Das NetworkParser-Framework

Ein Entwicklungtool für den SDM-Ansatz vom Stories bis zur fertigen Anwendung

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
(Dr. rer. nat.)

Elektrotechnik / Informatik
Research Group Software Engineering
Universität Kassel

vorgelegt von Stefan Lindel

bei Prof. Dr. Albert Zündorf

Kassel, den 10. März 2019

Danksagung

An erster Stelle gilt mein Dank meinem Doktorvater Herr Prof. Dr. Albert Zündorf für seine wissenschaftliche und methodische Unterstützung während der gesamten Bearbeitungsphase meiner Dissertation. Albert danke ich für die zahlreichen und unermüdlichen fachlichen Gespräche, Ratschläge und Anmerkungen, die mich auf dem Weg zur fertigen Arbeit immer wieder neue Aspekte und Ansätze entdecken ließen.

Vielen Dank richte ich auch an Frau Biehlig, welche immer ein offenes Ohr für mich und meine Probleme hatte.

Ein sehr großes Dankeschön geht an meine Frau, die mir die notwendigen Freiräume gab, indem sie sich noch intensiver um unsere gemeinsamen zwei Kinder kümmerte, um effektiv an meiner Dissertation arbeiten zu können. Während meiner Dissertation hatten wir gemeinsam leider ein paar Tiefschläge einstecken müssen, aber zusammen haben wir immer einen Weg gefunden.

Ein ausdrückliches Dankeschön geht an meine Schwester Silvia, welche mich beim Schreiben insbesondere beim Formatieren und Korrekturlesen sehr stark unterstützt hat.

Ein weiterer Dank geht an die zahlreichen Studierenden, die an den Umfragen freiwillig teilgenommen haben, bzw. bei den Lehrveranstaltungen “Programmiermethodik” und “Software Engineering 1” indirekt als Versuchskandidaten fungiert haben.

Die Disputation fand an der Universität Kassel am 27.09.2019 statt.

Zusammenfassung

Die Dissertation hat das Ziel ein Tool zur Verfügung zu stellen, um den Entwickler bei der Entwicklung eines Programms in jedem Entwicklungsschritt zu unterstützen, angefangen von dem Schreiben von Storys und Tasks. Diese werden dann im Mikado oder Kanbanboard organisiert. Die Abläufe sind dabei agil umgesetzt, sodass diese iterativ erweitert werden können. Das Tool soll so intuitiv bedienbar sein, dass Programmieranfänger damit arbeiten können und die Techniken erlernen können.

Nach dem Aufschreiben der Storys kann ein Programm mittels des Entwurfes eines Modells über die Codegenerierung bis zur Serialisierung und Verteilung zu anderen Rechnern entstehen. Darüber hinaus wird eine eigene auf Modelle angepasste Collection entwickelt, diese ist optimiert auf Zugriff, Speicher und Funktionalität. Es soll eine einheitliche Oberfläche integriert werden, welche plattformunabhängig, verteilbar im Netzwerk und auf bestehende etablierte Strukturen aufsetzt, sodass ein Entwerfen guter Oberflächen möglich ist. Die als Kernkomponente entwickelte Netzwerkkomponente wird mittels eines ad-hoc Netzwerk funktionieren.
Die Kommunikationsschnittstelle umfasst mittlerweile zahlreiche Wege wie TCP, E-Mail, XMPPP, aber auch verteilte Protokolle wie MQTT und AMPQ.
Durch die Einbeziehung von "Google Cloud Messaging für Android" [Goo12] und ad-hoc Techniken ist es möglich Smartphones, und Browser als Rechnerknoten zu integrieren. Bei den Smartphones ist das Betriebssystem Android sehr weit verbreitet. [LG10], [LG11], [LG12], [Joe17]
Die Vorgehensweise des Rahmenwerks ist "Agile" und passt sich somit immer auf die aktuellen Bedürfnisse des Entwicklers an. Ein weiteres Ziel ist die einfache Einbindung der NetworkParser Funktionalitäten und das aufeinander aufbauen der Software Entwicklungsschritte, sodass ein einfaches Überführen der Ergebnisse der einzelnen Entwicklungsphasen

in den Arbeitsschritten gegeben ist. Dem Endanwender wird lästiges Konfigurieren mit Standardeinstellungen abgenommen.
Der Ausgangspunkt der Systementwicklung sind Klassendiagramme oder Klassenmodelle. Das Erstellen von Klassenmodellen soll möglichst intuitiv sein, weiterhin soll eine Dokumentation generiert werden können. Durch das entwickelte NetworkParser-Framework kann die Codegenerierung mittels Templates durchgeführt werden. Die Templates sind individuell anpassbar. So können die Templates angepasst werden, um eine individuelle Sprache oder spezielle Methoden oder Klassen zu erzeugen.

Ein weiteres Feld, welches im Fokus stand, waren Verteilte Systeme. Es sollte ein Verfahren unterstützt oder entwickelt werden, welches ein gemeinsames Arbeiten wie Google Docs auf Modellebene unterstützt. Fokus ist schon die Entwicklung einer Anwendung bis hin zum gemeinsamen Benutzen der Anwendung unterstützt werden soll. Also der komplette Entwicklungsprozess und darüber hinaus. Diese arbeiten versionsbasierend zusammen und können wie CVS/SVN/GIT [DW06] Änderungen zwischen einzelnen Versionsständen ermitteln und Konflikte möglichst selbstständig auflösen.

Weiterhin soll es möglich sein, eine hierarchische Struktur zu definieren, um das Datenmodell zu strukturieren und so ähnlich wie MQTT-Topics die Synchronisation zu beeinflussen. Somit ist es einfach möglich einzelne Netzwerkknoten auch nur einen Teil des Datenmodells zur Verfügung zu stellen. Das soll zusätzlich den Datenverkehr reduzieren und Programmsicherheit bringen.
Das verteilte System soll möglichst ausfallsicher sein. Hierzu sollen die einzelnen Modelländerungen zur Laufzeit in einer Historie gespeichert werden, welche im Arbeitsspeicher gehalten werden. Durch den Java-Müllsammler (Garbage-Collection) der nicht mehr benötigten Änderungen löscht und probiert einzelne Änderungen immer zusammenzufassen wird der Speicherverbrauch in der Historie reduziert.
Mittels ModelChanges soll im besten Fall eine optimale Initialisierung des Datenmodells zur Laufzeit hergeleitet werden (Kürzester Änderungsverlauf).

Falls ein Rechnerknoten ausfällt oder nicht alle Nachrichten bekommen hat, soll es anhand dieser Historie möglich sein, die fehlenden Lücken zu schließen und unterschiedliche Ausprägungen des Datenmodells zu vereinheitlichen und Konflikte zu lösen.

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit gibt es unzählige Programmiersprachen und sehr viele speziell angepasste Entwicklungsumgebungen, die für jeden Anwendungseinsatz angepasst sind. Dazu zählen Standardsoftware für den Desktoprechner, mobile Anwendungen für das Smartphone oder Webanwendungen als Zwischenlösung. Diese Vielfalt wird verstärkt durch die Plattformabhängigkeit von Anwendungen, sodass diese nicht automatisch überall funktionieren geschweige denn gleich aussehen. Dieses zeigt sich gerade im Vergleich für die Betriebssysteme iOS, Windows, Android, Linux um nur einige zu nennen.

Dieses allein bietet schon eine große Vielfalt. Durch die unterschiedlichen genutzten Systemarchitekturen von Desktop, über Client/Server, Webanwendungen bis zu mobilen Anwendungen der einzelnen umzusetzenden Anwendungsfälle wird dieses noch gesteigert. Die Kommunikation zwischen den Anwendungen wird zusätzlich durch unterschiedliche Berechtigungen und Ansichten auf das Datenmodell erschwert.

Die Komplexität zeigt sich bereits in einem einfachen Beispiel. Für die Realisierung einer Gruppenkasse als Desktopanwendung wird ein Datenmodell modelliert und separat eine passende Benutzeroberfläche erstellt. Danach wird die Programmlogik geschrieben und die Anwendung ist fertig. Um die Anwendung nun auf unterschiedlichen Smartphones wie Android oder iPhone benutzen zu können, muss die Anwendung entweder als Webdienst umgeschrieben werden oder mittels Multibuilds für die unterschiedlichen Betriebssysteme übersetzt werden oder mehrfach implementiert werden. Soll der Dienst mehrbenutzerfähig sein, ist der Standardweg eine Datenbank anzubinden, oder einen zentralen Server einzurichten.

Für die objektorientierte Modellierung zum Start dieser Dissertation gab es nur das in die Jahre gekommene Tool Fujaba [NNZ00a, NNWZ00], welches nicht mehr weiterentwickelt wird. Dieses auf Swing basierende Tool war speziell für das Eclipse-Umfeld [MKF06] entwickelt worden. Es ist ein CASE Tool Framework, wodurch die Benutzung gut für die Lehre geeignet ist. Durch die strukturellen Änderungen in Eclipse und durch das populärer werden andere IDE wie InteliJ IDEA [Int11], war es notwendig neue Wege zu beschreiten.

Ein neues Tool wurde benötigt. Es sollte ein neues textbasiertes Modellierungstool entwickelt werden, welches frei zur Verfügung stehen sollte. SDMLib [LZ12] wurde somit entwickelt. Es gab eine Reihe von neuen Ideen, wie textorientierte Modellierung welches im Gegensatz zu den Designansatz von Fujaba nachvollziehbarer ist. Damit sollte die Abhängigkeit zu Benutzeroberflächenframeworks reduziert werden und die evaluierten Kritikpunkte (siehe Abbildung 47 auf Seite 153) an nicht nachvollziehbaren Designer (Editoren) behoben. Die komplexe Installation und die komplizierte Persistierung wurden weiterhin neu umgesetzt.

SDMLib stützt sich auf den im Rahmen dieser Arbeit entwickelten NetworkParser auf. Ich habe den NetworkParser seit dem 21.12.2011 entwickelt. Anfangs war der Hauptfokus des NetworkParser die automatische Serialisierung und Persistierung. Als dieses gelungen war, wurde der NetworkParser mit der Visualisierung von Modellen ergänzt. Bei der Entwicklung eines Peer-To-Peer System hat sich gezeigt, dass es notwendig ist, sich die Veränderungen von Datenmodellen der einzelnen Peers, zu Testzwecken und für Dokumentationszwecke, visuell aufzubereiten. Somit wurde der NetworkParser um die Möglichkeit erweitert Klassen- und Objektmodelle gelayoutet darzustellen.
Der Ansatz der Sourcecodegenerierung wurde von SDMLib überarbeitet und hinzugefügt. Diese reicht nun bis zur Model-Transformationen. Die Entwicklung vom NetworkParser wurde durch die Integration der GUI Programmierung vervollständigt. In der Entwicklungskette fehlte nur noch

die Projektmanagementfunktionalität. Diese wurde mittels der Requiments Engineering, User-Stories, Test und Doku Support integriert. Einige weitere Ideen für die Erweiterung der Projektmanagementfunktionalität stehen im Fazit.

Bei der Entwicklung hat sich gezeigt, dass die Aufteilung der Funktionalität auf die Projekte SDMLib und NetworkParser nicht immer gelingt und somit wurden die zusammengehörigen Punkte für die Modellierung und Generierung in überarbeiteter Form in den NetworkParser übernommen. Weiterhin gab es unterschiedliche Programmierstile, die aufeinandertrafen. SDMLib und NetworkParser sollten ein lehrunterstützendes Framework darstellen, welches für die Bedürfnisse der unterschiedlichen praxisorientierten Lehrveranstaltungen vom ersten bis sechstes Semester geeignet ist. Wir haben SDMLib und NetworkParser neben der Lehre auch in zahlreichen Forschungsprojekten und bei der Entwicklung von Kundenanwendungen wie ConfNet erfolgreich eingesetzt. SDMLib und NetworkParser haben sich darüber hinaus seit Jahren beim Internationalen Transformation Tool Contest (TTC) [Afc18] bewährt und dort zahlreiche Auszeichnungen erhalten. Durch die Teilnahme am TTC haben sich SDMLib und NetworkParser vor allen im Bereich Performance aber auch im Bereich Usability stark verbessert.

1.1 Darlegung des Problems

Es gibt eine Reihe von unterschiedlichen Tools und Frameworks, die auch die einzelnen Aspekte der oben beschriebenen Softwareentwicklungsaufgaben adressieren. Diese Tools sind meist nur für ihrem speziellen Anwendungsfall einsetzbar. Es kann also sein, dass die verschiedenen Probleme nur mit einer Vielzahl von unterschiedlichen Tools oder Framework realisiert werden können, welche in den seltensten Fällen aufeinander abgestimmt sind.

1.1.1 Party-App

Ein großer Anwendungsbereich für SDMLib und NetworkParser ist der Einsatz in der Lehre. Dies beginnt mit der jährlichen Vorlesung "Programmiermethodik" an der Uni Kassel. Als erstes Anwendungsbeispiel verwenden wir die Entwicklung einer Party-App. Diese wird im Laufe der Zeit zunehmend komplexer und spiegelt die unterschiedlichen Anwendungsgebiete wieder. Die Anwendung dient zum Planen einer Party. Dort gibt es verschiedene Teilnehmer, welche eine Reihe von Gegenständen für die Party kaufen wie Getränke oder Essen. Die Einkäufe werden mit der PartyApp erfasst und gegenseitig abgerechnet. Wir beginnen die PartyApp als Simple User-Desktopanwendung und entwickeln die Anwendung bis zur Multi-Useranwendung weiter. In späteren Veranstaltungen wird dies zu einer Web-Anwendung und zu einer Multi-Plattform MobilApp weiterentwickelt.

In der Vorlesung Programmiermethodik wird die PartyApp zunächst diskutiert und die Studierenden beschreiben die Funktionalität der Anwendung mit User-Stories. Die User-Stories (siehe Abschnitt 5.3 auf Seite 75) werden dann mit Komplexität geschätzt und nach Wichtigkeit sortiert. Aus diesen User-Stories werden dann die einzelnen Aufgaben (Tasks) abgeleitet und mit zeitlichen Aufwandsschätzungen versehen.
Aus den Storys wird auch das Klassenmodell abgeleitet. Das Klassenmodell kann mithilfe von SDMLib und NetworkParser als Quellcode geschrieben oder mit dem mitgelieferten Diagrammeditor vom NetworkParser designt werden. Anschließend kann eine Implementierung des Klassenmodells in Programmcode einer Programmsprache, hier Java-Sourcedateien zum Beispiel generiert werden.

Das Klassenmodell wird meist über mehrere Wochen hinweg iterativ weiterentwickelt. Gleichzeitig erweitern die Studierenden den generierten Sourcecode zum Beispiel um zusätzliche Methoden bis hin zur Businesslogik oder für einfache Protokollierungsanweisungen. Damit die manuellen Änderungen und Erweiterungen des Programmcodes bei der Neugenerierung

nicht verloren gehen, verändert der Generierungsprozess im einfachsten Fall die Dateien adaptiv. Das bedeutet, Änderungen am Klassenmodell, wie zum Beispiel Änderungen des Typs von Kardinalitäten von Assoziationen oder Datentypen von Attributen werden in den generierten Quellcode übertragen, ohne dass manuelle Änderungen verloren gehen. Dadurch bleiben angepasste Methoden und zusätzliche Informationen an Attribute und Methoden erhalten. Dieses ist enorm wichtig, um eine Akzeptanz beim Endanwender zu erreichen und durch das Verhalten des Generators keinen Programmcode zu verlieren.

Diese selektive Codegenerierung ist ein Alleinstellungsmerkmal und in meinem NetworkParser integriert (siehe Abschnitt 5.6 auf Seite 80). Überschreibt ein Codegenerator den Programmcode der Studierenden, führt dieses zu großem Unmut bei den Studierenden.
Dokumentation ist ein wichtiger Teil und kann automatisch mit generiert werden (siehe Abschnitt 5.9 auf Seite 93). So ist es möglich die Storys und abgeleitete Objekt- und Klassendiagramme als HTML Dateien zu generieren.

In der Lehrveranstaltung "Software Engineering 1" wird Wert auf Programmierqualität gelegt. Der Programmcode der Studierenden soll mindestens zu 75 Prozent getestet sein. Bei den Präsentationen hört man schnell, dass Teile nicht getestet sind, da Sie ja von SDMLib generiert wurden und somit nicht getestet werden müssen. Die Grenze verschwimmt jedoch, wenn in den generierten Programmcode eigene Programmzeile hinzugefügt werden oder gar Methoden mit Rumpf generiert werden.
Um den Programmcode dennoch zu testen habe ich im NetworkParser die BlackboxTesting-Klassen entwickelt, welche eine gute Testabdeckung mit Analyse der meisten Fehler garantiert. Das Test-Rahmenwerk ist weiterhin für Bibliotheken gedacht und somit können auch die APIs von Bibliotheken, gut getestet werden. (siehe Abschnitt 5.4 auf Seite 78)

Nachdem das Datenmodell erstellt wurde kann es nun mit der Implementierung der Programmlogik und der Benutzeroberfläche weitergehen. Im Moment wird als Benutzeroberfläche JavaFX benutzt, welche mittels SceneBuilder einfach zusammengestellt werden kann und mittels meines NetworkParser schnell bidirektional mit dem Datenmodell verknüpft werden kann (siehe Abschnitt 5.12 auf Seite 120).

Für andere Systemarchitekturen unterstützt der NetworkParser die Darstellung der Benutzeroberfläche als HTML (siehe Abschnitt 5.9.2 auf Seite 95). HTML ist eine Beschreibungssprache, die seit Jahrzehnten existiert und von allen gängigen Plattformen unterstützt wird. Es ist somit für alle Szenarios in dieser Dissertation geeignet. HTML existiert für Android, JavaFX und Webserver. Da HTML eine Beschreibungssprache ist und interpretiert wird und nicht wie SWT eine native Implementierung benutzt wird ist HTML per Definition plattformunabhängig.

Bei der Implementierung von Programmen ist es enorm schwierig zu verstehen, wie man die richtige Programmlogik realisiert. Diese wird meist durch Interviews mit Domainexperten erfasst und durch erfahrene Entwickler umgesetzt. Es wäre jedoch wünschenswert, wenn dem Entwickler das Erlernen diese Tätigkeit vereinfacht würde, oder der Domainexperte dieses direkt umsetzen könnte.
Für die Businesslogik ist es relativ schwierig, Assistenten und Modell-Transformationen zur Verfügung zu stellen. Allerdings stellt das NetworkParser-Framework hier Pattern-Klassen zur Verfügung mit denen der Benutzer komplexe Modelltransformationen erstellen kann. Diese Modelltransformationen können mit Hilfe von logischen Ablaufstrukturen zu komplexen Algorithmen verknüpft werden.

Als Nächstes soll die Anwendung in der Lage sein, Ihre Daten zu persistieren und zu laden (siehe Abschnitt 4.6 auf Seite 46). Die notwendige Komponente fällt unter den Begriff Model-Management und kümmert sich um die Persistierung, das Erkennen von Änderungen, das Zusammenführen

von verschiedenen Modellen oder Änderungen und die Versionierung von Modellen.
Das Erste und die am meiste entwickelte Komponente meines NetworkParser beschäftigt sich mit dem Umwandeln von Objekt-Modellen in die verschiedensten Serialisierungsformate (siehe Abschnitt 5.13 auf Seite 121). Die Persistierung ist somit für die Studierenden eine Kleinigkeit und ist mittels zwei Zeilen zusätzlichen Programmcodes erledigt.

Die Serialisierungfunktionalität meines NetworkParser erlaubt auch sehr einfach ein Multi-User Support und für die Kommunikation von mehreren Anwendungen auf einem oder mehreren PC und mobilen Endgeräten. Dafür kommen weiterhin lediglich drei Zeilen zur Anwendung hinzu (siehe Abschnitt 5.15 auf Seite 128).

1.1.2 Anwendungseinsatz

Die unterschiedlichen System-Architekturen, wie Desktop-, Web- oder Mobil-Anwendung, kristallisieren sich in speziell angepasste Programmiersprachen und den daraus angepassten Programmcode. Bevor die Anwendung umgesetzt werden kann muss man sich genaue Gedanken darum machen auf welchem Betriebssystem diese laufen soll und welche physikalische Architektur vorausgesetzt werden muss. So ist eine einheitliche Benutzung von Programmierlogik in den seltensten Fällen möglich. Die Anwendungen sind meist eng mit der Benutzeroberfläche verknüpft oder mit speziellen Eigenheiten des Anwendungseinsatzes. So ist es sinnvoll für eine Desktopanwendung mit JavaFX auch deren Umsetzung des Datenmodells zu benutzen.
Bei der Entwicklung von Webanwendungen wird die Programmlogik für die Serverebene und für die Clientebene entwickelt. Bei der Entwicklung von mobilen Anwendungen wird die Funktionalität durch die komplett andere Architektur und Vorgaben des mobilen Betriebssystems auf die unterstützen Features eingeschränkt.

1.1.3 Plattformunabhängigkeit

Der übersetzte Programmcode der Oberfläche ist meist direkt für eine Plattform kompiliert oder dessen Runtimesystem greift individuell auf plattformabhängige Methoden zurück, sodass eine Anwendung auf den unterschiedlichen Plattformen immer anders aussieht, oder sogar die notwendigen Funktionalitäten gar nicht unterstützt. Es existieren auch Frameworks, welche die Benutzeroberfläche (Swing,SWT,JavaFX,Javascript) komplett mit ausliefern, jedoch sind diese dann fest mit der Anwendung verzahnt, sodass die Anwendung schnell starr für eine Plattform ausgeliefert wird, oder sehr groß ist, da Sie alle nativen Bibliotheken mitliefern muss. Mein NetworkParser integriert eine Zwischenschicht womit eine nachträgliche Änderung der Plattform Android, Desktopanwendung oder Webanwendung und der Wechsel des Betriebssystems unter der parallelen Weiterentwicklung ohne Probleme möglich ist (siehe Abschnitt 5.9.3 auf Seite 96).

1.1.4 Netztopologie

Die einfachste Anwendungsfall ist eine Anwendung ohne Netzwerkkommunikation. Wenn Verteilung benötigt wird ist der einfachste Fall die Client-Server-Architektur. Dieses System ausfallsicher zu gestalten, ist nur mittels eines Rechnerverbundes inklusive Lastverteilung auf der Serverseite zu realisieren. Komplizierter wird es ohne festen Server. Ein solches Peer-To-Peer oder Verteiltes System ist nur mit sehr hohem Aufwand zu realisieren oder bedeutet die Integration eines großen Frameworks mit allen Vor- und Nachteilen.

Ein solches Peer-To-Peer System ist das JMS [HBS+02]. Dieses trennt die Programmlogik und Transportschicht. Allerdings wird dieses anhand der Schichten und nicht anhand von Anwendungsfällen gemacht. Die Kooperation der Peers wird anhand von Nachrichten realisiert. Diese bestehen aus einem Nachrichtenkopf und einen Nachrichteninhalt. Der Nachrichtenkopf enthält sämtliche Metainformationen, für das Steuern von Nachrichten zu speziellen Clients (Message-Selector), Zieladresse,

Ablaufzeitpunkt, Priorität, um nur einige zu nennen. Der Programmierer muss die Struktur der Nachrichten genau wissen, um die Vorteile von JMS nutzen zu können. Außerdem wurde die Transportschicht nur mittels Socket-Verbindung hergestellt. Neuere Techniken sind nicht vorgesehen.
In der heutigen Zeit gibt es viele Kommunikationswege, um zusätzlich Smartphones über Pushnachrichten oder Endgeräte über Bluetooth oder Ähnliches anzusprechen.

Es gibt noch kein einheitliches Verfahren, um ganze Modelle bzw. Teilmodelle in einem Peer-to-Peer System zu synchronisieren. Mit dieser Dissertation ist es möglich mit wenig Aufwand eine Synchronisierung von Modellen in einer Peer-to-Peer Anwendung zu erreichen (siehe Abschnitt 5.15.4 auf Seite 137).

Es sollen außerdem Techniken entwickelt werden, um die kleinsten Differenzen von Modellen zu finden und um Konflikte bei der Synchronisierung von Datenmodellen bei nebenläufigen, verteilten Anwendungen aufzulösen. Hierfür sollen die Modellelemente anhand von Checkwerten oder mittels Identifikationskennziffer verglichen werden um neue, verschobene, geänderte oder gelöschte Elemente oder sogar Attribute zu ermitteln. Dieses ist notwendig um Konflikte automatisch aufzulösen, die entstehen, wenn Peers zeitweise offline sind, jedoch Änderungen an dem Datenmodell vornehmen. Im Fokus stehen dabei die Szenarios, dass mehrere Peers getrennt starten und erst im Nachhinein zusammen kommunizieren und dass es ein oder mehrere Peers geben kann, die zwischenzeitlich offline sind.

1.1.5 Lehre

An der Universität Kassel gibt es zahlreiche praxisorientierte Veranstaltungen. Viele dieser Veranstaltungen benutzen ein Tool oder Framework, welches speziell auf die Bedürfnisse der Veranstaltung angepasst ist, oder diese ideal unterstützen. SDMLib und NetworkParser werden effektiv seit 2012 in verschiedenen Lehrveranstaltungen eingesetzt. Pro Jahr kommen dort ca. 275 Studierende zusammen, die den NetworkParser benutzen.

- ProgrammierMethodik ca. 120
- Software Engineering 1 ca. 80
- Design-Pattern ca. 20
- Graphentechnik ca. 20
- Projekte ca. 20
- Abschlussarbeiten ca. 5
- interne Projekte 10

Somit wurde SDMLib und NetworkParser bislang von ca. 2000 Studierenden ein oder mehrmals benutzt.

1.2 Motivation für meiner Lösung

Die Dissertation soll ein vollständiges Lösungspaket für die Entwicklung von Applikationen in mobilen und Desktopanwendungen sowie für die Entwicklung von einfachen Programmen für einen Rechner sowie für die Entwicklung von komplexen Systemen in verteilten Anwendungsfällen bieten. Weiterhin soll die Wiederverwendung von Programmlogik und Benutzeroberfläche vereinfacht und standardisiert werden. Im Zuge der Dissertation soll eine Abstraktionsschicht zwischen dem eigentlichen kompilierten Programm und der objektorientierten Modellierungsebene

entwickelt werden, sodass Programmieranfänger mittels grafischer Editoren Programmkomponenten auf der Modellebene entwerfen können. Es sollen dabei bereits etablierte und standardisierte Formate genutzt und besser miteinander verwoben werden.
Die so entstehenden Applikationen können mit verschiedenen Kollaborationsansätzen zusammenarbeiten. Der erste Ansatz ist, dass alle Netzwerkknoten gleichzeitig zur Laufzeit verfügbar sind, sodass das Peer-To-Peer System Probleme direkt bearbeiten kann und zum anderen versionsbasierend, welches den Vorteil hat, dass einzelne Peers zeitweise offline arbeiten können.

Hieraus resultieren sich folgende Fragen.

- Wie kann ich Anfängern mittels stabiler Programmier- und Modellierungshilfen die Software Entwicklung beibringen?
- Wie kann ich den kompletten SDM Prozess toolmäßig unterstützen?
- Welche standardmäßigen Techniken können verbessert und kombiniert werden?

2 Techniken

Für das gute Zusammenspiel der verschiedenen Problemlösungen ist es notwendig verschiedene Programmiersprachen und Frameworks miteinander zu verbinden, um die besten Eigenschaften nutzen zu können. Ziel ist dabei möglichst für alle Anwendungsfälle kompatibel zu sein und dennoch nur die notwendigsten Komponenten miteinander zu verbinden.

2.1 Programmiersprachen

Es wurden mehrere Programmiersprachen benutzt oder eine Kompatibilität hergestellt. Die funktionale Sprache Javascript und objektorientierte Sprache Java wurden integriert. Bei den funktionalen Sprachen sind alle Anweisungen gleichberechtigt und ein Programm ist eine Sammlung von Funktionen. Dadurch ist es möglich mit wenigen Programmcodezeilen komplexe und schnelle Algorithmen zu erstellen. Allerdings sind die funktionalen Programmiersprachen nicht gut für Anfänger geeignet. Die objektorientierten Programmiersprachen (OOP) probieren die Wirklichkeit abzubilden und zeichnen sich durch die Kapselung der Daten aus. Diese Vorgehensweise ist ideal für die Wiederverwendbarkeit von Programmcode und die statische Unterstützung von IDEs.

Java [Gos00]
Für die Verteilung und Bereitstellung von externen Ressourcen wurde die objektorientierte Programmiersprache Java benutzt. Die Programmiersprache Java ist eine plattformunabhängige Sprache, welche den Programmcode mittels eines Compilers in Bytecode übersetzt und in dessen JVM ausführt. Java wurde von der Firma Sun entwickelt und ist zu großen Teilen seit dem 8. Mai 2007 unter der GPL Version 2 [Fou] frei verfügbar.

Scala [LBB+16]
Scala ist eine rein objektorientierte Programmiersprache und ist mit Java kompatibel. Scala wird als Beschreibungssprache für die Gradle Tasks in dem Continuous Integration Prozess verwendet.

Javascript [Fla06] [ECM11]
Javascript ist eine Interpretersprache. Es wurde ursprünglich für dynamische Webanwendungen entwickelt, als Erweiterung von HTML und CSS benutzt. Aus diesem Schattendasein ist Javascript spätestens seit GWT [Dew07] getreten. Mittlerweile werden ganze Webseiten und Anwendungen mittels Javascript realisiert. Sogar ganze Server Anwendungen wie NodeJS [TV10] gibt es. Javascript wird in dieser Dissertation für die grafische Darstellung der Benutzeroberfläche, als dynamische Schicht der Businesslogik und als Debug-Oberfläche für die Entwicklung benutzt. Aktuell ist die Version 5. Javascript als Websprache entwickelt sich mittels Typescript zum Nachfolger von den der objektorientierten Sprache Java. Java hat gerade mit der Aufgabe von JavaFX die wichtigste Möglichkeit der Darstellung verloren.

Typescript [Wit12]
Microsoft hat Typescript entwickelt, welches eine eigene Sprache mit der Funktionserweiterung von Javascript darstellt. Sie stellt die in ECMA V6 [Int16] definierten Klassen zur Verfügung. Weiterhin wird der Programmcode in Module unterteilt, vergleichbar mit den Java-Packages. Es führt ein Typensystem ein, welches es erlaubt schon zur Entwicklungszeit Fehler im Programmcode zu finden. Typescript wird als Zwischensprache für das Datenmodell und die Programmlogik benutzt.

2.2 Frameworks, Dienste und Tools

In diesen Abschnitt wird auf die verwendeten Frameworks, Dienste und Tools eingegangen. Für das ehrgeizige Ziel, möglichst ein Framework ohne Grenzen zu realisieren und trotzdem den Entwicklungsaufwand im Rahmen zu halten, wurde auf bestehende Software zurückgegriffen.

IKVM [SE04]

IKVM ist ein Crosscompiler, welches es ermöglicht Java Programmcode in .Net Dll übersetzen kann. Aktuell ist die Version IKVM.NET 8.1. Es unterstützt den kompletten Umfang der Java 1.6 Spezifikation.

ANT [Edl02]

ANT (Another Neat Tool) ist ein in Java geschriebenes Framework, welches für das Erzeugen von Programmen genutzt werden kann. Es wird mittels einer XML-basierenden DSL realisiert. Durch den einfachen Aufbau und die Mächtigkeit ohne große Abhängigkeiten wurden mehrere Build-Scripte zum Überprüfen der Kompatibilität vom NetworkParser zu IKVM, GWT und Android realisiert. Aktuell ist die Version 1.10.5, es ist ein Top-Level Projekt der "Apache Software Foundation".

Templates

In der Datenverarbeitung findet man oft Templates, welche als Vorlage für die Generierung von Programmcode oder automatischer Ausführung von Programmcode dienen. Bei den Templates gibt es unzählige verschiedene in unterschiedlichen Ausprägungen und Komplexitäten. Fujaba nutzte für die Generierung "Velocity" [Vel18]. UML Lab nutzt die Templatesprache "xpand" [Skr18] in Kombination mit "Xtend" [ES18]. Monticore benutzt Apache FreeMarker.Diese waren nach Recherche allerdings nicht geeignet für den NetworkParser. Somit wurde eine eigene Template-Sprache angelehnt an HandleBars [Kat18] umgesetzt.

Android [Lin12]

Android ist in kürzester Zeit zu einem der beliebtesten Betriebssysteme für Smartphones und Tablets aufgestiegen. Das Android-Betriebssystem ist ein quelloffenes System, welches von der "Open Handset Alliance" entwickelt wird. Standardmäßig werden Anwendungen für Android unter Java geschrieben und mittels Dalvik-VM übersetzt und ausgeführt. [LG10, LG11, LG12, Joe17, RvdM18, LG18a, LG18b]

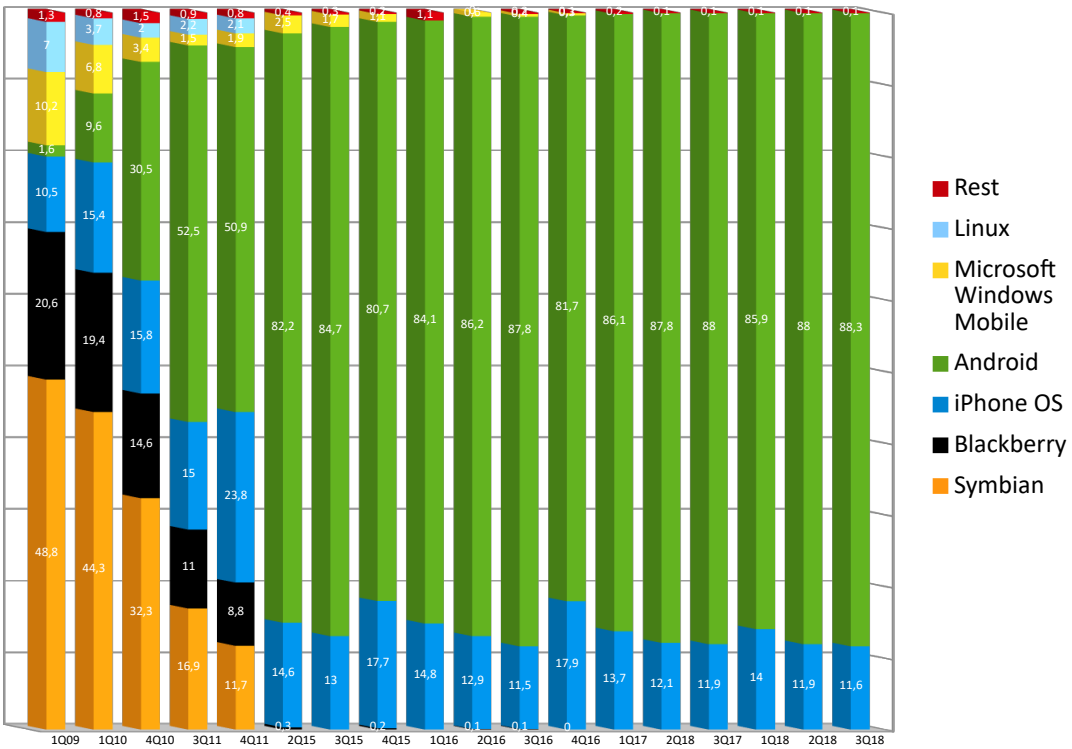

Abbildung 1: Gartner Statistik

Gource [Cau18]
Gource ist ein in C geschriebenes Tool zur Visualisierung von GIT Änderungen. Es dient zur einfachen Übersicht der kompletten Änderungen an einem Projekt. Ein Beispiel ist in der Abbildung 2 zu sehen [You18].

