Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Eine Anwendung für die Verwaltung von Experimenten im Testbed des Humboldt Wireless Lab

Studienarbeit

eingereicht von: Andreas Furtner
geboren am: 31.1.1979
geboren in: Berlin

Gutachter/innen: Prof. Dr. Jens-Peter Redlich

Betreut durch: Dipl.Inf Robert Sombrutzki
Mike Schulze, M.Eng.

eingereicht am: .....

Abstract

Der Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin betreibt unter dem Namen *Humboldt Wireless Lab* ein Testbed zum Testen neuer Routingverfahren. Zum Durchführen von Simulationen und Tests existiert bisher eine Sammlung von Unix-Werkzeugen und Shell-Skripten. Es bestand Bedarf an einer Anwendung zum Erstellen, Durchführen und Verwalten von Experimenten sowohl im Simulator als auch auf der realen Hardware. Dazu wurde eine webbasierte MVC-Anwendung auf Basis des Java/Scala-Frameworks Play2 entwickelt.

Inhaltsverzeichnis

1 Einleitung

Der Lehrstuhl für Systemarchitektur [22] der Humboldt-Universität zu Berlin betreibt unter dem Namen *Humboldt Wireless Lab* [23] (ehemals BRN[1]), ein „Testbed“ zum Testen zukunftsweisender Netzwerktechnologien. Das Testbed ist ein Netzwerk aus verschiedenen WLAN[2]-Routern und anderen Netzwerkkomponenten, in der folgenden Arbeit kurz Knoten genannt. Es werden zum Beispiel Maschennetzwerke[3], neue Routingprotokolle oder auch angepasste Firmware von Netzwerkkomponenten getestet.

Da die Komponenten des Testbed als Ressource begrenzt sind, wird in Ergänzung zum Testbed mit einem Simulator gearbeitet. Dabei handelt es sich um eine Software, mit der die gleichen Tests wie auf der eigentlichen Hardware durchgeführt werden können. Ein Simulator ist außerdem für die wissenschaftliche Forschung in diesem Bereich zwingend notwendig, um überhaupt eine Reproduzierbarkeit von Experimenten zu ermöglichen. Da sich die Umgebungsbedingungen in der Realität ständig ändern und nicht vollständig unter menschlicher Kontrolle liegen, lässt sich ein Experiment im Testbed niemals exakt wiederholen.

Zum Durchführen von Simulationen und Tests im Testbed existiert bisher eine Sammlung von Unix-Programmen und Shell-Skripten. Mit diesen werden Testbed und Simulator konfiguriert, die Tests durchgeführt und Testergebnisse gesammelt. Die Testergebnisse liegen anschließend in Form von Dateien im Dateisystem vor.

Die Konfiguration der Tools und Skripte erfolgt hauptsächlich durch Textdateien mit jeweils eigener, komplexer Syntax. Diese zu erstellen und anzupassen ist repetitiv und dadurch fehleranfällig und erfordert umfassende Einarbeitung. Syntax- oder Tippfehler, z.B. bei Parameternamen werden nicht vorab erkannt und realisieren sich erst bei der Ausführung zu Laufzeitfehlern, die dann im Testbed auch zu Hardwaredefekten führen können.

[1] *Berlin Roof Net*[26]
[2] Wireless Local Area Network, deutsch: „drahtloses lokales Netzwerk“
[3] geläufiger auf Englisch: mesh networks

Um Tests oder Verbesserungen an einzelnen Komponenten zu validieren, sollten die Testergebnisse unter gleichen Voraussetzungen reproduzierbar sein. Zudem muss die Konfiguration schnell änderbar sein, da viele unterschiedliche Experimente in der selben Umgebung stattfinden sollen.

Eine wichtige Herausforderung zur besseren Nutzung des Testbed, ist eine gute zeitliche Planbarkeit der einzelnen Testszenarien. Hier liegt auf der Hand, dass langlaufende Tests besser in der Nacht abgearbeitet werden, da hier wenig Interaktion von testbegleitenden Personen zu erwarten ist. Hingegen können tagsüber kurzlaufende Tests möglicherweise in Minuten von den Testern konfiguriert, durchgeführt und ausgewertet werden.

Den erwähnten Problemstellungen soll im Zuge dieser Arbeit mit einer zentralen Managementsoftware begegnet werden, die den testbegleitenden Personen viele Arbeitsschritte abnimmt und den Benutzer bei der Erstellung, Planung und Ausführung der Testfälle unterstützt.

Im Folgenden wird zunächst auf die Grundlagen und Rahmenbedingungen des Testbed und Simulators eingegangen und der bisherige Testablauf beschrieben. Bei dieser Betrachtung werden die aktuell bestehenden Probleme und Defizite im Umgang mit dem Testbed aufgedeckt. Das darauffolgende Kapitel beschreibt die aus dieser Analyse hervorgegangenen Anforderungen, bevor in Kapitel 4 auf die Architektur und Umsetzung der erstellten Softwarelösung im Detail eingegangen wird. Im letzten Abschnitt werden die bei der Umsetzung gewonnenen Erkenntnisse zusammengefasst und ein Ausblick auf mögliche Erweiterungen, die nicht Teil dieser Arbeit sind, gegeben.

2 Grundlagen

In den letzten zwanzig Jahren hat die Verbreitung von Mobilfunk- und Netzwerktechnologien stark zugenommen. Exemplarisch lässt sich das am Beispiel Deutschland zeigen. Laut statistischem Bundesamt hat sich der Anteil der Haushalte mit Mobiltelefonen von 11,2% im Jahr 1998 auf 95,5% in 2017 drastisch erhöht [38]. Die Endgeräte entwickelten sich im Laufe der Zeit von reinen Telefonen zu vollwertigen Netzwerkknoten, welche über WLAN, LTE[1] oder zukünftig 5G[2] ständig Teil eines Netzwerks sind [3]. Damit geht die allgemeine Verfügbarkeit von kabellosen Netzwerken einher.

2.1 Netzwerk-Testbed am Lehrstuhl für Systemarchitektur

Vor diesem Hintergrund engagiert sich der Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin bei der Erforschung von drahtlosen Kommunikationsprotokollen und Technologien. Dabei wird sich auf WLAN (IEEE-802.11) konzentriert, um robuste und selbstorganisierte Netzwerke für Anwendungsfälle wie beispielsweise Sensornetzwerke für Erdbebenfrühwarnsysteme zu ermöglichen. [21]

Für diese Forschung betreibt der Lehrstuhl das HWL[4]-Testbed. Unter einem Testbed wird in dieser Arbeit eine Umgebung für wissenschaftliche Experimente auf realer Hardware verstanden. Ein Testbed ist also ein Netzwerk aus verschiedenen Komponenten, wie zum Beispiel (WLAN-) Routern, Switches oder Sensorknoten. Mit diesem lassen sich Netzwerkprotokolle, zum Beispiel von Maschennetzwerken (mesh networks), im In- und Outdoor-Betrieb unter realen oder realitätsnahen Bedingungen evaluieren [25]. Diese Tests, die am Lehrstuhl auch *Measurements* genannt werden, erfolgen im Rahmen von Forschungsprojekten oder studentischen Abschlussarbeiten.

[1] Long Term Evolution, ein Mobilfunkstandard der vierten Generation
[2] Fifth Generation, geplanter Nachfolgestandard von LTE
[3] Stichwörter: always online, ubiquitious networking
[4] *Humboldt Wireless Lab*

So wurden unter anderem Performanceuntersuchungen mit 802.11n-Geräten, zum Beispiel zu Signalstärken und Durchsatz, in verschiedenen Experimenten durchgeführt [43] oder DHT-basierte Routingverfahren in drahtlosen Maschennetzwerken analysiert [2]. Viele weitere Beispiele für durch das Testbed unterstützte Forschungsarbeiten finden sich unter den Publikationen des Lehrstuhls.[5]

Der Lehrstuhl ist eine von mehreren Institutionen, die solche Testbeds betreiben. Ähnliche Installationen werden oder wurden etwa von der TU Berlin (BOWL[6]), der Universität Magedeburg (MIOT[7] Lab[8]) oder am MIT[9] betrieben.[10]

Das HWL-Testbed wird mit dem Framework[11]„Click Modular Router“ konfiguriert und betrieben. Der „Click Modular Router“ wurde von Eddie Kohler und anderen am MIT entwickelt. Mit diesem Framework wird ein Router aus einzelnen netzwerkpaketverarbeitenden Modulen kombiniert. Diese Module werden innerhalb des Frameworks „Elemente“ genannt. Einzelne Elemente implementieren dabei einfache Routerfunktionen wie Paketklassifizierung, Warteschlangen (Queueing), Scheduling und andere. Elemente werden dabei üblicherweise als Klassen in C++ implementiert und es gibt auch aktive Elemente, welche (neue) Netzwerkpakete erzeugen können. [18]

Zusätzlich definiert eine Click-Konfiguration, auch Click-Skript genannt, einen gerichteten Graphen mit den Elementen als Knoten sowie Inputs und Outputs. Netzwerkpakete können dabei entlang der Kanten dieses Graphen „fließen“. Unter zuhilfename des Click Compilers entsteht dann eine modulare Router Firmware, welche entweder als Linux-Kernelmodul oder auch im user space auf den einzelnen Routern laufen kann.[12] Das Click-Framework ist quelloffen und auf der OpenSource-Plattform github verfügbar.[13]

Über verschiedene Server im Testbed werden diverse globale Dienste angeboten, wie etwa der Zugriff auf Dateien via NFS und das Vorbereiten und Ausführen von Experimenten. Die Architektur des HWL-Testbed ist vergleichbar der in Abbildung

[5] Publikationen unter [24], vgl z.B. [19, 37, 20, 2]
[6] *Berlin Open Wireless Lab*
[7] *Magedeburg Internet of Things*
[8] ehemals Münster IOT Lab an der Westfälischen Wilhelms-Universität Münster und zuvor DES-Testbed an der FU Berlin
[9] Massachusetts Institute of Technology
[10] Vgl. dazu [32, 15, 16]
[11] *Framework* könnte man in etwa mit „Programmiergerüst“ ins Deutsche übersetzen, allerdings wird in Fachliteratur m.E. nur der Englische Begriff benutzt und so auch hier
[12] Eine Einführung zum „Click modular router“ findet sich unter [4]
[13] für die Quellen zum Click-Framework siehe [17]

2.1 dargestellten Architektur des Testbed, wie es am MIOT-Lab der Universität Magedeburg betrieben wird.

