicht Cloud-gebundene Architekturen haben diese Nachteile nicht. Sie sind jedoch meist schwieriger zu implementieren, da sie, wie im Falle von OpenHab, viel manuelle Konfiguration benötigen. Sie bieten dadurch jedoch auch mehr Sicherheit, da keine starke Abhängigkeit zu anderen Dienstleistern nötig ist Natürlich könnte man Smart Home Systeme noch weiter spezifizieren, indem man einzelne Komponenten genauer beleuchet und verschiedene Smart Home Anbieter miteinander vergleicht, jedoch ist dies, wie bereits erwähnt, im Rahmen dieser Arbeit

nicht notwendig. Der zu entwickelnde Algorithmus soll nämlich später theoretisch Platformunabhängig anwendbar sein. Dafür müssen logischerweise einige Aspekte vorausgesetzt werden. Zum Beispiel eine genaue Definition eines Smart Homes. Unter dem Begriff Smart Home stellen sich nämlich vermutlich verschiedene Personen verschiedene Dinge vor. Um dies zu vermeiden wird auf folgende Definition zurückgegriffen:

[4] *A smart home has a set of devices that are connected through a platform. Each device has a certain software version and a set of available updates. Each software version has a set of available predefined services. Devices can use services of other devices which creates dependencies. An update configuration is one of the finite states of the nondeterministic finite automaton (NFA) that can be constructed by using the configurations as states and connecting them using individual updates as transitions.* Wie man sieht ist diese Definition sehr generell gehalten. Auf Smart Home Architektur spezifische Aspekte wird nicht eingegangen, was im Rahmen dieser Arbeit völlig ausreichend ist.

Meist findet man selbst innerhalb eines Smart Home Systems bereits Geräte, die nicht vom selben Hersteller sind. Dies erschwert die Entwicklungs eines Algorithmus, welcher allgemein an allen Arten von Systemen anwendbar sein soll. Für die Implementation eines Algorithmus ist es daher notwendig gewisse Dinge vorauszusetzen und zu definieren. Zunächst die Definition eines Smart Homes:

2.2 Dependency Managment

Unter Dependency Management versteht man im Allgemeinen das Strukturieren von Abhängigkeiten verschiedener Systeme/Programme. Es ist oft so, dass Programme nur in Abhängigkeit von anderen Programmen starten können. Dies stellt auch kein Problem dar, solange die Abhängigkeiten in ihrer Anzahl überschaubar bleiben. Realität ist jedoch, dass Programme meist von mehreren Programmen abhängig sind und diese Prorgramme wiederum abhängig von anderen Programmen sind. Bei zu vielen Abhängigkeiten verliert man als Entwickler und auch als Nutzer schnell den Überblick. Dieses Wirrwarr an Abhängigkeiten nennt man auch "Dependency Hell". Gemeint ist damit, dass es ab einem bestimmten Punkt unmöglich wird den Überblick über alle Abhängigkeiten zu behalten. Je mehr Abhängigkeiten in einem System vorhanden sind, desto komplexer wird es, wodurch jede Änderung oder Erweiterung einen riesen Aufwand mit sich zieht.

In der Dependency Hell gibt es eine Zusammenfassung der am häufigsten auftretenden Probleme:

- Long Chain of Dependencies: Ein System A ist abhängig von einem System B. Das System B wiederum ist abhängig von einem System C. Möchte man nun System A nutzen benötigt man zusätzlich System B und damit auch System C. Solche lange Ketten können Konflikte verursachen. (Siehe Conflicting Dependencies)

- Conflicting Dependencies:
- Conflicting Dependencies:
- Conflicting Dependencies:
- Conflicting Dependencies:

Es existieren bereits viele Lösungen, um gegen diese Probleme vorzugehen. Dazu gehört:

- Version numbering:
- Version numbering:
- Version numbering:
- Version numbering:
- Version numbering:

Bei Smart Home Systemen ist es jedoch so, dass in diesem Bereich Dependency Management völliges Neuland ist.