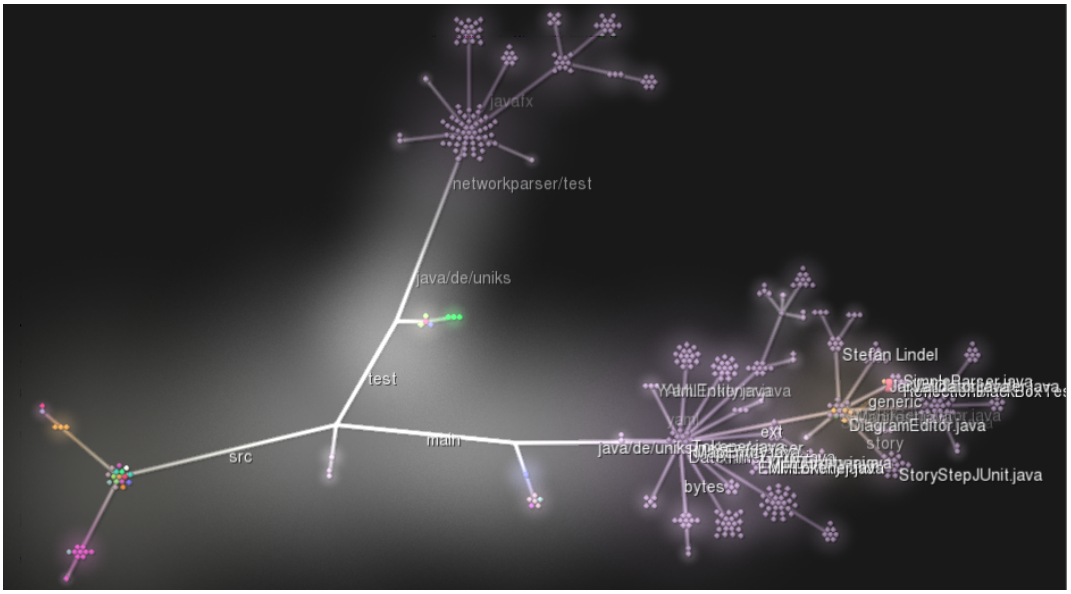

Abbildung 2: Gource NetworkParser 05.07.2018

Travis
Travis ist ein Continuous Integration Service, welcher es ermöglicht automatisch aufgrund von GIT-Veränderungen Tests auszuführen und neue Builds zu starten. Der Dienst bietet weiterhin die Möglichkeit nachgelagerte Dienste anzusteuern [Han16]. Die aktuellen Builds vom NetworkParser wurden mit Travis erstellt und in Sonatype veröffentlicht. Dieses geschieht für den Master-Zweig als Release und für den Develop-Zweig als Snapshot.

Weiterhin wird das Typescript Projekt DiagramJS [Lin18a] mit dem Travis gebaut. Dieses Projekt wird mittels NodeJS und NPM übersetzt. Nach erfolgreichem Übersetzen werden die Ergebnisse nach GitHub als Release gepushed und die Seite von GitHub [Lin18b] wird aktualisiert. Weiterhin wird für ein neues Release die Version auf NPMJS [npm18] veröffentlicht.

Jenkins [Blu18]
Jenkins ist ein Continuous Integration Service, den es sowohl als Onlinedienst, als auch als selfhosted Webanwendung gibt. Er ist ein Opensource Webserver, welcher in Java geschrieben ist. Der NetworkParser ist kompatibel zu dem Buildprozess im Jenkins und wurde früher auf dem Build-Server des Fachgebiets gebaut.

Gradle [Mus14]
Als weiteres Build-Tool wurde Gradle hinzugefügt. Es ist ein auf Java basierendes Tool, welches von diverses Continuous Integration Tools unterstützt wird. Es gibt acht wichtige Build Tools [Sch18, Moh18, WWR+18, Dü18]. Welches Tool verwendet wird ist eine persönliche Entscheidung (z.B. Maven, Ant, Gradle). Meine war Gradle, da es eine bessere Performance und klarerer Struktur für den Build-Prozess aufweist [Wen18].

SVN
Apache Subversion (SVN) existiert seit dem 5.Juni 2000. Es ist ein zentrales Versionsverwaltungssystem. Bei großen Binärdateien ist SVN besser als GIT, da auf den Clients nur die aktuelle Version vorhanden ist. Es ist allerdings immer notwendig eine Verbindung zu den zentralen Repository zu haben, da nur dort die Änderungen gespeichert werden [CSFP06]. Ursprünglich wurde der NetworkParser unter der Kontrolle von SVN entwickelt. Die Entwicklung an dem NetworkParser wurde am 21.12.2011 begonnen und auf dem fachgebietseigenen SVN-Server versioniert.

GIT [CS14]
GIT ist ein quelloffenes verteiltes Versionsverwaltungssystem, um textuelle Dateien zu verwalten. Es dient dazu, Änderungen transparent zu überwachen. Mit Hilfe von mehreren Branches ist GIT sehr gut als Hilfsmittel für die Projektverwaltung geeignet. Es ist selbst ein Peer2Peer System, welches lokal immer ein vollständiges Abbild des Repository darstellt.

GitHub [CW18], [AV18]
GitHub ist ein Onlinedienst, welcher als Filehoster von GIT-Repositories dient. Es bietet zusätzlich eine Web-Oberfläche an um projektbezogene Inhalte anzuzeigen und einfaches Projektmanagement zu realisieren. Hierzu zählen unter anderen Issues. Das Repository vom NetworkParser wird momentan unter anderem auf GitHub bereitgestellt [Zü15, Lin18e]. Auf GitHub findet man viele bekannte OpenSource-Projekte, und somit reiht sich der NetworkParser gut als OpenSource Projekt dort ein.

Gitlab
Gitlab und GitHub sind vom Funktionsumfang sehr ähnlich [Hof17]. Allerdings bietet Gitlab zusätzlich die Möglichkeit Builds mittels Continuous Integration und Deployment Pipelines zu realisieren. Der NetworkParser ist auch hierzu kompatibel, sodass der NetworkParser direkt in Gitlab verwaltet und neue Versionen erstellt werden können. Der NetworkParser wird auch auf Gitlab bereitgestellt [Lin18f]. Das Repository von dem OpenSource Typescript-Projekt DiagramJS befindet sich ebenfalls dort. Dieses wird in dieser Dissertation für die Darstellung von Benutzeroberflächen und Diagrammen und als Diagrammeditor zum Beispiel für eDobs benutzt.

Sonatype (Maven) [DAT17]
Dies ist ein Filehoster, welcher Maven Repositories hostet. Dort werden die Snapshots und Release Zweige von NetworkParser und SDMLib veröffentlicht [son19].

NPM [npm18], [IZSn18]
NPM ist ein offener Packetmanger für Javascript-Frameworks. Das Projekt DiagrammJS, welches für Benutzeroberflächen der Endanwendung und die Diagramme im NetworkParser benötigt wird mittels WebPack gebaut und in der NPM Registry hochgeladen [Lin19a].

WebPack [TKc18]
Webpack ist ein Open-Source-JavaScript-Modul-Bundler. Sein Hauptzweck ist es, JavaScript-Dateien für die Verwendung in einem Browser zu bündeln. Es ist auch in der Lage, nahezu jede Ressource oder Anlage zu transformieren, zu bündeln oder zu verpacken. Webpack nimmt Module mit Abhängigkeiten und generiert statische Assets, die diese Module repräsentieren. Es nimmt die Abhängigkeiten und erzeugt einen Abhängigkeitsgraphen, der es Webentwicklern ermöglicht einen modularen Ansatz für ihre Webanwendungsentwicklung zu verwenden. Der Bundler kann über die Befehlszeile oder über eine Konfigurationsdatei mit dem Namen webpack.config.js verwendet werden. Node.js wird für die Installation von Webpack benötigt.

https://coveralls.io
Dies ist ein freier Online Dienst zur Visualisierung und Speicherung von der Testabdeckung für die verschiedenen Versionen einer Bibliothek. Der Dienst bietet ein Badgesystem an, sodass der NetworkParser die Änderungen inklusive Historie auf GitHub anzeigt. Es zeigt sehr gut einen Verlauf der Codeabdeckung vom NetworkParser seit dem 18.12.2015. Dieser ist von 44 % auf 71 % gestiegen.

SonarQube [Son18b, Son18a]
SonarQube kann mittels statischer Code-Analyse das Projekt NetworkParser analysieren. Er trägt somit zur Qualitätssicherung bei.

scan.coverity.com
Coverity ist einen kostenloser Onlinedienst, welcher
statische Codeanalysen durchführt und die Ergebnisse mittels eines Issue Board dargestellt. Diese können auf der Webseite eingesehen werden. Mit Coverity ist es möglich Fehlerquellen zu finden und zu eliminieren. Coverity wurde im NetworkParser zur Qualitätssicherung verwendet und somit wurden bis zum 1.11.2017 298 Fehler behoben.

bestpractices.coreinfrastructure.org
Dieser Online-Dienst stellt eine standardisierte Plattform bereit um Bibliotheken zu kategorisieren. Auf dieser Webseite kann man einen Fragebogen ausfüllen, um sicherzustellen, dass das Projekt wie der NetworkParser nach dem "Best Practice" (optimale bzw. vorbildliche Methoden, Praktiken oder Vorgehensweisen) entwickelt wurde. Somit kann ein Entwickler auf einen Blick sehen, welche Wertigkeit das entsprechende Projekt hat.

https://www.openhub.net/p/NetworkParser/
OpenHub dient als Übersichtsseite zu verschiedenen Opensource-Projekten. Es stellt eine Übersicht zu den einzelnen Projekten im Web bereit. Hierbei werden Mitarbeit, Programmiersprachen und eine Codeanalyse grafisch dargestellt. OpenHub hat bereits über 474.291 Projekte indiziert. Die Seite dient damit als ein Einstiegspunkt für die Suche nach Frameworks.

JavaParser [vB18]
Der JavaParser ist ein Sourcecode-Parser, welcher Quellcode in einen abstrakten Syntaxbaum überführt. Der JavaParser soll als Alternative zu dem Tokenparser, welche anhand der Vorlesung "Compilerbau" entstanden ist, eingesetzt werden. Dieser soll dann ein Generieren von Modellen inklusive Refaktoring dienen.

Jacoco [Gmb18]
Jacoco ist ein Framework, welches Tests mittels der EMMA Bibliothek ausführt. Die so ausgeführten Tests werden durch einen speziellen ClassLoader analysiert und der ausgeführte Programmcode als HTML-Dateien visualisiert. Der so entstandene Report dient beim NetworkParser als CodeCoverage und wird mittels Travis und Gradle ausgeführt.

Findbugs [PL18]
Findbugs ist ein freies Framework, welches von "University of Maryland" entwickelt wurde. Es dient zur statischen Suche von Fehlern im Programmcode. Es analysiert dabei den Java-Bytecode mit statischer Codeanalysen nach bekanntem Fehlermuster. Mit diesem Tool wurden über 300 Fehler lokalisiert und behoben.

Checkstyle [Iva18]
Checkstyle wurde für die statische Codeanalyse entwickelt. Es kann direkt als Gradle Task aufgerufen werden und analysiert anhand des Apache Codestyle den Programmcode. Aktuell ist die Version 8.10.1. Dieses Tool wurde als Gradle Task eingebunden und trägt zur besseren Codelesbarkeit bei. Im Moment bestehen die 28633 Anmerkungen hauptsächlich aus:

- Javadoc-Kommentar fehlt: 6160
- Zeile länger als 80 Zeichen: 7492

SpotBugs [Col18]
SpotBugs ist der Nachfolger von FindBugs und dient auch zur Sicherung der Codequalität. Aktuell ist die Version 3.1.6. Momentan kann man den Report von SpotBugs vom NetworkParser über das Gradlescript generieren.
Aktueller Stand 4.10.2018: 46015 lines of code analyzed, in 462 classes, in 33 Packages.

Smack [Tuc18]
Smack ist ein Opensource Projekt, welches in Java geschrieben wurde. Es stellt die Kommunikation für den Instant-Messenger wie Jabber (XMPP) bereit. Angedacht ist das Framework über die NodeProxies einzubinden, um alle Funktionen von XMPP zu unterstützen. Momentan hat der NetworkParser eine abgespeckte Variante integriert.

Google FireBase [Inc18a]
Google FireBase ist eine Entwicklungsplattform für mobile und Webanwendungen. Sie stellt einige Dienste zur Verfügung wie das Cloud-Messaging. Weiterhin kann FireBase auch als Datei-Speicher dienen. Sie wird als Schnittstelle für das Verteilte-System vom NetworkParser eingesetzt. Es existiert ein spezieller Proxy, mit dessen Hilfe, Nachrichten per Cloud an Smartphones gesendet werden können.

NodeJs [Fou18], [Mar14]
NodeJs ist ein Programm, welches für Serveranwendungen konzipiert wurde. Es wird dabei in der Javascript-Laufzeitumgebung V8 ausgeführt. NodeJs unterstützt dabei Anwendungen, die in Javascript geschrieben sind. Es ist auf Performance ausgelegt. Durch die Benutzung von Javascript benötigt die Serveranwendung sehr wenig Arbeitsspeicher. NodeJS wird mit NPM für das Einbinden der Javascript-Bibliothek Dagre und DiagramJS während des Buildprozesses benutzt.

3 Ziele

Das Ziel dieser Dissertation ist es ein einfaches einheitliches Framework zur Verfügung zu stellen, welches dem Entwickler in jedem Schritt bei der Entwicklung einer objektorientierten Software zur Realisierung den NetworkParser bereitstellt. Eine objektorientierte Anwendung besteht typischerweise aus mehreren Komponenten: Einer Benutzeroberfläche, einem Datenmodell, eine Persistenzkomponente, einer Businesskomponente und einer Netzwerkkomponente.

Für die Entwicklung einer objektorientierten Anwendung sollte ein standardisiertes Vorgehensmodell verwendet werden. Dieses beginnt mit dem Aufschreiben der Anforderungen über den Entwurf des Datenmodells und der Implementierung der Businesskomponente und der Benutzeroberfläche, bis zu der Überprüfung und Dokumentation der Software mit Tests und Wartung mittels Fehlerprotokollen.

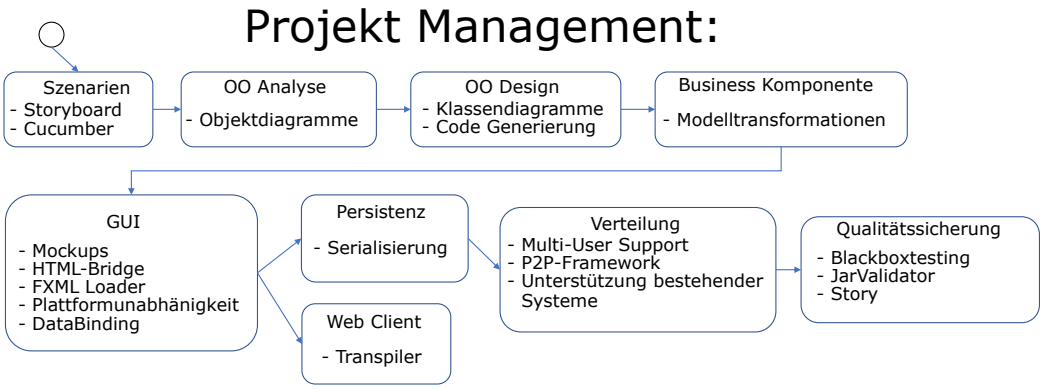

Abbildung 3: Ablaufdiagramm ProjektManagement

Es gibt mittlerweile unzählige Vorgehensmodelle in der Softwareentwicklung [Ack18, Sie18, Dil18]. Eine allgemeine Verteilung der benutzten modernen Verfahren ist nachfolgend aufgeführt:

Vorgehensmodell	**Anteil**
Scrum	26,9 %
V-Modell XT	22,6 %
Kanban	9,7 %
Extreme Programming	9,7 %
RUP	6,5 %
V-Modell 97	5,4 %
Rest	17 %
(kein)	2,2 %

Bei der Lehrveranstaltung "Programmiermethodik" wird "Story Driven Modelling" (SDM) eingesetzt. Es gehört zu den agilen Vorgehensmodellen. Dort wird empfohlen bei der Entwicklung mit dem Kunden mehrere Szenarien mit verschiedenen Stories zu entwickeln. [kla18]

Aus jeder Story wird dann ein passendes Objektdiagramm abgeleitet und aus diesem wird dann das Klassendiagramm [ZGK15] überführt. Der NetworkParser unterstützt mit seinem Storyboard den Entwickler bei der Beschreibung von Szenarien. Die Szenarien können in HTML basierte Dokumente generiert werden. Der NetworkParser erlaubt auch die Modellierung von Objektdiagrammen, die dann in ein Storyboard beziehungsweise die generierte Dokumentation grafisch eingebunden werden. Genauso unterstützt der NetworkParser die Modellierung von Klassendiagrammen. Klassendiagramme können grafisch in eine Storyboard-Dokumentation eingebunden werden. Zusätzlich bietet der NetworkParser einen CodeGenerator an, der aus einem Klassendiagramm die entsprechende Java / Typescript oder C++ Implementierungen des objektorientierten Modells erzeugt.

Die Anforderungen an die Benutzeroberfläche sind sehr unterschiedlich. Hier hat es sich bewährt dem Kunden ein Mockup zu geben oder ein GUI Dummy, der die groben Strukturen der Benutzeroberfläche beinhaltet. Der NetworkParser bietet die Basisfunktionalität, um ein Mockupdesign zu erstellen. Bei diesem ist es enorm wichtig, dass der Kunde merkt, dass es sich nicht um die endgültige Benutzeroberfläche handelt und dass seine Wünsche und Änderungen noch integriert werden können (sketch layout). Der NetworkParser stellt einige allgemeine GUI-Komponenten für die Benutzeroberfläche und für Use-Case Darstellung bereit. Weiterhin wäre eine Erweiterung für Mockup denkbar und einfach umsetzbar. Für die Phase der Entwicklung können noch weitere Komponenten mit HTML-Standardtechniken erstellt und integriert werden.

Bei den nächsten Schritten der Softwareentwicklung, bei denen es um die Businesskomponente geht, stellt der NetworkParser eine Reihe von Pattern-Matching und Modelltransformationen bereit.

Bei den letzten Schritten der Implementierung kümmert sich der Entwickler um Persistenz und Verteilung. Für die Persistierung stellt der NetworkParser eine automatische Serialisierung zur Verfügung. Für die Verteilung bietet der NetworkParser zusätzliche Komponenten. Der NetworkParser unterstützt hierbei zahlreiche Kommunikationswege und ist gut für mehrbenutzerfähige objektorientierte Anwendungen geeignet.
Mit der Erweiterung in Abbildung 4 ist es möglich die Applikationsdaten zu persistieren oder mit verschiedenen Knoten im Netzwerk zu teilen. Es wurde darauf geachtet, dass jede Komponente auch einzeln funktioniert und möglichst intuitiv bedienbar und erweiterbar ist.

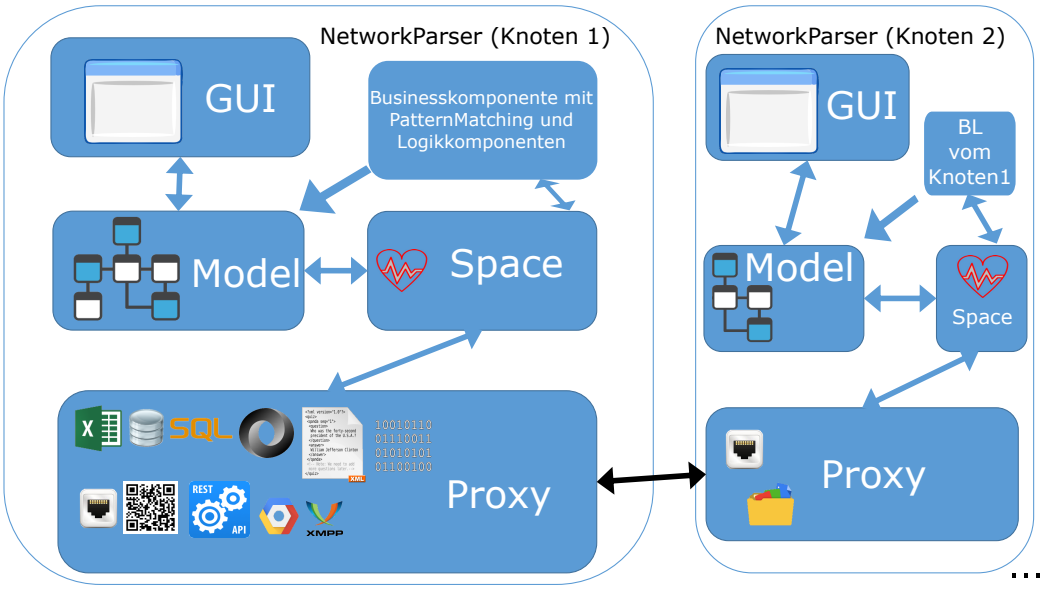

Abbildung 4: Struktur

3.1 Lehre

Die Dissertation hat das Ziel ein Software-Framework zu stellen, welches auch gut als Lehrmittel verwendet werden kann. Die Zielgruppe besteht aus Schülern der Oberstufe, Berufsschule bzw. Hochschule. Jede Vorlesung benutzt unterschiedliche Frameworks, die nicht immer exakt zum Lehrstoff passen. Die Studierenden lernen dadurch zwar ein spezielles Produkt kennen, jedoch ist es schwierig ein passendes, lizenzfreies Produkt für die speziellen Vorlesungen zu finden. Dieses erschwert die Vermittlung des Lehrstoffes und führt zu einem Mehraufwand für die Betreuung seitens der Tutoren und des Lehrpersonals. Bei der Lehre sollte der Fokus auf dem Lehrstoff liegen, da das Tool austauschbar sein sollte. Die Vergänglichkeit von Tools sieht man gerade im Webumfeld, wo ein Framework sich im Durchschnitt nur wenige Jahre hält [Mos17a].
Der Funktionsumfang des NetworkParser umfasst dabei das Erlernen und Vertiefen einer objektorientierten Programmiersprache, wie zum Beispiel Java.

Es soll weiterhin möglich sein, komplexe Systeme damit einfach zu vernetzen und zu testen. Damit können gerade einfache Programme realisiert werden. Als Zielplattform stehen Smartphone, Tablet, Desktop und Webanwendungen im Fokus.
Weiterhin soll eine Plattform entwickelt werden, mit deren Hilfe ein Projektmanagement realisiert werden kann, sodass es möglich ist agile Softwareentwicklung für kleine und mittlere Projekte im Umfang von zwei bis acht Personen zu planen und zu realisieren.
IKVM.Net und dynamische Frameworks wie GWT unterstützen Java in der Version 1.6 (Außer Reflection). Android unterstützt die Version 1.6 seit 2013 (Kitkat) die Version 1.7 und OpenJDK (Java) seit "Nougat" (Android 7.0) in der Version 1.8. Damit das Framework für alle denkbaren Szenarien eingesetzt werden kann, habe ich mich entschieden, dass der NetworkParser zu den Spezifikationen von Java 1.6 kompatibel sein soll.

Ziele der Promotion für die Lehre

- 1. Semester: Einführung in die Programmierung
  - Programmeditor wie BlueJ oder Greenfoot
  - Modellierungswerkzeug für Lehre zum Erlernen von objektorientierter Modellierung und Programmierung
- 3. Semester: Programmiermethodik
  - Bereitstellung eines Frameworks für agile Softwareentwicklung mittels textueller Szenarien und ausführbaren Teststories.
  - Einfache Erstellung von Benutzeroberflächen und vieler Diagrammtypen und Diagrammformate
  - Möglichkeit der einfachen Persistenz und Synchronisierung verschiedener Programmzuständen
  - Unterstützung der Hausaufgabenkorrektur für die Tutoren
- 4. Semester: Software Engineering I
  - Framework zum einfachen Erstellen von Desktop-, Web- und Smartphone-Programmen und deren einfache Vernetzung.
- 5-10 Semester: Projekte im Fachgebiet
  - Unterstützung für mehrsprachige Programme (i18n)
  - Vereinfachte Implementierung von Logmechanismen, Multisprachunterstützung und ErrorHandling
  - Sourcecodetranspiler für das Übersetzen von Programmcode von Java nach Javascript
  - Mögliche Basis für agile Softwaremanagementverwaltungssoftware. Dieses ist denkbar realisierbar mittels Versionsverwaltungssoftware wie GIT
  - Möglichst kompatibel zu den bestehenden Mechanismen wie SQL, Rest und EMF
  - Implementierung einer sehr übersichtlichen und performanten Value Liste oder Key-Value-Key Liste

und andere Anwendungsfälle aus der Wirtschaft

- Kernkomponenten von der Arztkongressmanagement-Software „ConfNet“
- Planung einer Verteilten Anwendung für die Niedersächsische Jugendfeuerwehr mit Hilfe der Kernkomponente. Kostenschätzung laut NJF 150 – 250k €
- Jahresberichteditor für die Niedersächsische Jugendfeuerwehr (Einfache Erweiterung von excelbasierten Dateien)

Bei der Entwicklung war die Einbeziehung der Studierenden (siehe Abbildung 47 auf Seite 153) sehr entscheidend. Bei mehreren Programmiermethodik-Vorlesungen wurde das Framework immer mehr optimiert und so auf die Anforderungen der Studierenden und der Tutoren angepasst. Das Diagramm zeigt die Auswertung aus der Vorlesung ”Programmiermethodik” aus dem Jahr 2013/2014 [Lin17a] von 87 Studierenden, dem Jahr 2014/2015 [Lin17b] von 28 Studierenden, dem Jahr 2016/2017 [Lin17c] von 12 Studierenden an und dem Jahr 2018/2019 [Lin18h] von 99 Studierenden.

3.2 Anwendungsszenario ConfNet

Wir haben einen Auftrag der Firma Röllmedia GmbH [Rö18] bekommen, die ein ausfallsicheres verteiltes System zur Verarbeitung und Verteilung von Präsentationen auf verschiedene Räume für Ärztekongresse benötigte. Weiterhin sollte eine intuitive Benutzeroberfläche für den Aufruf von Präsentationen geschrieben werden, welche durch Ärzte ohne vorhergehende Schulung selbst bedient werden, konnte. Das hierfür von mir maßgeblich entwickelte System heißt ConfNet. Die Firma setzt seit 2011 ConfNet erfolgreich als Kernsoftware auf Kongressen ($\sim$ 1500 Vorträge) und Workshops ($\sim$ 300 Vorträge) ein. ConfNet verwendet bereits wesentliche Komponenten des NetworkParsers, wie zum Beispiel die Serialisierung und die Verteilungs-Mechanismen. Auch die NetworkParser-Anteile für die Web-Anwendungen und Javascript Technologien sind in ConfNet eingeflossen. Zum Teil werden ConfNet und NetworkParser parallel entwickelt.

ConfNet ist allerdings in die Jahre gekommen und es gibt über 40 neue Anforderungen, welche implementiert werden sollten. Daher soll das bestehende System auf die neueren Entwicklungen dieser Dissertation migriert werden. Durch die Techniken dieser Dissertation wäre es möglich eine nachhaltige Software zu realisieren, welche mittels einer zeitlich begrenzten Lizenz kommerzialisiert werden kann.

Abbildung 5: ConfNet

Abbildung 6: Annahme

Abbildung 7: Room

3.3 Anwendungsszenario Niedersächsische Jugendfeuerwehr e. V.

Da die Komponenten der Dissertation sehr flexibel und universal einsetzbar sind, wurde bereits 2014 nach neuen Anwendungsgebieten gesucht. Es gab ein Gespräch mit dem Verein der Niedersächsischen Jugendfeuerwehr, wo eine Idee einer neuen Software vorgeschlagen wurde. In einem Protokoll vom 10.02.2014 wurde die Idee für eine Software für die einfache Erfassung der Jahresberichte und in der maximalen Ausbaustufe der Software für ein Dienstbuch für Kinder- und Jugendfeuerwehren mit einem Warenwert von 150k – 250k Euro geschätzt. Allerdings kam eine Auftragserteilung unter der damaligen Führung nicht zustande. Dieses müsste durch wiederholte Vorstellung bei der neuen Führung der NJF fokussiert werden.

3.4 Anwendungsszenario Programmiermethodik

Das Framework wird jetzt schon seit 2012 für die Lehrveranstaltung "Programmiermethodik" eingesetzt. Das Framework ist ideal an die Lehrveranstaltung angepasst, so sind textuelle Szenarios, Darstellungen von Objekt- und Klassendiagrammen und das Erstellen von Programmcodes darin enthalten. Es wurden weiterhin einige Adapterklassen geschrieben, um einen einfachen Einstieg in JavaFX und das Databinding zu realisieren.

Weiterhin wurde für die Tutoren einige Hilfen im NetworkParser eingebaut, welche für die Lehrveranstaltung "Programmiermethodik" und für "Software Engineering I" genutzt wurden.
In Programmiermethodik:

- In der 3. Hausaufgabe sollen die Studierenden ein Klassendiagramm erstellen:
  Die Studierende erstellen dieses mit dem NetworkParser textuell. Die Korrekteure können das Klassendiagramm einlesen und sortiert wieder speichern. Danach ist der Korrektor in der Lage das Klassendiagramm des Studierenden ganz einfach mittels eines Textvergleichstool mit der Musterlösung zu vergleichen.

- In der 4. Hausaufgabe, soll die Spielstartsituation erstellt werden. Die Ausgangssituation ist durch die Regeln definiert. Hier wurde ein Objektinstance-Differ eingebaut, der in einem Objektdiagramm die Objektinstanzen und Links mit der Häufigkeit anzeigt. Mittels PatternMatching können alle Objekte einfach abgefragt werden und mittels JUnit-Tests abgeprüft werden.

- In der 6. Hausaufgabe, beim Verknüpfen der Benutzeroberfläche mit dem Programmcode besteht die Möglichkeit über die globalen Felder zum Beispiel "Color" oder "Name" für alle Instanzen einen PropertyChangeListener zu setzten. So kann einfach kontrolliert werden, ob die zu implementierenden Controller richtig an allen Instanzen dem Datenmodell angemeldet sind.

In "Software Engineering 1" wurde zusätzlich der JarValidator im Bamboo verwendet, mit dessen Hilfe Codecoverage Tests ausgeführt wurden und die Abgabe der Studierenden auf Vollständigkeit und Zuverlässigkeit zu überprüfen. Zusätzlich wurde eine Liste der verwendeten Lizenzen angezeigt.

3.5 SynergieBox 2014

Loewe Projekt zur Entwicklung einer Soft- und Hardware-Lösung für ein adaptives Energiemanagement für Wohngebäude. Der NetworkParser wurde in diesem Projekt für die Serialisierung und Umformung der Daten der einzelnen Homeautomationsdienste benutzt. Die Dienste stellen unterschiedliche Daten im binären Format zur Verfügung, welcher der NetworkParser in ein Json-Format überführte.

3.6 EON ShiftScheduling 2013

Das industriefinanzierte Projekt dient der Vereinfachung der Planung und der Organisation des Rufbereitschaftsdienstes für Störungen im Gas- und Stromnetz. Es optimiert die Mitarbeiteranzahl zur Erfüllung der Rufbereitschaftsaufgaben. Der NetworkParser war für die Persistierung und Bereitstellung von effizienter Collection für die Berechnung der Rufbereitschaftsteams zuständig. Ein Energiekonzern hat gesetzlich die Pflicht eine Rufbereitschaft zu unterhalten, welche unter normalen Verkehrsverhältnissen einen Einsatzort innerhalb von 15 Minuten erreichen kann.

3.7 Loewe Projekt: eoda tableR 2014

Die Bearbeitung und Analyse von Erhebungsdaten gehört zu den ältesten Problemen, die mit Hilfe von Computern gelöst werden. Im Wesentlichen sind dies die Erstellung eines Fragebogens, die Eingabe der Daten in eine Matrix, die Erstellung von Tabellen, die Erzeugung von Grafiken und Diagrammen sowie schließlich die multivariate Datenanalyse. Die Software tableR versucht den Analyseprozess neu zu denken. Statt den Prozess mit einem Bündel spezialisierter Software für Teilprozesse anzugehen, wird tableR den Gesamtprozess ganzheitlich lösen. Kernelement der Software ist eine XML-Definition, mit deren Hilfe Daten von einem Aggregatzustand in einen anderen überführt werden können. Der NetworkParser wurde in tableR für die an Excel angelehnte Benutzeroberfläche und die Persistierung eingesetzt.

Abbildung 8: Use-Cases

4 Stand der Technik - Verwandte Arbeiten

Es gibt eine Reihe von Tools, die bereits die verschiedenen Teilfunktionalitäten wie die Persistenz und die Verteilung von Nachrichten, das Umwandeln von Programmcodes in andere Sprachen und die verschiedenen Implementierungen von Listen, Mengen und Key-Value-Listen realisieren. Es gibt unzählige Benutzeroberflächenframeworks, die jeweils immer alles besser machen wollen als ihre Vorgänger [Tru17]. Leider existiert noch kein Framework, welches komplett für alle Bedürfnisse geeignet ist.

4.1 Lehre

Mit dem NetworkParser soll es möglich sein, Anfängern das Programmieren beizubringen. Dieses gestaltet sich als schwierig, da die Vorkenntnisse der Schüler / Studierenden sehr unterschiedlich sind. An der Universität Kassel wird den Studierenden beigebracht, dass ein Problem erst ordentlich beschrieben werden muss, um es in einzelne Probleme aufspalten und diese anschließend lösen zu können. Projekte werden meist am Anfang mit allgemeinen User-Storys oder speziellen Use-Cases beschrieben [MM18].