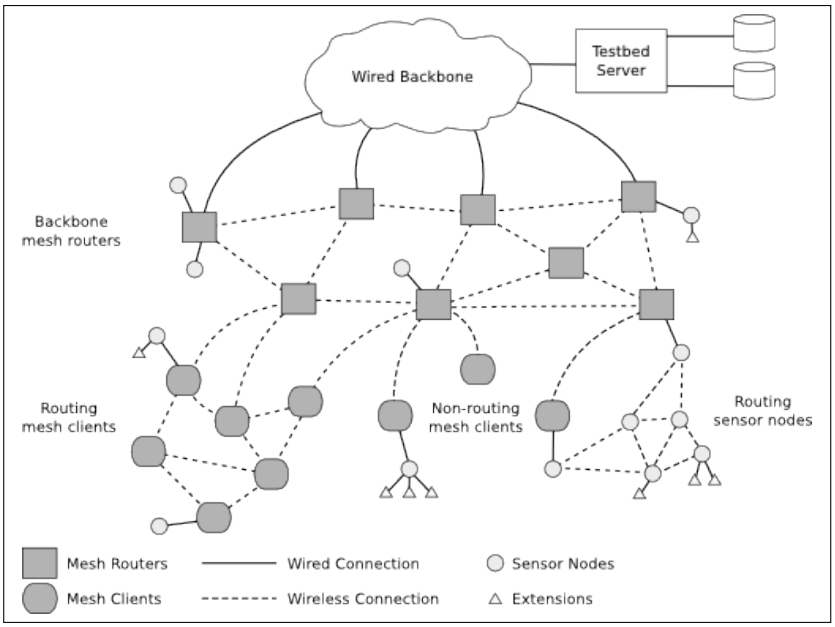

Abbildung 2.1: Architektur eines Testbed [14]

Die Hardware eines solchen Testbed muss überwacht, betreut, gewartet und gegebenenfalls erneuert werden. Demzufolge erfordert der Betrieb personelle und finanzielle Ressourcen. Die Kosten dafür können durch unsachgemäße oder fehlerhafte Testdurchführung steigen, wenn dadurch Geräte beschädigt oder zerstört werden. Bei Experimenten die nicht hardwarenah ausgeführt werden müssen, kann auf die Verwendung realer Hardware grundsätzlich verzichtet werden. Dazu zählt auch die Überprüfung von Testkonfigurationen, um sicherzustellen, dass die Hardware nicht durch Fehlkonfigurationen beschädigt wird. Aus diesem Grund betreibt der Lehrstuhl einen Simulator, welcher im nächsten Abschnitt näher beschrieben wird.

2.2 Simulator

Zur Simulation von Tests wird der Netzwerksimulator *ns-2* verwendet, der ursprünglich aus einem DARPA-Projekt entstanden ist. Die Simulation arbeitet zeitdiskret

[14] Quelle: Communication and Networked Systems an der Universität Magdeburg `http://www.comsys.ovgu.de/MIOT_Lab/Architecture.html`

und legt Ereignisse („events“) in einer Zeitreihe („timeline“) ab. Zum geplanten Zeitpunkt können diese dann von Ereignisbehandlungsroutinen („event handler“) verarbeitet werden. Die Event Handler können selbst wieder weitere Ereignisse erzeugen und in die Timeline einreihen. [40]

Im Simulator besteht ergänzend zu den schon aufgeführten Vorteilen auch die Möglichkeit, über verschiedene Radiomodelle, wie die vordefinierten „freespace“ oder „shadowing“, Hardware und Umgebungsbedingungen zu testen, welche im Testbed nicht vorhanden sind. Durch solche Radiomodelle werden unterschiedliche Eigenschaften der Hardware aber auch der Umgebung, z.B. die Dämpfungseigenschaften des Übertragungsmediums, simuliert. Es können somit neben Netzwerkprotokollen, also der oberen Schichten im ISO/OSI-Modell[15], ebenso neue MAC-Protokolle und Radiomodelle implementiert und simuliert werden. Diese sind eher in den unteren Schichten (1 und 2) des ISO/OSI-Modells einzuordnen.

Für die Simulation lassen sich ferner Umgebungsparameter wie die Knotenplatzierung (z.B node placement: Grid, random) oder Feldgröße, also die geografische Ausdehnung (z.B. field size: 100m x 100m) konfigurieren. Diese Parameter sind im Testbed durch die reale Umgebung und Knotenauswahl bestimmt und lassen sich nicht frei wählen.

Am Lehrstuhl wird ns-2 zusammen mit einem „Click-Adapter“ (*nsclick*) verwendet. Dadurch wird ermöglicht, dass mit Hilfe des Click-Frameworks umgesetzte Testbed-Experimente mit geringer Modifikation auch im Simulator ausgeführt werden können.

Es werden außerdem noch andere Simulatoren, beispielsweise *JiST*[16], verwendet. Da diese grundsätzlich mit ähnlichen Konzepten arbeiten, werden sie in dieser Arbeit nicht weiter betrachtet.

2.3 Derzeitiger Testablauf

Im Folgenden wird der bisherige Ablauf beim Durchführen von Experimenten beschrieben. Bei einem derartigen Test soll beispielsweise ein neues Routingprotokoll evaluiert werden.

Vor der Durchführung des Tests muss entschieden werden, auf welchen Netzwerkknoten das Experiment durchgeführt werden kann und soll. Für diese Entscheidung ist derzeit ein umfassendes Domänenwissen notwendig, da das Testbed eine Vielzahl

[15] Open Systems Interconnection Model der International Organization for Standardization, vlg. [41]

[16] *Java-in-Simulation-Time*, Vgl. [1, 5]

von unterschiedlichen Knoten umfasst. Diese sind an unterschiedlichen Standorten aufgestellt und unterscheiden sich im Hinblick auf ihre Ausstattung und technischen Merkmale. Ein Teil dieser Informationen könnte zumindest teilweise aus der Hardwaredatenbank des Lehrstuhls, im weiteren Verlauf „HWL-DB“ genannt, bezogen werden. Es gibt für diese MySQL-Datenbank jedoch nur einen beschränkten Zugriff über eine generische Datenbankverwaltungssoftware[17] für MySQL. Die Auswahl der Knoten für Experimente ist dementsprechend sehr kompliziert.

Nachdem die Entscheidung zur Knotenauswahl getroffen wurde, muss der Tester den Test mit diversen Konfigurationsdateien, im Folgenden auch Testinputs genannt, vorbereiten. Diese Dateien müssen für jeden Test neu erstellt werden. Da sie aber eine jeweils eigene, komplexe Syntax haben, werden sie jedoch in der Regel von bereits durchgeführten Experimenten kopiert und an das neue Experiment angepasst. Dieser Prozess ist repetitiv, durch die Komplexität fehleranfällig und kann schwer validiert werden. Durch die nicht zwingend vom Tester dokumentierten manuellen Anpassungen wird außerdem die Reproduzierbarkeit von einzelnen Experimenten erschwert. Einige Dateien müssen außerdem für jeden am Experiment beteiligten Knoten erstellt werden. Alle erforderlichen Dateien werden in einem speziellen Arbeitsverzeichnis gesammelt.

Die Tabelle 2.1 enthält eine Übersicht über erforderliche und optionale Testinputs.

Der eigentliche Test wird im Anschluss an die Anpassungen der Testinputs über die beiden Shell-Skripte „runSim“ und „runMeasurement“ durch den Nutzer entweder im Simulator oder direkt auf dem Testbed gestartet. Dabei erzeugt „runSim“ aus allen Konfigurationsdateien eine Konfiguration für den ns-2-Simulator und startet diesen, während „runMeasurement“ aus diesen Inputs mit den Click-Tools die Routersoftware zusammenbaut und mittels der Server im Testbed verteilt und startet. Dabei ist anzumerken, dass es aktuell keine Möglichkeit gibt, den Aufruf des Simulators vor dem Ausführen auf der Hardware zu erzwingen. Die Testausführung startet in der Regel sofort.

Durch die Art und Weise, wie ein Test gestartet wird und durch eingeschränkte Zugriffsrechte auf der Testumgebung, wird eine verzögerte oder zu einem späteren Zeitpunkt geplante Ausführung erschwert.[18] Da hier eine Überwachung des Ablaufs nötig ist, kann ein neuer Job effektiv erst nach Ablauf des vorherigen gestartet werden. Simulator und Testbed erzeugen während des Testlaufs Messdaten. Diese

[17] Und zwar phpMyAdmin, vgl. [34]

[18] Eine Verzögerung ist zwar mit Kommandozeilen- bzw. Administrationsprogrammen wie *sleep*, *nohup* und *cron* denkbar, erfordert im Falle von *cron* aber zusätzliche Benutzerrechte auf Systemebene. Verzögerte Ausführung mit *sleep* in einer mittels *nohup* offengehaltenen ssh-session ist wiederum nicht sehr robust und kann zu fehlgeschlagenen Experimenten führen.

Ergebnisse liegen nach Abschluss des Tests in Form von Dateien in zuvor konfigurierten Verzeichnissen vor. Im weiteren Verlauf werden diese Ergebnisdateien zusammen mit den Testinputs als Testartefakte bezeichnet.

Die Probleme und Defizite beim bisherigen Umgang mit Testbed und Simulator lassen sich in folgender Liste zusammenfassen:

- Reproduzierbarkeit von Experimenten:
  - Disziplin beim Archivieren von Testartefakten erforderlich
  - fehlerfreie Wiederholung wegen komplexer Konfiguration schwierig
- Expertenwissen bei Hardwareauswahl und Konfiguration nötig
- komplexe, schlecht dokumentierte, fehleranfällige Syntax der Konfigurationsdateien
- unüberwachte, verzögerte Testausführung nicht möglich
- Simulation vor Ausführung im Testbed nicht durch den Prozess erzwungen
- Hardwarefehler durch fehlerhafte Konfiguration damit nicht vermeidbar
- Ressourcen des Testbed schlecht ausgelastet, da hohe Einstiegshürden

Art der Datei (Datei/Endung)	oblig.	pro Knoten	Zweck/Beschreibung
Globale Testkonfiguration (.des)	ja	nein	globale Testbeschreibung, Knoten und Simulatorkonfiguration: z.B. (Arbeits-) verzeichnisse, Referenzen auf andere Konfigurationsdateien wie die .mes-Datei
Knotenkonfiguration (.mes)	ja	nein	tabellarische Konfiguration der beteiligten Knoten
Gerätekonfiguration (monitor-Datei)	ja	möglich	Gerätekonfiguration (z.B. Kanal, Antennen)
click-Konfiguration (click-Skript)	ja	möglich	Konfiguration Testprogramm/ Routersoftware als Graph von Click-Elementen mit In-/Outputs
control-Datei	nein	möglich	zeitliche Anpassung von Parametern während des Tests, z.B. Ändern des Kanals, Sendeleistung, etc.
application	nein	möglich	user space Applikation die zusätzlich auf Knoten laufen soll: muss nicht in „click“ implementiert sein
host application	nein	nein	globale (Steuer-) Anwendung, läuft auf Testcontroller (einem der Server im Testbed)

Tabelle 2.1: Testinputs: Übersicht für einen Test nötiger Konfigurationsdateien

3 Anforderungen

Aus dem beschriebenen Testablauf und Gesprächen mit Nutzern und Administratoren des Testbed sowie den in den Kapiteln 1 Einleitung und 2 Grundlagen geschilderten Problemen und Defiziten ergaben sich die im Folgenden aufgeführten Anforderungen an die zu erstellende Lösung.