Dabei werden Smart Home Systeme als Luxusgüter angesehen und aufgrund der hohen Preise erwartet der Verbraucher auch lange Lebensspannen der Geräte. Dies kann aber ohne vernünftiges Dependency Management nicht gewährleistet werden. Natürlich gibt es auch andere Aspekte wie z.B. discontinued services oder security issues, die die Lebenserwartung von Smart Home Systemen verringern. Dennoch sollte dabei Dependency Management nicht untergehen. Dies sieht man am Beispiel des Logitech Harmony Hub's. Logitech hatte eine neue Firmware Version für das Hub veröffentlicht, welche angeblich nur Sicherheitslücken schließen und Bugs fixen sollte. Nachdem Update kam es jedoch vermehrt zu Ausfällen zwischen dem Hub und third-party Geräten. Den Nutzern waren die Hände gebunden, sie konnten nur darauf hoffen, dass Logitech die Probleme wieder behebt. Dies hat Logitech in diesem Fall auch getan, dennoch zeigt dieses kleine Beispiel, wie leichtsinnig Updates von Nutzern installiert werden. Eine andere Möglichkeit besitzt man auch als Nutzer eines Smart Homes nicht wirklich, da es, wie eingangs erwähnt, zur Zeit keine Möglichkeiten gibt Updates vor der Installation zu überprüfen.

Was genau ist ein Smart Home System? Towards Longevity in Smart home Systems kopieren?

2.3 Code Listings

If there is some interesting code you would like to show in order to ease the understanding of the text, you can just include it using the `lstlisting` environment. Have a look at the source of this page to see how this is included:

```
x := from(42);
```

You could also put the code into an external file and include it in this document using the `lstinputlisting` command:

```
1 var x = new Node
2 if luck() {
3     var y = new Node
4     x.next = y
5 }
6 return
```

Be careful not to include large files as it hampers readability. If there is a short excerpt from a large file you would like to show, you can also extract an explicit range of lines from it without the need to modify the source file. This next listing only shows the conditional from the previous code:

```
2 if luck() {
3     var y = new Node
4     x.next = y
5 }
```

To use more advanced syntax highlighting have a look at the available options of the *listings* package or use the *minted* package[1], which has more extensive language support and additional themes. Both can be configured in the main file.

2.4 Math

In case you need to include some math, the *amsmath* package[2] is already included in this document.

To properly display some short formula like $e^{i\pi} = -1$, you can use the `\( \)` inline command. For larger formulas, the `math` environment is more appropriate. If you need to reference the formula multiple times, e.g. in case it is used in theorems, you should use the `equation` environment:

[1] `https://texdoc.net/texmf-dist/doc/latex/minted/minted.pdf`
[2] `https://texdoc.net/texmf-dist/doc/latex/amsmath/amsmath.pdf`

$$\vec{\nabla} \times \vec{B} = \mu_0 \vec{j} + \mu_0 \varepsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{2.1}$$

To reference it as 2.1 using the `\ref{}` command, remember to use a `\label{}`.

2.5 Miscellaneous

You can use the `\todo{}` command to put obvious reminders on the side of the document.

TODO: like this!

Chapter 3

Design und Implementation

Dieses Kapitel beschreibt die Implementation des Algorithmus und erklärt Entscheidungen, die im Laufe der Implementation getroffen wurden.

3.1 Design

Basierend auf den Erkenntnissen aus Chapter 1 und 2 kann nun der Algorithmus implementiert werden, welcher im Paper ... beschrieben wird. Die Implementation zielt darauf unabhängig vom darunterliegenden Smart Home System anwendbar zu sein. Das Ziel ist es theoretisch ein zukunftsfähiges Tool zu entwickeln, welches allen Smart Home Nutzern ermöglicht sich vor Abhhängigkeits zerstörenden Updates zu schützen.Oft werden Geräte automatisch über eine aktive Internetverbindung geupdatet oder auch durch andere Geräte wie zum Beispiel einem Hub. Idealerweise sollten Nutzer jedoch vor jedem Update gefragt werden, ob sie dieses installieren wollen. Es wäre also von Vorteil, wenn es für den Nutzer eine Übersicht über alle möglichen Updates geben würde, in der er sich dann für passende Updates entscheiden kann. Für solch eine Übersicht bietet sich ein Smartphone an

Nun benötigt man noch eine Möglichkeit die Informationen über die Services und Updates eines Geräts leicht abfragen zu können. Eine solche Abfrage müsste in der Realität für jedes Gerät individuell angepasst werden. Dies ist jedoch nicht umsetzbar, weswegen es nötig ist einen einheitlichen Weg zu finden. Geräte besitzen üblicherweise Metadaten, welche Informationen über Firmware Version, Größe usw. enthalten. Diesen Metadaten könnte man zusätzlich Informationen über Services und Updates hinzufügen, sodass man einen schnellen Überblick über alle Geräte innerhalb eines Netzwerks erhalten kann.