Objektdiagramme

Es gibt verschiedene Programmierumgebungen, welche speziell für das Erlernen vereinfacht sind. Diese Mini-Umgebungen sind meist nur mit wenig Funktionalität ausgestattet, um das Erlernen zu erleichtern. Robot Karol [Fre18b] ist eine deutsche Mini-Programmiersprache, welche einen Interpreter und eine Eingabeoberfläche mit sich bringt. Damit sind einfache funktionale Programme für den virtuellen Roboter umsetzbar. Von der Universität München gibt es eine Weiterentwicklung von Robot Karol für Java. Hier können die vereinfachten Programme importiert werden und mittels der Programmiersprache Java erweitert werden. Dem Anwender steht nun die komplette Java-Umgebung zur Verfügung. Diese zeichnet sich durch die objektorientierte Programmierung und den Zugriff auf die komplette Java Bibliothek und externe Bibliotheken aus. Als Programmeditor wird BlueJ oder Eclipse empfohlen. [Fre18a]

Die Mini-Sprache Karol ist sehr natürlich sprachlich und ermöglicht das komplexe Thema "Programmieren" sehr früh in den Lehrplan aufzunehmen [BD09] (siehe Quellcode 1). Diese ist sehr umgangssprachlich und bietet mit der Pseudosprache einen guten einstieg. Ein Umstieg auf eine andere Programmiersprache gestaltet sich aber als schwierig.

```
1 Anweisung SucheWand
2   Wiederhole solange NichtIstWand
3     Schritt
4   *Wiederhole
5 *Anweisung
```

Quellcode 1: Karol Beispiel

Greenfoot [Str17]

Greenfoot ist eine vereinfachte IDE, mit dessen Hilfe einfache Spiele realisiert werden können. Die Darstellung der Spiele wird mittels "Tiles" zur Verfügung gestellt. Weiterhin bietet es Klassen mit vorgefertigten Konstruktoren und einer beispielhaften Implementierung einer individuellen Methode an (siehe Abbildung 9). Auch hier modelliert der Benutzer auf einer höheren Abstraktionsebene, womit ein Umstieg zu erhöhten Betreuungsaufwand führt.

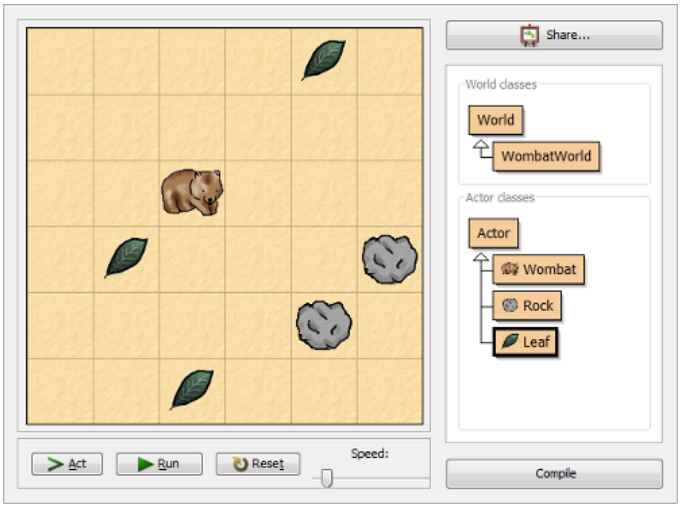

Abbildung 9: Greenfoot

Scratch [Gro18, Res18]

Scratch ist ein online basierter Editor, welcher für Jugendliche im Alter von 8-16 Jahren für das systematische, grundlegende Erlernen von einfachen Programmabläufen geeignet ist. In diesem Editor wird bewusst ohne Programmiersyntax gearbeitet. Ein Beispiel ist in Abbildung 10 zu sehen. Die Lehrveranstaltung "Programmiermethodik" ist im dritten Semester angesiedelt und dient unter anderen als Vorbereitung auf "Software Engineering I". Die didaktische Struktur passt eher als Einstiegsveranstaltung in das erste Semester.

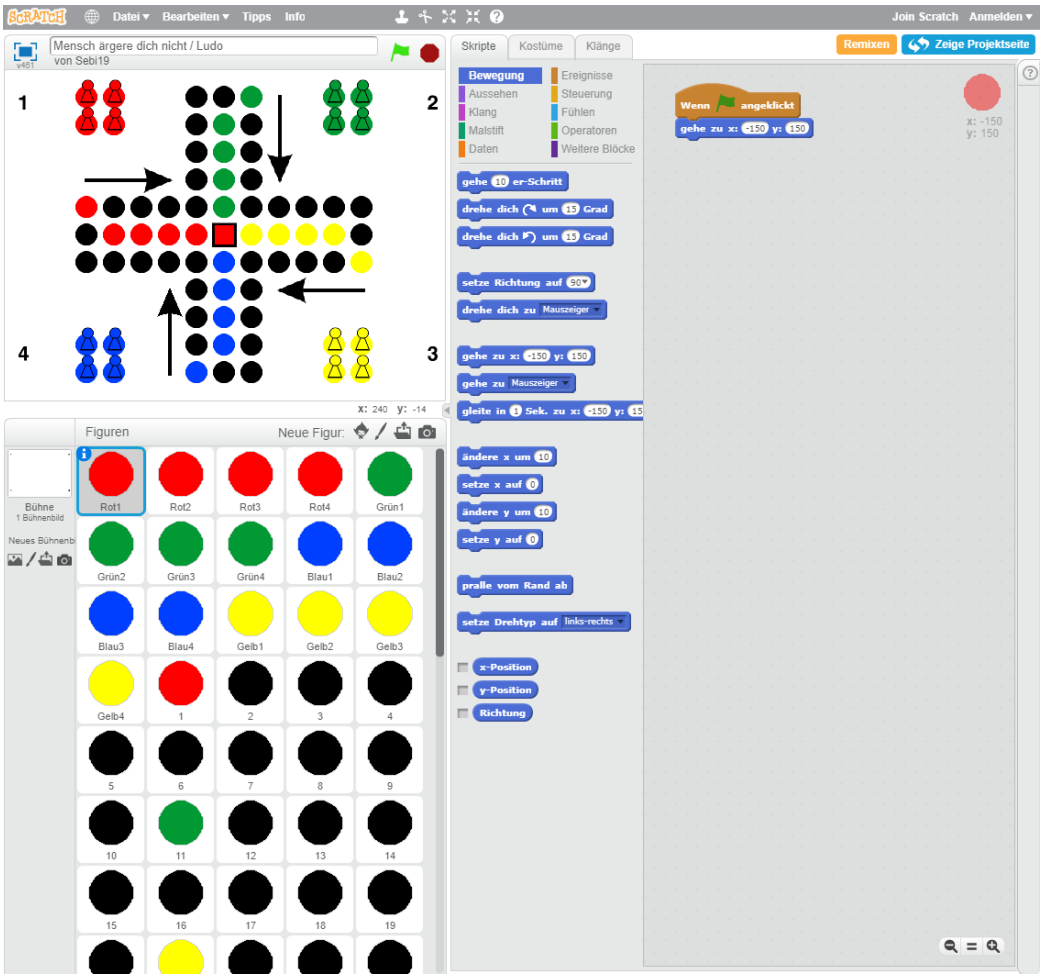

Abbildung 10: Scratch

BlueJ [Lon17]
BlueJ ist eine IDE, welche es ermöglicht einfache Klassenmodelle zu erstellen und die vollen Klassenrümpfe zu editieren. Dieser Editor bietet weiterhin eine Simulation an, in der der entworfene Programmcode ausgeführt werden kann und wo der aktuelle Zustand als Objektdiagramm angezeigt wird. Weiterhin bietet der Editor auch eine Integration von GIT und SVN. BlueJ wird neben Scratch an vielen Schulen eingesetzt. Es ist ein gutes Tool und es gibt viele Schüler/Studierende, welche die Grundlagen damit gut lernen können. Theoretisch können mit BlueJ alle Probleme gelöst werden, jedoch bedarf dieses einen großen Betreuungsaufwand.

WhiteSocks [DJKZ07]
WhiteSocks ist eine Erweiterung von Fujaba [FNTZ98], welche es erlaubt eine grafische Modellierung von Klassendiagrammen und Statecharts mittels Fujaba zu erstellen. Diese können dann mit der GUI-Bibliothek Greenfoot und JFrame dargestellt werden. Da WhiteSocks auf dem nicht mehr weiterentwickelten Framework Fujaba aufbaut, wurde sich gegen WhiteSocks entschieden.

BlackSocks [ADH+09]
BlackSocks ist eine web orientierte auf GWT [Dew07] basierte Version von WhiteSocks. Die Architektur von BlackSocks inklusive des Java-Javascript Compilers GWT ist zu komplex und unwartbar, dass sich gegen dieses Framework entschieden wurde.

Ein guter Vergleich von diversen Arbeiten findet sich in der Ausarbeitung von Herrn Ries [Rie18]. Dort wird eine eigene Lösung gerade für den Lernstoff an Gymnasien gefordert (”Informatik, Mathematik und Physik”).

eDobs [GZ06a]

eDOBS ist eine Weiterentwicklung von Dobs aus Fujaba. eDobs stellt den aktuellen Zustand von Java-Heap-Objekten und Java-Referenzen im Objektbrowser in der IDE Eclipse dar. Die Abbildung 11 zeigt ein Beispiel aus dem originalen Paper von 2006 [GZ06a]. Die Funktionalität von EDobs stellt dabei allerdings nur ein Hilfsmodul für das Programmieren. EDobs setzt dabei auf bestehenden Programmcode auf und stellt die Instanzen als Objektdiagramm dar, so dass im Nachhinein auf Fehler besser und verständlicher reagiert werden kann. Es stellt kein designen von Modellen und textuellen Szenarien bereit. Weiterhin ist es sehr eng mit Eclipse verzahnt und ist ohne dieses nicht benutzbar.

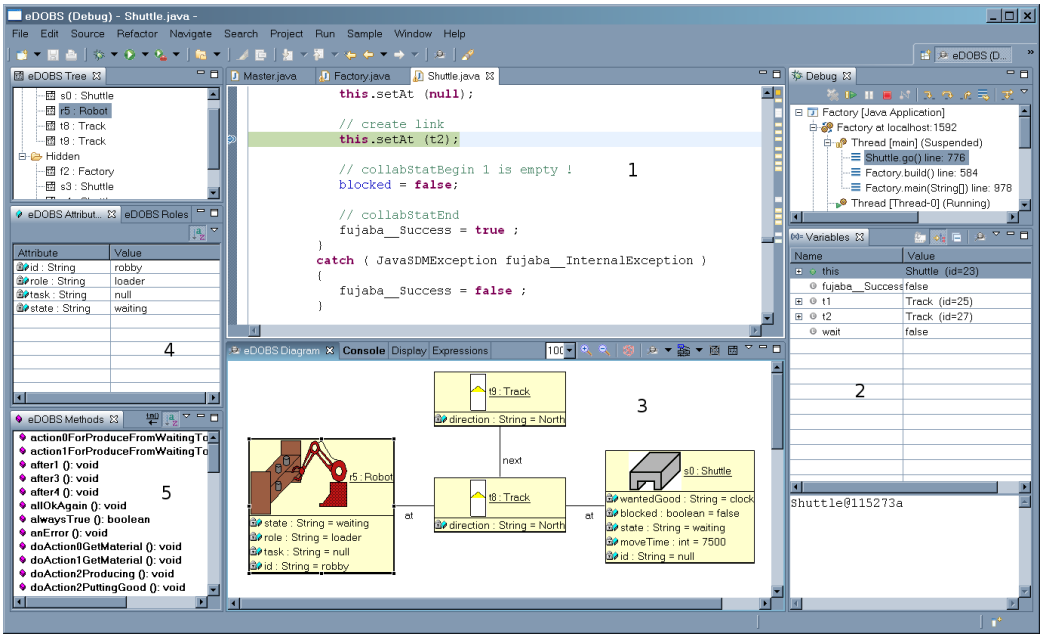

Abbildung 11: EDobs

User-Storys

Für die Modellierung gerade mit Domainexperten sind textuelle Szenarien von Vorteil. Die textuellen Szenarien sind einfach zu verstehen und beinhalten genau die gewünschten Use-Cases. Allerdings ist die automatische Übersetzung sehr schwierig. Hierfür wurde mit der Beschreibungssprache Gherkin eine Grammatik für die standardisierte textuellen Beschreibungen

von Kent Beck verfasst [BG00].
In der Dissertation von Jörn Dreyer [Dre15] wurde ausgiebig das Thema der automatischen Überführung von User-Storys in Objektdiagramme diskutiert. Dort findet man auch die "Given-When-Then" Schritte (steps), welche im Cucumber-Framework [Hel18] umgesetzt wurden. In der Cucumber-JVM für Java besteht die Möglichkeit textuelle Szenarien in "IDL-Dateien" zu definieren. Mittels Ersetzungsdefinitionen kann der Cucumber-Compiler ein Objekt-Szenario erzeugen. Durch die Trennung der Definitionen von Cucumbertext, Logikprogrammcode und dem übersetzten Ergebnis binden sich die Stories nicht in den direkten Entwicklungsfluss ein. Weiterhin muss der Entwickler zu diesem Zeitpunkt ein grobes Verständnis der Logik und des verhalten besitzen und mit regulären Ausdrücken umgehen können.

```
Scenario: Defining the start player

Given Alice and Seb play ludo
Given the players has tokens on startingArea
Given Alice has dice with value 5
When Bob has dice with 1
Then Alice is currentplayer from ludo
```

Quellcode 2: Cucumber Ludo

4.2 Transpiler

Programmcode-Dateien welche eine Repräsentation von Klassenmodellen darstellen unterscheiden sich von Programmiersprache zu Programmiersprache. Für die Unterstützung bzw. Generierung eines Klassenmodells ist es am Anfang wichtig zu entscheiden, welche Programmiersprache unterstützt oder verwendet werden soll. Es gibt unzählige Programmiersprachen. Für meine Dissertation habe ich mich für Java und JavaScript entschieden. Java hat den Vorteil, dass es auf Servern, Desktops und Mobiltelefonen funktioniert. JavaScript dagegen ist mittlerweile auf jedem Endgerät verfügbar, hat allerdings einige gewollte

sicherheitsrelevanten Einschränkungen. Ein Ziel des NetworkParser ist es ein einfaches Framework bereitzustellen, welches auf den oben genannten Geräten ausführbar ist. Dabei soll die einfache Handhabbarkeit und sichere Wartbarkeit von Java mit der universellen Einsetzbarkeit von Javascript kombiniert werden. Für die Benutzeroberfläche soll die Beschreibungssprache HTML, welche auf allen Geräten verfügbar ist, mit der Möglichkeit der bidirektionalen Verknüpfung mit der Businesskomponente umgesetzt werden. Um dieses zu erreichen wird ein Transpiler für Java-Programmcode benutzt. Ein Transpiler übersetzt den Programmcode von einer Programmiersprache in eine andere. Er arbeitet dabei so, dass der Programmcode nahezu identisch ist, bei gleichem Ergebnis bei der Ausführung.

JSSweet

JSSweet wandelt Java Code in Javascript Code um. Das Typensystem wird nicht komplett umgesetzt, sondern es gibt Hilfstypen wie jsweet.lang.Object. Es wandelt den Sourcecode in Typescript und Headerdateien um. Diese können dann mittels Typescript-Compiler nach JavaScript umgewandelt werden. Der originale Source-Code bleibt dabei selten erhalten. Somit wurde sich gegen dieses Framework entschieden [Fel15, Paw17].

4.3 PropertyChange

JavaBeans bezeichnet das allgemeine Binden von Listener an Objekte,um Änderungen weiterzureichen. Die ursprüngliche Entwicklung diente nur zur Aktualisierung der Benutzeroberfläche des Datenmodells. Heutzutage wird Beans in vielen Situationen eingesetzt, zum Beispiel zum Serialisieren oder für automatische Abläufe [Ham18]. Die verwendeten Klassen und Interfaces sind in Abbildung 12 zu sehen. In Abbildung 13 ist das Grundprinzip aufgezeichnet. Die zwei unterschiedlichen Views kommunizieren mit dem Controller und Handler. Änderungen an dem Model wird anhand der blauen Pfeilrichtung wieder an die Controller weitergeleitet.

JavaBeans wird oft mit dem PropertyChangeSupport assoziiert. In diesem gibt es eine Liste von Listener, welche auf alle Änderungen reagiert, und eine Liste von Listener für einzelne Änderungen. Bei jeder Änderung werden die allgemeinen Listener und die Listener für das entsprechende Property mit einem PropertyChangeEvent informiert. Die PropertyChange werden im NetworkParser als Basisklasse für Änderungen für Logikalgorithmen und Benachrichtigungen benutzt.

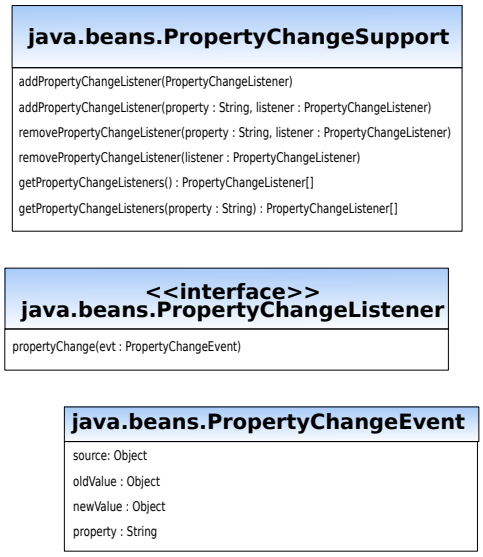

Abbildung 12: PropertyChange

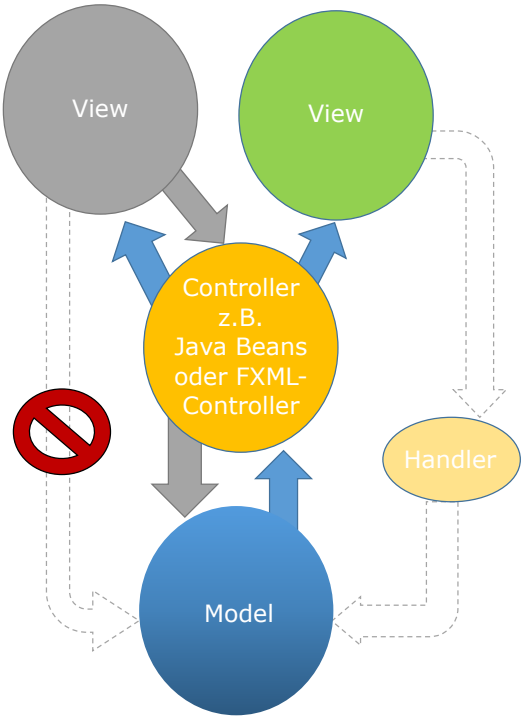

Abbildung 13: Ablauf

4.4 Bidirektionale Assoziation

Für die Implementierung von Assoziation mit der Kardinalität "Many" bei Klassenmodellen müssen einige Anforderungen erfüllt werden. So sollen in der Mengenliste keine doppelten Einträge vorhanden sein. Weiterhin ist es wünschenswert eine Reihenfolgesicherheit zu erhalten. Dieses hat mehrere Vorteile, zum einen sind die Modelle testbar, zum anderen ist es notwendig die Reihenfolge bei der Verwendung von Containern zu

erhalten, um Konflikte bei überschneidenden Nachrichten zu erkennen und zu beheben. Alle Anforderungen sind nur in Kombination von mehreren "Collections" erfüllbar. Die sind allerdings nicht so performant und damit wäre der Vorteil gegenüber einer Standardimplementierung nicht gegeben. In Fujaba [FNTZ98, NNZ00b] wurde aus diesem Grunde eine eigene "Collection", welche auf die `java.util.LinkedHashSet` aufsetzt, entwickelt.

In SDMLib [ZLGE14b, NJZ13] wurde auf eine neue Implementierung gesetzt. Bei dem Transformation Tool Contest (TTC) 2014 und einigen Projekten hat sich herausgestellt, dass die Modelle ungleichmäßig viele "To-Many" Kanten besitzen. So besitzen sehr viele Objekte keine oder nur wenige Kanten. Meist gibt es jedoch auch ein Objekt, welches viele Kanten besitzt (World-Objekt oder Root-Objekt) [TH14, ZLGE14b, ZLGE14a]. Weiterhin wurde die bidirektionale Verbindung zwischen zwei Objekten als Standardimplementierung und intuitives Verhalten verwendet. Nachfolgender Quellcode 3 ist das standardmäßige Verhalten von dem Datenmodell (siehe Abbildung 14) zu sehen.

```
1 University uni = new University();
2 uni.setRooms(new HashSet<>());
3 Room corn = new Room(1337)
4 uni.getRooms().add(corn);
5 corn.setUniversity(uni);
```

Quellcode 3: AddLink OldJava

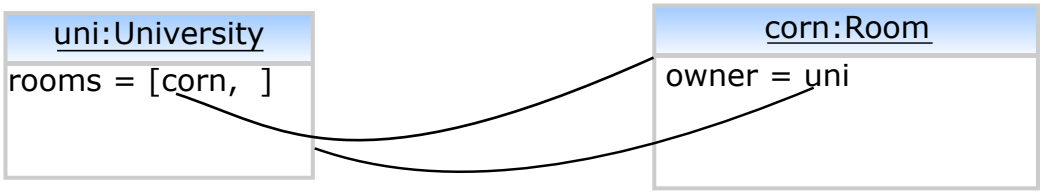

Abbildung 14: Objektdiagramm Beispiel University Kanten

In Fujaba war es zusätzlich notwendig eine eigene Collection zu benutzen, da die Assoziationen immer bidirektional sind. Bei der Lehrveranstaltung "Programmiermethodik" hat sich gezeigt, dass die Studenten, die noch nicht sehr geübt mit dem Umgang von Objekten und Eclipse sind, meist ein Objekt zu einer Collection wie in Codebeispiel siehe Quellcode 4 benutzen und vergessen die Rückrichtung zu setzen.

Das heißt, dort entsteht keine bidirektionale Verbindung. Das Setzen eines Elements in einer zu-n-Menge wird als Nebeneffekt mittels des Aufrufes des Getters hergestellt. Hierfür wird der Getter gerufen und auf den Rückgabewert das Element hinzugefügt.

```
1 uni.getRooms().add(new Room(1337));
```

Quellcode 4: AddLink

In EMF wird eine eigene Collection benutzt. Dort wird die "add"-Methode überschrieben, welche die Rückrichtung des Links setzt.
Durch weitere Optimierungen wird im NetworkParser bei der "getRooms()"-Methode bei der leeren Menge eine statische leere Menge zurückgegeben wie im Codebeispiel 5. Hierfür ist es außerdem notwendig, die zurückgegebene Liste nur lesend zugreifbar zu machen. Zugriff erfolgt jetzt durch die "get"-Methoden. Die "RoomSet.EMPTY"-Collection vermeidet fehlerhafte Benutzungen. Dieses Konzept hat der NetworkParser übernommen.

```
1  public class University {
2    private RoomSet rooms;
3    public RoomSet getRooms() {
4      if(rooms==null) {return RoomSet.EMPTY;}
5      return rooms;
6    }
7    public boolean setRooms(Room... values) {
8      boolean result=true;
9      if (this.rooms == null) {this.rooms = new RoomSet();}
10     for (Room item : values) {
11       this.rooms.withVisible(true);
12       boolean changed = this.rooms.add(item);
13       this.rooms.withVisible(false);
14       if (changed) {
15         result = result & item.setUni(this);
16         firePropertyChange(PROPERTY_ROOMS, null, item);
17       }
18     }
19     return result;
20   }
21   public University withRooms(Object... values) {
22     for (Object item : values) {
23       if (item == null) {continue;}
24       if (item instanceof Collection<?>) {
25         setRooms(((Collection<?>) item).toArray());
26       } else {
27         setRooms((Room) item);
28       }
29     }
30     return this;
31   }
32 }
33 public class Room {
34   private University uni;
35   public University getUni() {return uni;}
36   public void setUni(University value) {
37     if(value!= this.uni){
38       University oldValue = this.uni;
39       this.uni=value;
40       firePropertyChange(PROPERTY_UNI, oldValue, value);
41     }
42   }
43 }
```

Quellcode 5: ClassCode

4.5 Java Collection

Es gibt eine Reihe von Collections, die bereits mit Java ausgeliefert werden. Für den normalen Einsatz wird meist die java.util.ArrayList [Doc15a] benutzt. Für eindeutige Listen wird häufig die java.util.HashSet [Doc15b] oder die java.util.LinkedList [Doc15d] verwendet. Für festgelegte Reihenfolgen und schnelle Listen wird dann die java.util.LinkedHashSet [Doc15c] benutzt. Im EMF-Kontext wird die org.eclipse.emf.common.util.BasicEList oder die org.eclipse.emf.ecore.util.EObjectResolvingEList [Fou15] gebraucht, die auf AbstractList aufbauen. Als Vergleich wurde zusätzlich die Brownies-Collection in der Version 0.9.10 hinzugenommen. Aus der Sammlung der dort zur Verfügung stehenden Collections wurde die GapList [Mau15b] und die BigList [Mau15a] ausgewählt.

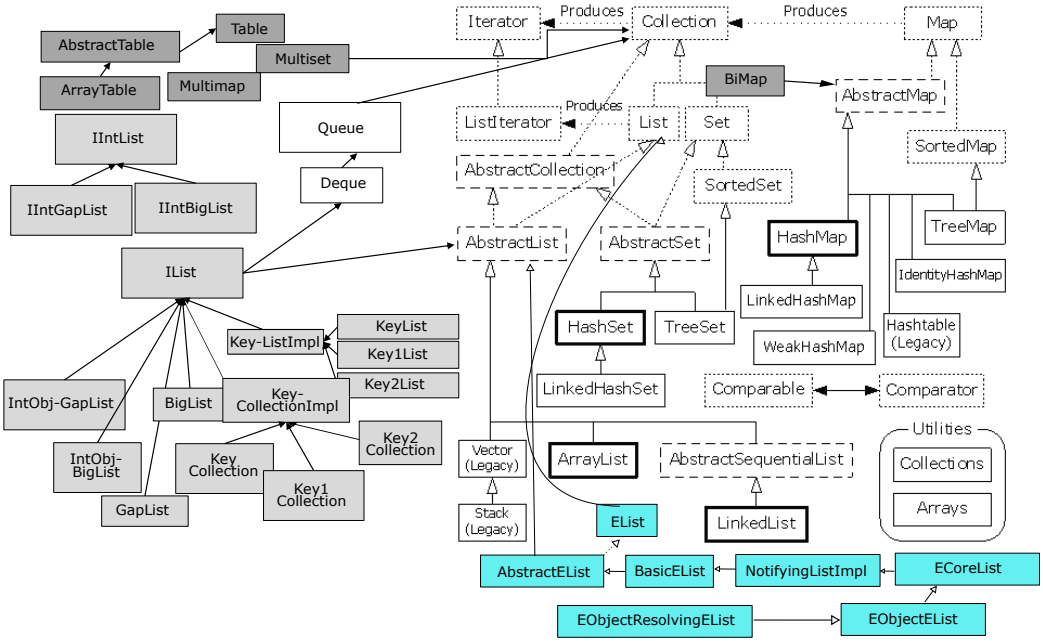

Abbildung 15: Java Collection + Brownies + EMF Collection + Guava

Diese Collections haben jedoch auch einige Nachteile. Sie sind meist für einen Anwendungsfall ausgerichtet und sind im Nachhinein schwer, beziehungsweise gar nicht austauschbar. Die Speicherprobleme bei

LinkedHashSet und BasicEList sind in der Messung (siehe Abbildung 37 auf Seite 119) aufgeführt oder in der Zeitmessung (siehe Abbildung 36 auf Seite 119) dargestellt. Insgesamt verbraucht das LinkedHashSet zu viel Speicher und die BasicEList arbeitet zu langsam. In Abbildung 15 sieht man eine Auflistung einiger Collections. Die unterschiedlichen Farben repräsentieren die unterschiedlichen Frameworks.

4.6 Serialisierung

Für das Speichern und Übertragen ist es notwendig das Modell und Änderungen in ein Austauschformat zu überführen. Da es eine Reihe von unterschiedlichen Anwendungsszenarien gibt, wurde auch überlegt nicht nur ein Datenformat (Serialisierungssprache) zu unterstützen, sondern möglichst viele verschiedene Datenformate. Um die neuesten Techniken zu unterstützen, müssen mehrere Formate benutzt werden, wie JavaScript Object Notation (Json), Extensible Markup Language (XML) und Byte-Kodierung (Byte) [Fri16]. Das Thema wird ausführlich in der Masterarbeit diskutiert [Lin12], welche bereits über 180-mal bei Reseachgate [Inc18c] gelesen wurde.

4.6.1 json.org

Json ist mittlerweile ein sehr verbreitetes Format für den Datenaustausch, es ist sehr kompakt und in vielen Sprachen verfügbar. Das Format besteht aus Listen (JsonArray) und Key-Value-Objekten (JsonObject). Die genaue Sprachdefinition ist auf json.org [Cro06] nachzulesen. Die Json Serialisierung des NetworkParser ist nach der Referenzimplementierung "json.org" von 2011 nach implementiert. Allerdings stand hier der Fokus auf einer Unterstützung von mehreren Sprachen wie Json, XML und einem proprietären Format.

4.6.2 Google GSON

Google hat eine Java Bibliothek entwickelt, welche das einfache Umformen von Java-Objekten nach Json ermöglicht. Es ist auf jedes Modell anpassbar allerdings bietet es nur die Konvertierung nach Json an. Es funktioniert

mittels Reflection und kann somit auch auf unbekannte Modelle angewendet werden. Allerdings gibt es noch Probleme mit bidirektionalen Kanten. Hierfür wird die externe Klasse GraphAdapterBuilder [ind17b] benötigt. Alternativ kann auch der Entwickler bewusst Kanten mit transient versehen (Nicht permanent), damit dieser Link zwischen zwei Objekten ignoriert wird. Hierfür ist ein enormes Wissen über das Modell notwendig. [ind17a]

4.6.3 json-io

Das Framework json-io ist eine sehr schnelle Persistenzschicht, die es ermöglicht Nachrichten sehr schnell nach Json zu überführen. Ganze Modelle automatisch umzuwandeln, funktioniert nicht. [DeR17]

4.6.4 json-Simple

Dies ist einfache Java Bibliothek zum Erstellen von Json. Diese Bibliothek ist nur 24kb groß und dessen Name beschreibt die Eigenschaft. Es bietet nur die notwendigen Dateien wie JsonObject und JsonArray [fan17].

4.6.5 json-lib

Diese Bibliothek ermöglicht es bequem von Listen, Maps und XML nach JSON zu konvertieren. Das automatische Transformieren von Modellen ist relativ schwierig, da es spezielle Datentypen gibt, die explizit abgeprüft werden müssen. Die Bibliothek ist überschaubar und mit 156 kb sehr klein. [Alm17]

4.6.6 SAX Parser

Der ”Simple API for XML-Parser” (SAX) ist eine Standardimplementierung für Java. Sie wird mit dem JDK mitgeliefert und dient zum seriellen Parsen von XML-Dateien, welches mittels Callbacks angepasst werden kann. [Bro02] Der SAX Parser ist die Standardimplementierung bei der Entwicklung von Android Anwendungen. Allerdings war er nicht besonders effizient und hatte eine Größenbeschränkung von 64k-Byte von XML Dateien.

4.6.7 Yaml

Die neueste Entwicklung geht in Richtung von MarkDown und YAML. YAML (Ain't Markup Language) [yam19] ist eine vereinfachte Auszeichnungssprache. Die Definition hat die Version 1.2 2019. Tools zum einlesen wären zum Beispiel die SnakeYAML Engine [sna19] in Java.

4.7 Benutzeroberflächen und Diagramme

Es gibt eine Reihe von Benutzeroberflächenframeworks in den unterschiedlichsten Programmiersprachen. In dieser Dissertation soll der Fokus auf Java liegen. Allein in Java existieren mehrere Lösungen für Desktop, Mobile und Webanwendungen. Es gibt viele Frameworks speziell für einzelne Anwendungsfälle. In dieser Arbeit soll ein Benutzeroberflächenframework ausgesucht werden, welches es ermöglicht die Benutzeroberfläche einmal zu modellieren und [AG17] auf allen Plattformen auszuführen. Nicht zu verachten ist im Moment der Wandel der Frameworks [Sch17b] von nativen Ansätzen zu einer Lösung mittels HTML5-Webtechnologie. Hierbei sieht man jedoch, dass zunehmend die Entwicklung Richtung mobiler Webseiten geht.

Java - Desktop

AWT, Swing, SWT und JavaFX gehören zu den bekanntesten GUI-Frameworks in Java. Bei der neuesten Entwicklung JavaFX hat man sehr viele Ansätze aus der Webrichtung übernommen und neu umgesetzt. So gibt es eine eigene Interpretation von Layout (CSS) und nur eine begrenzte Anzahl von Eingabeelementen. Dieses ist mit Java 9 vollendet werden.

Allerdings führen jetzt schon mehrere Artikel [Sch17c, Tho15] auf, dass JavaFX auch von Oracle mittlerweile nicht mehr aktiv weiterentwickelt wird. Dieses zeigt sich auch in dem Oberflächendesigner SceneBuilder. Dieser wird mittlerweile von der OpenSource Community Gluon [Glu16] weiterentwickelt. JavaFX ist ein richtiger und großer Schritt in die richtige Richtung, allerdings hat er bei Weitem nicht so viel Verbreitung gefunden

wie die bisherigen Benutzeroberflächen-Frameworks. Auch zu diesem Thema gibt es verschiedene Ansichten [Sch17c]. Oracle hat die Weiterentwicklung schlussendlich aufgegeben und hat JavaFX der freien Community zur Verfügung gestellt. Es ist somit nicht mehr Bestandteil des Java JDK.

Java - Android

In Android existiert nur ein Benutzeroberflächenframework, welches von Google mittels des ADT ausgeliefert wird. [Mos17b, Inc17]. Die Entwicklungsumgebung war früher Eclipse. Seit November 2016 ist dies ein angepasstes InteliJ. Auch hier sieht man die voranschreitende Entwicklung, mit der der Programmierer gezwungen ist, sogar seine IDE anzupassen oder zu wechseln. Weiterhin besteht bis zur Version Android 5.0 eine Dalvik-Dex Einschränkung, dass Datei-Ressourcen mit dem eingebauten XML-Parser nur 64k-Bytes groß sein dürfen.

Web

In Java wurden die Java-Servlets [Ora18a] in der Spezifikation 116 [Kri18] am 07.03.2003 veröffentlicht. Seitdem hat sich zwar einiges dort getan, diese werden heutzutage aber nicht mehr eingesetzt. Java Servlets ermöglichen die Ausführung von JavaCode in Webseiten auf dem Client. Auf dem Client wird ein Javaprogramm gestartet, welches standardmäßig die Eingrenzungen wie ein Browser besitzt. Die Businesslogik wird auf einen Applicationserver in einem Servlet-Container ausgeführt. Die Wartung dieser Servlets stellt ein großes Problem dar.