3.1 Unterstützung bei der Auswahl der Knoten

Der Tester soll bei der Auswahl der Netzwerkknoten für seinen Test unterstützt werden. Dies ist notwendig, da für diese Auswahl aktuell ein großes Domänenwissen, bezogen auf die Standorte und Eigenschaften der Knoten, erforderlich ist. Es liegen aber in einer Hardware-Datenbank schon für viele Knoten Informationen beispielsweise auch Geo-Koordinaten vor, die den Tester bei der Auswahl unterstützen könnten. Diese Informationen sollen von der Applikation verwendet und präsentiert werden. Außerdem ist eine Suchfunktion wünschenswert.

3.2 Vereinfachung der Konfiguration

Aktuell erfolgt die Konfiguration der Tests und der am Test beteiligten Hardware durch Kopieren und manuelles Anpassen diverser Textdateien. Diese existieren in vielfältigen Verzeichnissen der einzelnen Mitarbeiterinnen und Studierenden. Die Anwendung soll diese Konfigurationen bündeln und deren Bereitstellung für den Test weitgehend automatisieren, um die manuellen Schritte zu minimieren.

3.3 Reproduzierbarkeit von Experimenten

Ein weitere wichtige Anforderung für die Forschung mit dem Testbed umfasst die Reproduzierbarkeit und Nachvollziehbarkeit der durchgeführten Experimente. Bisher

ist dafür die Disziplin des Testers nötig, da dieser die Testartefakte (Testinputs und -ergebnisse) archivieren und anderen Forschern zugänglich machen muss. Durch eine zentrale Verwaltung und Archivierung der Testartefakte soll deshalb die wissenschaftliche Arbeit unterstützt werden.

3.4 Terminplanung

Bisher ist nur eine sofortige Ausführung eines Tests möglich. Das heißt es muss immer eine testbegleitende Person, zumindest für den Start des Tests, anwesend sein. Testausführungen sollen zeitlich geplant und automatisiert werden können. Zudem muss sichergestellt werden, dass Tests während der Ausführung nicht um Ressourcen konkurrieren.

3.5 Simulator

Vor einem Test auf der Hardware muss ein erfolgreicher Lauf auf dem Simulator sichergestellt sein. Dies soll verhindern, dass Fehler in der Software zu Ausfällen der Hardware im Testbed führen.

3.6 Anbindung der HWL-Datenbank

Es existiert eine Datenbank („HWL-DB“) [27] mit Informationen zu den einzelnen Harwarekomponenten, die im Testbed verwendet werden. Dazu zählen beispielsweise Namen, MAC-Adressen, IP-Adressen oder auch Standorte (geo location). Diese Informationen zu verwenden, so dass Hardware-Informationen in der Anwendung nicht redundant gepflegt werden. Der Zugriff auf diese Datenbank darf nur lesend erfolgen.

3.7 Zugriffsbeschränkung, Benutzerverwaltung

Der Zugriff auf die bisherigen Shell-Skripte und Unix-Tools wird bisher über Nutzeraccounts beschränkt. Eine vergleichbare Zugriffsbeschränkung muss durch die

zu erstellende grafische Benutzeroberfläche gewährleistet sein. Die Verwaltung der Zugänge soll durch Mitarbeiter des Lehrstuhls erfolgen können.

4 Architektur, Funktionen und Implementierung der Anwendung

Zur Umsetzung der im 3. Kapitel gestellten Anforderungen wurde eine Webanwendung unter der Projektbezeichnung „testbedr“ auf Basis des Java- und Scala-Frameworks Play2[28] entwickelt.[1]

In diesem Kapitel soll diese Anwendung vorgestellt und dabei getroffene Architekturentscheidungen erläutert, Funktionen der Anwendung und das zugehörige Domänenmodell herausgestellt, sowie auf einzelne Aspekte der Implementierung eingegangen werden. Die Startseite der erstellten Anwendung ist in Abbildung 4.1 dargestellt.

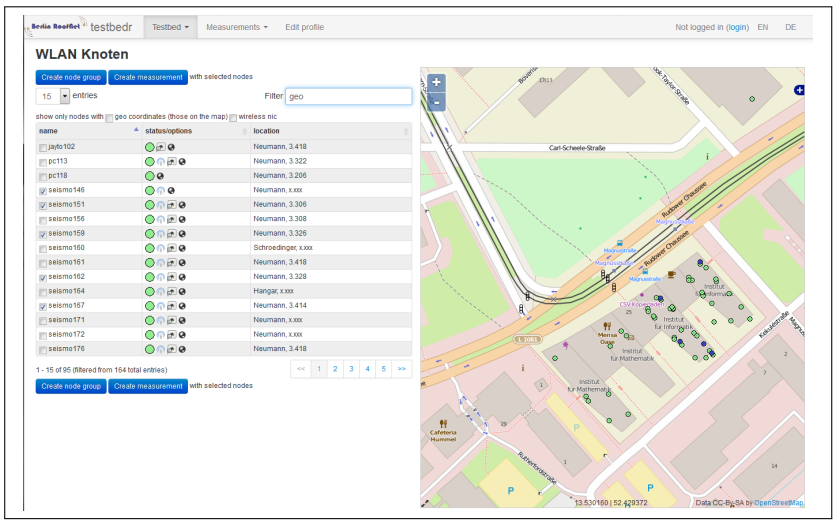

Abbildung 4.1: Screenshot der Webanwendung

[1] Der Quelltext dieser Anwendung wird als Git-Repository über die gitlab-Installation des Instituts für Informatik zur Verfügung gestellt. [11]

4.1 Architekturentscheidungen

4.1.1 Basisarchitektur, Implementierungssprache und Frameworks

Da die bisherigen Tools eine UNIX-Umgebung voraussetzen und von der Anwendung weiterhin verwendet werden sollen, muss diese in der gleichen Umgebung lauffähig sein. Obwohl Testbed und Simulator an zentraler Stelle vorhanden sind, soll die Testplanung dezentral durchgeführt werden können. Die Anwendung soll den Tester durch eine grafische Benutzeroberfläche (GUI[2]) beim gesamten Testprozess unterstützen. Diese Punkte führten zur Entscheidung, ein serverseitige Anwendung mit Weboberfläche zu entwickeln. So ist die Anwendung für die Tester möglichst plattformunabhängig zugänglich, also ein Zugriff mit den unterschiedlichsten Endgeräten[3] möglich.

Für die Erstellung von Benutzeroberflächen hat sich die MVC[4]-Architektur als de-facto-Standard etabliert. MVC ist eine Sammlung von Entwurfsmustern bei der Anwendungslogik, interne Datenstrukturen und deren Darstellung voneinander getrennt umgesetzt werden. Dabei umfasst das *Model* die anwendungsspezifischen Informationen, die dem Nutzer in einer (oder auch mehreren) *View* dargestellt werden. Der *Controller* reagiert auf Nutzereingaben und steuert die Veränderung der Daten im *Model*. Dies veranschaulicht Abbildung 4.2.

Die Grundprinzipien von MVC wurden 1978-1979 von Trygve Reenskaug am Xerox PARC[5] in Smalltalk-76 [35] entwickelt. In den 1980er Jahren setzten sich Prinzipien und Begriff allmählich durch.[42] Spätestens nach der Bezugnahme auf MVC durch Erich Gamma et al. in der Einleitung ihres Standardwerks „Design Patterns“ [12] waren Begriff und Prinzipien in der Welt der Softwarearchitektur dann allgemein bekannt und anerkannt.

Als Implementierungssprache wurde zunächst Java ausgewählt, da diese Sprache an den Hochschulen gelehrt wird und damit eine breite Nutzerbasis, auch unter den Studierenden, hat. Dadurch sollte eine zukünftige Wartbarkeit und Weiterentwicklung der Anwendung am Lehrstuhl erleichtert werden.

[2] Graphical User Interface
[3] sofern ein Webbrowser vorhanden ist
[4] Model-view-controller
[5] Palo Alto Research Center

Zur Unterstützung bei der Erstellung von webbasierten MVC-Anwendungen existieren im Java-Umfeld eine Vielzahl von Frameworks, zum Beispiel JSF[6] oder Spring WebMVC. Die meisten dieser Frameworks setzen auf Java Servlets auf. Diese Technologie stammt ursprünglich aus der Java Enterprise Edition und setzt einen Java-Webserver[7] wie z.B. „Tomcat“ oder einen Java Enterprise Application Server (z.B. „Websphere“ oder „JBoss“) als Ablaufumgebung voraus. Der Autor hat sich stattdessen für die Verwendung von Play2 entschieden, da dieses Framework, im Vergleich zu den genannten, leichtgewichtiger ist und somit geringere Anforderungen an die Laufzeitumgebung und Hardware stellt. Es wird nur eine JVM[8] vorausgesetzt. Das Play2-Framework selbst ist zwar in Scala geschrieben, wird aber zu Java-Bytecode übersetzt und läuft damit auf der JVM, wodurch die Möglichkeit entstand, sowohl Java als auch Scala innerhalb der Anwendung zu verwenden.

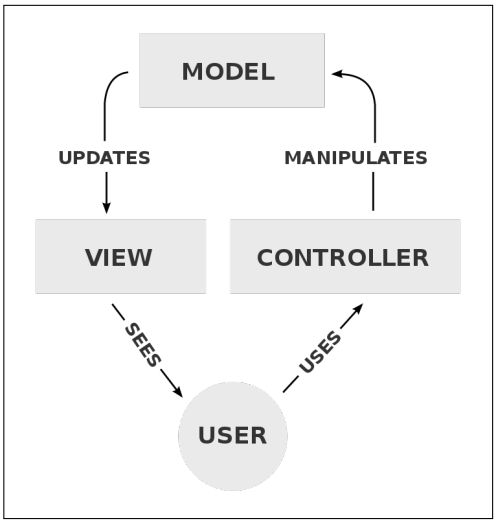

Abbildung 4.2: Interaktion von Model, View und Controller[9]

4.1.2 Anbindung mehrerer Datenbanken mittels JPA

Bezogen auf die Datenhaltung soll die Anforderung 3.6, auf die bestehenden Informationen in der HWL-DB zuzugreifen, erfüllt werden. Bei dieser handelt es sich um eine relationale MySQL-Datenbank (MySQL-DB), auf welche nur lesend zugegriffen werden darf.