Der entwickelte Algorithmus wird nicht an realen Smart Home Systemen getestet, weswegen sich diese Frage zunächst erübrigt. Es bieten sich zwei Möglichkeiten an den Algorithmus zu testen. Eine Möglichkeit ist eine statische Liste, welche man man manuell mit Geräten und deren Updates füllt, oder ein Generator, welcher abhängig von Parametern ein "künstliches" Smart Home System kreiert. Parameter wären dann zum Beispiel die

Anzahl der Geräte oder die Anzahl der Updates, die ein einzelnes Gerät besitzt. Der Generator bietet sich mehr an, da der Algorithmus nicht nur getestet, sondern auch evaluiert werden soll. Die Evaluation kann dann an unterschiedlich komplexen Smart Home Systemen durchgeführt werden, indem man die Parameter dementsprechend verändert. Parameter wären zum Beispiel die Anzahl der Geräte, die Anzahl der Updates die ein Gerät besitzt oder auch die Anzahl der Services.

Daher sollte der Algorithmus möglichst unabhängig vom darunterliegenden Smart Home System sein. Zu diesem Zwecke war es zunächst notwendig einen Generator zu entwickeln, auf welchem dann der später implementierte Algorithmus getestet werden kann. Der Generator funktioniert so, dass er Geräte erzeugt, welche in Form von Objekten gespeichert werden. Diese Geräte enthalten Informationen über alle ihre Dienstleistungen und alle möglichen Updates. Bei realen Geräten müssten diese Informationen in den Metadaten gespeichert werden, um sie leicht abfragen zu können. Dies ist eine Voraussetzung für den Algorithmus, ohne die es nicht möglich wäre die Implementation Systemunabhängig zu gestalten. Da Metadaten bei den meisten Geräten bereits existieren, sollte es für die Entwickler dieser Geräte kein großes Problem darstellen die Metadaten um diese Informationen zu ergänzen. Nachdem die Geräte erstellt wurden, kreiert der Generator künstliche Abhängigkeiten zwischen diesen Geräten, um so ein Netzwerk zu simulieren. Eine Abhängigkeit sieht so aus, dass ein Gerät Zugriff auf die Dienstleistungen eines anderen Geräts benötigt. Solch ein simples Netzwerk reicht im Rahmen dieser Arbeit völlig aus, da für den Algorithmus die Struktur des Smart Homes später völlig egal sein wird. Lediglich die Abhängigkeiten zwischen den einzelnen Geräten sind interessant. Außerdem ist es ist möglich in Abhängigkeit von verschiedenen Parametern die Komplexität der erzeugten Smart Home Netzwerke zu variieren. So können die Anzahl der Geräte, die Anzahl der möglichen Updates pro Gerät, die Anzahl der Dienstleistungen pro Gerät oder auch die Anzahl der Abhängigkeiten zwischen den Geräten variiert werden. Dies ermöglich den später implementierten an Algorithmus an verschiedenen Systemen zu evaluieren, um so ein gutes Fazit über Effiziens und Lauftzeit treffen zu können.

Der Algorithmus selbst funktioniert folgendermaßen. Im ersten Schritt

The evaluation usually consists of three main steps: first it defines the goals intended to be achieved by the software in detail; next is a description of the methodology used to measure the satisfaction of the software in relation to these goals; finally, the measurements are depicted and assessed.

Chapter 4

Conclusion

The conclusion quickly summarizes the results of the paper in relation to the previously defined goals and hypotheses. It usually also includes some information on next steps and further research that might be required or possible on the presented subject.

4.1 Bibliograhpy

Bibliography

[1] https://www.verifiedmarketresearch.com/product/global-smart-home-market-size-and-forecast-to-2025/

[2] https://www.homeandsmart.de/

[3] Peter Zdankin, Matthias Schaffeld, Marian Waltereit, Oskar Carl, Torben Weis, "An Algorithm for Dependency-Preserving Smart" Home Updates"

Versicherung an Eides Statt

Ich versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach §156 und §163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

Duisburg, 2. September 2021
(Ort, Datum)

(Vorname Nachname)