Die Java Server Pages (JSP) [Poi18, Pan18] sind eine Servertechnologie, welche auf dem Server dynamische Inhalte aufbaut und zum Client eine statische Seite schickt. Diese müssen keine Voraussetzungen erfüllen.

JSF [Ora18b] ist eine modernere Variante von Java Server Pages, welche auf dem Server ausgeführt wird. Es benutzt das MVC-Pattern und bietet eine Verknüpfung von View und Model an, ohne die Inhalte direkt in JSF zu implementieren. Aktuell ist die Version 2.3. JSF wird auf einem

Server in einem JSF-Servlet-Container gestartet. In dieser Arbeit wurde sich bewusst gegen diese Technik entschieden, da Sie von dem Entwickler eine Serverinstanz mit JSF-Architektur verlangt.

Die beliebtesten clientseitige Javascript-GUI-Frameworks sind "AngularJS" und "React". Im direkten Vergleich [Gmb17a, Sch17a, Kor17] werden diese beiden Frameworks als momentaner Standard für Webframeworks dargestellt. Allerdings sind sie im Moment speziell für das Web und deren Umgebung zugeschnitten.

4.7.1 Diagramme

Ein Klassendiagramm oder Objektdiagramm darzustellen ist eigentlich keine Herausforderung. Allerdings hat sich schnell herausgestellt, dass die Schwierigkeit darin besteht, ein automatisches Layout zu finden, das nur eine minimale Anzahl von Kanten Kreuzungen enthält. Nach Möglichkeit soll das dargestellte Diagramm plättbar bzw. planar sein. Nach dem eulerschen Polyedersatz kann für einen planaren Graphen die Menge der Knoten V nach der Anzahl der Kanten E abgeschätzt werden.

$$G = (V, E)$$

$$|E| \leq 3|V| - 6.$$

Wenn man nach Layout-Algorithmen im Internet sucht, gibt es unzählige Lösungen, die es in verschiedenen Programmiersprachen und Ausprägungen gibt. Zwei plattformabhängige Lösungen sind GraphViz [EGK+02, GKNV93] und yFiles [yWo15]. Diese sind je nach Ausprägungen entweder als Webservice oder lokal einsetzbar. Einige Lösungen sind in der Bachelorarbeit [Sch14] gut verglichen. Interessant sind die Lösungen, die Betriebssystemunabhängig sind und Javascriptbasiert funktionieren. Jointjs [Joi15], Cola.js [Mar15] und Soyatec [Soy15] sind solche Lösungen. Diese besitzen die Darstellungsmöglichkeit, jedoch können sie eine vorhandene Knotenwolke mit unterschiedlich großen Knoten nicht darstellen. Dieses

Problem besitzt auch das D3-Framework [Bos15]. Hier gibt es eine Reihe von Algorithmen, die die Berechnung des Layouts realisieren. Zwei davon sind Dagre [cpe15] und Klay [Ope15b]. Die entstehenden Graphen sind allerdings eine Mischung aus SVG und HTML. Gerade Objekt- und Klassendiagrammen mit den unterschiedlichen großen Elementen und den Kanten mit extra Informationen kann das D3-Framework schwer darstellen. Daher wurde nach langer Suche eine eigene Implementierung realisiert, welches mittels des Layoutframework Dagre die Diagramme layoutet und als SVG-Bilder darstellt (siehe Abschnitt 5.9 auf Seite 93).

4.8 Verteilte Systeme

Für die Verteilung von Nachrichten und Modellen hat sich herausgestellt, dass es zwar einige Frameworks gibt, die für sich allein gut funktionieren, die meist jedoch mit einfachen Nachrichten funktionieren und sich nicht um das Verschmelzen von unterschiedlichen Modellversionen kümmern. Es existiert allerdings keine einfache Implementierung für das Serialisieren und Verteilen mit unterschiedlichen Techniken oder unterschiedlichen Formaten inklusive Rechten.

4.8.1 JMS

[Cur18] Es gibt für die Verteilung von Clients verschiedene Ansätze. Es existieren bereits auch Rahmenwerke die als "Message Oriented Middleware" funktionieren. In diesem Kontext seien die Java Message Service (JMS) wie ActiveMQ von Apache oder Rabbit erwähnt. Diese Rahmenwerke bilden eine Abstraktionsebene zwischen den Anwendungen und kommunizieren über Sockets. Sie transportieren die Daten in einfachen Formaten wie Zeichenketten und Bytestreams. Es existieren JMS Provider (Anbieter) und Broker (Makler). Das entspricht einem Sender mit Empfänger.

Die bisherigen Techniken lösen die Verteilung von Nachrichten sehr effizient. Wie in dem Artikel [qiCT17] zu sehen, ist die Verteilung von Nachrichten, darauf ausgelegt, dass sehr viele Nachrichten gleichzeitig an mehrere Knoten verteilt wird (siehe Abbildung 16 auf Seite 53).
ActiveMQ und RabbitMQ sind Client-Server Rahmenwerke, wobei der Server im Cluster gespiegelt werden kann. Die zwei Frameworks kümmern sich um die Verteilung von Daten im einfachen Format. Diese bauen alle auf TCP-Verbindungen auf und benutzen Channels. Zu Anfang der Dissertation waren diese zwei Verfahren nur über native Clientprogramme möglich. Mittlerweile gibt es Clients, welche in Java geschrieben sind. Auf dieser Arbeiten habe ich ein einfaches Verfahren implementiert und in den NetworkParser integriert (Siehe 5.15.3 auf Seite 136) [Wei18].

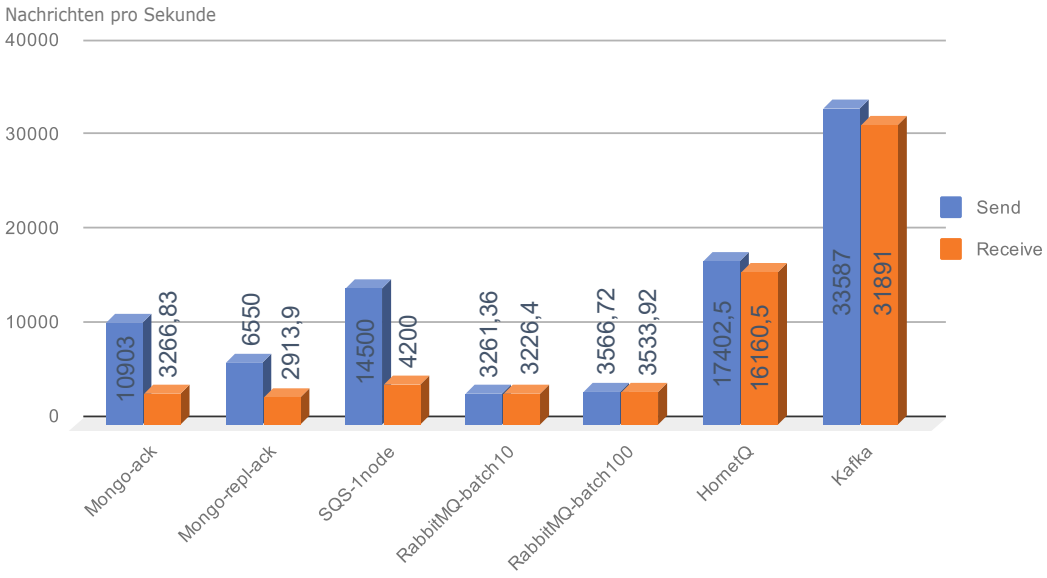

Abbildung 16: Kafka und RabbitMQ und andere Tools

4.8.2 Akka

Akka ist eine Implementierung, die auf dem Aktormodell basiert [See17]. Es stellt ein sehr robustes System zur Verfügung, welches fehlertolerant ist, soweit die Nachrichten eine lose Kopplung besitzen. Es dient zum Austausch von einfachen Nachrichten. Für die Integration von Modellen muss der Programmierer wieder Hand anlegen. Akka stand nicht im Fokus dieser Arbeit. Denkbar ist eine Erweiterung zu schaffen, damit das AKKA-System unterstützt wird. Allerdings muss ein individueller Programmcode für den jeweiligen USE-Case geschrieben werden und die Knoten müssen möglichst in hierarchischer Struktur vorliegen [Nor18].

4.8.3 EMFStore

Das ”Eclipse Modeling Framework” (EMF) [Fou15] ist ein Modellierungsframework und Tool zur Code-Generierung basierend auf einem strukturierten Datenmodell. Mittels XMI können Modelle serialisiert und verteilt werden. Für die Verteilung kann der EMFStore benutzt werden, wo die Modelle in einer relationalen Datenbank abspeichern werden. EMFStore wurde nicht verwendet, da es einen EMFStore-Server voraussetzt.

Außerdem funktioniert es nur im Eclipse-Umfeld, weil es auf EMF aufbaut oder viele exrerne Bibliotheken benötigt.

4.8.4 Distributed Objects

Distributed Objects [Nit18] dient als Oberbegriff von mehreren Verteilungsansätzen. Dieses findet besonders Verwendung bei Agentensystemen. Es gibt eine Reihe von Frameworks, die dieses bereits unterstützen: DCOM (Distributed Common Object Model), CORBA (Common Object Request Broker Architecture) und JINI.

Corba

Die "Common Object Request Broker Architecture" wurde von "Object Management Group" (OMG) [MR98] entwickelt. Dieses unterstützt die Verteilung von Objekten über Rechnergrenzen hinweg. Hierfür wird auf lokalen Netzen das TCP-Protokoll und General Inter-ORB Protocol (GIOP) und für das Internet das Internet-Inter-ORB Protocol (IIOP) benutzt. Dieses wurde im NetworkParser nicht genutzt, da es nur mittels IDL-Modelle und individueller Programmcodegenerierung funktioniert.

Java remote method invocation

RMI dient zum einfachen Aufrufen von Methoden auf anderen JVM [Hei18]. Dabei stellt einer einen RMI-Registry Server bereit, wo sich mehrere Clients registrieren können, um danach direkt untereinander zu kommunizieren. Der NetworkParser unterstützt momentan nicht das RMI-Verfahren, da sich bewusst für ein menschenlesbares Format entschieden wurde und nicht für das binäre Format von RMI.

4.8.5 MQTT

MQTT ist ein Client-Server-Protokoll [Pip18]. Clients senden Nachrichten mittels Topics an einen Server ("Broker"). Jeder Client kann sich daraufhin auf spezielle Topics registrieren. Es gibt verschiedene Optionen für das Senden von Nachrichten. Die häufigste ist die

Fire-and-forget-Technik oder einfach ausgedrückt das einfache Senden ohne Sicherstellung, dass der Empfänger die Nachricht auch erhalten hat. Es wurde ein NodeProxy im NetworkParser implementiert, welcher nach diesem Prinzip asynchrone Nachrichten versenden kann.
Ein weiteres Verfahren ist das quittierende Senden, dass bedeutet, dass solange gesendet wird, bis die Nachricht quittiert wird. Dieses kann bedeutet, dass die Nachricht öfter ankommt. Oder mit dem Ziel, dass die Nachricht nur einmal ankommt. Der Server bietet auch die Möglichkeit Nachrichten bei erfolglosem Senden zwischenzuspeichern, für ggf. Ausfälle von Clients. MQTT wird üblicherweise über TCP benutzt.

4.8.6 REST

"Representational State Transfer" ist eine einheitliche, vereinfachte Schnittstelle für verteilte Systeme [Cam18]. Diese standardisierte Schnittstelle ermöglicht eine schnelle Erweiterung von bestehenden Systemen und die einfache Einbindung neuer Dienste. Es wurde 1997 in einer Dissertation erarbeitet und entwickelt sich seit 2014 als Standard für Online-Dienste. Im NetworkParser wurde ein Service implementiert, welcher es ermöglicht mittels wenigen Zeilen siehe Quellcode 52 eine REST-Schnittstelle anzubieten (siehe Abschnitt 5.15.3 auf Seite 134).

4.8.7 Resilient Software Design

Bei diesem Design geht es darum stabile (unverwüstliche) Software zu realisieren. Das bedeutet, dass das System möglichst nicht ausfällt. Die Verarbeitung soll in keiner Situation ins Stocken geraten und der Benutzer soll ggf. unerwartete Fehler nicht bemerken [Fri17]. Für die Robustheit einer Software wird die Formel der Verfügbarkeit benutzt:

$$\frac{MTTF}{(MTTF + MTTR)}$$

NetworkParser ist genau nach diesem Prinzip aufgebaut [Han13].

4.8.8 Versionierung vs. Kollaboration

Bei der Online Kollaboration wird ein paralleles Arbeiten an einem Dokument umgesetzt. Hierfür werden alle Änderungen direkt an die anderen Netzwerkknoten weitergesendet. Um Konflikte zu umgehen, wird empfohlen nur einfache Informationen oder Befehle zu teilen. Ein Konflikt bei überschneidenden Nachrichten ist dann schwer möglich.
Bei der Versionierung von Datenmodellen ergibt sich der Vorteil, dass alle Änderungen nachvollziehbar sind und die unterschiedlichen Netzwerkknoten in einem verteilten System auftretende Konflikte lösen können. Dieses geschieht immer nacheinander, sodass nur ein verzögertes Verteilen realisiert werden kann. Um die Konfliktwahrscheinlichkeit zu reduzieren, kann man ACID-Transaktionen [KE04], [Zor12] verwenden. In der IdMap wurde ein ModelChange-Mechanismus implementiert, welcher es ermöglicht, nur relevante Nachrichten an andere Knoten weiterzuleiten, sodass die Anzahl der Datenpakete reduziert wird. Die IdMap hat einen speziellen UpdateFilter der normale Änderungen in atomare Änderungen aufspalten kann oder in einfache Transaktionen zusammenfassen kann (siehe Abschnitt 5.15.1 auf Seite 129).

4.9 MontiCore

MontiCore ist ein Werkzeugframework, welches für die Softwareentwicklung zum schnellen und effektiven Erstellen von domänenspezischen Sprachen dient.

	NetworkParser	**MontiCore**
Version	4.7.1235	4.5.3
Ursprung	Kassel	Aachen
Start	21.12.2013	23.08.2006
Größe	1.4 MB	31.9 MB
Verbreitung	Gradle, Maven	Maven
Verwendung	Lehre, Industrie	Lehre, Industrie
Modelle	Klassen	Interfaces und abgeleitet Klassen
IDE	Gleiche Unterstützung in allen IDE's	**Eclipse**, funktioniert auch unter InteliJ und Java UML/P(programmierbar)
Diagramme	textuell: - Klassendiagramm - Objektdiagramm - Story - Cucumber - Javacode - GUI	Visuell: - Klassendiagramm - Objektdiagramm - Statechart - Sequenzdiagramm textuell: - OCL - Java Codeteil
Assoziations-standard	bidirektional	unidirektional
Methodenanzahl für Assoziation	3 / 4	4 / 6

Abhänigkeiten	Keine	- de.monticore.commons:se-commons-utilities - de.monticore.commons:se-commons-groovy - de.monticore.commons:se-commons-logging - com.google.guava:guava - org.reflections:reflections - commons-io:commons-io - com.google.code.findbugs:jsr305 - com.google.inject:guice - com.google.inject.extensions: guice-assistedinject - javax.inject:javax.inject - org.codehaus.groovy:groovy - org.slf4j:slf4j-api - ch.qos.logback:logback-classic - ch.qos.logback:logback-core - org.jgrapht:jgrapht-core - jline:jline - org.antlr:antlr4 - org.antlr:antlr4-runtime - org.freemarker:freemarker - org.mod4j.org.eclipse.emf:ecore - org.mod4j.org.eclipse.emf.ecore:xmi - org.mod4j.org.eclipse.emf:common - de.monticore:monticore-runtime - de.monticore:monticore-grammar - de.monticore:monticore-generator - de.monticore:monticore-emf-runtime - de.monticore:javaDSL - de.monticore.lang:cd4analysis

5 Umsetzung

Alle genannten Herausforderungen wurden im NetworkParser vereint und eng miteinander verzahnt. Eine Übersicht über die wichtigsten Komponenten des NetworkParser Framework ist auf der folgenden Seite dargestellt (siehe Abbildung 17 auf Seite 60). Einige Komponenten wurden als Projekte, Bachelorarbeiten (BA) und Masterarbeiten (MA) realisiert. Die wichtigsten Komponenten des NetworkParser sind:

- IdMap für das Persistieren und zentrale Klasse
- Codegenerator und Parser für das Round-Trip Engineering von Modell-Code
- Das NetworkParser-Framework kümmert sich um die Verteilung von Nachrichten und die Auflösung von unterschiedlichen Modellkonflikten
- Die Netzwerkschicht dient zur Bereitstellung verschiedener physikalischer Verbindungen
- Die GUI Komponenten dienen zur allgemeinen, Plattform und Programmiersprachen unabhängigen Realisierung
- Die optimierten Collections
- Dokumentation und Projektplanung mittels Story und Cucumber
- Test und Validatoren Komponenten zur Qualitätssicherung
- Objekt- und Klassendiagramm inklusive Editoren
- Projektmanagement

Diese Komponenten werden in den folgenden Abschnitten vorgestellt.

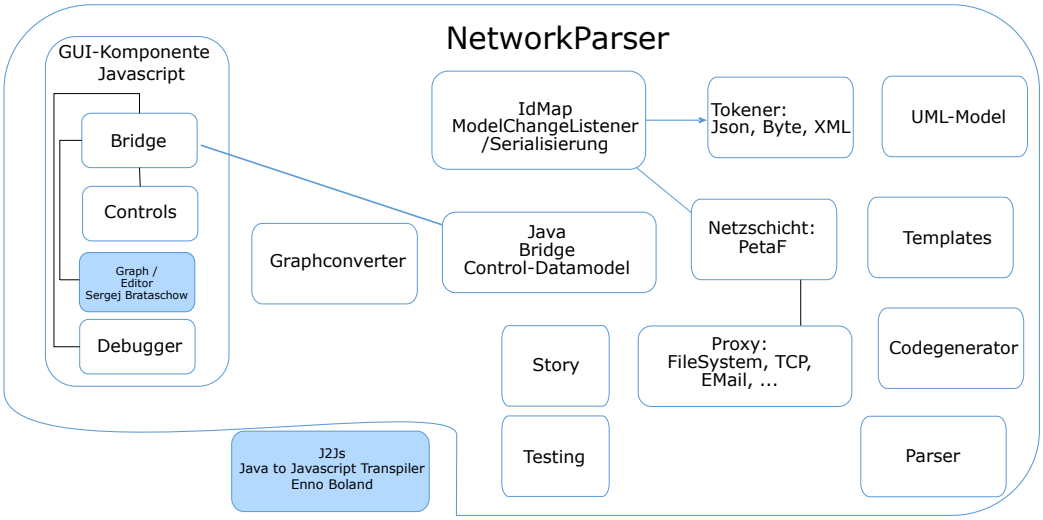

Abbildung 17: Komponentendiagramm

Alle Komponenten (siehe Abbildung 17) wurden nach und nach aus organisatorischen Gründen und damit die weitere Entwicklung möglichst effektiv ist in den NetworkParser übernommen. Ein guter Nebeneffekt ist die enge Verzahnung der Komponenten. Im Moment fehlt nur noch die Bachelorarbeit von Enno Boland (J2Js). Der Transpiler wurde allerdings komplett als eigenes System entwickelt und soll so nicht komplett übernommen werden.

5.1 Projekthistorie

Das Projekt "NetworkParser" wurde am 21. Dezember 2011 19:03 gestartet und unter `https://tracsvn.cs.uni-kassel.de/svn/SL_MasterAndroid` als SVN-Projekt versioniert. Es war ein Teil meiner Masterarbeit [Lin12], welche sich aus folgenden Komponenten zusammensetze: NetworkParser, NetworkMessages, Network, PeerCenter, PeerCenterAndroid und PeerCenterJava. Als Buildsystem wurde ANT eingesetzt. Der NetworkParser war zu diesem Zeitpunkt einzig für das Serialisieren von Objektinstanzen verantwortlich. Hierfür existierten mehrere IdMaps wie JsonIdMap, XMLIdMap, ByteIdMap und SQLIdMap. SDMLib wurde am 8. März 2012 gestartet. Seit dem 14. März 2012 wurde zu SDMLib eine Exportmöglichkeit nach Dot implementiert. Dot ist ein Teil der GraphViz Bibliothek und erlaubt z.B. die Generierung von Diagrammen und das automatische anordnen von Objektdiagrammen und Klassendiagrammen. Die Quellcodedateien vom NetworkParser wurden seit dem 16. März 2012 immer wieder in das Projekt SDMLib hineinkopiert. Dieses erwies sich als sehr unpraktisch. Am 31. Oktober 2012 wurden GUI Elemente zum eigenen Source-Path von SDMLib (srcSWT) hinzugefügt. Am 18. März 2014 wurde der NetworkParser als Subkomponente von SDMLib transformiert.

Am 21. September 2012 wurden die Elemente von PeerMessage dem NetworkParser hinzugefügt. Am 21. Juni 2013 wurden zwei Projekte ins Leben gerufen, welche sich um die Darstellung der Oberfläche kümmerten, NetworkParserFX und NetworkParserSWT. Mittels dieser Projekte wurde der UseCase EONShifting (siehe Abschnitt 3.6 auf Seite 32) realisiert und die Tabellenansicht für eine einfache und übersichtliche Darstellung der Daten realisiert (siehe Abschnitt 3.7 auf Seite 33).

Am 11. November 2013 wurde das Versionsverwaltungssystem von dem veralteten SVN auf GIT gewechselt. Durch die Verwendung von SDMLib und NetworkParser in der Lehre hat sich herauskristallisiert, dass die Studierenden immer wieder Probleme mit dem Aufrufen von GraphViz haben, da dieses Betriebssystemabhängig ist. Es wurde zuerst probiert mit

Installationspaketen zu arbeiten, so dass die Studierenden ein Paket für ihr Betriebssystem herunterladen und entpacken (200MB). Da dieses nicht unter MAC funktionierte wurde seit dem 15. Dezember 2013 eine alternative gesucht und am 19. Juli 2014 wurde zum ersten Mal die Javascriptdatei für die Graphdarstellung mit dem NetworkParser ausgeliefert. Weiterhin wurde das Buildsystem an diesen Tag von ANT auf Gradle umgestellt. Am 8. Oktober 2015 wurde als Kontinuierliche Integration Server der Travis hinzugefügt. 20. Februar 2016 wurden die verschiedenen IdMaps in eine einzige überführt und die verschiedenen Sprachspezifikationen in einen Tokener verschoben.
Ab dem 13. Juni 2016 ist der NetworkParser Oberflächenunabhängig. Zu diesem Zeitpunkt wurden die Dependencies per Soft-Dependencies ersetzt, so dass der NetworkParser komplett OpenJDK kompatibel ist.

5.2 IdMap

Der "Story Driven Modeling" Ansatz benötigt auch einen Mechanismus zur Serialisierung von Objektmodellen. Für Fujaba wurde schon ein changelogbasierter Mechanismus für die Serialisierung, Persistenz und Versionierung von Objektmodellen entwickelt [Sch07]. Dafür wurden drei Dinge vorausgesetzt, ObjektIds, ChangeListener und Reflektion. Die Idee der IdMap ist in der Software-Entwicklung nicht neu und wurde aber aus verschiedenen Gründen in dieser Arbeit neu implementiert. Als Vergleich wird die IdMap mit dem Json-Framework Jackson verglichen. Zur einfachen Benutzung wurde die Erzeugung und Konfiguration vereinfacht, so dass mittels einfachen Aufrufes in einer Hilfsklasse des Modells eine vollständige IdMap für das Modell erzeugt werden kann.

Bei Jackson kann das Modell mittels Annotationen so konfiguriert werden, dass mit der Klasse "ObjektMapper" eine "ID" bezogene Serialisierung zumindest mit Json möglich ist. Dieses bedeutet aber eine gewisse Modellabhängigkeit. Eine ID-bezogene Serialisierung wird auch im NetzwerkParser verwendet. Der NetworkParser benutzt dieses Verfahren für Modelle und für Events von PropertyChanges. Letzteres ist auch mittels Jackson möglich, allerdings bedarf dieses eines zusätzlichen Listeners und des ObjektMappers. Dann muss noch ein Weg gefunden werden, die vergebene ID vom "ObjektMapper" zu ermitteln.

Weiterhin soll die Serialisierung nach Möglichkeit sehr universell einsetzbar sein, so dass das Format frei gewählt werden kann. Jackson bietet zwar eine Factory an, ist aber auf Json beschränkt. Die IdMap ist sprachunabhängig. Sie bietet einfache Filter und Benachrichtigungsmöglichkeiten, um die Serialisierung anzupassen.

Es existieren einfache baumartige Modelle und häufiger komplexe Modelle mit Zyklen. In der Abbildung 18 sind beide Modellarten gegenübergestellt. Ein baumartiges Modell hat den Vorteil, dass es sich bei der Serialisierung

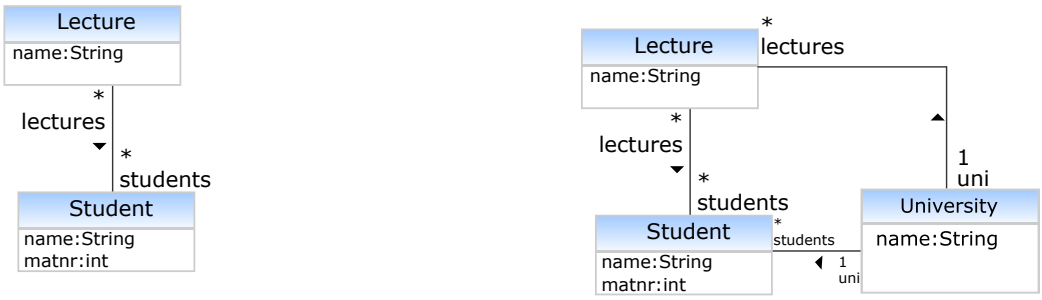

Abbildung 18: baumartige und zyklisches Modelle

leicht durch geschachtelte Objektbeschreibungen darstellen lässt. Ein einfaches Objekt-Beispiel, wie in Abbildung 19 dargestellt, wird zum Beispiel durch den Jackson Parser nach Auflistung 6 übersetzt.
Wie man sieht wird das Wurzelobjekt von Typ ”Lecture” durch das erste paar geschweifter Klammern dargestellt. Darin sind die Beschreibungen der Attribute ”name” und ”students” geschachtelt. Die Beschreibungen der Studentenlinks wird durch eckige Klammern eingeschlossen, die die Beschreibung der zwei Studenten-Objekte umfassen. Dieses Beispiel wurde mit bidirektionalen Kanten realisiert, wo die Rückkante mit ”@JsonIgnore” markiert wurde.

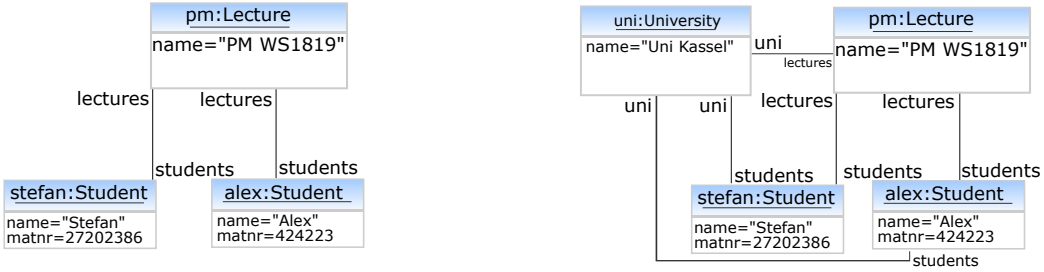

Abbildung 19: baumartige und zyklisches Objektmodelle

```
{
  "name" : "PM WS1819",
  "students" : [ {
  "name" : "Alex",
  "matnr" : 424223,
  }, {
    "name" : "Stefan",
    "matnr" : 27202386,
  } ]
}
```

Quellcode 6: Json Jackson 2.10.0-SNAPSHOT

Diese geschachtelte Darstellung ist nur möglich, wenn es keine Querkanten (Zyklen) im Objektmodell gibt. Zyklen und bidirektionale Verbindungen stellen für die Standardimplementierungen zur Serialisierung von Modellen ein großes Problem dar. Dieses Problem wird in der Regel an den Datenarchitekten verschoben, welcher das Klassenmodell mit Transitionskanten und unidirektionalen Kanten versehen soll.
Bei Jackson ohne zusätzliche Annotationen führen Zyklen oder bidirektionale Kanten zu dem Fehler "StackOverflow". Eine Möglichkeit zur Darstellung von Querreferenzen ist die Verwendung von relativen oder absoluten "Pfaden" innerhalb der Baumstruktur wie es bei EMF üblich ist (siehe Quellcode 7).

```
<University xsi:type="University"
  name="Uni Kassel"
  student="/University/lecture.0/student.0
        /University/lecture.0/student.1">
  <lecture xsi:type="Lecture"
    name="PM WS1819"
    uni="/University">
    <student xsi:type="Student"
      name="Alex"
      lecture="/University/lecture.0"
      matnr="424223"
      university="/University"/>
    <student xsi:type="Student"
      name="Stefan"
      lecture="/University/lecture.0"
      matnr="27202386"
      university="/University"/>
  </lecture>
</University>
```

Quellcode 7: XMI Beispiel bidirektionale Kanten

In Google Json kann man exklusive Regeln mittel der Methode ”setExclusionStategies” im JSONBuilder setzen. Diese Methode wird bei der Serialisierung für jeden Link aufgerufen. Somit ist dies die perfekte Stelle für das Ignorieren der Rückkante.

Ab der Version 2.0 in Jackson gibt es die Annotation für ”Identitäts-Referenzen”. Bei der Generierung muss explizit mittels ”@JsonIdentityReference” und ”@JsonBackReference” die Richtungen angegeben werden. Eine andere Möglichkeit ist mittels ”@JsonIdentityInfo” eine ”ID” zu generieren.
Dieses kann mittels eines ObjectIdGenerators automatisch erstellt werden, oder mittels einer Resolver Klasse. Für das obige Szenario ist ein Resolver

denkbar, der die ”UUID” für die zu serialisierende Objekte erstellt. Allerdings müsste dieser mittels Singleton eingebunden werden, so dass die verschiedenen Nachrichten dieselbe ”ID” benutzen.
Bei der Serialisierung eines (baumartigen) Modells wird im Allgemeinen nur das Root-Objekt angegeben. Der Serialisierungsmechanismus durchläuft dann alle Kanten und sammelt dabei alle zum Modell gehörenden Kindobjekte ein. Dabei muss auf Objekte aus anderen Java-Bibliotheken aufgepasst werden zum Beispiel ”java.util.Date” oder ”java.io.File” wie im Quellcode 8. Jackson, GSON und NetworkParser rufen auf Objekten mit unbekanntem Typen die ”toString” Methode auf.

```
1 class Model_File {
2   private String name;
3   private java.util.Date date;
4   private File file;
5 }
```

Quellcode 8: Simple Model

Die bisherigen Lösungen sind nur bedingt für die Zielgruppe von Studierende aus dem dritten Semester geeignet. Es wurde also eine Lösung gesucht, welche keine Einschränkungen an das Klassenmodell stellt und wo der Anwender keine speziellen Annotationen benötigt, welche die Studierende in ihrem bisherigen Studium noch nicht gehört haben. Die IdMap vom NetworkParser unterstützt seit Anfang an bidirektionale und zyklische Assoziationen. Die Grundidee stammt von der Referenzimplementierung auf json.org.

Bei dem NetworkParser kann allerdings eine ”Creator”-Klasse als Reflektionsschicht hinzugefügt werden, um eine spezielle Serialisierung zu ermöglichen oder spezielle Attribute zu ignorieren.

Als Basis dient das SendableEntityCreator Interface siehe Quellcode 9. In diesem Interface sind die grundlegenden Dinge für die Beschreibungsklasse definiert. In der Beschreibungsklasse "UniversityCreator" sind diese implementiert (siehe Quellcode 10). Dort gibt es einen Getter "getProperties()" (Zeile 4) für die zu serialisierenden Attribute. Diese werden als Liste von Attributnamen zurückgegeben. Weiterhin muss es eine Möglichkeit geben eine neue Instanz zu erstellen (Fabrik- Pattern). Des Weiteren gibt es eine Möglichkeit für die Objekte zentral Werte zu schreiben "setValue()" (Zeile 15) und zu lesen "getValue()" (Zeile 8). Somit kann die IdMap jedes beliebige Objekt verwalten und persistieren.

```
1 public interface SendableEntityCreator {
2   public String[] getProperties();
3   public Object getValue(Object entity, String attribute);
4   public boolean setValue(Object entity, String attribute,
        Object value, String type);
5   Object getSendableInstance(boolean prototyp);
6 }
```

Quellcode 9: SendableEntityCreator

```
1  public class UniversityCreator implements
       SendableEntityCreator {
2   private final String[] properties = new String[]
3    {University.PROPERTY_NAME, University.PROPERTY_STUDENTS};
4   public String[] getProperties() {return properties;}
5   public Object getSendableInstance(boolean reference) {
6    return new University();
7   }
8   public Object getValue(Object target, String name) {
9    if (PROPERTY_NAME.equalsIgnoreCase(name))
10     return ((University) target).getName();
11   if (PROPERTY_STUDENTS.equalsIgnoreCase(name))
12     return ((University) target).getStudents();
13   return null;
14  }
15  public boolean setValue(Object target, String attrName,
       Object value, String type) {
16   University item = (University)target;
17   if (PROPERTY_NAME.equalsIgnoreCase(attrName))
18     return item.setName((String) value);
19   if (REMOVE.equals(type) && value != null) {
20   if (PROPERTY_STUDENTS.equalsIgnoreCase(attrName))
21     return item.removeStudents((Student) value);
22   }
23   if (PROPERTY_STUDENTS.equalsIgnoreCase(attrName))
24     return item.addStudents((Student) value);
25   return false;
26  }
27  public static IdMap createIdMap(String sessionID) {
28   return CreatorCreator.createIdMap(sessionID);
29  }
30 }
```

Quellcode 10: UniversityCreator

Mit den bisherigen vorgestellten Serialisierungsmechanismen wird eine vollständige Beschreibung des gesamten aktuellen Objektmodells erstellt. In vielen Anwendungsfällen soll eine Reihe von Versionen eines Objektmodells gespeichert werden, wobei sich die einzelnen Versionen nur durch kleine Änderungen unterscheiden.
Daher wird in den sogenannten Model-Repositories meist eine deltabasierte Speicherung der Versionshistorie verwendet.
Eine Variante der deltabasierten Speicherung sind sogenannte Changelogs, in denen alle Änderungsoperationen, die auf dem Modell ausgeführt wurden mit protokolliert werden. Solche Changelogs werden in Datenbanken auch häufig für Recovery-Mechanismen verwendet.
Bei dem UseCase von EONShiftScheduling (siehe Abschnitt 3.6 auf Seite 32) war es eine Grundvoraussetzung, dass Modelländerungen mitprotokolliert werden. Dieses sollte eine Transparenz schaffen wer und wann das Datenmodell geändert wurde.