[6] *Java Server Faces*
[7] eher geläufig ist in diesem Kontext die Bezeichnung Java *web container*
[8] *Java Virtual Machine*
[9] Quelle: wikipedia, Autor: Regis Frey, public domain

Im Java-Umfeld hat sich für den objektorientierten Umgang mit relationalen Daten das ORM[10]-Framework JPA[11] etabliert, welches im Laufe der vergangenen Jahre auch funktional sehr erweitert wurde. Durch die hohe Verbreitung und den Status als Industriestandard kann von einer Zukunftssicherheit ausgegangen werden. Von den Entwicklern des Play2-Frameworks war für einfache Java-Anwendungen eBean [7] als Persistenztechnologie vorgesehen. Da es sich bei *testbedr* größtenteils um eine Java-Anwendung handelt und die Community und damit die Unterstüzung durch diese für eBean wesentlich kleiner als die für JPA ist, fiel die Wahl auf den Einsatz der letzteren Technologie. Da hier keine reine Scala-Anwendung entwickelt werden sollte, wurde der ebenfalls mögliche Einsatz von Slick[12] ausgeschlossen.

Für die Speicherung der anwendungseigenen Daten, wie Benutzerinformationen, Testkonfigurationen und Testergebnisse u.a., benötigt die Anwendung eine eigene Datenhaltung. Zur Vereinheitlichung des Datenzugriffs wird für diesen die gleiche Technologie wie für die HWL-DB benutzt.

4.1.3 Domänenmodell und Persistenz

Bei Erstellung einer Anwendung, die ein ORM-Framework für die Datenhaltung verwendet, muss eine Entscheidung getroffen werden, auf welche Art und Weise dieses in die Anwendung integriert wird. Zum einen existiert der Ansatz, die Speicherung von Anwendungsdaten in eine eigene Persistenzschicht mit dedizierten Services auszulagern. Zum anderen gibt es aus dem *domain-driven-design* [8] den Ansatz der rich models. Bei letzterem werden Domänenobjekte nicht bloß als Datencontainer ohne Verhalten betrachtet, sondern können auch Anwendungslogik und Querschnittsfunktionalität wie Persistenz enthalten. Für die Umsetzung fiel die Wahl auf die Verwendung von rich models, da das Domänenmodell hier recht überschaubar[13] ist. Die Alternative eines eigenen Service- und Persistenzlayers, welcher bei umfangreicheren Domänen die Übersichtlickeit verbessert, verkompliziert bei einem kleinen Domänenmodell die Architektur und hebt damit die Einstiegshürde für zukünftige Wartung und Weiterentwicklung an. Zusammengefasst wurden in der Anwendung die wesentlichen Persistenzfunktionen wie Speichern oder Löschen direkt in der Basisklasse *BaseModel*) aller Domänenmodellklassen implementiert.

[10] Object relational mapping
[11] *Java Persistence API*
[12] Vgl. [29]
[13] in Bezug auf die Anzahl der modellierten Objekte, hier in Form von Java-Klassen

Eine ausführlichere Betrachtung der Vor- und Nachteile der beschriebenen beiden Ansätze ist nicht Bestandteil dieser Arbeit.[14]

4.1.4 Darstellung von Standorten

Zu den einzelnen Netzwerkknoten existieren schon Standortinformationen in Form von Geo-Koordinaten in der HWL-DB. Da eine Webanwendung entwickelt wurde, war es naheliegend diese Daten in einer Kartenansicht darzustellen. Die Integration von Karten in Webanwendungen wird von diversen Anbietern, wie Google, Microsoft, Apple und OSM[15] durch Bereitstellung von Softwarebibliotheken[16] unterstützt. Diese sind proprietär und somit nicht untereinander austauschbar. Als Alternative und Ergänzung zu den genannten Anbietern gibt es mit dem *OpenLayers*-Projekt [33] eine Bibiliothek, welche mit einer generischen API[17] die Verwendung von Kartenmaterial verschiedenster Anbieter ermöglicht. Durch die zusätzliche Möglichkeit offline-Karten einzubinden und die freie Lizenz[18], fiel die Wahl auf diese.

4.1.5 Speicherung von Testartefakten in der Datenbank

Bei der Konfiguration, der Durchführung und Auswertung der Tests fallen zu archivierende[19] Dateien an. Diese wurden weiter oben unter Testartefakte zusammengefasst und umfassen zum Beispiel Click-Skripte und die Testergebnisse. Letztere werden von der Anwendung in einem ZIP-Archiv zusammengefasst. Um alle anwendungsrelevanten Daten an einem zentralen Punkt vorzuhalten, wurde entschieden, diese Dateien als BLOB[20] in der Anwendungsdatenbank zu speichern. Das hat den Vorteil, dass externe Dateisystemablagen nicht mit Datenbankinhalten synchronisiert werden müssen und die Reproduzierbarkeit von Experimenten und Testdaten damit verbessert wird.

[14] Vgl dazu bspw. [9] [10] [36] [3] [31]
[15] *Open Street Map*
[16] libraries
[17] Application programming interface
[18] BSD-Lizenz, BSD: Berkeley Software Distribution
[19] siehe auch Anforderung 3.3 zur Reproduzierbarkeit
[20] *Binary Large Object*

4.1.6 weitere Softwarebibliotheken (3rd party libraries)

Zur Vereinheitlichung des Oberflächendesigns wird die Komponentenbibliothek Bootstrap [14] verwendet. Für die Verbesserung der Tabellendarstellung, insbesondere bezogen auf Durchsuchbarkeit und Filterbarkeit, kommen außerdem die Javascript-Bibliotheken jQuery [39] und DataTables [6] zum Einsatz.

4.2 Domänenmodell

In Abbildung 4.3 ist das Domänenmodell[21] – der *model*-Teil von MVC – als Klassendiagramm mit Vererbungshierarchie dargestellt. Wie in Abschnitt 4.1.3 schon erwähnt, wurde der Zugriff auf die Datenbanken nicht in einer eigenen Schicht umgesetzt. Stattdessen werden Persistenzfunktionen wie Speichern oder Löschen einzelner Objekte bereits durch die abstrakte Basisklasse *BaseModel* generisch implementiert. Diese Klasse verkörpert also eine Art „persistentes“ *model*. Alle Klassen des Domänenmodells leiten sich von diesem *BaseModel* ab, wodurch diese Querschnittsfunktionen bei allen Domänenobjekten zur Verfügung stehen und nicht jeweils neu ausprogrammiert werden müssen. In Java ist dieses Vorgehen wegen der Einfachvererbung im Gegensatz zu etwa C++ nicht immer möglich, da verwendete Bibliotheken oft Restriktionen bezüglich der Vererbungshierarchie auferlegen. Bei den an dieser Stelle relevanten Frameworks Play2 und JPA gibt es keine solche Einschränkungen, somit konnte generisches Verhalten auf diese Art und Weise modelliert und implementiert werden.

Die konkreten Klassen wie *Node*, *Measurement* oder *User* repräsentieren im Sinne des *Domain-Driven-Design* [8] jeweils konkrete Objekte aus dem modellierten Fachbereich.

Dateien werden, wie in Abbildung 4.4 dargestellt, durch die Klassen *FileStore* und *FileBlob* und deren gemeinsame abstrakte Basisklasse *BaseFile* modelliert. Dabei stellt *FileStore* nur die Verwaltungsinformation einer Datei dar. Im Gegensatz dazu enthält *FileBlob* auch den Inhalt der entsprechenden Datei – *FileStore* ist eine reduzierte Sicht auf dasselbe *FileBlob*-Objekt. Sie werden dazu auf denselben Datensatz der Tabelle `filestores` abgebildet. Nur beim Speichern und für den Download von Artefakten, wenn also direkt auf den Dateiinhalt zugegriffen werden muss, wird *FileBlob* verwendet, an anderen Stellen reicht eine Referenz mittels *FileStore*. Das Übertragen von BLOBs stellt oft einen Flaschenhals beim Datenbankzugriff dar.

[21] oft eher Englisch: domain model

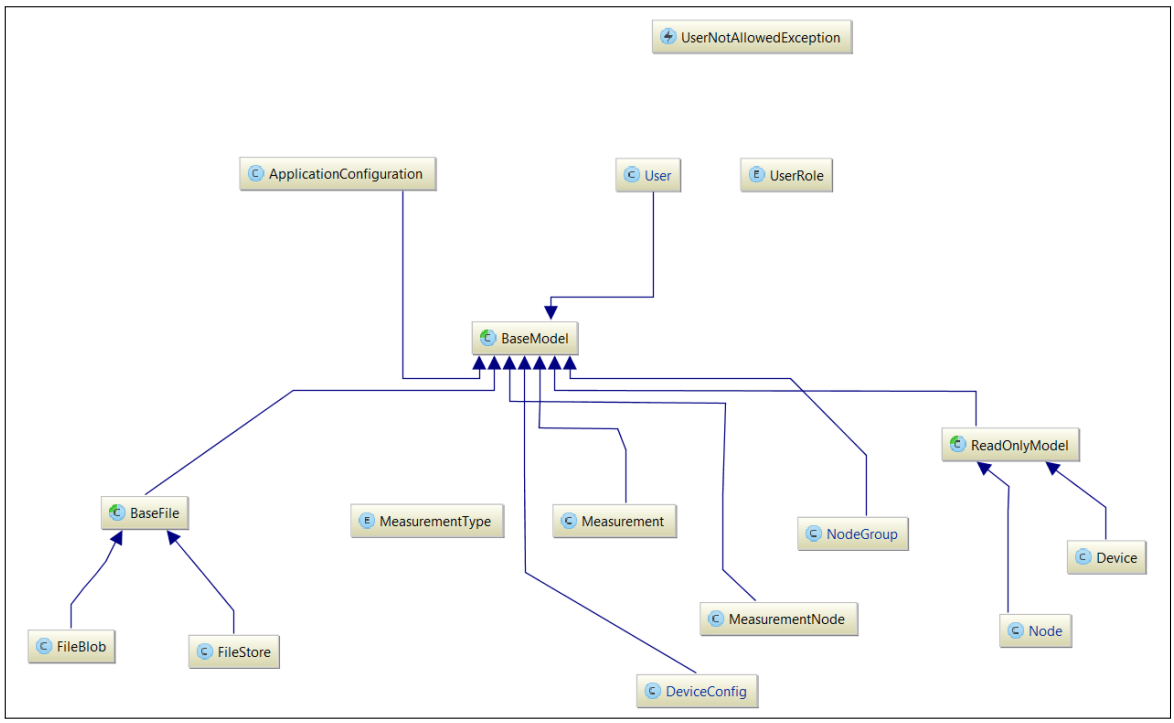

Abbildung 4.3: Domänenmodell

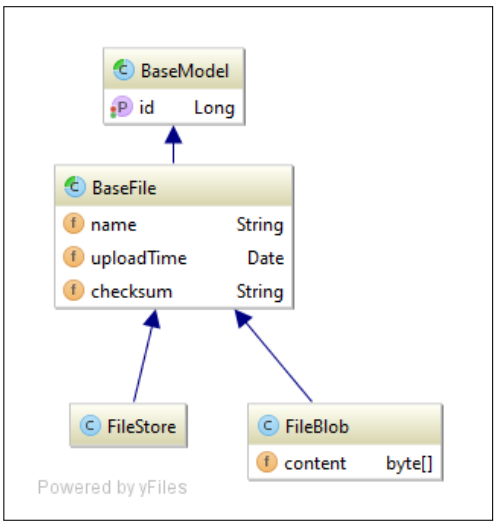

Abbildung 4.4: Domänenmodell für Dateien und Inhalte

Durch diese Art der Modellierung werden Performanzprobleme verhindert – das ORM-Framework muss nicht den ganzen Dateiinhalt aus der Datenbank laden, wenn auch eine Referenz ausreicht.