Multi-User Anwendungen verwenden Deltas oder Changelogs für die Synchronisation der Modelle von mehreren Anwendungen. Dieses vereinfacht die Entwicklung und der Kommunikationsaufwand wird minimiert.

In einem Changelog beziehen sich einzelne Changes meist auf ein Objekt und ein Attribut dieses Objekts, dass einem neuen Wert erhält. Meist wird auch noch der alte Attributwert protokolliert. Dies ermöglicht zum Beispiel ein Undo des Changes. Quellcode 11 zeigt ein Beispiel für eine JSON Serialisierung eines Changes an dem Objekt S1. Wie man sieht, wird für die Serialisierung von Changes wieder die ObjektIds verwendet.

```
{"class":"Student",
  "id":"S1",
  "rem":{"name":"Stefan", "credits":0},
  "upd":{"name":"StefanL","credits":42}
}
```

Quellcode 11: Json Change

Bei zyklischen Modellen oder Modellen mit bidirektionalen Kanten ist die Idee, für alle Objekte eine eindeutige ID zu generieren und diese ID für alle Links zu verwenden. In der IdMap gibt es daher eine Methode `"String getId(Object)"`, die zu einem Modellobjekt die zugehörige Objekt-ID liefert. Die Methode `"Object getObject(String)"` gibt zu einer ID das zugehörige Modellobjekt zurück. Falls ein Modellobjekt noch keine ID besitzt generiert die Methode `"String getId(Object)"` eine neue eindeutige ID. Intern verwendet die Klasse IdMap eine bidirektionale HashMap zur Verwaltung von Modellobjekten und der zugehörigen ID. Für diese bidirektionale HashMap stellt der NetworkParser eine speziell angepasste Klasse bereit (siehe Abschnitt 5.11 auf Seite 109).
Auf dieser Basis bietet der NetworkParser eine einfache Unterstützung für die Erstellung und Nutzung von Changelogs an.

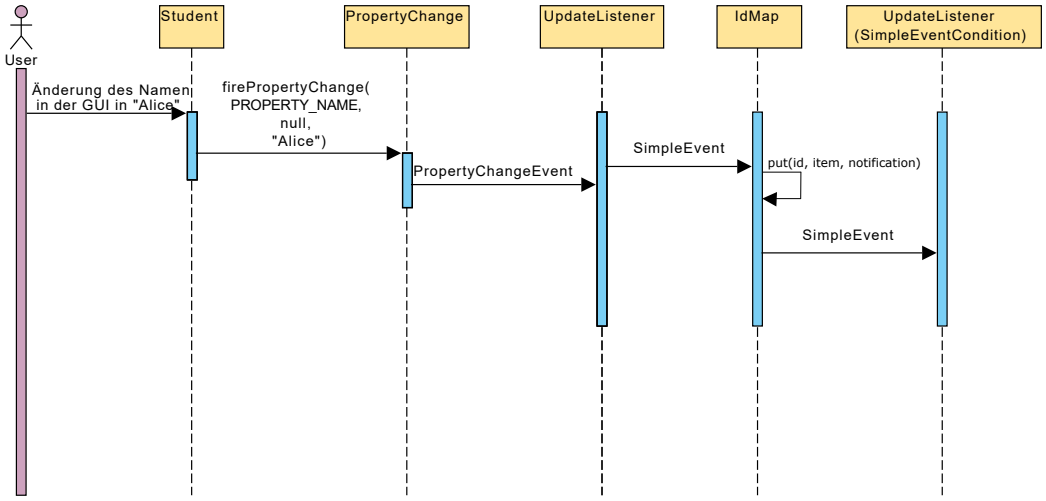

Abbildung 20: Sequenzdiagramm PropertyChange

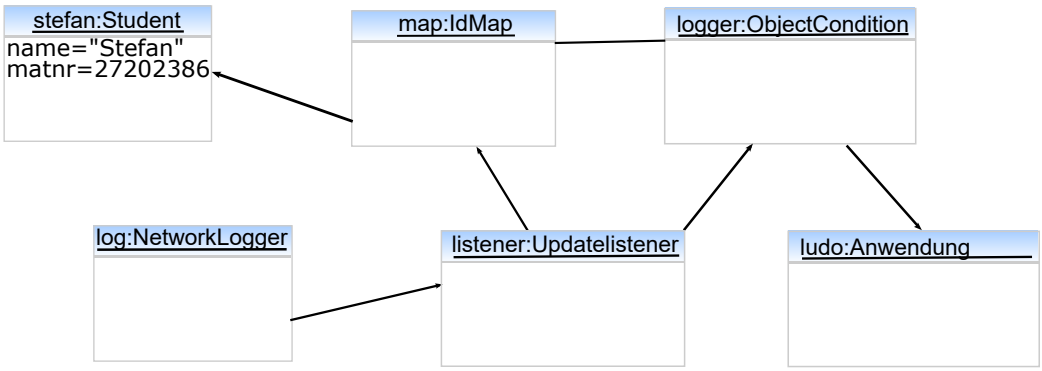

Abbildung 21: firePropertyChange

Für eine einfache Unterstützung der Änderungsnachrichten wurde das Verfahren des ”java.bean.PropteryChange”-Mechanismus verwendet. Wie in Abbildung 21 zu sehen meldet sich ein UpdateListener bei jedem Objekt des Modells an, welches in der IdMap bekannt ist und übermittelt die Änderungsnachrichten an das Objekt ”map:IdMap”. Die Anwendung kann sich dort ebenfalls als Listener registrieren und bekommt so die Änderungen als Json Nachricht übermittelt (ludo:Anwendung). Damit dieser Mechanismus funktioniert, ist es notwendig, das allen Setter-Methoden des Modells die Methode ”firePropertyChange()” aufrufen.
Zur besseren Wartbarkeit und Übersicht werden die Änderungen bei der Json-Nachricht in dem Attribut ”upd” verpackt. Falls ein Objekt entfernt werden soll, kommt das Attribut ”rem” zum Einsatz. In diesen Fall wird auch der Listener von dem entsprechenden Objekt entfernt. Bei neuen Objekten fehlen die ”rem” Attribute.

Ein Problem kann auftreten, wenn das Modell aus Teilen besteht, welches nicht oder nur teilweise serialisiert werden sollen. Bei EMF gibt es einen ”EContentAdapter” mit dem es möglich ist auf Änderungen des Datenmodells zu reagieren. Dieses ist allerdings nicht einfach umsetzbar. Dafür müssen die beiden Tutorial [Vog18, LVc18] beachtet werden. Ein anderes Verfahren ist mittels eines Treewalkers, welcher rekursiv über das Modell läuft zu realisieren [HFBO05]. Bei dem NetworkParser können spezielle Filter das Serialisieren oder deserialisieren verhindern. Weiterhin besteht die

Möglichkeit über die eigene Reflektionsschicht "Creator" Einfluss auf den Parser zu nehmen. Bei baumartigen Modellen können auch Pfade eingegeben werden welche Elemente persistiert werden sollen. Ansonsten steht dem Entwickler noch die Möglichkeit Instanzen oder Klassen per WhiteList für das Persistieren zu registrieren oder mit der BlackList von dem Persistieren auszuschließen.

Das vorgestellte Konzept zur Serialisierung von Objektmodellen und Modell-Änderungen basiert im Wesentlichen auf der Verwaltung von Objekt Ids in der IDMap. Die konkrete Serialisierung ist konzeptionell nicht auf Json beschränkt. Die eigentliche Serialisierung kann genauso gut im XML, YAML oder in einem Binärformat erfolgen. Aus Gründen der Flexibilität und der Kompatibilität mit anderen Ansätzen und Frameworks unterstützt der NetworkParser daher eine Vielzahl von Serialisierungsformaten. [Mae12] Die einzelnen Formate können mit verschiedenen "Tokener" Klassen (zum Beispiel Json, XML, YAML, SQL, Byte, Excel, CSV oder EMF) und der "Grammatikklasse" für Grammatikstrukturen eingestellt werden. So ist es einfach möglich zwischen mehreren Formaten zu wechseln (siehe Quellcode 12). Für die Serialisierung und Versionierung stehen weiterhin mehrere Filter zur Verfügung, welche auf dem "java.beans.PropertyChange"-Mechanismus basieren. Die Deserialisierung von Elementen wird mittels einer eigenen Reflektionsschicht realisiert. Somit können spezifische Formatierungen vorgenommen werden. Eine eigene 'cloneObject'-Methode ermöglicht es einzelne Objekte oder Objektstrukturen zu duplizieren ohne Reflektion. Mit diesen ist es rudimentär möglich Erreichbarkeitsgraphen zu erstellen. Dieses wurde speziell für den TTC 2014 umgesetzt. Durch diesen Mechanismus war SDMLib das beste Tool in der Kategorie "Performance" [ZLGE14b].

```
1 String xml="<de.uni.kassel.peermessage.model.ChatMessage
    sender=\"Stefan\" value=\"Test\"/>";
2 JsonObject testjson = new JsonObject().withEntity(new
    XMLEntity().withValue(xml));
3 String json="{\"class\":\"de.uni.kassel.peermessage.model.
    ChatMessage\","+
4   "\"prop\":{\"sender\":\"Stefan\",\"value\":\"Test\"}}";
5 Assert.assertEquals(json, testjson.toString());
```

Quellcode 12: Cross Parsing

Für das einfache Protokollieren wurde eine Klasse ähnlich SL4J implementiert und in der IDMap standardmäßig registriert. Der NetworkParser stellt mit der Klasse ”NetworkParserLog” ein zentrales protokollieren von Ereignissen bereit. Es wurde weiterhin das Interface ”org.slf4j.spi.SLF4JServiceProvider” bei der Klasse ”NetworkParserLoger4J” hinzugefügt. Diese Klasse steht in der Jar mit GIT zur Verfügung. Da hier eine Klasse von SL4J erweitert werden musste existiert somit eine direkte Abhängigkeit zu der Bibliothek ”slf4j-api-1.8.x.jar”. Mit der Implementierung steht dann ein Service für die Schnittstelle von SL4J dem Entwickler zur Verfügung.

5.3 Story

Bei Story Driven Modeling werden Szenarien in Tests umgesetzt. Hierbei wird die Startsituation beschrieben. Danach werden Aktionen definiert und es können beliebig viele Folgesituationen definiert werden.
Die Idee der Storyboard-Diagramme + Codegenerierung kommt aus der Diplomarbeit von Leif Geiger [Gei04]. Zusätzlich werden Storyboards bei SDMLib und NetworkParser als Dokumentation des Projektes verwendet. Bei SDMLib.org findet man zirka 160 Storys. Bei dem NetworkParser existieren bis jetzt drei.
Die Modellierung von Programmen fängt meist mit Use-Cases oder User-Stories an. Beide sind sehr ähnlich. User-Stories beschreiben das große Ganze und sollen zur Diskussion anregen, wogegen Use-Cases detaillierter und mehr den genaueren Ablauf beschreiben. Der NetworkParser unterstützt dabei den Entwickler durch die Klasse "Story". Diese umfasst die Möglichkeit verschiedene Schritte zu Dokumentationszwecken in einer HTML-Datei zusammenzufügen. Die Stories können dabei aus verschiedenen "Steps" wie Tests (JUNIT-Überprüfungen), Bilder, Texte, Sourcecodes, Diagramme und benutzerdefinierte Programmcodes bestehen (siehe Quellcode 13).

```
1  ClassModel ludoModel=getModel(); //new ClassModel() ...
2  Ludo ludo = getLudo(); //new Ludo() ...
3  Story story = new Story().withLabel("playing Ludo");
4  story.addText("This is the Ludo Model");
5  story.addDiagram(ludoModel);
6  story.addText("next Step is Testing");
7  story.add(new StoryStepJUnit());
8  story.addText("Result is Fine");
9  story.addSourceCode(1, 12);
10 story.assertEquals("PLayer must 2", ludo.getPlayer().getSize(),
       2);
11 story.addText("Mockup");
12 story.addImage("doc/gui_mockup.png");
13 story.dumpHTML();
```

Quellcode 13: Story Example

Es stehen einige Validierungsmethoden in der Story zur Verfügung. Diese überprüfen die aktuelle Situation per JUNIT-Framework. Das Framework wird mittels Soft-Dependencies hinzugefügt. Als Fallback wird die Logikkomponenten benutzt.

- assertEquals
- assertTrue
- assertFalse
- assertFail
- assertNotNull
- assertNull

Das Verknüpfen von verschiedenen Steps und Stories miteinander ist mittels StoryBook möglich. So ist das hierarchische Gliedern von Stories in beliebiger Tiefe und Granularität möglich. Dieses eignet sich speziell für Beschreibung von ganzen Tools mit einzelnen Stories als für spezielle Features.

5.3.1 StoryTest

Zur Verfeinerung der Stories wurden in der Diplomarbeit [Gei04] die Storyboards um Tests erweitert. Dieses wurde in der Grundidee in der Klasse GraphConverter in der Methode convertToTestCode nach implementiert. Diese Methode formt ein Objektmodell in ein Testcode Fragment um (siehe Quellcode 14 und 15), um diesen und die Attribute und Links zu testen. Dieses wurde nach der letzten Veranstaltung von "Software Engineering 1" entwickelt. Die Studierenden sollten ihr Programmcode testen, allerdings kam zu Recht der Einwand, dass der generierte Programmcode von NetworkParser nicht getestet werden muss. Bei der Bewertung wurde allerdings der C0 Prozentsatz von jeder Programmklasse betrachtet. Somit wurden Tests entwickeln, welche Programmcode erzeugen, um ein Modell zu testen.

```
1 ObjectModel model = new ObjectModel();
2 ObjectInstance alice = model.createObject("alice", "Person");
3 alice.createAttribute("name",DataType.STRING).withValue("Alice");
4 ObjectInstance uni = model.createObject("uni", "University");
5 uni.createAttribute("name",DataType.STRING).withValue("Uni Kassel
      ");
6 uni.withLink(alice, "student");
7
8 GraphConverter converter = new GraphConverter();
9 TemplateResultFragment convertToTestCode = converter.
      convertToTestCode(model, true);
```

Quellcode 14: Test Fragment

```
1 Story story = new Story();
2 University uni = new University();
3 story.assertNotNull("Object 'uni' NULL", uni);
4 uni.setName("Uni Kassel");
5 story.assertEquals("Attribute name wrong value", "Uni Kassel",
      uni.getName());
6 Person alice = new Person();
7 story.assertNotNull("Object 'alice' NULL", alice);
8 alice.setName("Alice");
9 story.assertEquals("Attribute name wrong value", "Alice", alice.
      getName());
```

Quellcode 15: Test Fragment

In der Diplomarbeit von Herrn Geiger wurden alle Tests als TestSuite und TestCases implementiert. Bei dem NeworkParser wurde das Interface TestListener hinzugefügt. Mit dessen Hilfe kann bei Ausführung von Tests zur Laufzeit der Objektbrowser angezeigt werden.

5.4 Testing

Dem Entwickler steht eine Test-Suite zur Verfügung, welche auf dem JUNIT-Framework aufbaut, mit deren Hilfe er automatische Tests ausführen kann. Mittels Blackboxtesting ist es möglich, automatisch seinen Sourcecode zu validieren. Hierfür wurde eine Klasse implementiert, welche alle Methoden des entwickelten Programmcodes mittels Reflektion aufruft.
Es können weiterhin einzelne Methoden oder Klassen ausgeklammert werden. Zur Verfügung stehen verschiedene Testkategorien. Die ermittelten Methoden können mit minimalem Wert, Null-Werten, maximalem Wert, einem zufälligen Wert oder individuell übergebenem Parameter aufgerufen werden. Somit wird probiert mittels der Grenzwerte und möglichst sinnvollen Werten verschiedene Szenarien automatisch zu testen.
Hierbei ist aber zu beachten, dass die Modelle nicht speziell nach ihrer Verwendung überprüft werden. Durch die einfache Überprüfung wird eine C0-Codeabdeckung des generierten Codes von 98 % erzeugt.

```
1 ReflectionBlackBoxTester.execute("de.uniks.ludo.model");
```

Quellcode 16: BlackboxTesting

Die Ergebnisse können entweder mit einem Listener programmatisch verarbeitet werden oder in einer Datei gespeichert werden. Weiterhin ist es möglich Tests und insbesondere den BlackboxTest in einer Story zu integrieren. So ist eine lückenlose Dokumentation gegeben. Die Ergebnisse können auch in die HTML des Jacoco Builds eingebunden werden.

5.5 JarValidator

Gerade bei Lehrveranstaltungen wie ”Software Engineering I” hat sich gezeigt, dass die Anforderungen nach guter Codequalität von den Studierenden von dem Tutor überprüft werden sollte. Hierfür wurde ein JarValidator geschrieben, welcher es ermöglicht, sämtliche Tests in einem Repository auszuführen. Diese werden im Nachhinein mittels CodeCoverage

[Jan18] (Testabdeckung in der Softwaretechnik) ausgewertet und neutral dargestellt. Dieses basiert auf dem Sourcecode (siehe Aufruf 17).

```
1 java -jar NetworkParser.jar JARVALIDATOR time=6000 path=
    build/libs fatjar coverage=70
```

Quellcode 17: JARVALIDATOR

In "Software Engineering I" wird von den Studierenden verlangt, einen Spielclient zu schreiben mit sechs Entwicklern. Da hier eine reelle Situation nachempfunden wird, inklusive Unternehmensstruktur und Kunden, wird auch das Ergebnis als vollwertiges Produkt betrachtet. So muss der Produktowner das Produkt richtig bewerben und vermarkten, dazu zählen auch die enthaltenen Lizenzen.
Die Studierenden sollen eine FatJar erstellen, welche fehlerfrei ist und für den übersetzten Programmcode alle notwendigen Bibliotheken beinhaltet. Der JarValidator analysiert diese Datei und prüft anhand der Bibliotheken, welche Lizenzen verwendet wurden und ob alles Notwendige enthalten ist. Der Validator hat zusätzlich die Möglichkeit herauszufinden, ob in der abgegebenen Datei Elemente enthalten sind, die nicht zur Ausführung notwendig sind.

PomValidator

Für das Erzeugen von Releases (Rollout) bei Maven ist es notwendig eine gültige POM-Datei zu erzeugen. Diese wird von Gradle erzeugt. Allerdings dürfen zum Beispiel die Dependencies maximal mathematische Ausdrücke enthalten.

In Gradle verwendet man für Abhängigkeiten meist eine feste Version oder ein mathematischen Ausdruck wie 4.2.+ . Dieser führt allerdings zu einer falschen Erstellung der POM Datei. Richtig wäre hier [4.2). Dabei wird die untere Kante durch die '[' definiert und die Obergrenze für die Revision freigelassen.

Hierfür wurde ein POM-Validator integriert, welcher die notwendigen Konfigurationen einer POM-Datei überprüfen kann. Dieses war gerade enorm wichtig bei der Release Phase des NetworkParsers. Für das erste Release mussten innerhalb einer Woche einige Fehler korrigiert werden. Aus diesen Erfahrungen ist der PomValidator entstanden.

5.6 Code Generierung

Bei den Datenmodellen gibt es unzählige Codegeneratoren. Allein für Java existieren XCore [Ed.17], UMLLab [Gmb17b], Fujaba [FNTZ98], SDMLib [Zü15], Enterprise Architekt [Ltd18], Jenesis4Java [rit18], ANTLR [Par18], Jmr [Yu18], EMF [SBMP08], fulib [Zü18] und viele mehr. Eine kurze Übersicht findet sich auch in der Arbeit von Ruben Jubeh auf Seite 166 und folgende [Jub17]. Wichtig für eine gute Unterstützung und Akzeptanz der Benutzer vom NetworkParser ist eine einfache Bedienung des Codegenerators und eine gute Einbindung mit der Oberfläche (PropertyChange-Unterstützung) und Kommunikationskomponenten (Persistenz und Netzwerkverteilung). Durch die Implementierung eines eigenen Codegenerator konnten mehrere Anpassungen vorgenommen werden. So werden für alle Attribute und Assoziationen zusätzliche Methoden generiert. Für zu Many-Assoziationen werden eigene Set-Klassen erzeugt, welche zusätzlich als Reflektionsschicht und als zusätzliche Klasse für PatternMatching dienen. Zusätzlich wird eine eigene Klasse mit einer Hilfsmethode zum Erzeugen von der Persistenzklasse erzeugt.

In der Lehre und für die Erarbeitung von diversen Drittmittelprojekten hat sich herausgestellt, dass eine gut Datenmodellimplementierung sich durch bidirektionale und reihenfolgesichere Kanten und eine allgemeine Schnittstelle zum Serialisieren auszeichnet. Eine gute Generierung von bidirektionalen Kanten liefert aktuelle UMLab, SDMLib, NetworkParser, fulib und EMF. Die Lösung von EMF für Kanten ist sehr ineffektiv. Der NetworkParser hat das Codegenerierungskonzept von SDMLib übernommen. Für die Flexibilität wurde allerdings auf Templates umgestellt.

```
1 ClassModel model = new ClassModel("de.uniks");
2 Clazz uni = model.createClazz("Uni");
3 Clazz student = model.createClazz("Student");
4 uni.withAssoc(student, MANY);
5 model.generate("gen");
```

Quellcode 18: Einfache Assoziation

5.6.1 Assoziation

Bei der Befragung der Studierenden hat sich herausgestellt, dass gerade das manuelle Erstellen von Assoziationen für einen unerfahrenen Programmierer schwierig ist. Hierfür wurde die API nochmal erweitert. Bei dem Beispiel 18 wird eine Assoziation erstellt wie im Klassenmodell Abbildung 22 zwischen "University" und "Student" dargestellt. Hierbei wird automatisch eine bidirektionale Assoziation mit den Kardinalitäten "1 zu *" und den Rollennamen "students" und "uni" erzeugt. Um dieses zu vereinfachen wurde eine einfache Übersetzung für den englischen Rollennamen geschaffen, welche mittels den fünf Regeln einen plural Rollennamen erstellt. Die Rückkante wird automatisch mit "1"-Kante hinzugefügt.

Abbildung 22: Einfache Assoziation

In dem Metamodell sind für die Kardinalitäten Werte im Bereich "0 - *" und "0 - 1" möglich. Erstellt man eine Assoziation mit dem maximalen Wert 0 wird die Assoziation automatisch in eine unidirektionale Assoziation umgewandelt.

Weiterhin wurde eine bidirektionale Selbstassoziation erstellt, welche mit demselben Rollennamen umgehen kann. Dieses ist im Beispiel 23 dargestellt. Das hat allerdings die Einschränkung, dass beide Kardinalitäten den gleichen Wert besitzen.

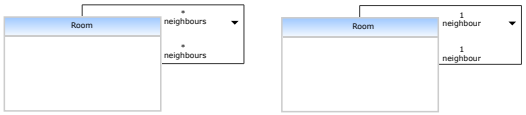

Abbildung 23: Self Assoziation

```
1 ClassModel model = new ClassModel("de.uniks");
2 Clazz room = model.createClazz("Room");
3 room.createBidirectional(room, 1);
4 //room.createBidirectional(rooms, 42);
5 model.generate("gen");
```

Quellcode 19: Selbstassoziation

5.6.2 dynamische Modelle

Bei der Codegenerierung existiert die Möglichkeit, die Datenmodelle dynamisch erweitern zu lassen.

```
1  private SimpleKeyValueList<String, Object> dynamicValues=
       new SimpleKeyValueList<String, Object>();
2  public Object getDynamicValue(String key) {
3    return this.dynamicValues.getValue(key);
4  }
5  public Student withDynamicValue(String key, Object value) {
6    this.dynamicValues.put(key, value);
7    return this;
8  }
9  public Object[][] getDynamicValues() {
10   return this.dynamicValues.toTable();
11 }
```

Quellcode 20: DynamicValues

So können unbekannte Argumente in jedem Element über eine KeyValue-Liste gespeichert werden. Damit ist eine Evolution des Datenmodells möglich. Dieses findet sich auch bei der funktionalen Programmiersprache Javascript wieder. So sind Szenarien möglich, wo das Datenmodell noch nicht komplett ausgearbeitet ist oder verschiedene Versionen einer Anwendung miteinander kommunizieren sollen. Die nicht zugeordneten Werte können im Datenmodell gespeichert werden. Diese werden dann bei der Persistierung und bei der Verteilung berücksichtigt. Für den außenstehenden ist kein Unterschied zwischen dynamischen und festen Attributen zu erkennen. Die erzeugten Methoden sind im Beispiel 22 aufgelistet.

5.6.3 Set Klassen

Der NetwerkParser erzeugt auch Hilfsklassen für ”0..*” Assoziationen. Dieses hat mehrere Vorteile. So wurden in den Klassen einige Methoden implementiert um unnötige Überprüfungen von Null-Werten zu vermeiden.

Weiterhin unterstützt die Klasse ein Methoden-Chaining und der Hauptgrund ist die Implementierung einer eigenen Set-Klasse. Diese ist auf Geschwindigkeit und Speicher optimiert. Dieses Konzept wurde von SDMLib übernommen. Diese Hilfsklassen basieren standardmäßig auf der SimpleSet-Klasse (siehe OCL-Ausdruck 21 und Quellcode 22). Sie können aber auch auf anderen Collections aufbauen. Sie dienen zum Beispiel als Speicher für mehrere Instanzen. Die Set-Klassen sind bestens für "Many" Assoziationen optimiert. Es wurde zusätzlich eine raw-Methode implementiert, um eine optimale Unterstützung von "1 ..*" zu erzielen. Diese erlaubt es beim Hinzufügen immer auf der zu 1 Kante zu überprüfen, ob der Eintag schon gesetzt ist. Dieses vermeidet eine Überprüfung in der Set-Klasse.

```
1 context University
2 inv: Set{Set{self->collect(rooms)}.collect(students)}.
     forAll(s->s.setCredits42)
```

Quellcode 21: OCL Chaining

```
1 //Stream
2 uni.getRooms().stream().map(p -> p.getStudents())
3   .forEach(s->s.forEach(st->st.setCredits(42)));
4 //SDMLIB
5 uni.getRooms().getStudents().setCredits(42);
```

Quellcode 22: DynamicValues

Die Set-Klasse besitzt die `createIdMap(String session)`-Methode (siehe Quellcode 23 und Abschnitt 5.2), mit dieser kann eine IdMap erzeugt werden, welche zum Beispiel für das Persistieren notwendig ist.

```
1 public static IdMap createIdMap(String session) {
2   return CreatorCreator.createIdMap(session);
3 }
```

Quellcode 23: CreatorCreator Method

Bei der aktuellen Codegenerierung wurden die Creator- und Set-Klassen vereint. So dienen die Set-Klassen als Reflektionsschicht um mittels `getValue(Object entity, String attribute)` und `setValue(Object entity,String attribute,Object value,String type)` Werte in dem Modell zu setzen oder auszulesen (siehe Quellcode 24). Das PatternMatching benutzt intern auch diese Klasse. Dieses hat den Vorteil den Studierenden den Umstieg auf PatternMatching und Graphersetzungsregeln zu erleichtern (siehe Abschnitt 5.7 auf Seite 90).

```
1  public Object getValue(Object entity, String attrName) {
2    if(entity==null||entity instanceof University==false){
3      return null;
4    }
5    University item = (University) entity;
6    if(University.PROPERTY_NAME.equalsIgnoreCase(attribute)){
7      return item.getName();
8    }
9    return null;
10 }
11 public boolean setValue(Object entity, String attrName,
       Object value, String typ) {
12   if(entity==null||entity instanceof University==false){
13     return null;
14   }
15   University item = (University) entity;
16   if(University.PROPERTY_NAME.equalsIgnoreCase(attrName)){
17     return item.setName((String) value);
18   }
19   return false;
20 }
```

Quellcode 24: Setter and Getter-Method

5.6.4 UML-Graph-Metamodell

Im Folgenden sind die Java-Klassen des Modells für die Repräsentation des NetworkParser aufgelistet. Sie repräsentieren das komplette Metamodell der Code Generierung. Somit ist es möglich, ein neues bzw. bestehendes Modell auf abstrakter Weise darzustellen.

- Annotation
- Association
- Attribute
- Clazz
- DataType
- DataTypeMap
- DataTypeSet
- Feature
- GraphEntity
- GraphImage
- GraphLabel
- GraphList
- GraphMetric
- GraphModel
- GraphNode
- GraphOptions
- GraphPattern
- GraphPatternChange
- GraphPatternMatch
- GraphSimpleSet
- GraphTokener
- GraphUtil
- Import
- Literal
- Match
- Method
- Modifier
- ModifyEntry
- Parameter
- Pattern
- SourceCode
- Throws
- Value

Um ein automatisches Anzeigen zu vereinfachen wurde eine ”FileClassModel”-Klasse geschrieben, welche von der ”ClassModel”-Klasse erbt. Diese ermöglicht ein automatisches Erstellen des Metamodell aus Dateien.

5.6.5 Java Converter

Klassendiagramme oder genauer Klassenmodelle, die auf den gerade beschriebenen Metamodellen aufbauen können in vier Textformate überführt werden. Zum einen existiert ein YUML-Converter, welcher ein Klassendiagramm in die Struktur des Onlinekonverter YUML umwandeln, YUML.me [Yum15] ist ein verbreiteter Webservice. Das Beispiel vom Quellcode 32 wird in den Text 25 umgeformt.

```
1 [Ludo|name:String|init()]-[Player|name:String;color:String]
```

Quellcode 25: yUML Modell

Dieser Webdienst besitzt eine einfache Grammatik und kann Diagramme mittels URL-Request zurückgeben.

Weiterhin existiert auch ein Json-Converter, welcher ein Diagramm in eine Jsonstruktur umwandelt. Die so entstandene Struktur wird für das hier beschriebene DiagramJS-Framework benötigt. Die Grammatik ist unterteilt nach Nodes und Edges (siehe Abbildung 24). Das Klassenmodell

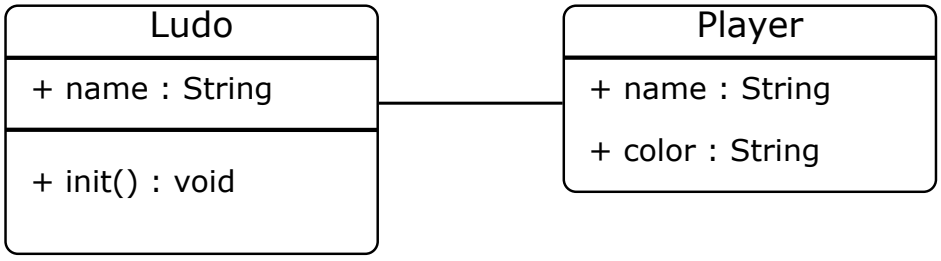

Abbildung 24: University Example

kann zusätzlich noch um zahlreiche Informationen angereichert werden. Diese können beispielsweise Differenzen zwischen mehreren Versionen oder Codemetriken beinhalten. Die Differenzen können für die Bewertung für die Lehrveranstaltung "Programmiermethodik" verwendet werden. Die Metriken können dann aufbereitet angezeigt werden [WLR11]. CodeCity ist eine integrierte Umgebung zur Softwareanalyse, in der Softwaresysteme als interaktive, navigierbare 3D-Städte visualisiert werden. Die Klassen werden

als Gebäude in der Stadt dargestellt, während die Pakete als die Stadtteile dargestellt werden, in denen sich die Gebäude befinden. Die sichtbaren Eigenschaften der Stadtartefakte stellen eine Reihe von ausgewählten Softwaremetriken dar. Es wurde ein Converter geschrieben, welcher das Metamodell mit diesen Metriken zum Beispiel LoC und InBounds in das MSE Format [Nie18] überführt, sodass das Tool CodeCity Javacode wie von dem NetworkParser darstellen kann (siehe Abbildung 25).

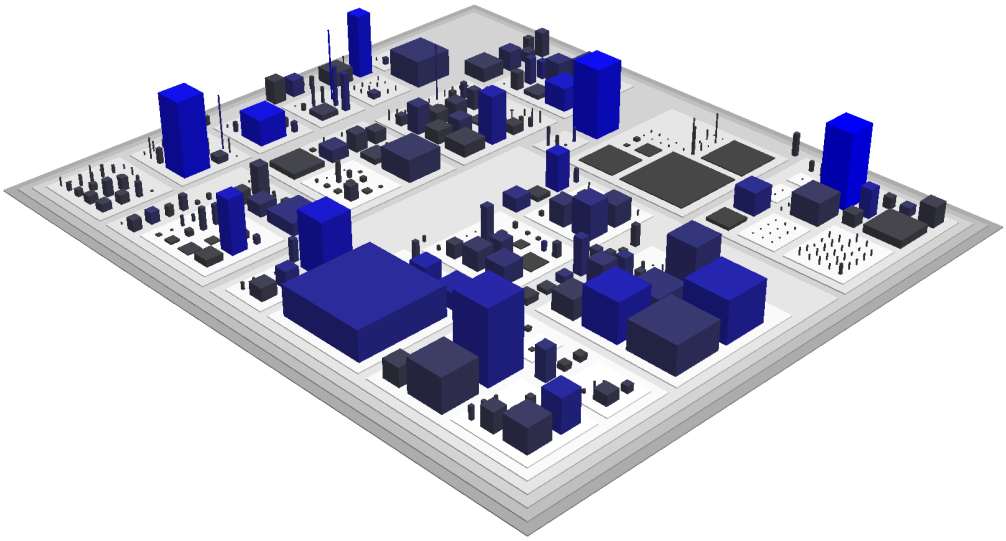

Abbildung 25: NetworkParser CodeCity

Für die Kompatibilität mit anderen Tools wurde außerdem ein Dot-Converter integriert, welcher das Klassenmodell in die ”Dot-Language” umwandelt. Diese Struktur wird unter anderem in dem Tool GraphViz benutzt [EGK+02] (siehe Quellcode 26 und Abbildung 26).

```
1  digraph ClassDiagram {
2     node [shape = none, fontsize = 10, fontname = "Arial"];
3     edge [fontsize = 10, fontname = "Arial"];
4     compound=true;
5
6  University[label=<<table border='0' cellborder='1'
       cellspacing='0'><tr><td><b>University</b></td></tr>
7  <tr><td><table border='0' cellborder='0' cellspacing='0'>
8  <tr><td align='left'>name : String</td></tr>
9  </td></tr>
10 <tr><td><table border='0' cellborder='0' cellspacing='0'><
       tr><td align='left'>init()</td></tr>
11 </table></td></tr></table>>]
12 Student[label=<<table border='0' cellborder='1' cellspacing
       ='0'><tr><td><b>Student</b></td></tr>
13 <tr><td><table border='0' cellborder='0' cellspacing='0'>
14 <tr><td align='left'>name : String</td></tr>
15 <tr><td align='left'>color:String</td></tr>
16 </table></td></tr></table>>];
17 University[taillabel = "students"] -> Student[taillabel = "
       uni"];
18 }
```

Quellcode 26: Dot Example

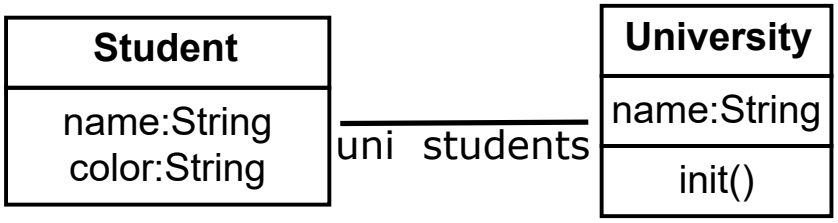

Abbildung 26: Dot Example

5.7 Pattern Klassen

Die Programmlogik zu erstellen ist eine herausfordernde Arbeit und somit wurde probiert wie schon in Fujaba und SDMLib eine eigene Teilsprache für Modell-Queries und Modelltransformationen anzubieten. In dem Beispielcode 27 sieht man das neue Verfahren, welches mittels der eigenen CollectionKlassen funktioniert. Somit ist eine Typsicherheit gegeben und es mussten keine eigenen Pattern-Klassen generiert werden wie in SDMLib. Für das besondere Verwenden der Set-Klasse wurde der Listener einer Set-Klasse gesetzt, welche anhand des Patterns die Collection verwendet. In der Standardverwendung der Pattern-Klassen sind mehrere identische Werte in einer Collection erlaubt.
Der momentane Ansatz umfasst wieder eine textuelle Lösung und soll im Nachgang in dem Editor modelliert werden können. Die PatternMatching funktionieren wie eine WhiteList. Hier wird ein Objektmodell modelliert, welches dann auf das gesamte Modell angewendet wird, um mögliche passende Stellen zu finden. Weiterhin kann auch mittels PatternObject einfache Änderungen wie das Setzen von Attributen oder das Setzen von Links getätigt werden.

```
1 University uni = new University();
2 PatternCondition patternCondition = PatternCondition.
    createPatternPair(uni);
3 UniversitySet set = patternCondition.getRoot();
4 RoomSet rooms = set.getRooms();
5 AssignmentSet assignments = rooms.getAssignments();
6 double sum = assignments.getPoints().sum();
```

Quellcode 27: PatternMatching

5.8 Template Codegenerator mit Reverse Engineering

Es wurde eine eigene Templatesprache entwickelt, welche auf der Grundidee des Generators von SDMLib basiert. Der dort entwickelte Generator hatte den Vorteil, dass die Templates in dem Generator selbst definiert wurden, so dass eine schnelle Entwicklung anhand von Beispielcode realisiert werden konnte. Allerdings musste für eine Weiterentwicklung der Generator immer neu übersetzt werden und die Templates hatten immer sprachspezifische Anteile. Der NetworkParser trennt die Templates von dem ausführbaren Generator. Es hat sich gezeigt, dass die Einbindung von Fremdbibliotheken zur Generierung zu diesem Zeitpunkt (2010) zu umständlich und zu unflexibel war. Die Generatoren waren nur als externe Fremdbibliotheken verfügbar und führten zu Problemen bei dem Installieren bei den Studierenden.

Der Generator basiert auf Templates, diese werden eingelesen und in eine logische Struktur überführt. Diese dienen als Vorlage für die zu erzeugenden Java Dateien. Die einzelnen Generierungsschritte sind in Abbildung 27 dargestellt. Die Templates sind an handlebars [Kat18] angelehnt (siehe Quellcode 28). Die Templates untergliedern sich in Template-Fragmente. Die Template-Fragmente sind mit Platzhaltern versehen, um eine gute Übersichtlichkeit zwischen statischen und dynamischen Templateteilen zu bilden. Diese Platzhalter sind mit doppelten geschweiften Klammern eingegrenzt. Weiterhin besteht auch die Möglichkeit Programmcodes explizit vor der Codegenerierung zu schützen. Hierfür muss nur der Kommentar `//XXX NOGEN` eingefügt werden. Dieser verhindert erneutes Überschreiben. Der Code-Generator erzeugt anhand von dynamischen Templates Sourcecode-Fragmente. Diese können vor der Persistierung (Schreiben von Sourcecode-Dateien) noch angepasst werden. Im Beispiel 28 ist das Template für eine Java Methode abgebildet.

```
{{#foreach {{parameter}}}}
  {{#if {{#AND}}{{item.typeClazz.type}}==class
    {{#NOT}}{{item.packagename}}=={{file.member.packagename}}
    {{#ENDNOT}}{{#ENDAND}}}}
    {{#import {{item.type(false)}}}}
  {{#endif}}
{{#endfor}}
{{visibility}} {{modifiers} }{{returnType}} {{name}}{{parameterName}}
  {{#if {{file.member.type}}==interface}};
  {{#else}}
    {{#methodbody}}
  {{#endif}}
```

Quellcode 28: Template Beispiel Methode

```
{{#if {{#feature DYNAMICVALUES}}}}
  {{#import "+SimpleKeyValueList.class.getName()+"}}
  private SimpleKeyValueList<String, Object> dynamicValues=
      new SimpleKeyValueList<String, Object>();
  public Object getDynamicValue(String key) {
    return this.dynamicValues.getValue(key);
  }

  public {{name}} withDynamicValue(String key, Object value) {
    this.dynamicValues.put(key, value);
    return this;
  }
  public Object[][] getDynamicValues() {
    return this.dynamicValues.toTable();
  }
{{#ENDIF}}
```

Quellcode 29: Template DynamicValues

Weiterhin besteht auch die Möglichkeit sich bei jedem Element als Listener zu registrieren, so dass Codeschnipsel während der Generierung noch angepasst und verändert werden können.

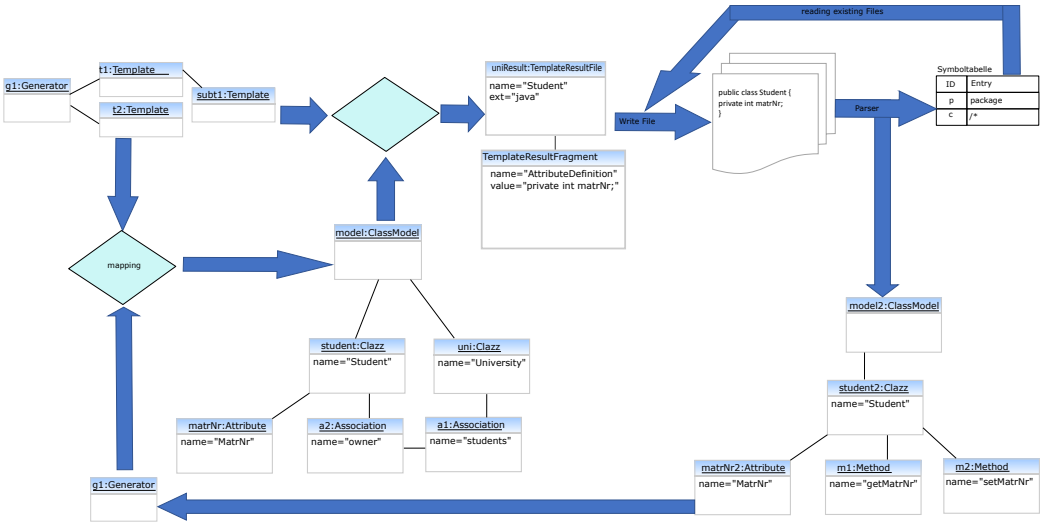

Abbildung 27: Generierungsschritte+Reverse

In der Abbildung 27 ist der komplette Ablauf der Generierung aufgezeichnet. Die Templates werden aus Dateien geladen und mittels Metamodell in Fragmente überführt. Im Standardfall werden die bereits existierenden Dateien in das Modell eingepflegt und aus diesen werden die Java-Dateien geschrieben.

Der Parser kann die Dateien wieder einlesen. Das so entstehende Metamodell unterscheidet sich von dem ursprünglichen Metamodell. Dieses sieht man an dem Attribut "MatrNr". Dieses wird zum einen als Attribute hinzugefügt, aber die gesamten Zugriffsmethoden werden einzeln hinzugefügt. Somit ist der Parser unabhängig von den Templates und kann ausgetauscht werden. Das so entstandene Metamodell kann mittels des Template auf das alte Metamodell abgestimmt werden. Hierfür werden die Templates erneut mit dem geparsten Metamodell ausgeführt und die Ergebnisse verglichen. So können doppelte Methoden herausgefiltert werden.

5.9 Diagramme als HTML darstellen

Bei der Entwicklung von objektorientierten Anwendungen hat sich gezeigt, dass es notwendig ist sich die Veränderungen von Datenmodellen der

Anwendungen, zu Testzwecke und für Dokumentationszwecke, visuell aufzubereiten. Somit wurde ein Framework entwickelt, welches ein Datenmodell darstellen und layouten kann.
Bei der Entwicklung wurde Wert daraufgelegt, dass möglichst eine Darstellung gefunden wird, welche layoutstabil und trotzdem ohne Überlappungen dargestellt werden kann. Hierfür wurden das Dagre-Javascript Framework von Chris Pettitt [cpe15] adaptiert, welcher auf dem Dagre-Algorithmus von GraphViz aufbaut. Einen passenden Layout-Algorithmus zu finden gestaltet sich als sehr schwierig, da es in der Softwareentwicklung und UML sehr viele verschiedene Diagrammarten gibt. Die Grundsätze für die Wahl von einem passenden Layout finden sich im Buch Software Visualisierung [Die07]. Die meisten Layoutalgorithmen gehen von Knoten aus, welche eine gleiche Größe Besitzen, dieses ist bei Klassen- und Objektdiagrammen anders. Wenn man nach Layout-Algorithmen im Internet sucht, gibt es unzählige Lösungen in verschiedenen Programmiersprachen und Ausprägungen. Eine binäre Lösung ist GraphViz [EGK+02] [GKNV93] und yFiles [yWo15]. Diese sind durch ihre Ausprägung entweder als Webservice oder lokal einsetzbar. Einige Lösungen sind in der Bachelorarbeit [Sch14] gut verglichen. Interessant sind die Lösungen, die betriebssystemunabhängig sind und auf Javascriptbasis funktionieren. Jointjs [Joi15], Cola.js [Mar15] und Soyatec [Soy15] sind solche Lösungen. Diese besitzen zwar die Darstellungsmöglichkeit, jedoch können sie eine vorhandene Knotenwolke mit unterschiedlich großen Knoten nicht darstellen. Dieses Problem besitzt auch das D3-Framework [Bos15]. Hier gibt es eine Reihe von Algorithmen, die die Berechnung des Layouts realisieren. Zwei davon sind Dagre [cpe15], Klay [Klo12, Ope15b]. Die entstehenden Graphen sind eine Mischung aus SVG und HTML.

5.9.1 Webview

Oracle hat eine native Webview mit WebEngine [jav12] implementiert, mit deren Hilfe es möglich ist die GUI einer Desktop-Applikation als lokal gehostete Webseite (im Speicher liegende, zur Laufzeit generierte

Webseite) zu implementieren. Eine Webengine dient zum Rendern von HTML-Inhalten, meist Browser-Engine genannt. Die Programmoberfläche steht dem Endanwender als eine html5 basierte Webseite zur Verfügung.

Oracle hat somit eine bidirektionale Verbindung zwischen Java und JavaScript erreicht. Diese bidirektionale Verbindung ermöglicht es aus der Sandbox der Webseite (Webengine) auszubrechen und direkt Java Code aufzurufen. Damit funktionieren moderne Webanwendungen wie Desktopanwendungen. Die Webengine ist unterschiedlich implementiert, existiert allerdings in der Java Runtime für den Desktop und für Android. Bei OpenJDK ist JavaFX nicht standardmäßig enthalten. JavaFX ist selbst Open-Source und wird in der Community von OpenJDK gepflegt.
Die Anzeigekomponente des NetworkParser kann also auch als zusätzliche Bibliothek zu OpenJDK hinzugefügt werden und steht somit zur Verfügung.

5.9.2 HTML

Tim Berners-Lee hat 1990 [FGM+99] das HTML erfunden, welches angelehnt ist an "Standard Generalized Markup Language" (SGML) [Cor18] ist. Die Weiterentwicklung HTML5 und CSS3 [Hog11] ist für Offline-Anwendungen gedacht. HTML5 erlaubt Javascriptcode eingebettet in HTMLCode. Das "Document Object Model" (DOM) repräsentiert das HTML Dokument. Der Javascriptcode kann den (DOM) [Hé18] manipulieren. Durch geschicktes Hinzufügen von Elementen zum DOM-Baum können die GUI Elemente des NetworkParser angezeigt werden. So sind die interaktiven Elemente gut für Anwendungen geeignet. Weiterhin können Webanwendungen bis zu 5 MB Anwendungsdaten in einer browserlokalen Datenbank speichern. Das Input-Element wurde um verschiedene Typen erweitert, zum Beispiel zur Eingabe von Suchbegriffen, Texten, Datums- und Zeitangaben, Telefonnummern, URL- und E-Mail-Adressen, numerische Werte sowie Farbangaben. Diese Elemente beinhalten auch clientseitige Validierung wie numerische Felder oder EMail-Felder. Auch existieren native Unterstützungen wie der ColorPicker.

5.9.3 Bridge

Für die Entwicklung von Desktopanwendungen mit HTML basierter GUI bietet der NetwerkParser eine Bridge an, die zwischen den Javascript-Anteilen der GUI und den Java-Anteilen in der Applikationslogik vermittelt. Für die Oberfläche einer Anwendung wurde eine allgemeingültige Darstellung von HTML-Elementen gewählt, welche nicht von der Programmiersprache abhängig ist. Hierfür wurde eine Java-Bridge und eine JavaScript-Bridge geschrieben. Mit dessen Hilfe ist ein einfaches Benutzen der bereits implementierten Bedienelemente und das bidirektionale Verknüpfen von Bedienelementen in JavaScript und dem Listener in Java möglich siehe Quellcode 30. Zum Darstellen von neuen Controls kann man entweder manuell eine neue Instanz eines Controls erstellen und der Bridge mitteilen oder dieses dynamisch über Json laden lassen.

Bei SDMLib [Zü15] werden die Diagramme mit dem Graph-Control gezeichnet. Die Bridge besitzt eine Control-Fabrik zum automatischen Erstellen von Controls. Die Fabrik kann dynamisch um neue Control-Typen erweitert werden.

```
1 public class TestBridge {
2   public static void main(String[] args) {
3     SimpleController controller = SimpleController.create(
          new JavaBridgeFX(), null, true, false);
4     controller.getBridge().enableFirebug();
5     controller.getBridge().addControl(new Button().
          withValue("Click Me"));
6   }
7 }
```

Quellcode 30: Bridge Beispiel

5.9.4 Controls

JavaScript in der Version 5 ist am meisten verbreitet, jedoch bietet erst die Version 6 die Möglichkeit objektorientiert zu arbeiten. Hier schließt Typescript und Coffeescript die Lücke. Typescript wurde von Microsoft entwickelt und ist eine eigene DSL. Es bietet die Möglichkeit Klassen inklusive Vererbung zu nutzen. Die Bedienelemente der Oberfläche wurden in Typescript entwickelt. Alle implementierten Controls erben von der abstrakten Typescript-Klasse Control. Diese können mit einer Klasse in Java verbunden werden. Somit ist das einfache Verwenden von diesen Bedienelementen gewährleistet. In Java wurden die folgenden generischen Struktur-Controls implementiert:

Label

Das Label-Item kann direkt mit einem Modellelement verknüpft werden und dient zum Anzeigen von Texten. Diese werden als Div-Element mit Text in den DOM-Baum eingepflegt.

BR

Das Elemente BR dient zum Formatieren von Webseiten und fügt Zeilenumbrüche hinzu.

DIV

Das DIV-Control ist ein Containerelement und dient zum Gruppieren von Elementen.

FORM

Das Form-Element dient zum einfachen dynamischen Bearbeiten von Modellelementen. Es konfiguriert sich automatisch anhand des Elements. Dieses eignet sich hervorragend für Eigenschaftsfenster oder Konfigurationsoberflächen (siehe Abbildung 28 links).

Input

Das Input-Control unterstützt sämtliche Eingabefelder und kann direkt mit

Datenmodellelementen verknüpfen werden. Der Typ kann automatisch von dem Datentyp des Modelles konfiguriert werden.

Symbol

Das Symbol-Control dient zum systematischen Anzeigen von SVG Elementen. Es existieren hier schon eine Reihe von vorgefertigten Elementen. Diese bilden jetzt bereits einen Grundstock für weitere Diagrammarten wie Use-Cases oder User-Stories. Die momentan vordefinierten Symbole sind:

- Hamburger
- Smiley
- Database
- Letter
- Max
- Wall
- Actor
- Lamp
- Stop
- Min
- Arrow
- Class
- Button
- Dropdown
- Classicon
- ClassWithEdgeicon
- Mobilephone
- Edgeicon
- Copynode
- Basket
- Pencil

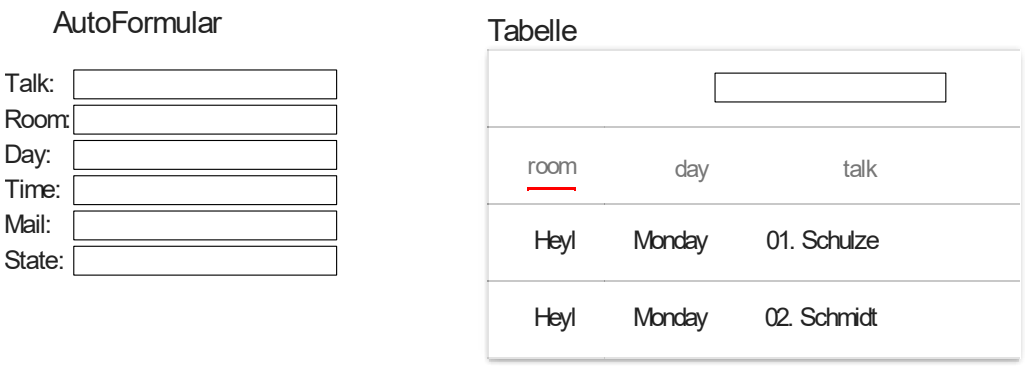

Abbildung 28: HTML Controls Links Formular, Rechts Tabelle

Table

Das Table-Control ist das komplizierteste Control (Abbildung 28 rechts). Mit dessen Hilfe ist es möglich eine tabellarische Ansicht von Listen von Anwendungsmodellelementen. zu erstellen. Es bietet eine Suchfunktion und eine Sortierfunktion. Das Eingabeelement bietet eine einfache Möglichkeit mittels des Inlineeditor Elemente zu bearbeiten (siehe Quellcode 31).

```
1  public class ClickCounter {
2    public static void main(String[] args) {
3      JavaBridgeFX javaBridge = new JavaBridgeFX();
4      SimpleController controller = SimpleController.create(
           javaBridge, null, true, false);
5      SimpleObject blub=new SimpleObject();
6
7      NumberField numberField=new NumberField().withElement(blub);
8      Button button = new Button().withValue("add");
9      javaBridge.addEventListener(button, EventTypes.CLICK,
10       new ObjectCondition() {
11       public boolean update(Object value) {
12        return blub.setValue("value",blub.getValue("value")+1);
13       } });
14     javaBridge.addControl(numberField, button);
15   }
16 }
```

Quellcode 31: ClickCounter Beispiel

Virtual Keyboard

Um möglichst für alle zukünftigen Plattformen und Eingaben vorbereitet zu sein wurde eine virtuelle Tastatur implementiert. Eine virtuelle Tastatur existiert auch als Javascript Implementierung oder es kann unter Android die mitgelieferte Tastatur benutzt werden. Bei einer vordefinierten Lösung muss allerdings komplett die Rückrichtung und Anzeige angepasst werden. Bei Windows steht eine eigene Bildschirmtastatur-Anwendung zur Verfügung und ist nicht Bestandteil der eigenen Anwendung.

Dieses ist gerade für mobile Einsatzzwecke mit Touchoberflächen sinnvoll. So können relativ einfach mobile Endgeräte integriert werden (siehe Abbildung 29).

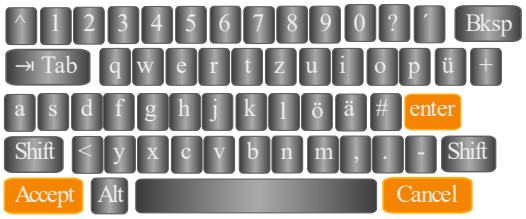

Abbildung 29: Virutal Keyboard

5.10 DiagramJS

Der neue Ansatz vom NetworkParser Klassendiagramme und Objektdiagramme mit einer Inline-DSL zu erstellen hat allerdings den Nachteil, dass nicht eingearbeitete Personen den Programmcode der Inline-DSL erst verstehen müssen. Die Diagramme führen nicht nur zu einem besseren Verständnis, sondern es hat sich gezeigt, dass es auch einige Studierende gibt, welche besser mit Diagrammen als mit Programmcode umgehen können. Deswegen wurde relativ früh eine Darstellung von Diagrammen in den NetworkParser integriert (seit 2014).
Bei der Entwicklung wurde ein Typescript-Framework erstellt, welches eine Graphdarstellung mittels anpassbaren Layouts erzeugen kann. Die so

erstellten Diagramme sind nicht nur einfache Abbildungen, sondern sind interaktiv nutzbar. So sind die Knoten per "Drag and Drop" verschiebbar. Somit kann das berechnete Layout individuell angepasst werden.

Für die Graphdarstellung wurde die von mir betreute Bachelorarbeit von Sergej Brataschow eingebunden [Bra18]. Diese wurde in Typescript umgesetzt und nach JavaScript übersetzt. Der Kern des Frameworks umfasst einen Diagrammeditor mit einem Eventbus. Das Framework ist mit einem Plugin-Factory Pattern versehen, so dass es ohne Probleme erweitert werden kann.
Das Framework "DiagramJS" wurde objektorientiert in Typescript entwickelt und im Package "elements" wurden die Darstellungsanteile von den Eingabeelement-Klassen integriert. "DiagramJS" wird mittels Webpack [TKc18] nach Javascript übersetzt und in eine einzige Datei zusammengefasst. Die diagram.js dient zum Erstellen einer internen Diagrammstruktur und zum Übergeben der als Json vorliegender Diagramme an den eingestellten Layout-Algorithmus. Der standardmäßig benutzte Layout-Algorithmus ist der dagre-JS [cpe15], welcher unter anderen im Tool GraphViz Verwendung findet.
Das Framework wurde mit zwei weiteren Dateien um die Funktionalität des Exports erweitert. Somit können Diagramme für Paper, Dokumentationen oder Präsentationen verwendet werden. Es wurde eine Klasse "svgToEPS" in Typescript entwickelt, welches es ermöglicht SVG-Bilder (Diagramme) als EPS-Dateien abzulegen. Hierfür wurde Dropdownfeld in das Eigenschaftsfenster integriert. Auch wurde das GitHub-Projekt jspdf [Git13] eingebunden. Hierfür wurde eine Fassadenklasse geschrieben. Dieses kann ein Diagramm als PDF exportieren. Somit ist es möglich die dargestellten Diagramme in folgende Formate zu exportieren:

- svg
- eps
- png
- pdf

5.10.1 Elemente

Ein Diagramm besteht aus Knoten und Kanten. Verschiedene Diagrammarten stellen die Knoten unterschiedliche dar, z.B. UML Objektdiagramme und UML Klassendiagramme. DiagramJS unterstützt im Moment verschiedene grafische Elemente. Dies sind neben einfachen Nodes auch PatternObjekte, UML-Objekte und UML-Klassen, Bilder, SVG-Symbol, Klassendiagramme, Objektdiagramme, Patterndiagramme und Textelemente.

```
1  {
2    "type": "classdiagram", //["node", "patternobject", "
       classdiagram", "objectdiagram"]
3    "id":"model42",
4    // "content":{"src":<image-url>,  "html":<value>}
5    width: 200, height: 200,
6    x: 0, y: 0,
7    "head":{
8      "src":"img/se.png",
9      "html":"Uni Kassel - SE"
10   },
11   Options: {},
12   nodes:[
13 {"type":"class","id":"Ludo","attributes":["name:String"],
       methods:["init()"]},
14 {"type":"class","id":"Player","attributes":["name:String,
     color:String"]}
15   ],
16   edges:[
17   {"type":"assoc","source":{"property":"game","cardinality"
       :1,"id":"Ludo"},"target":{"property":"fields","
       cardinality":42,"id":"Player"}}
18   ]
19 }
```

Quellcode 32: Node-Struktur

Es besteht die Möglichkeit Diagramme mit Subdiagrammen zu erstellen. Die so hinzugefügten Komponenten können ausgeblendet werden. So besteht die Möglichkeit zu große Klassendiagramme hierarchisch darzustellen. Ein ausführliches kommentiertes Beispiel findet sich in Quellcode 32. Pflichtanteil ist nur der "Type" und die "Id". Bei Diagrammgruppen (Klassendiagramm, Objektdiagramm, Patterngraph) wie im Quellcode 33 gibt es zusätzlich das Attribut Options. Mit diesen Options können die Diagramme individuell eingestellt werden.

```
1  Options:{
2    raster: true, // boolean
3    display: "html", //["svg", "html"]
4    font: {"font-size": "10px", "font-family": "Verdana"},
5    layout: {name: "Dagre"},  // Dagre TB, LR
6    CardinalityInfo: true, // boolean
7    propertyinfo: true, // boolean
8    rotatetext: true, // boolean
9    linetyp: "center",
10   buttons: ["HTML", "SVG", "PNG", "PDF"]
11 }
```

Quellcode 33: Node-Struktur

5.10.2 Kanten

Bei den unterstützten Kanten werden die Kantentypen aus Abbildung 30 unterstützt. Dieses sind Kompositionen, Vererbung, Unidirektionale Assoziationen, Bidirektionale Assoziationen, Aggregationen. Alle Kanten können zusätzliche Informationen besitzen. So können Kanten und/oder die Kantenenden eine Beschriftung besitzen. Die Beschriftung kann entweder waagerecht oder in Kantenrichtung angezeigt werden.

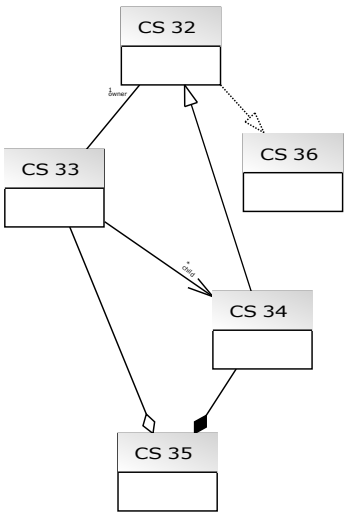

Abbildung 30: All Edges

5.10.3 Layouting

In dem Typescriptpackage ”layouts” sind Layout-Algorithmen integriert. Diese sind

- Spring-Layout
- Circular-Layout
- Fixed-Layout
- Ordered-Layout

Es handelt sich bei den aufgelisteten Algorithmen um sehr einfache Layouts, welche nur bedingt für komplexere Diagramme geeignet sind. In dem ”package” layout existiert weiterhin das Dagre-Layout, welches eine Fassade zu dem Github-Projekt von Chris Pettitt [cpe15] ist.

5.10.4 Controls

In der ”diagram.js” wurden zwei Ansichten implementiert. Im Moment gibt es eine HTML5-Ansicht und einen SVG-Ansicht. Die HTML Ansicht ist farbenfroh und bietet die Möglichkeit interaktive Elemente einzubinden. Die SVG-Ansicht ist durch ihre Verwendung von nur SVG-Elementen kompatibel zu den externen Bibliotheken. Es ist möglich mittels Dropdownbutton zwischen den verschiedenen Layouts zur Laufzeit zu wechseln.

5.10.5 Design

Das Design der Diagrammknoten wurde mittels CSS realisiert. So sind einfache Anpassungen an dem Design möglich, ohne den Sourcecode von JavaScript anpassen zu müssen. Jedes einzelne Element besitzt eine Id über die es angesprochen werden kann.

5.10.6 Qualitätssicherheit

Für die Qualitätssicherheit wurde JSLint [Cro10] genutzt. Der Sourcecode wurde mit dem statistischen Analysewerkzeug überprüft. JSLint dient zum Finden klassischer statischer Fehler und überprüft den Typescript-Sourcecode bei der Übersetzung nach Javascript. Zusätzlich werden Formatierungsfehler aufgedeckt.

5.10.7 Java-Schnittstelle

Mittels der Kombination von einigen Komponenten ist es möglich, den aktueller Zustand des Datenmodells zu serialisieren und über das Netzwerk vom NetworkParser an einen anderen Knoten zu senden. Damit die Objektdiagramme aus dem verteilten System angesprochen werden können, gibt es zu jedem JavaScript-Objekt eine entsprechende JavaScript Klasse.
Eine einfache Erstellung der Diagramme ist im Codebeispiel 34 dargestellt. Die Zeilen 1 - 9 erstellen ein Klassenmodell und Zeile 12 erstellt daraus eine HTML Seite. Die Schnittstelle umfasst im Moment 32 Klassen. Weiterhin existieren auch Subklassen von GraphList, GraphClazz und

GraphEdges, mit deren Hilfe ein Vergleich von zwei Klassen oder sogar ganzen Datenmodellen möglich ist. Dieses wird in der Lehrveranstaltung "Programmiermethodik" zum Vergleich abgegebener Hausaufgaben genutzt. Somit wurde der Korrekturaufwand für die Hausaufgaben der Studierenden minimiert. So dass ein einfacher Textvergleich Unterschiede feststellen kann.

```
1  // Create Classmodel
2  GraphList model = new GraphList();
3  Clazz uni = model.with(new Clazz());
4  uni.withClassName("University");
5  uni.withAttribute("name", DataType.STRING);
6  Clazz stud = model.with(new Clazz());
7  stud.withClassName("Person");
8
9  uni.withbidiAssoc(stud, "has", Cardinality.MANY, "studis",
       Cardinality.ONE);
10
11 // Generate HTML-File
12 HTMLEntity htmlEntity = new HTMLEntity().withGraph(model);
13 System.out.println(htmlEntity.toString());
```

Quellcode 34: HTML Generator

5.10.8 eDobs

Die Java-Schnittstelle ist so erweiterbar, dass die Java-Bridge genutzt wurde, um einen interaktiven Objektdiagrammeditor zu erstellen. Dieses erleichtert das Entwickeln wie schon in dem Paper [GZ06a] gezeigt. Der interaktive Objektdiagrammeditor ist mittels einer einfachen statischen Methode aufrufbar. In Eclipse oder InteliJ gibt es eine separate Ansicht für Expression. Diese wird jeweils beim Erreichen eines Haltepunktes ausgewertet. Mittels des Aufrufs siehe Quellcode 35 kann ein separates Fenster geöffnet werden, wo die aktuellen Elemente als Objektdiagramm dargestellt werden. Die Elemente sind verschiebbar und editierbar, somit ist ein einfaches Ändern der aktuellen Situation möglich. Da die IdMap vom

NetworkParser für das Umformen benutzt wird, ist sogar eine Darstellung von Objekten von anderen Knoten im Netzwerk denkbar. In Abbildung 31 ist das separate Fenster zu sehen. In der Abbildung ist ein Beispiel aus SDMLib zu sehen. Interessant ist das "assignment"-Attribut in dem Objekt "Room". Falls ein Objekt nicht bekannt ist, wird hierauf die "toString"-Methode aufgerufen. Es wurde sich bewusst gegen eine Integration in eine IDE entschieden um möglich allgemein einsetzbar zu sein [GZ06b, GZ06c, KZ15]. Es ist somit dem Anwender überlassen welche IDE er benutzt, da der eDobs IDE unabhängig arbeitet.

```
1 DiagramEditor.edobs(studyRight, mathRoom, alice);
```

Quellcode 35: eDobs Java

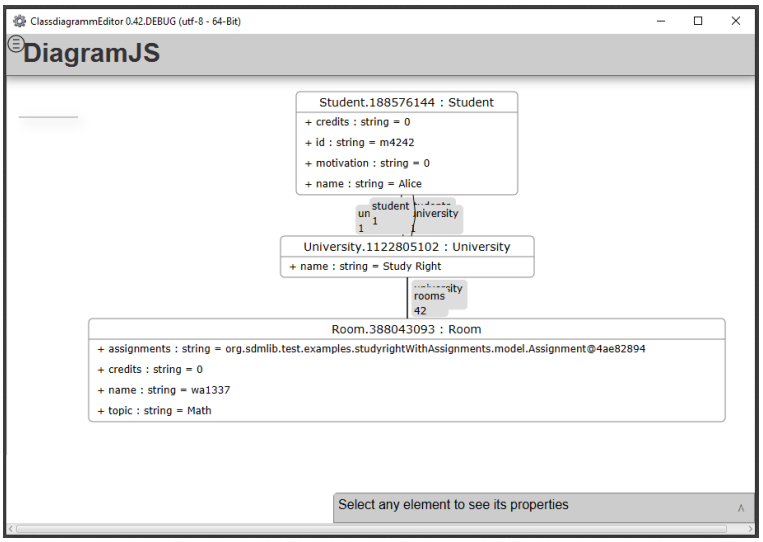

Abbildung 31: eDobs

5.10.9 JavaFX-Schnittstelle

In der Lehre bei "Programmiermethodik" hat sich herausgestellt, dass es enorm wichtig ist einen guten und intuitiven GUI-Designer zu benutzen. Einen entsprechenden GUI Designer zu entwerfen würde allerdings an dieser Stelle zu weit gehen. Deswegen wurde auf das bekannte Tool SceneBuilder

aufgesetzt. Dieser wurde von Oracle 2014 entwickelt und wird seit 2015 von der Firma Gluon weiterentwickelt [Glu16]. Die Studierenden hatten in der Vorlesung keine Probleme ganze Spieloberflächen mit dem Designer zu realisieren. Die so entstandenen Oberflächen werden als FXML abgespeichert und können mit dem NetworkParser eingelesen werden und in die neue HTML-Darstellung übersetzt werden.

5.10.10 Klassendiagramm-Editor

Für die Lehrveranstaltung "Programmiermethodik" wurde der NetworkParser um einen Editor erweitert. Der Klassendiagramm-Editor ist in Abbildung 32 zu sehen. Dadurch ist es möglich ein Klassendiagramm visuell zu zeichnen. Das erstellte Klassendiagramm kann dann mit der JavaFX-Bridge als Json abgelegt werden. Die so abgespeicherten Datenmodelle können wieder per "Drag and drop" im Editor geladen werden. Alternativ kann mittels SDMLib [ZLGE14b, NJZ13] oder NetworkParser Sourcecode generiert werden.