Weitere Aspekte des Domänenmodells werden in den folgenden Abschnitten zu einzelnen Funktionen der Anwendung vorgestellt oder behandelt.

4.3 Funktionen der Anwendung

4.3.1 Anbindung der Hardware-Datenbank

Die Informationen aus der HWL-DB werden von der Anwendung genutzt, um den Nutzer bei der Auswahl von Netzwerkknoten für Experimente im Testbed effektiv zu unterstützen. Die Darstellung von Netzwerkknoten erfolgt dazu einerseits tabellarisch mit schnell erfassbaren Symbolen (Icons) für wichtige vorhandene Eigenschaften wie WLAN-Fähigkeit oder vorhandene Standortinformationen. Nach diesen und anderen Eigenschaften lässt sich filtern, so dass eine schnelle Auswahl für ein Experiment geeigneter Knoten leichter möglich ist. Die bekannten Standorte einzelner Knoten werden gleichzeitig auf einer zugehörigen interaktiven Karte dargestellt. Auch in der Karte lassen sich direkt einzelne Knoten für nachfolgende Aktionen auswählen. Karte und Tabelle verhalten sich bei der Auswahl synchron zueinander, das heißt Auswählen in der Karte selektiert die Knoten auch in der Tabelle und umgekehrt. Dieser Anwendungsbereich ist in Abbildung 4.5 dargestellt.

Wenn eine Menge an Knoten ausgewählt wurde, kann der Nutzer folgende weiterführende Aktionen mit diesen ausführen:

- „Create measurement“: direktes Erstellen eines Experimentes
- „Create node group“: Merken der Knoten für spätere wiederholte Verwendung in Experimenten

Darauf wird im Abschnitt 4.3.2 weiter eingegangen.

Auf die Daten aus den vorgegebenen Tabellen der HWL-DB wird mit eigenständigen Model-Klassen objektorientiert zugegriffen. [22] Sie werden also als Teil des Domänenmodells betrachtet. Im Ausschnitt des Domänenmodells in Abbildung 4.6 sind die Klassen *Node* und *Device* und ihre Beziehungen dargestellt. Die beiden Klassen

[22] Objekte werden durch das objektrelationale Mapping auf relationale Tabellen „gemappt“, also zugeordnet bzw. abgebildet

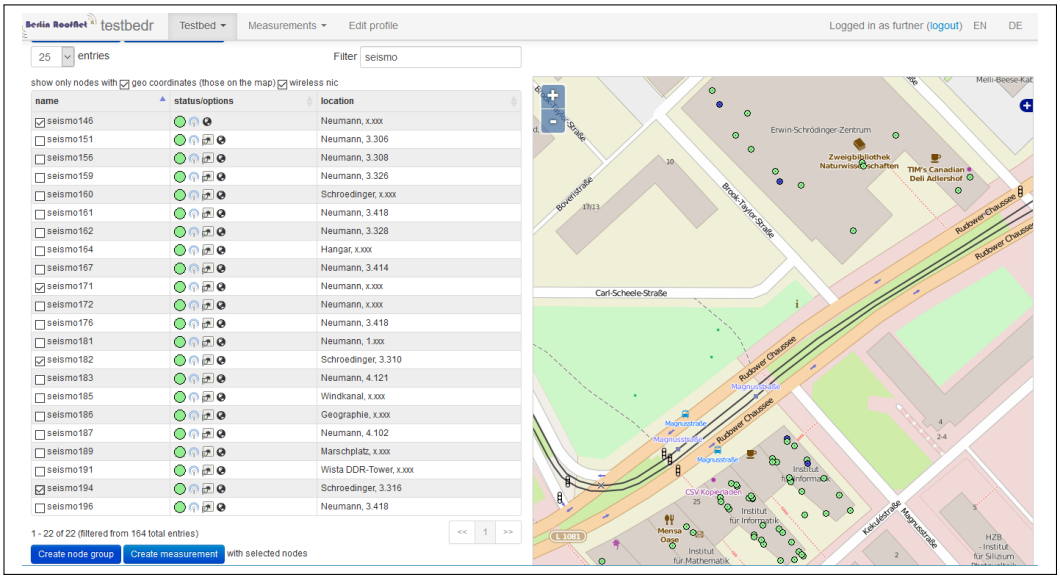

Abbildung 4.5: Testvorbereitung: Auswahl der Knoten

modellieren Netzwerkknoten (*Node*) und deren Netzwerkgeräte (*Device*), im Sinne von Netzwerkkarten oder -schnittstellen. Ein Knoten kann dabei eine oder mehrere solcher Schnittstellen besitzen. *Node* ist der HWL-Datenbanktabelle `testbed_nodes` und *Device* der DB-Tabelle `testbed_devices` zugeordnet.

Wie in Abbildung 4.6 ebenfalls zu sehen ist, wird dabei die Nur-Lese-Eigenschaft für diese Entitäten direkt durch die abstrakte Basisklasse *ReadOnlyModel* modelliert. Die Persistenzfunktionen `save()` und `delete()` von *BaseModel* sind dort explizit überschrieben, so dass diese beim Aufruf eine *UnsupportedOperationException* werfen. Zusätzlich wurden Controller-Methoden, welche mit diesen Klassen interagieren, mit der Play2-Annotation `@Transactional(readOnly=true)` versehen. Diese bewirkt, dass eine zugehörige Datenbankverbindung nur lesend geöffnet wird[23]. So wird durch die Architektur weitestgehend sichergestellt, dass kein unbeabsichtigtes Speichern geänderter Daten in diese Datenbank möglich ist.[24]

4.3.2 Verwaltung von Experimenten

Die Kernfunktionen der Anwendung beziehen sich auf die Verwaltung von Experimenten, ermöglichen also die Unterstützung beim Erstellen, Konfigurieren und

[23] wenn dies von Datenbank und DB-Treiber unterstützt wird.
[24] Vgl. dazu Anforderung 3.6

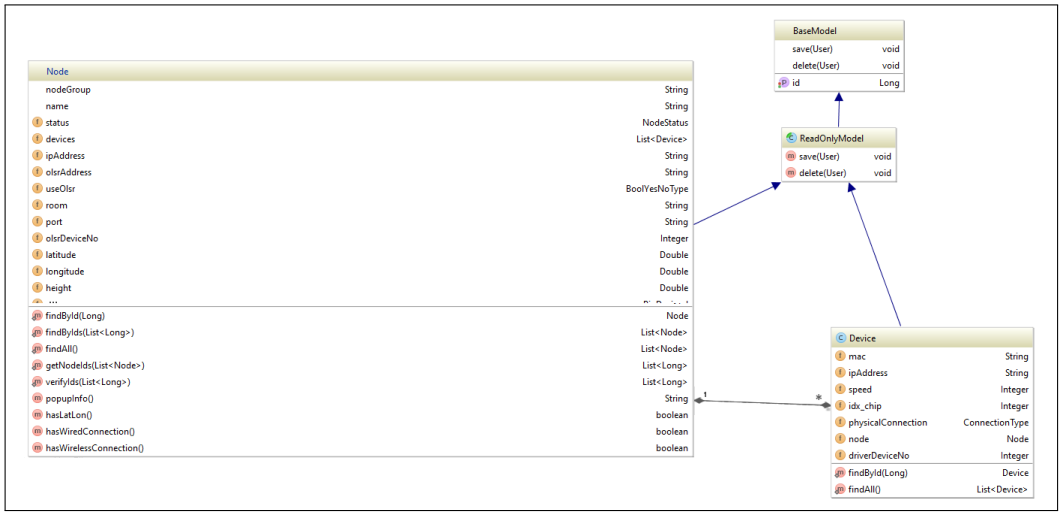

Abbildung 4.6: Domänenmodell (Ausschnitt Anbindung HWL)

zeitlichem Planen von Tests im Simulator und Testbed.

Dabei war ein Leitfaden, dem Anwender so viel wie möglich Arbeit durch Standardkonfigurationen abzunehmen, aber zugleich flexibles Überschreiben von Vorgaben zu ermöglichen. Daher ist eine Voraussetzung für das Erstellen eines Experimentes das vorherige Anlegen mindestens einer wiederverwendbaren Gerätekonfiguration (*DeviceConfig*). Gerätekonfigurationen fassen mehrere Parameter, wie Kanal oder Sende-/Empfangsleistung von Netzwerkgeräten unter einem Namen zusammen und können wie in Abbildung 4.7 eingestellt werden. Diese Vorbereitung soll nicht jedes Mal vor einem Experiment durch den Tester erfolgen. Stattdessen ist vorgesehen, dass ein Experte eine Reihe von gängigen Gerätekonfigurationen bereitstellt, die dann bei der Testerstellung nur noch ausgewählt werden.

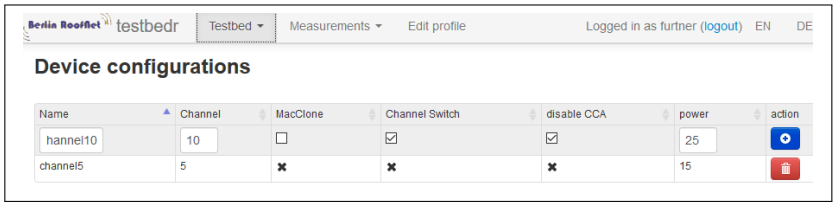

Abbildung 4.7: Testvorbereitung: device configs

Nun gibt es mehrere Wege zum Erstellen eines Experimentes. Eine Möglichkeit umfasst die Auswahl einer Menge von Knoten, die als Grundlage zur Erstellung eines neuen „leeren“ Experimentes dient. Dies wurde in Abschnitt 4.3.1 schon beschrieben

und in Abbildung 4.5 dargestellt. Eine Knotenauswahl lässt sich auch als Gruppe *NodeGroup* speichern, wodurch eine schnelle und unkomplizierte Konfiguration wiederholter, aber angepasster Experimente auf denselben Knoten vereinfacht wird.