Der Klassendiagrammeditor umfasst eine Toolbar auf der rechten Seite, um Klassen und deren Attribute und Methoden hinzuzufügen. An der unteren Seite findet man die Propertybar, welche das Speichern, den Packagenamen des Klassenmodells und ggf. den Generierungsbutton enthält. An der unteren rechten Ecke kann man sich eine Vorschau des Datenmodells in Json anzeigen lassen. Das Javascript ist so implementiert, dass nur die im Moment zur Verfügung stehenden Features angezeigt werden. Falls auf den Rechner kein JavaFX oder OpenFX vorhanden ist, startet der NetworkParser eine Server-Instanz und startet, den lokalen Browser. Somit ist es möglich ein Datenmodell im Browser zu entwickeln. Für das Hinzufügen oder Editieren von Eigenschaften einer Klasse wurde ein Inlineeditor geschrieben. Dieser wird aktiv durch einfaches selektieren einer Klasse und Eingabe auf der Tastatur. Dadurch erscheint unterhalb der Klasse ein Eingabefeld, welches mittels Syntaxanalyse die Eingabe von Klassennamen, Attributen und Methoden unterstützt.

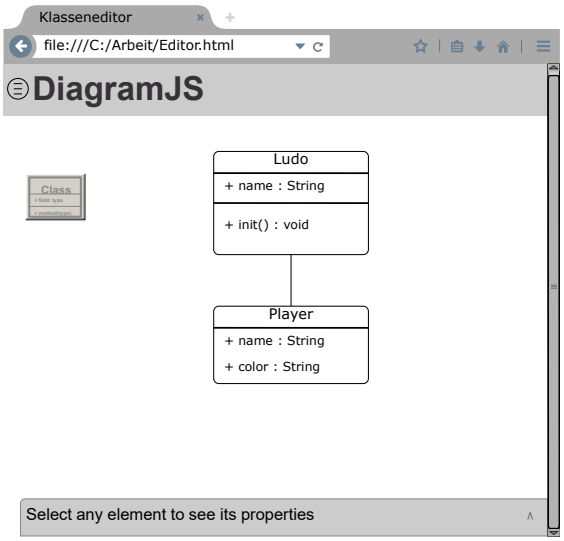

Abbildung 32: ClassEditor

5.11 Collection

In Java gibt es zahlreiche Collections. Die implementierten Lösungen für Datenmodelle benutzen sehr unterschiedliche Collections. Bei EMF wird die ArrayList benutzt. SDMLib hat lange Zeit die LinkedhashSet benutzt. Dieses führte schon zum Gewinnen des TTC 2015 mit Petri Netzen [ZL15]. Aber die LinkedHashSet Collection braucht viel Speicher und hat intern viele Hilfsobjekte. Das Hauptargument von EMF, dass die Anzahl der Nachbarn eines Objekts fast immer relativ klein ist stimmt irgendwie auch. Die Idee ab 420 Elemente von ArrayList auf LinkedHashSet wechseln wurde in einer eigenen Collection realisiert. Das Verhalten zeigt sich auch bei binären Bäumen und Datenmodellen, diese besitzen viele Objekte/Knoten mit keinen oder wenigen Nachbarn (Links). Bei Datenmodellen kommt meist ein Objekt (Wurzel) hinzu, an dem sehr viele Knoten hängen.
So ist es notwendig eine optimale Speicherung für die unterschiedlichen Fälle zu gewährleisten. Eine eigene Hash-Liste mit Überlauflisten im Array selbst wurde implementiert. Dadurch sind keine Hilfsobjekte notwendig.

Modelle haben spezielle Anforderungen an Container-Klassen. Die bestehenden Collections haben immer eine Spezialisierung. Somit wurde die

eigene Collection mit den bestehenden Collections verglichen.

Name	**Vorteile/Eigenschaften**	**Nachteile**
LinkedHashSet	Reihenfolgestabil	viele interne Instanz Objekte
ArrayList	Reihenfolgestabil	langsames Durchsuchen
		bei vielen Elementen
		langsames löschen
HashSet	Schnell	Zufällige Reihenfolge,
		viele interne Instanz-Objekte
Vector	Reihenfolgestabil	viele interne Instanz Objekte
Stack	Reihenfolgestabil	Zugriff auf erstes / letztes Element
TreeSet	Reihenfolge einstellbar	Nur Reihenfolgestabil
		mit eigenem Comparator
LinkedList	Reihenfolgestabil	Iteration und Contains sehr langsam
SimpleList	Reihenfolgestabil, schnell	Contains langsam
SimpleSet	Reihenfolgestabil, schnell	Contains langsam

Bei der Implementierung der Collections der Simple-Collection (SimpleList und SimpleSet) wurde zum einen auf einen speichersparenden Container geachtet und zum anderen auf eine Vielzahl von Einsatzmöglichkeiten unteranderen ein optimales schnelles zugreifen auf die Elemente. Die Implementierung ähnelt vom Prinzip der Collection ”java.util.ArrayList” [Doc15a]. Da die ArrayList, sowie die SimpleList und SimpleSet auf ein internes Array zugreifen, besitzen Sie die Einschränkung nicht ”threadübergreifend sicher” zu sein.

Für die Benutzung im verteilten System wurde die ”SimpleList” Collection noch für die Benutzung als Queue optimiert. Weiterhin hat die Arbeit von der Brownies-Collection gezeigt, dass der Löschungen bei Collection am Anfang öfter eintreten und von der Performance am schlechtesten sind. Es wurde zusätzlich ein Pointer integriert, um das Löschen und Einfügen am Anfang zu optimieren. Bei der Entwicklung wurde weiterhin darauf geachtet,

dass die zur Verfügung gestellte Schnittstelle möglichst übersichtlich ist. So wurden so wenig wie nötig "Public"-Methoden integriert, jedoch so viel wie nötig um eine gute Funktionalität abzudecken. Bei den Collections sind 59 Methoden sichtbar, 20 Methoden kommen aus dem java.util.List Interface. Die meisten Methoden wurden nach der Methoden-Verkettung (Chaining) Prinzip aufgebaut.

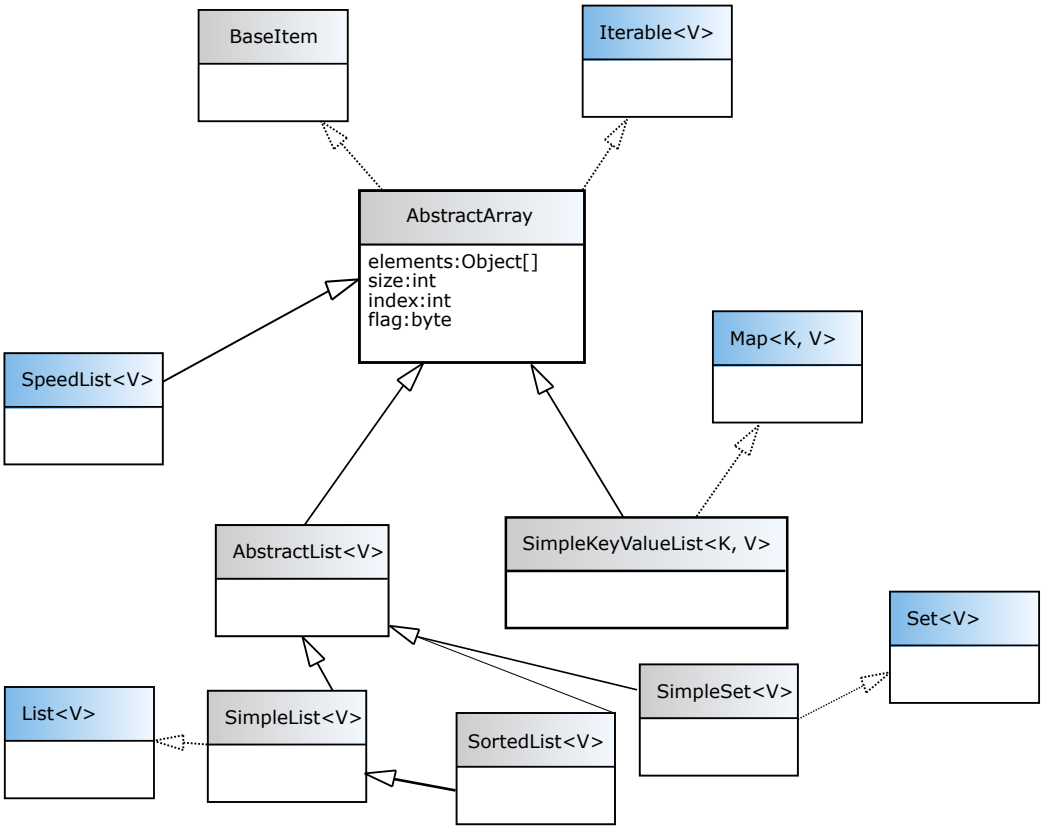

Abbildung 33: Modell von Simple - Collection

Eine einfache Instanz auf einer 64-Bit JVM [Tys18] hat somit 32 Bytes.

Mittels zwei Methoden wird das Sortieren mit Hilfe eines Komparators realisiert. So entstand die SortedList.

Selbst den Bereich der KeyValue-Liste und der KeyValueKey-Liste [Bid] wurde berücksichtigt. Die Bereiche (KeyValueKey-Liste), die in Java gar

nicht oder nur mit mehreren Klassen realisiert sind, wurden hier mit einer Klasse und wenigen speziellen Methoden umgesetzt.

In Abbildung 33 ist die Klassenstruktur zu sehen. In grau sind die selbstimplementierten Klassen und in blau die Klassen aus dem SDK dargestellt. Bei der Modellierung wurde darauf geachtet, dass eine sehr übersichtliche Klassenstruktur benutzt wurde. So benutzen die Collections für Auflistung, für Mengen und KeyValue-Listen alle eine Basisklasse AbstractArray. In den speziellen Containern (SimpleSet und SimpleList) wurden mehrere Interfaces wie Iterable, BaseItem, Map, Set, List integriert. Somit können mittels fünf Klassen (siehe Abbildung 33) alle bisherigen Szenarien abgedeckt werden. Als zusätzliche Optimierung wurde eine SpeedList integriert. Diese erbt direkt von AbstractArray und ist zusätzlich auf Geschwindigkeit optimiert. Allerdings fehlen ihr einige Funktionalitäten, welche für die Verwendung von Modellen und PatternMatching benötigt werden.

Die Hauptfunktionalität findet sich in der AbstractArray-Klasse. Somit haben alle Collections dieselbe Codebasis. Die einzigen Unterschiede sind die implementierten Interfaces und das gesetzte Flag für das "Erlauben von Duplikaten" ((siehe Quellcode 36 auf Seite 113)). Die zwei Hauptcollections SimpleList (6 Methoden in 42 Zeilen) und SimpleSet (18 Methoden in 150 Zeilen) besitzen selbst nur sehr wenige Methoden.
In der SimpleList dürfen Einträge doppelt vorkommen. Die SimpleSet ist speziell für Mengen ohne Duplikate ausgelegt. Die AbstractList umfasst alle Methoden der List und Set-Klasse, die identisch sind allerdings im Widerspruch mit dem "Map" Interface stehen.

```
1  /** Is Allow Duplicate Items in List */
2  public static final byte ALLOWDUPLICATE = 0x01;
3  /** Is Allow Empty Value in List (null) */
4  public static final byte ALLOWEMPTYVALUE = 0x02;
5  /** Is The List is Visible for Tree Editors */
6  public static final byte VISIBLE = 0x04;
7  /** Is Key is String and is Allow Casesensitive */
8  public static final byte CASESENSITIVE = 0x08;
9  /** Is List is ReadOnly */
10 public static final byte READONLY = 0x10;
11
12 /** Is The List has Key,Value */
13 public static final byte MAP = 0x20;
14 /** Is List is Key,Value and Value, Key */
15 public static final byte BIDI = 0x40;
```

Quellcode 36: Flags

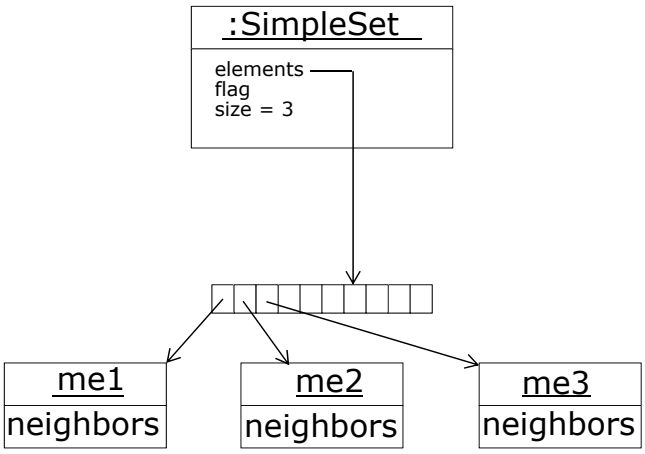

Abbildung 34: Small SimpleSet

Bei kleinen Mengen werden die Werte ähnlich wie in einer ArrayList in einer eindimensionalen Array gespeichert (siehe Abbildung 34). Bei größeren Mengen werden die Elemente in einem zweidimensionalen Array gespeichert (siehe Abbildung 35). In dem ersten Array werden die Elemente gespeichert,

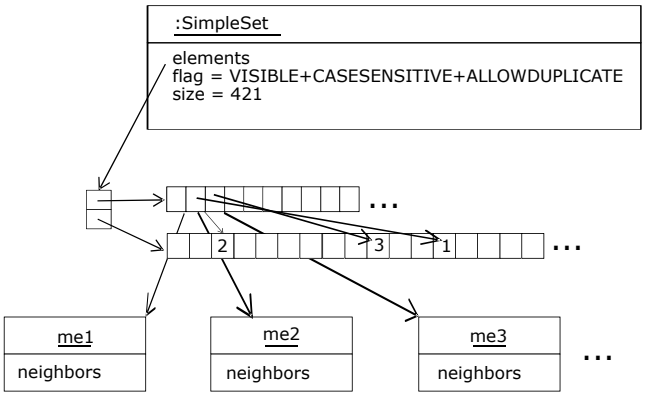

Abbildung 35: Large SimpleSet

in dem zweiten werden die Positionen der Werte aus dem ersten Array gestreut (gehashet) abgelegt. Dadurch ist bei allen Collections der Zugriff ähnlich schnell wie bei der HashSet-Collection. Beim Löschen wird eine Liste mit gelöschten Elementen gepflegt, welche maximal 42 Elemente groß wird. Dadurch wird nicht nach jeden löschen die Arrays neu angeordnet. Falls mehrere Elemente denselben Hash-Wert besitzen, werden die Werte hintereinander gespeichert. So gibt es eine horizontale Überlaufliste. Bei so einer Anordnung werden keine Hilfsobjekte benötigt. Das bedeutet, dass einfach die Werte nacheinander in dem Array gespeichert werden. Dadurch kann ein schneller Zugriff (iterator) gewährleistet werden. Die implementierte Collection wurde mittels des Guava Framework [Git15] von Google getestet. Dieses wurde bereits für die Brownies Collection genutzt und somit konnten die eigenen Collectionklassen gut mit den Brownies Collection verglichen werden. In dem Framework sind mehrere eigene Collections und viele standardisierte Tests, welche benutzt wurden um die Funktionalität der entwickelten Simple-Collection sicherzustellen.

5.11.1 Wichtigkeit der Methoden

Es gibt eine Reihe von Zugriffsmethoden für Collections. Bei der Nutzung von Collections im Modellumfeld werden im Normalfall allerdings nicht alle genutzt. Die Zugriffsmethoden werden auch unterschiedlich oft verwendet. Die häufigsten verwendeten Zugriffsmethoden sind in der Reihenfolge

1. contains(Object), indexOf(Object)
2. iterator, get(pos), size()
3. add(Object), remove(Object)
4. clear() [reset]
5. clone()

Die Reihenfolge basiert darauf, dass bei Modellen schon intern überprüft wird, ob ein Element in einer Menge enthalten ist.

Die entstandenen Collections sollten auch für das verteilte System eingesetzt werden. Dort werden für die Abarbeitung der Aufgaben Queues benutzt.
Für Queues sind andere Methoden wichtig:

1. first()
2. removeFirst()
3. addFirst()

5.11.2 Messung der Methodengeschwindigkeit

Für das Messen der Geschwindigkeit der wichtigsten Methoden wurde das JMH [Ope15a] Framework von Oracle [Ora18c] benutzt. Dieses wurde speziell entwickelt, um Performancetests durchzuführen. Manuelle Tests auf der JVM durchzuführen, welche aussagekräfte Zeiten protokollieren ist schwierig, da der JIT-Compiler dynamische Optimierungen zur Laufzeit vornimmt. Die Einstellungen der Tests geschieht mittels Annotations und beinhaltet eine Reihe von Einstellungen, welche sogar die Warmups der JVM ermöglichen. Dadurch werden Schwankungen des JIT-Compilers abgefangen. Bei der Entwicklung des Frameworks JMH wurde Wert daraufgelegt, dass die Auslastungen der CPUs vernachlässigt werden können. Der Performancetest, wie im Quellcodebeispiel 37 zeigt das einfache Hinzufügen eines Elements zu einer Collection, welches mittels des JMH Framework getestet wird.

```
1  @State(Scope.Thread)
2  @OutputTimeUnit(TimeUnit.NANOSECONDS)
3  public class CollectionTest_AddSmall {
4    private void test(Collection<Integer> list,int len) {
5      for(int i=0;i<len;i++)
6        list.add( i );
7    }
8
9    @Benchmark
10   @Warmup(iterations = 5, time = 1)
11   @Measurement(iterations = 10, time = 1)
12   @BenchmarkMode(Mode.AverageTime)
13   public void AddSet100() {
14     test(new SimpleSet<Integer>(), 100);}
15 }
```

Quellcode 37: JMH

Bei der Simple-Collection wurden zwei Messungen vorgenommen. Für die erste Messung wurde eine kleine Menge (100 Elemente) benutzt. Für die zweite Messung wurde eine große Menge (1.000.000 Elemente) verwendet

(siehe Abbildung 36 auf Seite 119). Für eine etwaige Abweichung wurden zehn Iterationen pro Messung durchgeführt, wobei vor jeder Iteration fünf Warmups durchgeführt wurden.
Mittels des JMH Framework wurden wie in Abbildung 37 mehrere Methoden getestet. Für den Vergleich der verschiedenen Collection für meinen Einsatzzweck wurden die jeweiligen Methoden benutzt, welche in dem java.util.Collecion-Interface definiert sind. Die Messung wurde mit der jmh.gradle im Projektroot durchgeführt (siehe Quellcode 38). Die einzelnen Tests findet man unter src/jmh/java. Somit sind die Tests bei den Programmcode vom NetworkParser auf GitHub enthalten.

```
1 gradlew -b jmh.gradle jmh
```

Quellcode 38: JMH Befehl

Durch die Optimierungen ist das Ziel erreicht, eine schnelle Collection bei "Contains", "Remove" und "Iterator" zu implementieren. Bei "Add" und "Add+Contains" ist es die schnellste Collection, welche mit Objekten umgehen kann. Die schnellere "LinkedHashSet" hat bei größeren Datenmengen einen erheblich größeren Speicherbedarf.

5.11.3 Optimieren nach Speicherbedarf

Bei der Implementierung der Simple-Collections wurde auf speichersparendes Verhalten Wert gelegt (siehe Abbildung 37 auf Seite 119). Bei den Transformation Tool Contest (TTC) 2014 - 2017 mussten mittels Graphtransformation Probleme mit großen Datenmengen gelöst werden. Hierbei war die Umsetzung speichersparender Container wichtig [TH14, ZLGE14b, ZLGE14a, ZL15, ZE16, ZW17].
Der Speicherbedarf wurde mit dem Framework Classmexer [Jav15] von "javamex" gemessen (siehe Quellcode 40). Es ist ein einfaches Verfahren, welches wiederholbare zuverlässige Werte ermitteln kann.

```
1 gradlew -b jmh.gradle classmexer
```

Quellcode 39: Classmexer Befehl

Dieses Framework ermöglicht ein einfaches Messen von Speicherbedarf von Objekten und komplexen Strukturen im Arbeitsspeicher zur Laufzeit. Die SimpleList- und SimpleSet-Collection ist laut Classmexer nach der Initialisierung 32 Byte groß. Wie im Beispiel 40 wird der gesamte Speicherbedarf gemessen. Die SpeedList benötigt bei der Initialisierung sogar nur 31 Bytes.

```
1 System.out.printf("%12d", com.javamex.classmexer.MemoryUtil
      .deepMemoryUsageOf(list));
```

Quellcode 40: Classmexer

100 Elements / Zeit in Nanosekunden	Add	Add+Contains	Contains	Remove	Iterator
java.util.ArrayList	15571	1465509	1777	160	6
java.util.ArrayDeque	16556	2075649	5015	206	125
java.util.LinkedList	15571	3210539	8697	155	109
org.magicwerk.brownies.collections.GapList	12133	1736146	4876	152	9
org.magicwerk.brownies.collections.BigList	5661	6187960	40596	152	661
org.magicwerk.brownies.collections.primitive.IntObjGapList	901	2546	2052	185	103
de.uniks.networkparser.list.SimpleList	3079	1071504	1808	171	5
de.uniks.networkparser.list.SpeedList	2622	5200	1614	151	3
org.eclipse.emf.common.util.BasicEList	23589	1469481	1721	154	5
org.eclipse.emf.ecore.util.EObjectResolvingEList	22268	2321	1901	157	111
java.util.HashSet	20118	1375	421	700	278
java.util.LinkedHashSet	42921	1188	424	694	155
java.util.TreeSet	32488	2361	1960	223	357
de.uniks.networkparser.list.SimpleSet	3027	10256	1808	218	5

1M Elements / Zeit in Millisekunden	Add	Add+Contains	Contains	Remove	Iterator
java.util.ArrayList	148	1388455	166061	97776	0
java.util.ArrayDeque	206	1501633	465423	2	1
java.util.LinkedList	147	2680258	1170215	1	3
org.magicwerk.brownies.collections.GapList	82	1545912	1826597	2	0
org.magicwerk.brownies.collections.BigList	69	4084035	4339899	2	8
org.magicwerk.brownies.collections.primitive.IntObjGapList	7	165903	180114	3	6
de.uniks.networkparser.list.SimpleList	189	409	125	316	1
de.uniks.networkparser.list.SpeedList	178	390	115	313	0
org.eclipse.emf.common.util.BasicEList	196	1517707	201537	92706	0
org.eclipse.emf.ecore.util.EObjectResolvingEList	168	58	224616	92626	3
java.util.HashSet	208	217	60	30	33
java.util.LinkedHashSet	355	213	48	28	4
java.util.TreeSet	257	159	172	3	31
de.uniks.networkparser.list.SimpleSet	318	490	144	504	0

Abbildung 36: Zeitmessung

Elements / Speicher	0	10	100	1000	1000000
java.util.ArrayList	40	240	2080	20976	20858280
java.util.ArrayDeque	104	264	2152	20136	20190120
java.util.LinkedList	32	432	4032	40032	3064720
org.magicwerk.brownies.collections.GapList	104	264	2200	20144	20017568
org.magicwerk.brownies.collections.BigList	168	368	2304	24368	20315400
org.magicwerk.brownies.collections.primitive.IntObjGapList	120	120	616	4160	4021312
de.uniks.networkparser.list.SimpleList	40	280	2168	20288	20167784
de.uniks.networkparser.list.SpeedList	31	248	2048	20048	19996560
org.eclipse.emf.common.util.BasicEList	24	240	2152	20096	20017296
org.eclipse.emf.ecore.util.EObjectResolvingEList	544	760	2672	20616	20018488
java.util.HashSet	64	640	5920	56288	56358384
java.util.LinkedHashSet	72	728	6728	64296	64357672
java.util.TreeSet	64	640	5680	56080	55699040
de.uniks.networkparser.list.SimpleSet	32	280	2168	42064	42376600

Abbildung 37: Speichermessung

5.12 PropertyChange

In dem NetworkParser wurde der PropertyChange-Mechanismus an mehreren Stellen verwendet. So benutzt die Codegenerierung diesen Ansatz, um Callback-Programmcode zu erstellen. Somit kann der Entwickler in jeden Schritt der Generierung eingreifen.
Bei der Serialisierung kann der Entwickler sich als Listener registrieren um spezielle Serialisierungen zu beeinflussen oder zu verhindern (siehe Quellcode 41). Die GUI wird komplett mittels PropertyChange aktualisiert. Das verteilte System ist an vielen Stellen individuell anpassbar, dadurch können verschiedene Szenarios ohne viel Programmcode realisiert werden. Dieses wurde mittels PropertyChange realisiert. Die Logikkomponenten benötigen ein SimpleEvent, welches von PropertyChangeEvent abgeleitet ist (siehe Abschnitt 5.17.2 auf Seite 139). Zusätzlich existiert ein SimpleLogger, welcher auch als SL4J Logger funktioniert. Der SimpleLogger kann die Events per PropertyChange weiterleiten und auch PropertyChange-Nachrichten speichern oder Ausgeben. Dieses hat sich für die Entwicklung bewährt. Somit sind in allen Bereichen standardisierte Schnittstellen (PropertyChangeListener) integriert um das gesamte System schnell zu erweitern.

```
1  IdMap map = new IdMap().withTimeStamp(1);
2  User user= new User().withName("Albert").withPassword("geheim");
3  map.with(new UserCreator());
4  map.getMapListener().getFilter()
5     .withPropertyRegard(new ObjectCondition() {
6          public boolean update(Object evt) {
7            SimpleEvent simpleEvent = (SimpleEvent) evt;
8            return (User.PROPERTY_PASSWORD.equals(simpleEvent.
                 getPropertyName()) == false);
9          }
10       });
11 map.toJsonObject(user);
```

Quellcode 41: Filter

5.13 Serialisierung

Für das Austauschen von Datenmodellen und das Verteilen von Modelländerungen wurden diese in das Json-Format serialisiert (siehe meine Masterarbeit [Lin12]). So wurde eine eigene Reflektionsschicht geschrieben, um eine allgemeine und überall benutzbare Zugriffsschicht zu realisieren. Dieses hatte mehrere Gründe, zum einen war 2011 Reflektion nicht überall verfügbar (GWT, Android), außerdem sollte der Entwickler zu jeden Zeitpunkt genau über die zu serialisierten Elemente zugriff haben und zum anderen hatte dieses Performancegründe, da die Reflektionsschicht von Java immer deutlich langsamer ist als die eigene generierte Reflektionsschicht [Fra16]. Da die damaligen Standard Serialisierungframeworks keine Linkzyklen unterstützten, musste ein eigener Mechanismus integriert werden um eine Auflösung der bidirektionalen Links zu realisieren. Da sich herausgestellt hat, dass einige Frameworks wie EMF [Fou15] nur XML-Dateien unterstützen, oder im Bereich Heimautomation oder Verschlüsselung ein Binäres Format vorausgesetzt wird, wurde eine Kontrollschicht (IdMap) entwickelt, welche es mittlerweile erlaubt, die Daten in viele gängige Formate zu überführen. Die Entwicklung des NetworkParser startete mittels einer Neuimplementierung der Java-Musterimplementierung von json.org [dou10]. Die komplette Syntax ist in Abbildung 38 auf Seite 123 zu sehen.

Der NetworkParser wurde mittlerweile erweitert, sodass das neue Format hjson (Human Json) [Zan16] unterstützt wird. Dieses wurde von C. Zang 2016 entwickelt und hat das Ziel noch einfacher und menschenfreundlicher zu sein, so dass auch Kommentare erlaubt sind. Im NetworkParser wurde die Unterstützung der Kommentare bei XML ebenfalls hinzugefügt. Standardmäßig implementieren die Creator(Set)-Klassen das SendableEntityCreator Interface. Es gibt noch weitere wie der SendableEntityCreatorWrapper, welcher für Klassen geeignet ist, die nicht iterativ erstellt werden (zum Beispiel "java.util.Date" mit einem Konstruktor), wie zum Beispiel die "java.util.Date"-Klasse. Mittels des SendableEntityCreatorTag-Interface kann ein spezielles Mapping

realisiert werden, wie bei XML. Für Laufzeitelemente sollen keine IDs verwendet werden. Hierfür kann das SendableEntityCreatorNoIndex-Interface hinzugefügt werden. Dieses ist gerade bei Tasks oder für Data Transfer Objects von Vorteil. Falls für eine spezielle Klasse der Creator selbst für die Id zuständig ist, kann dies mittels SendableEntityCreatorIndexId-Interface realisiert werden.

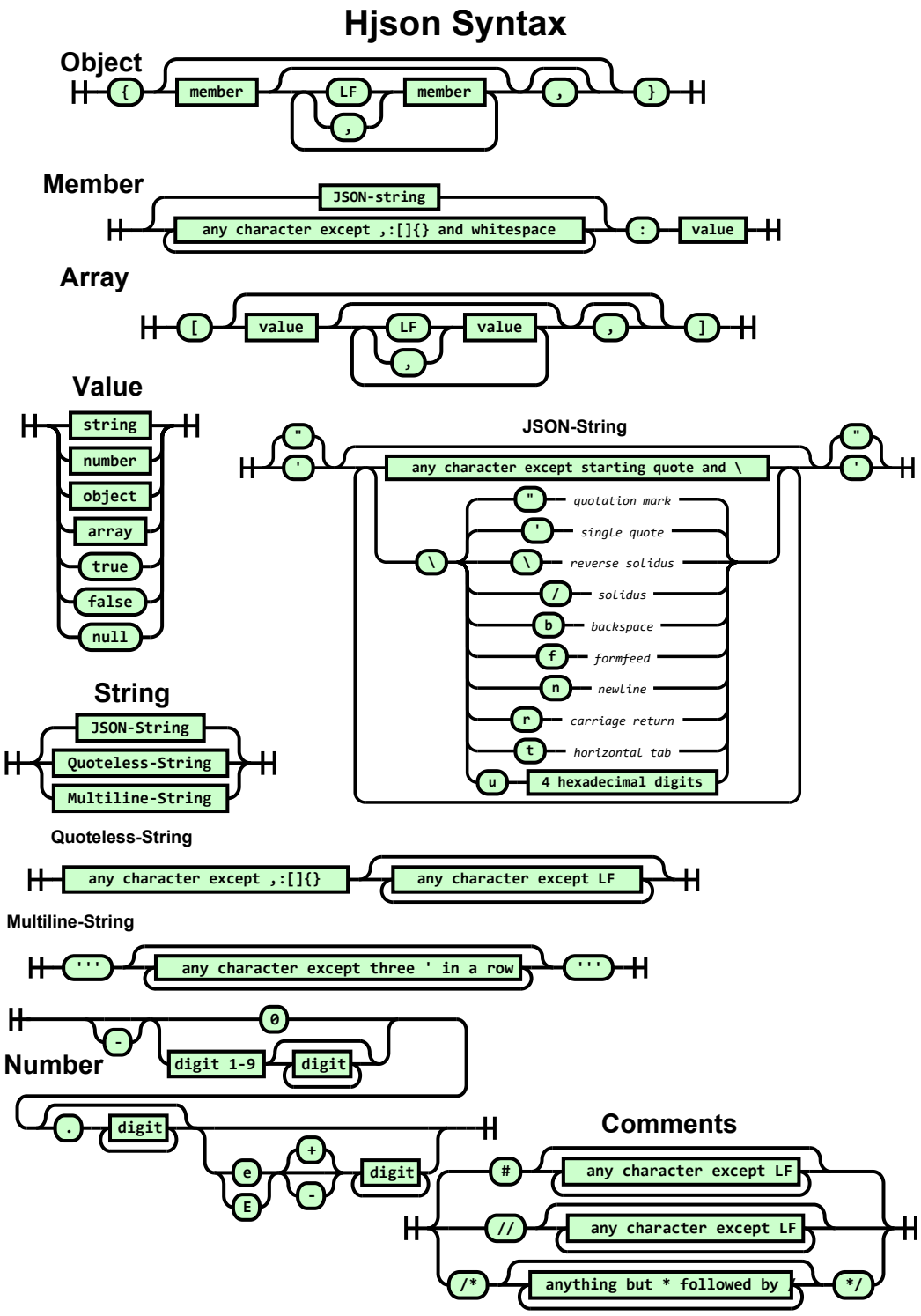

Abbildung 38: Syntax HJSON

Durch die eigene Implementierung der Reflektionsschicht ist es möglich, verschiedenste Grammatiken zu implementieren mit wenigen Klassen. So unterstützt der NetworkParser mittlerweile unter anderen die Formate Bit, Byte, XML, HXML, XSD, HJSON, JSON, YUML, EMF, GXL, TileMap, Dot.

5.13.1 Transaktion

Änderungen an einem Datenmodell, die per PropertyChange an die IdMap geleitet werden können aus einfachen Änderungen eines Attributes bestehen, können aber auch mehrere Attribute oder sogar mehrere Objekte betreffen. Die Änderungen werden zusammengefasst und mittels PropertyChange versendet. Bei dem Neuanlegen eines Objektes und dem Hinzufügen zu dem bisherigen Modell wird das neue Element komplett serialisiert und verteilt.

```
{"class":"Student",
  "id":"S1",
  "upd":{"name":"StefanL","credits":42},
  "rem":{"name":"Stefan", "credits":0}
}
```

Quellcode 42: Json Transaktion

Dieses kann allerdings auch in atomarer Form aufgesplittet werden. Somit ist es möglich auf jede einzelne Änderung zu reagieren. Dieses hat dann die Eigenschaft, dass ein neues Element angelegt wird und dann für jedes Attribut ein eigenes Event erzeugt und weitergegeben wird.

Das Senden von Änderungen in atomarer Form hat allerdings einige Nachteile. So werden alle Änderungen einzeln versendet und dadurch kommt es vor, dass sich das Datenmodell beim Empfänger in einem nicht gültigen Zwischenzustand befindet.

Seit der Erfindung von Datenbanken gibt es Datenbanktransaktionen, die dieses verhindern. In einer Transaktion sind mehrere Änderungen zusammengefasst, die in einem speziellen Anwendungsfall zusammengehören.