Durch Auswahl von „Create measurement“ wird ein Dialog zum Erstellen neuer Experimente angezeigt, der in Abbildung 4.8 zu sehen ist.[25] Der Nutzer kann dort zunächst Verwaltungsdaten, wie Name und Kurzbeschreibung des Experiments eintragen. Diese dienen dazu, (abgeschlossene) Experimente später wiederzufinden. Anzupassende Konfigurationsparameter eines Experimentes werden, soweit für den Wertebereich möglich, durch diskrete GUI-Elemente, wie Drop-Down-Menüs, Radiobuttons oder Checkboxen ausgewählt. Möglich wird das durch die Vorgabe von Gerätekonfigurationen, die zuvor erstellt wurden. Diese kapseln fast alle frei einstellbaren Parameter, die eine nicht begrenzte Wertemenge [26] haben. Bei der Testerstellung kann der Nutzer nun nur noch solche „bewährten“ Gerätekonfigurationen auswählen.

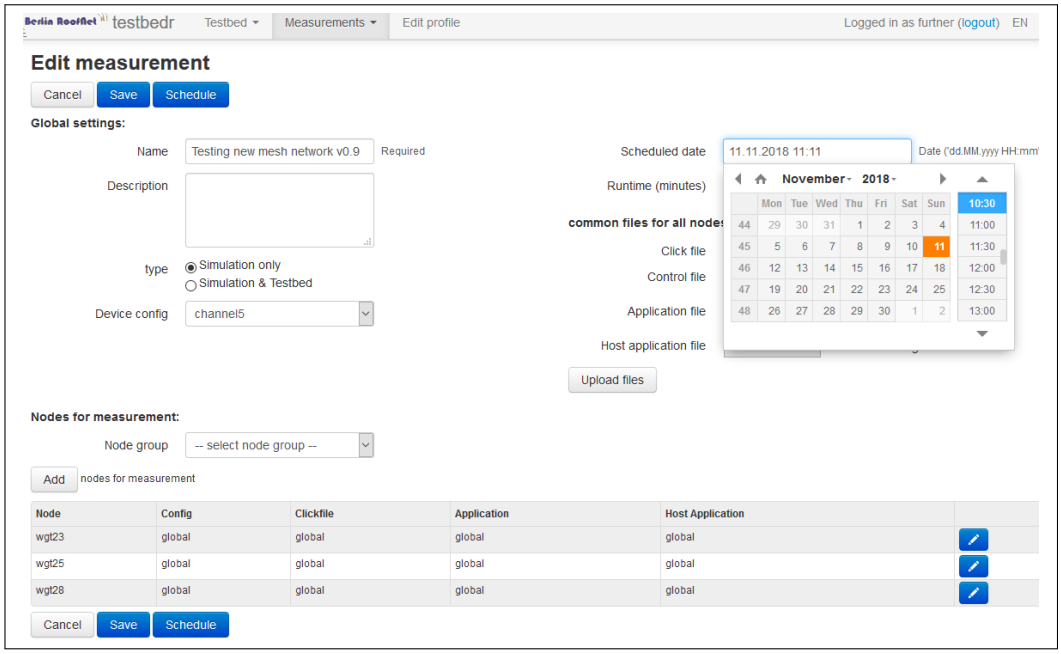

Abbildung 4.8: Hinzufügen und Anpassen einen Testlaufs (measurement)

Zunächst muss für alle Knoten nur eine Basiskonfiguration gewählt werden. Einstellungen, die pro Knoten angepasst werden sollen, lassen sich dann für ausgewählte Knoten „überschreiben“, was in Abbildung 4.9 dargestellt ist.

[25] Die Anwendung verwendet, wenn möglich, dieselben *Views* für Neuerstellen (*create*) und Verän-

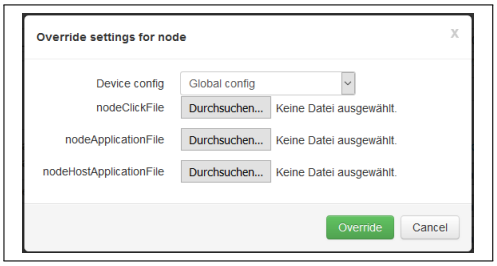

Abbildung 4.9: Überschreiben der Basiskonfiguration eines Netzwerkknotens

Experimente können jederzeit gespeichert oder, wenn alle nötigen Einstellungen vorgenommen wurden, zur vorgesehenen Ausführung zeitlich geplant (*to schedule*) werden. Bei letzterem wird noch eine erweiterte Eingabevalidierung vorgenommen. Der Nutzer muss sich dabei entscheiden, ob der Test nur im Simulator oder zuerst im Simulator und, bei fehlerfreiem Abschluss der Simulation, anschließend automatisch im Testbed ausgeführt werden soll. Ein direktes Ausführen im Testbed ist so nicht mehr möglich. Auf das „Scheduling“ und Ausführen von Experimenten wird in Abschnitt 4.3.4 noch weiter eingegangen.

Experimente haben eine Art Lebenszyklus und durchlaufen verschiedene Zustände. Gespeicherte, zur Ausführung geplante, abgeschlossene und fehlgeschlagene Experimente können in einer Übersicht angezeigt, sortiert, gefiltert und gesucht werden. Diese zeigt Abbildung 4.10.

Dort finden sich weitere Möglichkeiten zum Neuanlegen und Anpassen von Experimenten. Alle Experimente – außer abgeschlossene – lassen sich zum Editieren öffnen. So können zum Beispiel fehlgeschlagene Experimente schnell angepasst und zur erneuten Ausführung gebracht werden. Für einfache Reproduzierbarkeit von Experimenten kann einfach ein vorheriges Experiment kopiert werden. Außerdem hat der Nutzer hier Zugriff auf die Testergebnisse abgeschlossener Experimente. Diese können als Dateidownload abgerufen werden.

4.3.3 Authentifizierung, Autorisierung und Nutzerverwaltung

Bedingt durch die Anforderung nach Beschränkung des Zugangs bei gleichzeitiger Öffnung für bisher nicht unterstützte Nutzergruppen wurde es notwendig, in die Anwendung ein Rechte-/Rollen-Konzept zu integrieren.

dern (*edit*) von Objekten.

[26] begrenzt, im Sinne einer grafisch unterstützten Auswahl

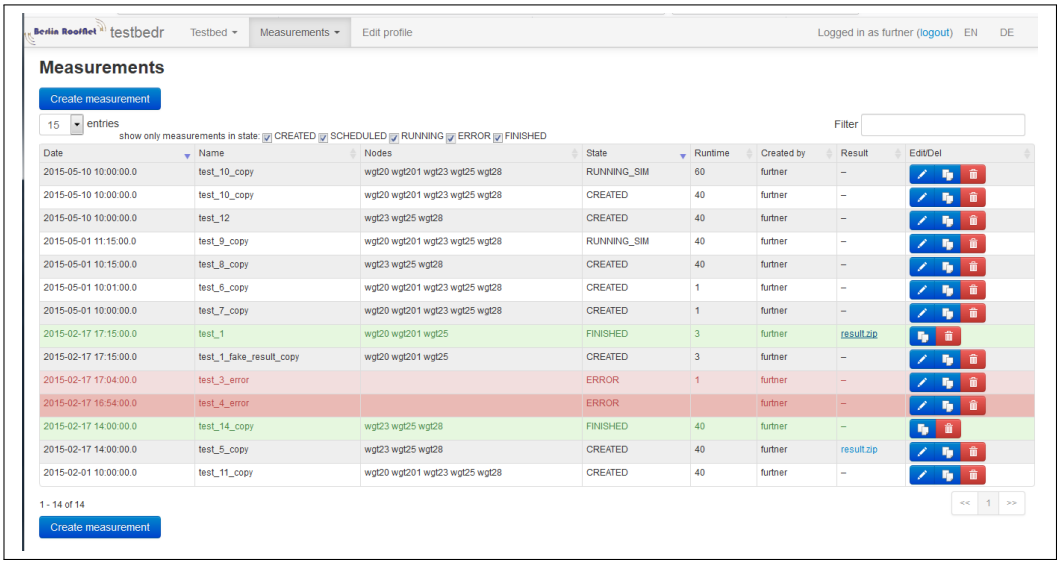

Abbildung 4.10: Übersicht geplante und ausgeführte Testläufe

Wenn der Benutzer nicht angemeldet ist oder für die Aktion nicht die erforderlichen Rechte besitzt, informiert die Anwendung über die Beschränkung des Zugangs und leitet auf eine Login-Seite weiter. Vgl. dazu Abbildung 4.11.

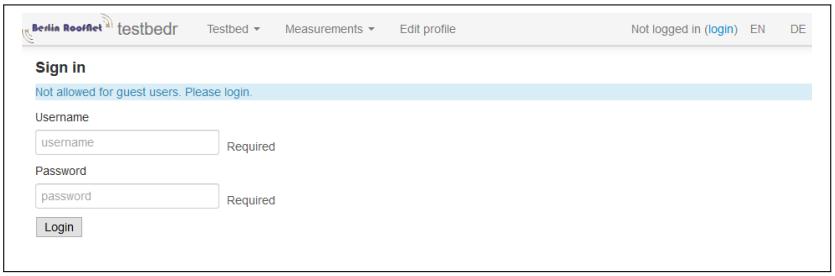

Abbildung 4.11: Login/Security-funktionalität

Um verschiedene Zugriffsszenarien umzusetzen, wurden Nutzerrollen eingeführt, modelliert als die enum-Werte *UserRole.USER* und *UserRole.ADMIN*. Es gibt mindestens einen *ADMIN*-Nutzer. Dieser und andere *ADMIN*-Nutzer können weitere Benutzer, auch administrative, anlegen. Als Unterstützung wurde eine grafische Nutzerverwaltung erstellt. In Abbildung 4.12 ist die Übersicht bestehender Nutzer zu sehen, von der aus Nutzerkonten angepasst und auch gelöscht werden können.

Zusätzlich zur Benutzerverwaltung können Administratoren die globale Anwendungskonfiguration (siehe Abbildung 4.13) anpassen.

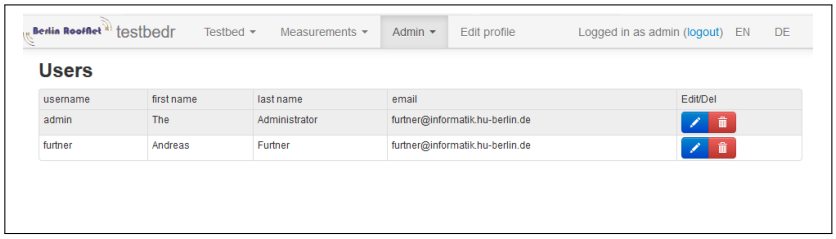

Abbildung 4.12: Nutzerverwaltung

Angemeldete Benutzer mit der Rolle *USER* können Gerätekonfigurationen erstellen und bearbeiten. Weiterhin dürfen sie Experimente erstellen, verändern, planen (schedule) und ausführen sowie auf Ergebnisse vergangener Tests zugreifen.

Dahingegen erhalten nichtangemeldete Benutzer lediglich eine Übersicht und eine Karte der im Testbed vorhandenen Netzwerkknoten.[27]

Zur technischen Umsetzung der oben genannten Punkte wurden sämtliche Endpunkte (Controller actions) abgesichert, die von Nutzern aufgerufen werden können. Dies erfolgt durch die Annotation *@Restricted* an den einzelnen Methoden. Beispielhaft sei hier die Funktion „Anzeigen der Gerätekonfigurationen“ aufgeführt, auf die Benutzer mit der Rolle *USER* oder *ADMIN* Zugriff erhalten, während nichtangemeldete Benutzer beim versuchten Zugriff auf die Anmeldungsseite umgeleitet werden:

```
@Restricted(userRoles = {UserRole.USER, UserRole.ADMIN})
public static Result listDeviceConfigs() {
    Form<DeviceConfig> configForm = form(DeviceConfig.class);
    return ok(
      deviceConfigs.render(DeviceConfig.findAll(), configForm)
    );
}
```

Die Annotation *@Restricted* führt dazu, dass mit Hilfe von Play2-Infrastruktur ein generischer Proxy[13] den Aufruf der entsprechend annotierten Controller-Methode abfangen kann. In der den Proxy implementierenden Klasse *RestrictedAction* wird die Authentifizierung und Autorisierung durchgeführt und bei Erfolg der Aufruf an die obige Methode delegiert.

[27] wie schon vorher in Abbildung 4.1 dargestellt

4.3.4 Scheduling und Testausführung

Die Anwender können beim Erstellen oder späteren Anpassen eines Experimentes Ausführungszeitpunkt und geplante Laufzeit einstellen.[28] Soll das Experiment zur Ausführung eingestellt werden, muss der „Schedule“-Knopf gedrückt werden. Wie auch beim einfachen Speichern (ohne „Schedule“) finden diverse Eingabevalidierungen, z.B. für Zahlenwerte, statt. Fürs Scheduling werden aber noch weitere Plausibilitätsprüfungen vorgenommen, ohne die eine Bereitstellung zur Ausführung nicht möglich ist. So muss z.B. mindestens eine Gerätekonfiguration für alle Knoten eingestellt sein, der Ausführungszeitpunkt in der Zukunft liegen und eine maximale Laufzeit für das Experiment angegeben werden.

Außerdem dürfen Experimente nicht miteinander kollidieren. Hierzu wird überprüft, dass der Ausführungszeitpunkt noch nicht von anderen Tests „gebucht“ ist. Die Ressourcen des Testbed werden also vorerst exklusiv zur Verfügung gestellt, so dass immer nur ein Test laufen kann. Diese Prüfung erfolgt sofort beim Scheduling.

Um die Tests zum geplanten Zeitpunkt auszuführen, musste eine Möglichkeit zum zeitgesteuertem Starten von Jobs auf dem Server umgesetzt werden. Play2 bringt eine dazu benötigte Infrastruktur in Form von *akka*[30] mit. Akka ist ein Framework zur Komposition von Diensten mittels nebenläufiger, sogenannter Aktoren, welche untereinander Nachrichten austauschen. Es bietet auch die Möglichkeit, solche Aktoren zeitgesteuert zu starten. Mit *akka* war es naheliegend, einen Aktor *MeasurementScheduler* zu implementieren, der in regelmäßigen Intervallen die Datenbank prüft, ob Experimente zur Ausführung zu diesem Zeitpunkt vorliegen.

Wenn ein Experiment zur Ausführung kommen soll, wird dieses entsprechend an den *MeasurementExecutor* übergeben. Von diesem werden analog zum bisherigen manuellen Testablauf aus der Konfiguration und festen Templates, die von den Shell-Skripten erwarteten Konfigurationsdateien in einem Arbeitsverzeichnis im Dateisystem angelegt. Die Templates nutzen dieselbe Technologie wie zur Erzeugung der *Views*, also der Weboberfläche der Anwendung. Play2 stellt dafür die Scala-basierte *Template Engine* „Twirl“ bereit. *Twirl*-Templates werden kompiliert und sind typsicher. Damit können zusammen mit der richtigen Modellierung der möglichen Parameterwerte und der Eingabevalidierung nur noch syntaktisch korrekte Konfigurationsdateien erzeugt werden.

Zum Starten des Experimentes werden nun Betriebssystemprozesse gestartet, welche die, in der Anwendung global konfigurierten, Shellskripte für „runSim“ und „runMeasurement“ entsprechend zur Ausführung bringen. Es wird dabei auf die konfigurierte

[28] siehe auch Abbildung 4.10

Laufzeit geachtet und – falls erforderlich – diese Prozesse auch vorzeitig beendet. Deren Statuscodes werden ausgewertet und eine Ausführung im Testbed nur nach erfolgreichem Beenden der Simulation gestartet.

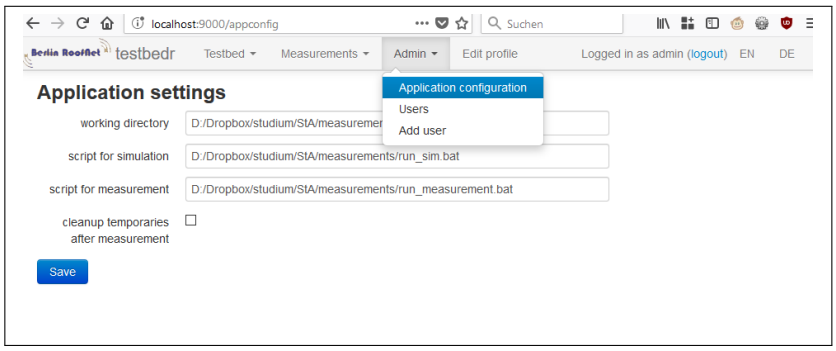

Abbildung 4.13: Konfiguration der Startskripte

Nach Beendigung des Experimentes liegen die Testergebnisse zunächst wie bisher im Dateisystem vor. Diese werden durch die Anwendung dann gesammelt, archiviert, komprimiert und zusammen mit den anderen Artefakten in der Datenbank gespeichert. Sie sind dann, wie schon beschrieben, in der Übersicht der Experimente verfügbar. Damit ist eine spätere Nachvollziehbarkeit und Reproduzierbarkeit der Experimente gegeben und durch das beschriebene Scheduling, wird ein unbeaufsichtigtes, automatisches Ausführen von Experimenten möglich.

5 Fazit und Ausblick

Die Anwendung wurde den Anforderungen gemäß fertiggestellt und, wie im Kapitel 4 ausgeführt, wird der Tester nun bei der Vorbereitung, Erstellung, Planung und Auswertung von Experimenten deutlich besser unterstützt. Fehler bei der Erstellung der Testkonfigurationen werden durch den geführten Prozess und die erweiterten Möglichkeiten einer grafischen Oberfläche minimiert oder ganz ausgeschlossen. Durch die automatische Archivierung von Testinputs und -ergebnissen und die Möglichkeit, vergangene Experimente zu klonen, ist zu erwarten, dass Transparenz, Reproduzierbarkeit und damit die wissenschaftliche Qualität von Forschungsarbeiten in Zukunft noch verbessert werden.

Für den Lehrstuhl und den Betrieb des Testbed können sich nach Inbetriebnahme der Testmanagementsoftware weitere Vorteile ergeben. So werden dann Nutzerzugänge am Lehrstuhl auf Betriebssystemebene nicht mehr unbedingt nötig sein, um Experimente im Simulator oder Testbed auszuführen. Auch Externen könnte somit der Zugang zum Testbed ermöglicht werden. Es ist zwar nicht ausgeschlossen, den bisherigen Prozess der manuellen Konfiguration und Testausführung auch in Zukunft beizubehalten, eine exklusiven Benutzung der Software würde aber eine deutlich bessere Gesamtübersicht über Experimente und Forschungsprojekte ermöglichen. Auch können bestehenden Ressourcen wie die HWL-DB durch die Anbindung an die Software besser oder überhaupt erst benutzt werden. Da die Anwendung die Einstiegshürde zur Benutzung des Testbed senkt, wird möglicherweise auch die Auslastung der Ressourcen besser.

Bei der Umsetzung der Anforderungen sind einige Probleme aufgefallen. Eine Zugriffsbeschränkung mit (nur zwei) Nutzerrollen reicht eventuell nicht aus, um in Zukunft bei vielen, möglicherweise auch externen Nutzern, ein ausreichendes Rechtemanagement zu gewährleisten. Hier müsste vom Lehrstuhl bewertet werden, ob es eingeschränkte Nutzerrollen geben soll, bei denen Nutzern nur der Zugriff auf die *eigenen* Experimente gestattet wird. Auch ein Gruppenkonzept oder ACLs[1] für einzelne Experimente könnten sich als sinnvoll herausstellen.

[1] Access Control Lists

Außerdem ergaben sich Probleme bei der Integration der bestehenden Systeme, mit denen die Anwendung interagiert. So geben die Shellskripte zum Starten von Simulationen und Messungen im Testbed aktuell nur „erfolgreich“ oder „fehlerhaft“ als Exit-Status zurück. Daher kann die Anwendung dazu auch keine weiteren Informationen anzeigen. Hier wären für ein besseres Nutzererlebnis, insbesondere im Fehlerfall, weitere Status- bzw. Fehlercodes wünschenswert gewesen.

Das durch die HWL-DB vorgegebene Datenbankschema ist nicht normalisiert und die Tabellenstruktur mit nur zwei Tabellen führte daher beim objektrelationalem Mapping zu noch verbesserungswürdiger Modellierung. Der Datenbestand dieser Datenbank ist nicht vollständig und nicht ausreichend gepflegt. Es existieren zum Beispiel nicht zu allen Knoten Standortkoordinaten oder diese stimmen nicht mit dem tatsächlichen Standort überein, so dass die Visualisierung in der Kartenansicht nicht die eigentliche Mächtigkeit des Testbed wiedergeben kann. Die obigen Punkte leiten schon einen Ausblick auf zukünftige Arbeiten und Erweiterungsmöglichkeiten ein.

Erst während des praktischen Einsatzes der Software wird sich herausstellen, ob die bisherigen Anforderungen ausreichend waren oder angepasst werden müssen. Erste Gespräche noch während der Umsetzung haben allerdings schon ergeben, dass einige funktionale Erweiterungen wünschenswert sind. Dabei haben sich vor allem zwei Themengebiete herauskristallisiert:

- Ausbau der Anwendung zur Verwaltungsoberfläche für nahezu alle Aspekte des Testbed
- Ermöglichung von mehr Experimenten durch weitere Automatisierung und verbessertes Scheduling

Der erste Aspekt zielt darauf, mehr Testbed- bzw. Hardwareinformationen umfangreicher, nutzerfreundlicher und genauer darzustellen. Da schon eine Kartendarstellung integriert ist, wäre auch eine einfachere Prüfung und Pflege der Standortdaten über die Anwendung vorstellbar. Dazu müsste diese dann allerdings auch Daten in die Hardwaredatenbank zurückschreiben dürfen. Das würde die Datenqualität in HWL-DB und Anwendung verbessern. Außerdem wäre die Anbindung weiterer schon existierender Schnittstellen mit Status-Informationen (Nagios, SNMP) hilfreich. Diese Informationen sollten dann zusammen mit dem Status aktuell laufender Experimente auch live in der Oberfläche visualisiert und aktualisiert werden.

Zum zweiten Punkt wurde festgestellt, dass eine größere Automatisierung beim Scheduling und Ausführen von Experimenten erstrebenswert wäre. Durch Mitarbeiter des Lehrstuhls wurden neue Shellkripte erstellt, die weitere, bisher in der Anwendung nicht berücksichtigte, Parameterreihen entgegen nehmen. Diese neuen Skripte rufen

die bisherigen, auch von der Anwendung aufgerufenen Startskripte auf und ermöglichen das wiederholte aufeinanderfolgende Ausführen des gleichen Experimentes, aber mit automatisch innerhalb eines Wertebereiches variierten Parametern. Eine Unterstützung dieser Art von Experimenten sollte durch wenige Anpassungen in der Anwendung möglich sein, zumal die aufzurufenden Startskripte bereits konfigurierbar sind.

Ein weiterer Aspekt des zweiten Punktes betrifft die gleichzeitige und geteilte Nutzung der Ressourcen des Testbed. Da es sich um viele, auch räumlich voneinander entfernte Netzwerkknoten handelt, liegt es nahe, mehrere Experimente gleichzeitig durchführen zu wollen. Die Anwendung wurde jedoch bewusst so umgesetzt, dass nur ein Experiment pro Zeitraum zugelassen ist. Eine Entscheidung, welche Ressourcen in welcher Art und Weise aufgeteilt und damit gleichzeitig benutzt werden können, ist in dieser Domäne nicht trivial und führte über die Zielstellung dieser Arbeit hinaus. Dazu müssen zunächst Kriterien entwickelt werden, unter welcher Bedingungen Experimente miteinander „verträglich“ oder „ausgeschlossen“ sind. Diese Kriterien könnten zum Beispiel räumliche Entfernung, benutze Kanäle oder Signalstärken der an zwei Experimenten beteiligten Geräte mit einschließen. Damit wäre es denkbar, dass zukünftig ein Anwendungsmodul automatische regelbasierte Entscheidungen trifft, ob Experimente gleichzeitig laufen dürfen. In einer weiteren Ausbaustufe wäre zudem eine automatische Zuteilung von Ausführungszeiträumen für ins System eingestellte Experimente vorstellbar. Der Tester gäbe in diesem Fall keinen Termin vor, sondern überließe das Scheduling allein der Anwendung und wartete lediglich die Ergebnisse nach einer von ihm definierten Frist ab.

Eine ausführliche Analyse der aufgeführten Themenkomplexe bleibt zukünftigen Arbeiten vorbehalten.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

5G Fifth Generation, geplanter Nachfolgestandard von LTE 3
ACL Access Control List 29
API Application programming interface 17
BLOB *Binary Large Object* 17
BOWL *Berlin Open Wireless Lab* 4
BRN *Berlin Roof Net*
BSD Berkeley Software Distribution
GUI Graphical User Interface 14
HWL *Humboldt Wireless Lab* 3
ISO/OSI-Modell Open Systems Interconnection Model der International Organization for Standardization
JiST *Java-in-Simulation-Time*
JPA *Java Persistence API* 16
JSF *Java Server Faces* 15
JVM *Java Virtual Machine* 15
LTE Long Term Evolution, ein Mobilfunkstandard der vierten Generation 3
MIT Massachusetts Institute of Technology 4
MIOT *Magedeburg Internet of Things* 4
MVC Model-view-controller 14
ORM Object relational mapping 16
OSM *Open Street Map* 17
PARC Palo Alto Research Center 14
SAR Lehrstuhl für Systemarchitektur
WLAN Wireless Local Area Network, deutsch: „drahtloses lokales Netzwerk“ 1

Literaturverzeichnis

[1] Barr, Rimon: An efficient, unifying approach to simulation using virtual machines, Cornell University, Ph.D. dissertation, 2004. ttp://jist.ece.cornell.edu/docs/040517-thesis.pdf

[2] Bauer, Christoph: Analyse von DHT-basierten Routingverfahren in drahtlosen Maschennetzwerken, Humboldt-Universität zu Berlin, Studienarbeit, 2013. https://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2013-07/SAR-PR-2013-07_.pdf

[3] Berdychowski, Kamil: Domain-Driven Design vs. anemic model. How do they differ. (2017). https://blog.pragmatists.com/domain-driven-design-vs-anemic-model-how-do-they-differ-ffdee9371a86

[4] Centre Tecnològic de Telecomunicacions de Catalunya (CTTC): Click Modular Router. http://networks.cttc.es/mobile-networks/software-tools/click/,

[5] Cornell University: JiST / SWANS, Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator - Documentation. http://jist.ece.cornell.edu/docs.html

[6] datatables.net: Datatables (Homepage). https://datatables.net/,

[7] Ebean community: Ebean ORM for Java (Projektseite bei github). https://ebean-orm.github.io/,

[8] Evans, Eric: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003. – ISBN 0321125215

[9] Fowler, Martin: AnemicDomainModel. https://www.martinfowler.com/bliki/AnemicDomainModel.html

[10] Fowler, Martin: Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. – ISBN 0321127420

[11] Furtner, Andreas: testbedr: Eine Anwendung für die Verwaltung von Experimenten im Testbed des Humboldt Wireless Lab (Quelltext). https://gitlab.informatik.hu-berlin.de/furtner/hwl-testbedr. Version: 2018

[12] Kapitel 1. In: Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissiedes, John: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. – ISBN 0–201–63361–2, S. 4–6

[13] Kapitel 4. In: Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissiedes, John: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. – ISBN 0–201–63361–2, S. 207–217

[14] getbootstrap.com: Bootstrap Homepage. https://getbootstrap.com/,

[15] Günes, Mesut et al.: Magdeburg Internet of Things Testbed - Homepage. http://www.comsys.ovgu.de/MIOT_Lab-p-174.html

[16] Günes, Mesut et al.: Münster Internet of Things Testbed - Homepage. https://www.uni-muenster.de/MIOT-Lab/

[17] Kohler, Eddie: Click Modular Router (Quellcode bei github). https://github.com/kohler/click,

[18] Kohler, Eddie ; Morris, Robert ; Chen, Benjie ; Jannotti, John ; Kaashoek, M. F.: The Click Modular Router. (2000). https://pdos.csail.mit.edu/papers/click:tocs00/paper.pdf

[19] Kurth, Mathias: Opportunistic Routing with Adaptive CSMA/CA in Wireless Mesh Networks / Humboldt University, Systems architecture group. 2010. – Technical Report

[20] Lange, Frank: Evaluation of alternative backoff schemes for 802.11, Humboldt-Universität zu Berlin, Bachelorarbeit, 2014. http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2014-06/SAR-PR-2014-06_.pdf

[21] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: Geo-Services Communication Infrastructure. http://sar.informatik.hu-berlin.de/research/geo_sensor_networks/geo_sensor_networks.htm,

[22] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: Homepage des Lehrstuhls. http://sar.informatik.hu-berlin.de

[23] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: Humboldt Wireless Lab Github Homepage. http://humboldtwirelesslab.github.io/

[24] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: Seite Publications am Lehrstuhl für Systemarchitektur. https://sar.informatik.hu-berlin.de/research/publications/index.htm

[25] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: Wireless Mesh (Community) Networks. http://sar.informatik.hu-berlin.de/research/wireless_mesh/wireless_mesh.htm,

[26] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: SAR-Wiki: Brn.Sim. http://sarwiki.informatik.hu-berlin.de/Brn.Sim, September 2010

[27] Lehrstuhl für Systemarchitektur der Humboldt-Universität zu Berlin: SAR-Wiki: HWL-Testbed Computers, Networks & Simulations. https://sarwiki.informatik.hu-berlin.de/HWL-Testbed_Computers,_Networks_%26_Simulations, 2010

[28] Lightbend, Inc.: play – The High Velocity Web Framework For Java and Scala. https://www.playframework.com/,

[29] Lightbend, Inc.: Slick - Functional Relational Mapping for Scala (Homepage). http://slick.lightbend.com/,

[30] Lightbend, Inc.: akka – Build powerful reactive, concurrent, and distributed applications more easily (Documentation). https://akka.io/docs/, 2018

[31] Makalancheche, Mahlatse: Anemic vs. Rich Domain Objects—Finding the Balance. https://dzone.com/articles/anaemic-vs-rich-domain-objects-finding-the-balance

[32] Merz, Ruben ; Schiöberg, Harald ; Sengul, Cigdem: Design of a Configurable Wireless Network Testbed with Live Traffic. In: Testbeds and Research Infrastructures, Development of Networks and Communities: 6th International ICST Conference, TridentCom 2010, Berlin, Germany, ... and Telecommunications Engineering), Springer, 2011. – ISBN 3642178502, S. 189–198

[33] OPENLAYERS.ORG: OpenLayers - A high-performance, feature-packed library for all your mapping needs, Homepage. http://openlayers.org/,

[34] PHPMYADMIN.NET: phpMyAdmin – Bringing MySQL to the web (homepage). https://www.phpmyadmin.net/

[35] REENSKAUG, Trygve: Notes and Historical documents: MVC XEROX PARC 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html,

[36] SAPM, Course B.: The Anaemic Domain Model is no Anti-Pattern, it's a SOLID design. https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/

[37] SOMBRUTZKI, Robert: Kanalzuweisung und verteilte Antennen in IEEE 802.11 Infrastruktur-Netzwerken, Humboldt-Universität zu Berlin, Diplomarbeit, 2009. http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2009-19/SAR-PR-2009-19_.pdf

[38] STATISTISCHES BUNDESAMT: Ausstattung privater Haushalte mit Informations- und Kommunikationstechnik - Deutschland. https://www.destatis.de/DE/ZahlenFakten/GesellschaftStaat/EinkommenKonsumLebensbedingungen/AusstattungGebrauchsguetern/Tabellen/Infotechnik_D.html,

[39] THE JQUERY FOUNDATION: jQuery: write less, do more, Homepage. http://jquery.com/,

[40] USC, Information Science I.: The Network Simulator - ns-2. https://www.isi.edu/nsnam/ns/,

[41] WIKIPEDIA: OSI-Modell. https://de.wikipedia.org/wiki/OSI-Modell,

[42] WIKIPEDIA: Wikipedia Artikel Model-view-controller. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller,

[43] ZUBOW, Anatolij ; SOMBRUTZKI, Robert: A low cost MIMO mesh testbed based on 80211n. In: 2012 IEEE Wireless Communications and Networking Conference: Services, Applications, and Business, 2012