Das Finden von Transaktionen aus den Änderungen bedarf gewisser Regeln, welche durch einen Domainexperten, der die Probleme und notwendigen Änderungen definieren kann, richtig modelliert werden müssen. Ein einfaches Zusammenfassen von einigen Änderungsnachrichten reicht nicht aus. Das Verhalten wurde in einer Transaktion-Klasse zusammengefasst und ermöglicht dem Entwickler mit wenig Aufwand Änderungen zusammenzufassen. Im einfachsten Fall braucht dieser nur das Anfangsereignis und das Endereignis definieren. In anderen Fällen können eigene Listener registriert werden, um komplexe Überprüfungen zu tätigen.

```
1  University uni = new University();
2  IdMap map=new IdMap();
3  map.with(new UniversityCreator(), new StudentCreator());
4  UpdateCondition fT = UpdateCondition.createTransaction(map);
5  map.withListener(fT);
6  fT.withStart(Student.class).withEnd(Student.PROPERTY_LASTNAME);
7  map.toJsonObject(uni);
8
9  Student student = new Student();
10 uni.addToStudents(student);
11 student.withFirstName("Albert");
12 student.withLastName("Zuendorf");
```

Quellcode 43: TestCode Transaktion

```
{"class":"Student", "id":"S13522776366800",
  "timestamp":"13522776366800", "upd":{"university":
  {"class":"University","id":"U13522762684400"},
  "firstName":"Albert","lastName":"Zuendorf"}}
```

Quellcode 44: Update Transaktion

5.13.2 JSON Light

Der NeworkParser formt eine beliebige Modell-Instanz standardmäßig in nachfolgenden JSON (Code 45) um. Die Grammatik kann jedoch für spezielle Use-Cases anders aussehen. Im NetworkParser wurde eine ”SimpleGrammar”

geschrieben, welche für die Standardimplementierung zuständig ist. Um Datendurchsatz zu reduzieren, gibt es auch eine "FlatFlag" wodurch die eigenen Attribute nicht separat gekapselt werden. Dieses kann auch durch Verwendung von speziellen Interfaces SendableEntityCreatorWrapper aus dem NetworkParser unterstützt werden.
Um keinen Namenskonflikt mit den Metadaten zu erzielen, wird bei diesen ein "." vorangesetzt, da dieses für Klassenvariablen ungültig ist. Dieses ist in Code 46) dargestellt.

```
{"class":"Student","id":"P42","timestamp":"934738178045517",
  "prop":{"name":"Alice"}}
```

Quellcode 45: Json Serialisierung

```
{".class":"Student",".id":"P42",".timestamp":"934738178045517",
  "name":"Alice"}
```

Quellcode 46: Json Light Serialisierung

5.13.3 JSON Patch

Mittels JSON Patch wird eine standardisierte Struktur definiert, mit welcher Änderungen an einem Modell im Web übertragen werden können. Die Struktur gibt vor, dass in einer kurzen Nachricht die Operation und eine atomare Änderung enthalten ist. Die Beispiel-Patches (47) kommen aus dem Spezifikationmanuskript von P. Bryan und M. Nottingham [PB18].

```
[
  { "op": "test", "path": "/a/b/c", "value": "foo" },
  { "op": "remove", "path": "/a/b/c" },
  { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
  { "op": "replace", "path": "/a/b/c", "value": 42 },
  { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
  { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

Quellcode 47: Json Patch

5.14 Deployment

Die komplette "DiagramJS"-Bibliothek wurde mit Typescript realisiert und Sie wird auf dem Travis [Tra19] übersetzt und in dem NPM-Registry bereitgestellt. Die so zur Verfügung stehende Bibliothek wird dann mittels Node.JS und NPM im NetworkParser integriert.

Der gesamte Build-Prozess wird dokumentiert und ist im BuilderRahmen (siehe Abbildung 39 auf Seite 127) zu sehen. In der Abbildung 40 wird die Abhängigkeit von Gradle Tasks dargestellt. Diese Darstellung stammt ebenfalls aus dem NetworkParser. Hierfür wurde eine Klasse namens "Gradle" erstellt. Diese kann direkt aus den GradleWrapper aufgerufen werden und erweitert den GradleProzess um einige Features wie die Darstellung der Task-Dependencies.

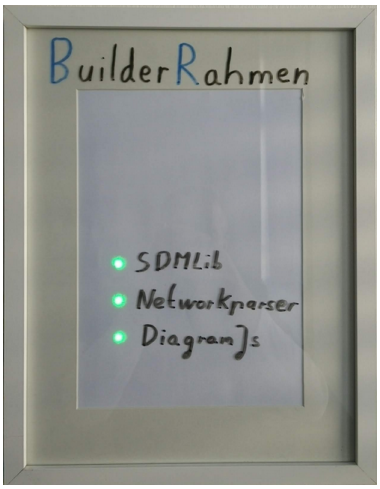

Abbildung 39: BuilderRahmen

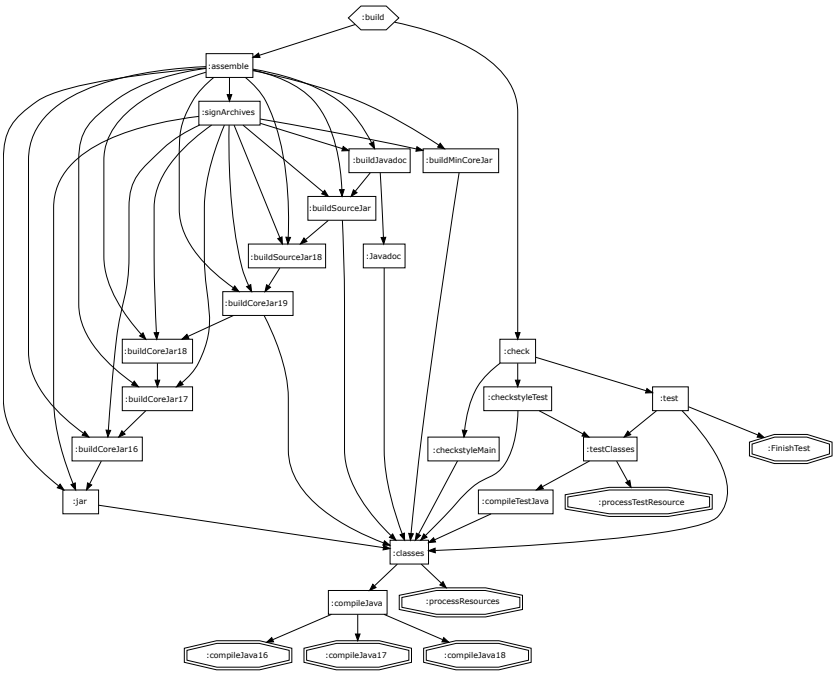

Abbildung 40: GradleTasks

Es gibt verschiedene Ausprägungen "Classifier" des NetworkParser.

- core16
- core17
- core18
- core19
- javadoc
- min
- sources
- sources18
- jar
- git

Der NetworkParser ist unter Sonatype zum Beispiel als "NetworkParser-4.7.1357-git.jar" verfügbar. Diese spezielle Jar, besitzt neben den Klassen vom NetworkParser auch noch die Klassen von JGit für das Projektmanagement (siehe Abschnitt 5.18). Außerdem besitzt dieser noch eine Gradle Taskdatei, welches es unter anderem ermöglicht die Versionierung eines Projekts mittels GIT Historie aufzubauen. Zusätzlich kann der BlackboxTest ausgeführt werden. Dieser ermöglicht es bei dem Erstellen einer neuen Version im "Continuous Integration" den eigenen Programmcode zu testen. Beides kann durch die erstellten GradleTasks nach dem Anlegen eines Projektes benutzt werden (siehe Abschnitt 5.18 auf Seite 142).

5.15 Verteiltes System

Das Verteilen von Anwendungen auf mehrere Nodes wurde vereinfacht (siehe Abbildung 41) und mittels der Grundidee der Masterarbeit von Andreas Kiefer implementiert. Für die Verwaltung wurde die zentrale Klasse Space entwickelt. Die NodeProxies sind die Ein- und Ausgabeschnittstellen des Systems. Die IdMap kümmert sich um die Umformung und das Filtern des Modells und der Änderungsnachrichten.

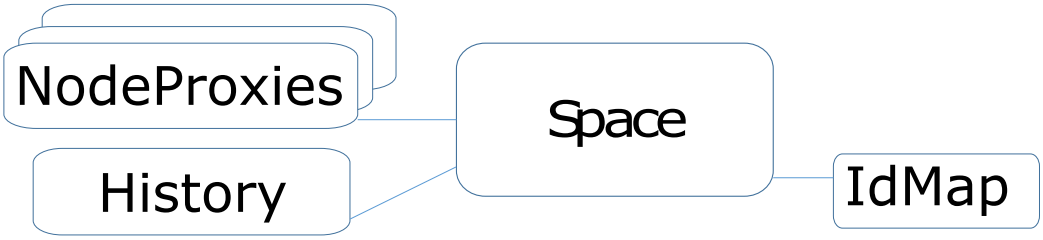

Abbildung 41: P2P

5.15.1 History

Die Historie wird anhand der Change-Nachrichten aufgebaut. Dabei ist es wichtig, dass jede Historie auf allen Rechnerknoten dieselbe Sortierung der Change-Nachrichten hat, um ein eindeutiges Verhalten zu garantieren.

Change-Nachricht

Eine Änderungsnachricht enthält eine Reihe von Metainformationen.

- eindeutige Änderungs-ID
- ID des Vorgängers
- Die Rechnerknoten, welche die Nachricht schon bekommen haben (Erster ist Absender)
- Die Rechnerknoten, die die Nachricht noch bekommen sollen

Refactoring

Die Änderungsnachrichten werden in einer Historie gespeichert, diese wird im Arbeitsspeicher vorgehalten. Da dieser allerdings begrenzt ist, werden die Änderungsnachrichten zusammengefasst. Es gibt zwei unterschiedliche Fälle. Zum einen wird probiert, die Initialisierung des Datenmodells anzupassen und zum anderen werden Änderungen an demselben Objekt zusammengefasst. Da es vorkommen kann, dass nicht alle Clients im Netzwerk dieselbe Version besitzen, werden nur die Änderungen zusammengefasst, welche allen bekannt sind. Weiterhin können so auch Änderungen in der Mitte aller Änderungen zusammengefasst werden, um Patches zu erstellen.

```
{session:"42",class:"ChangeMessage",id:"da39a3ee5e6b4b0d3255bfef95601890afd80709",
    received:["192.168.1.42"],prevChange:"da39a3ee5e6b4b0d3255bfef95601890afd80709",
    changeid:"S810276874033685",property:"first",new:"Alberto",changeclass:"Student"}
{session:"42",class:"ChangeMessage",id:"9wgzqtkml31ggjupxpiufchtsnj8n839ns8u32vz",
    received:["192.168.1.42"],prevChange:"da39a3ee5e6b4b0d3255bfef95601890afd80709",
    changeid:"S810276874033685",property:"name",new:"Albert",old:"Alberto", changeclass:"Student"}
{session:"42",class:"ChangeMessage",id:"g14ww8p0ypibnz6efhkgw1t4mhgkqyhxo6mu9x4z",
    received:["192.168.1.42"],prevChange:"9wgzqtkml31ggjupxpiufchtsnj8n839ns8u32vz",
    changeid:"S810276874033685",property:"age",new:"42", changeclass:"Student"}
```

Quellcode 48: Change Refactoring

```
{session:"42",class:"ChangeMessage",id:"1c9s1p0mnr804652zouqn9oty535l6m5lqv1e1bf",received:["192.168.1.42"]
    prevChange:"1c9s1p0mnr804652zouqn9oty535l6m5lqv1e1bf",changeid:"S810276874033685",
    changes:[{property:"name",new:"Albert",old:"Alberto"},{property:"age",new:"42"}], changeclass:"Student"}
```

Quellcode 49: Change Patch

5.15.2 Space

Die Klasse Space ist der zentrale Punkt für das verteilte System im NetworkParser. Hier werden alle notwendigen Informationen gesammelt und Rechnerknoten initialisiert. So stehen einige Standardmethoden zum Erstellen von NodeProxies zur Verfügung. Die Basisverteilung nutzt eine Ringverteilung mit 2 Sicherheitsringen. Die Sicherheitsringe sind jeweils um einige Knoten versetzt, so dass ein Ausfall oder eine langsame Verknüpfung von Knoten umgangen werden kann. Weiterhin sind dort die ChangeNachrichten und Reconnectintervalle hinterlegt. In der Space-Klasse ist die Anzahl der Sicherheitsringe für die Ringverteilung einstellbar. Es gibt einfache Einstellungen für das Einstellen der zu benachrichtigen Rechnerknoten. Die Rechnerknoten können mit einfachen Filtern eingestellt werden beziehungsweise kann die komplette Ermittlung der zu benachrichtigen Rechnerknoten durch eine eigene Klasse angepasst werden.
Bei der Arbeit von Christian Schneider [Sch07] werden Modelle im CTR Format gespeichert. Änderungen können dort auch auf mehreren Knoten ermittelt werden und ausgetauscht werden. Das Verfahren setzt allerdings einige Dinge voraus:

1. Gleiches Modell-Schema bei allen Rechnerknoten
2. Gleiches Root-Element
3. Reihenfolge bei Änderungen: Für das Ändern eines neuen Elements muss die Nachricht des Erzeugens vorher gesendet und verarbeitet sein
4. Gleiche Modelle auf allen Rechnerknoten
5. Senden inklusive quittieren

Es gibt in der Arbeit aber auch interessante Ansätze wie die Konfliktbehebung von Change-Nachrichten mit unterschiedlichen Inhalten. Bei Konflikten werden die fremden Änderungen übernommen und die lokalen Änderungen verworfen. Dieses hat den Vorteil, dass der Anwender visuell sieht, welchen Zustand das Modell hat. Für die Ermittlung von Änderungen können auch "Id" mittels Prefix eindeutig für Rechnerknoten erstellt werden. Ein anderer Ansatz ist in der Arbeit von Florian Heerdegen beschrieben,

[Hee14] wo ein generischer Vergleich von Modellen gezeigt wird, um Modelle und Teilmodelle zusammenzuführen.

Eine IdMap für das automatische Senden und Anwenden von Änderungen, steht in einem Space automatisch zur Verfügung oder kann eingestellt werden. Die Änderungen werden in der Historie gespeichert. Für das Abarbeiten steht ein eigener Executor (threadsafe) zur Verfügung. Für das Serialisieren und Deserialisieren steht ein Tokener und ein ByteConverter zur Verfügung. Das Datenmodell kann dann mittels Path und einstellbaren ProxyFilter umverteilt werden. Bei der ProxyFilter-Klasse können für jede Modellklasse verschiedene Proxies registriert werden. Ein automatisches Backupsystem, ein LogSystem und ein FehlerHandler stehen bereit und können individuell angepasst oder ersetzt werden. Die Merge-Konfliktbehebung geschieht in der History-Klasse. In der History wird geschaut, ob die Änderung bekannt ist, oder ohne Konflikte an die Vorgängeränderung angehängt werden kann. Falls die Vorgängeränderung nicht bekannt ist wird diese bei anderen NodeProxies nachgefragt. Ansonsten wird probiert anhand der zusätzlichen Information wie z.B. alter Wert und Position den Konflikt aufzuheben. Falls der Konflikt nicht automatisch aufgelöst wird, wird die Änderung trotzdem in das Modell übernommen und die Space-Klasse über einen speziellen Listener informiert. Das Standardverhalten beinhaltet dann ein Persistieren des Konfliktes.
Der Entwickler kann Listener registrieren, um seine Oberflächen auf spezielle Events aus dem verteilten System reagieren zu lassen. Somit ist das einfache Benutzen mit Standardeinstellungen mittels drei Zeilen möglich und kann dennoch auf jeden Spezialfall angepasst werden.

```
1 University uni = new University();
2 IdMap map = UniversitySet.createIdMap("42");
3 Space space=new Space();
4 space.withModel(map, uni);
5 space.with(NodeProxyTCP.createServer(4242), NodeProxyTCP.
    search(4242));
```

Quellcode 50: Simple Space

5.15.3 Kommunikationswege

Damit die Kommunikationsschnittstelle möglichst auf alle denkbaren Einsatzgebiet passt, wurde die Kommunikation zwischen den Rechnerknoten losgelöst von dem Verarbeiten und der Bereitstellung des Modells implementiert. Weiterhin wird auch eine Reihe von unterschiedlichen Kommunikationswegen (siehe folgende Seiten) unterstützt. Diese erlauben eine synchrone und asynchrone Kommunikation. Es wird außerdem zwischen eingehenden und ausgehenden Proxies unterschieden. Somit sind verschiedene Variationen möglich. So können Proxies für nur die eingehende oder ausgehende Kommunikation benutzt werden, es besteht aber auch die Möglichkeit, dass ein Proxy für beide Richtung zur Verfügung steht. Im Folgenden sind die wichtigsten Kommunikationswege beschrieben.

FileSystem

Mit dem FileSystem-Proxy ist es möglich Textdateien zu erstellen. Diese bilden entweder das vollständige Modell ab oder beinhalten nur die Änderungen als Historie. Die FileSystem-Proxies dienen zur Übergabe (Versionierungsdateien) und als Backupsystem. Weiterhin ist es möglich für jeden Rechnerknoten, eine eigene Datei anzulegen, so dass in einem geteiltem Verzeichnis jeder Benutzer seine Datei besitzt und diese mittels Verzeichnissynchronisationsdienst wie DropBox [dro18], Google Drive [Inc18b], GIT [TH10] oder FreeFileSync [zen18] synchron zu halten. Der NetworkParser stellt hierfür die Klasse FileWatcher bereit, welche entweder per ”java.nio” einen FileWatcher registrieren kann, oder per Intervall die Dateien auf Änderung untersucht.

Excel

Die Unterstützung von Excel bietet sich gerade für Daten an, welche in einer zweidimensionalen Form vorkommen, oder mittels einfachen Konverters für Tabellenauswertungen wie bei ConfNet. Es ist somit schnell möglich automatische Reports zu generieren und Daten aus dem System für weitere Bearbeitung zu exportieren.

E-Mail

Mit der "SMTPSession" Klasse ist es möglich Plain und HTML-EMails über TLS [rfc19, Res08] an andere Absender oder Programme zu senden. So ist es denkbar Nachrichten oder Änderungen von Modellen als EMail-Reports zu senden. Der Proxy kann über einen Listener individuell angepasst werden, so dass einer einfachen Benutzung nichts mehr im Wege steht.

Google Cloud Messaging

Das Google Cloud Messaging [Goo12] erlaubt es Push Nachrichten an Geräte mit dem Betriebssystem Android zu senden. Hierfür wird auf dem Android-Gerät nur eine Internetverbindung und der Google PlayStore benötigt. Die Nachrichten werden im XMPP-Format übermittelt. Dadurch ist es möglich Nachrichten an Tablets, Smartphones und SmartTV zu senden. In meiner Masterarbeit [Lin12] wird damit der Use-Case des serverunabhängigen Babyphone realisiert.

Local

Der locale Proxy ist ideal für Tests, GUI-Komponenten und sonstige Erweiterungen geeignet. Er besitzt die Möglichkeit einen einfachen Listener per "ObjectCondition" zu registrieren, welcher bei allen Änderungen benachrichtigt wird.

```
1 Space space=new Space().with(NodeProxyLocal.create(e -> {
2   System.out.println(e);
3   return true;
4 }));
```

Quellcode 51: Local Listener

ModelProxy

Bei dem ModelProxy können Datenmodelle gehalten und synchronisiert werden. Der Proxy unterscheidet sich nicht von den anderen Kommunikationswegen. Jedes Modell oder Sammlung von Objekten wird ebenfalls als Ein- und Ausgabeschnittstelle der Space-Klasse betrachtet. In dem Proxy können spezielle Filter zu definieren werden, um einem Rechnerknoten nur ein Teilmodell zuzuordnen. Diese schränken ein Modell durch eine hierarchische Struktur oder durch eine Scope-View, welche es erlaubt verschiedene Instanzen eines Klassentypen von einer Menge zu halten, ein.

Rest

Zur Kommunikation ist es weiterhin möglich ein RESTServiceTask als zentraler REST-Server zu initialisieren. Der Start eines solchen Rest-Service ist im Quellcode 52 zu sehen. Der implementierte REST-Service umfasst im Moment die Nachrichtentypen "GET" und "POST", "DELETE". Damit ist es möglich eine einfache Schnittstelle zwischen verschiedenen Programmen zu realisieren. Mit dem "GET" Befehl können einfache Elemente oder Collections abgerufen werden (siehe Quellcode 53). Mit dem "DELETE" Befehl können einzelne Elemente gelöscht werden und mit dem "POST" Befehl können neue Elemente angelegt oder verändert werden. Der Service bietet einen einfachen Listener an, welcher mittels SimpleEvent erweitert werden kann und welcher die Berechtigungen der aufrufenden Netzwerkverbindung überprüft, um ein Zugriff ggf. abzulehnen. Das einfache Initialisieren ist mittels weniger Zeilen möglich. Das Beispiel 52 zeigt einen HTTP-Server (Restdienst) auf Port 8080, welcher mit der übergebene IdMap und dem Root-Objekt "uni" funktioniert. Eine Schnittstellenbeschreibung wurde unter [Lin19b] angelegt.

```
1 IdMap map = UniversityCreator.createIdMap("42");
2 University uni = new University().withName("Uni Kassel");
3 RESTServiceTask task = new RESTServiceTask(8080, map, uni)
4 // new Thread(task).start();
5 Executors.newSingleThreadExecutor().execute(task);
```

Quellcode 52: Rest-Service

```
1 task.executeRequest("/json/") -> {"class":"University","id":"U1",
    "prop":{"name":"Uni Kassel"}}
2 task.executeRequest("/changes/") -> Listener
```

Quellcode 53: Rest-Beispiel Java-Aufruf und Response

SQL

Relationale Datenbanken werden durch den SQL-Tokener unterstützt. Dieser kann als Update-Listener an einer IdMap registriert werden und formt die Änderungen in SQLStatement um. Mittels der ”encode”-Methode in dem SQLTokener kann aus einem ClassModel oder einer Objektstruktur ein Datenbank-Schema ableiten werden. Bei den Datenbank-Schemas werden ”zu eins”-Kanten mit der direkten Element-ID des Targets realisiert. Der NodeProxySQL beherrscht zwei Arten von Persistierung. Zum einen als relationale Datenbank, welche jedes Attribut und Assoziation in eine eigene Datenbankspalte umformt. Zum anderen eine Tripelpersistierung, wo die Werte mittels ”Objekt-Id”, ”Propertyname” und ”Value” gespeichert werden. Mit diesen Verfahren ist es möglich relationale Datenbanken für Szenarien zu nutzen, wobei davon auszugehen ist, dass sich das Datenmodell im Laufe der Wartungszeit noch ändert, oder die Einträge sehr starke Unterschiede von der Anzahl der Werten besitzen.

TCP/UDP

Hier wurde ein Adhoc-Proxy implementiert. Dieses ist der Standardfall im ConfNet Use-Case (3.2). Die Kommunikation wird ohne Bestätigungsnachrichten im Ring [TW14] mit einstellbarer Anzahl der doppelten Nachrichten verteilt. So wurde ein einfaches ausfallsicherer Netzwerk modelliert. Der TCPProxy kann auch als BroadCast Proxy eingesetzt werden. Dieser hat die Möglichkeit einen UDP-Service aufzusetzen, mit dessen Hilfe sich andere Clients automatisch finden und konfigurieren lassen (siehe Quellcode 50 auf Seite 131).

XMPP [Shi02]

Die Wichtigkeit der bestehenden verteilten Nachrichtensysteme darf nicht unterschätzt werden. Es wurde somit das XMPP-Verfahren adaptiert. Allerdings hat sich herausgestellt, dass Google seit dem 27.6.2017 und Facebook seit dem 30.04.2015 diesen Dienst nicht mehr in ihren Nachrichten-Messenger unterstützen. Die zwei Unternehmen setzen auf interne Entwicklungen, um die Benutzer besser an sich binden zu können. Es gibt allerdings gerade in den universitären Umkreisen und in der freien Community zahlreiche freie XMPP Server, mit dessen Hilfe eine freie verteilte Kommunikation möglich ist. Die freien XMPP-Server, dienen als Chatserver und sind teilweise sogar als OpenSource Variante verfügbar.

MQTT [Pip18]

Dieses offene Nachrichtenprotokoll ist ideal für die Machine-to-Machine-Kommunikation. Das Übertragen funktioniert damit wie gewohnt mit einfachen Nachrichten. Diese Dienste sind weit verbreitet und dienen gerade zur Integration von leistungsschwachen Systemen oder einfachen Sensoren. Die Nachricht hat einen kurzen binären Header und wird mittels der MessageSession Klasse kodiert.

AMQ [SDB11]

AMQ ist ähnlich wie MQTT. Es handelt sich ebenfalls um ein binäres Format. Es ist minimal größer, bietet aber durch die mehr Headerbytes eine besser hierarchische Verteilung von Kommunikationen und Nachrichten.

```
1 NodeProxyBroker broker = new NodeProxyBroker("avocado.uniks.de
    :32777");
2 broker.withFormat(MessageSession.TYPE_AMQ);
3 // MessageSession.TYPE_MQTT
4 broker.connect();
```

Quellcode 54: MQTT/AMQ-Beispiel

5.15.4 SimpleReplikation

Aufgrund einer Neugestaltung der Lehrveranstaltung "Programmiermethodik" wurde das Thema von Persistierung und verteilten Systemen in den Lehrplan aufgenommen. Für das Heranführen der Studierenden an die Thematik wurde die Klasse SimpleReplikation implementiert. Diese verknüpft sich mit einer JavaFX Oberfläche und einem Datenmodellelement. Für jedes Element kann der Primärschlüssel angegeben werden. Dieses kann mittels Übergabeparameter oder per Creator mitgegeben werden. Das Binden an der Oberfläche funktioniert bidirektional und wird anhand des GUIKey-Event "Key-Release" ausgelöst (siehe Quellcode 55). Die Klasse steht den Studierenden in der SimpleController zur Verfügung. Die SimpleController kümmert sich um das Anzeigen der Oberflächenelemente. Die Klasse ModelListenerProperty dient als Listener der Oberfläche und Veränderungen an das Datenmodell weiterleitet. Der DiceController ist eine Standardkomponente, welche ein Würfel mit einer Maximalen neun Augen verwalten kann.
Das Element wird erst in der IdMap unter dem Primärschlüssel gespeichert, wenn die Eingabe des Primärschlüssel beendet wird oder per "Enter" Taste bestätigt wird. Zum Verteilen wird die Klasse "Space" verwendet. Dort wird ein gemeinsames Modell für alle Knoten in Netzwerk erstellt.

```
1  SimpleController controller = SimpleController.createFX();
2  controller.withMap(LudoSet.createIdMap("42"));
3  controller.withMap(new DiceController(), "dice");
4  Ludo game = new Ludo();
5  game.init(new Player().withName("Albert"), new Player().withName("Stefan"));
6  ModelListenerProperty controllerFactory = new ModelListenerProperty();
7  for(int p=1;p<=game.getPlayers().size();p++) {
8    Player player = game.getPlayers().get(p-1);
9    controller.withMap(controllerFactory,"p"+p, player, Player.PROPERTY_NAME);
10   controller.withMap(controllerFactory, "p"+p+"_textfield", player, Player.
         PROPERTY_NAME);
11 }
```

Quellcode 55: SimpleController

5.16 Transpiler j2js

Eine weitere Idee war es eine verteilte Anwendung zu schreiben, welche eine feste kaum veränderte Basis besitzt und wo die Anwendung nachgeladen wird. Bei einer agilen verteilten Anwendung wird eine plattformabhängige Basis entwickelt und verteilt. Die Anwendung soll aber losgelöst von der Applikation sein. Also dem Datenmodell, Programmlogik und Oberfläche. Diese werden mittels Javascript und HTML nachgeladen.

Als Programmiersprache steht dem typischen Programmiermethodik-Entwickler Java zur Verfügung. Damit dieser nicht zwischen den Programmiersprachen wechseln muss und als besseren Einstieg wurde ein Java Transpiler implementiert. Dieser ermöglicht einen Programmcode von Java nach möglichst gleich aussehenden Javascript zu portieren ohne Verluste, wie zum Beispiel Kommentare und Variablennamen. Hierfür wurde die Projektarbeit und die Bachelorarbeit von Enno Boland verwendet. Das Tool steht im Moment als externes Projekt zur Verfügung [jav17].

5.17 Tools

Im Zuge der Entwicklung vom NetworkParser wurden eine Reihe von Hilfskomponenten entwickelt, welche für die Funktionalität benötigt werden. Darunter zählen gerade numerische und logische Ausdrücke. Diese Tools werden mit dem NetworkParser ausgeliefert.

5.17.1 Calculator

Der Calculator wurde bei einem Drittmittelprojekt für die dynamische Berechnung von Tabellenzellen benötigt. Er umfasst einfache Rechenoperationen, Klammersetzung und Funktionen. Es ist somit möglich eigene Funktionen zu definieren. Angelehnt an das Verhalten von Microsoft Excel/LibreOffce Calc.

5.17.2 Logik

Die Logikkomponente wurde in Anlehnung einer Bachelorarbeit entworfen, mit dessen Hilfe ein Fragebogen komplett in einem Designer modelliert werden kann [Wit15].
Die Logikkomponente wird zur Modellierung von Programmlogik und im Codegenerator verwendet. Weiterhin können damit die Proxies im verteilten System konfiguriert werden. Da es sich bei den Logikkomponenten um eigene Klassen handelt, beinhalten diese bereits alle Möglichkeiten der Persistierung. Somit können Logikanteile von einem Rechnerknoten zum anderen übertragen werden.

Die Logikkomponenten bestehen aus

- And
- Between
- BooleanCondition
- ChainCondition
- CompareTo
- CustomCondition
- Equals
- FeatureCondition
- ForeachCondition
- IdFilterElements
- IfCondition
- ImportCondition
- InstanceOf
- ListCondition
- MapFilter
- Not
- Or
- PatternCondition
- SimpleObjectFilter
- StringCondition
- TemplateCondition
- TemplateFragmentCondition
- VariableCondition
- WhiteListCondition

und bieten so die Möglichkeit einen UML-Aktivitätsdiagramm Editor zu implementieren.

5.17.3 Soft Dependencies

Um den NetworkParser unabhängig zu gestalten und trotzdem die Features für SDM zu realisieren wurden die verwendeten ThirdParty Framework ohne feste Abhängigkeiten eingerichtet. Einige Third-Party Frameworks sind nicht in jedem Anwendungsszenario verfügbar. Weiterhin verringern Abhängigkeiten die Wiederverwendung. Die Wiederverwendung wirkt sich häufig positiv auf die Entwicklungsgeschwindigkeit, die Codequalität, die Lesbarkeit des Codes aus.
So wurden AWT, JavaFX, EMF, JUNIT, GIT per Soft-Abhängigkeiten im ReflectionLoader hinzugefügt. Diese Klasse kann dynamisch per ClassLoader die Fremdbibliotheken laden. Der ReflectionLoader vereinfacht die Reflektionsschicht von Java und fängt die Fehler ab beziehungsweise leitet diese weiter. Somit ist es möglich auf nicht eingebundene oder vorhandene Bibliotheken zu reagieren. So wird der GUI Editor entweder mit der JavaFX Bridge geöffnet, oder mittels Server im Standard-Webbrowser. Der ReflectionLoader wird auch nicht in jeder Jar-Ausprägung ausgeliefert. So wird bei der ”core16” und ”core17” diese aufgrund von Kompatibilität zum Beispiel mit ”GWT” ignoriert.

5.17.4 EntityUtil

Bei vielen Features des NetworkParser gibt es immer wieder dieselben Hilfsmethoden, welche in der Klasse EntityUtil zusammengefasst. Diese sind auch nach außen benutzbar, so dass ein Entwickler diese statischen Methoden in seinem Programmcode integrieren kann. Diese Klasse enthält allgemeine Konvertierungen für Texte und Zahlen. Weiterhin sind dort eine Reihe von Methoden implementiert, welche für die Codegenerierung benötigt werden.

- Quote und UnQuote
- Clone
- compareEntity
- containsAny
- convertPrimitiveToObjectType
- countMatches
- doubleToString
- equalsAny
- getExcelRange
- getPath und getRelativePath
- getStringType
- isEMFType
- isEmpty
- isModifier
- isNumeric, isNumericType und isNumericTypeContainer
- isPrimitive und isPrimitiveType
- isValidJavaId
- randomString
- repeat
- replaceAll
- shortClassName
- stringEquals
- strZero
- upFirstChar
- valueToString

5.18 Projektmanagement

Es hat sich gezeigt, dass es schwierig ist Studierende mit den ganzen Techniken zu schulen und gleichzeitig den neuen Lehrstoff mit Hausaufgaben zu stellen. Somit wurde angefangen bei dem NetworkParser einige Hilfen zum Projektmanagement bereitzustellen. So wird mittels Aufruf 56 ein vollständiges Projekt initialisiert. Die einzelnen Schritte sind

- Ordner Ludo-Project anlegen
- Die Projektdateien für die IDE anlegen
- Das Projekt mit der Gradle Struktur versehen und den aktuellen Gradle Wrapper hinzufügen
- Ein "git init" ausführen
- Das übergebene Remote Repository hinzufügen. (Falls angegeben)
- Das Projekt wird mittels Angabe des notwendigen Lizenztexts versehen. Weiterhin wird auch eine Manifest-Datei hinzugefügt.

```
1 java -jar NetworkParser.jar init LudoProject MIT
2   https://github.com/StefanLindel/LudoProject.git
```

Quellcode 56: Projekt Initialisierung

5.19 Cucumber

Es hat sich gezeigt, dass es für Programmieranfänger oder für Programmierfremde, wie zum Beispiel Kunden, nach kurzer Schulung möglichst sich mittels Cucumber textuelle Ausdrücke zu formulieren. Um die textuellen Szenarien besser zu unterstützen wurde ähnlich wie in Dissertation [Dre15] ein Übersetzer geschrieben, welcher es ermöglicht textuelle Szenarien in Objektdiagramme zu übersetzen.
Im NetworkParser wurden zwei Klassen implementiert, die "Cucumber"-Klasse und "CucumberStdRule"-Klasse. In der Klasse Cucumber kann ein Wörterbuch definiert werden, um eine Übersetzung vom Text zu ermöglichen. In diesem Wörterbuch kann zur Laufzeit die Cucumberklasse spezielle Wörter übersetzen. Die "CucumberStdRule"-Klasse erlaubt ein einfaches Parsen von

Sätzen. So werden von den Wörtern die Wortstämme gesucht. Weiterhin kann man mittels eines einfachen Wörterbuchs Wörter ersetzen oder auf die Ignore-Liste setzen. Die Standardimplementierung probiert dann die Nomen und Verben zu identifizieren. Weiterhin werden dann Attribute und Links mit einem separaten Wörterbuch hinzugefügt. Aus Nomen, die mit 'and' getrennt sind werden "Many"-Kanten. Die vereinfachte Grammatik sieht wie folgt aus:

$$Assoziationen : Nomen + Verb\ Nomen+$$

$$Attribut : Nomen\ Object(Nomen)Value$$

```
Title: Defining the start player
Definition: Alice is a Player
Definition: Bob is a Player
Definition: dice is a Dice
Definition: ludo is a Game

Given: Alice and Bob play ludo
Given: Alice has the dice with value 5
```

Quellcode 57: Einfaches Cucumberbeispiel

```
1 Cucumber scenario = Cucumber.createScenario("defining the start
    player");
2 scenario.Definition("Alice is a Player");
3 scenario.Definition("Bob is a Player");
4 scenario.Definition("dice is a Dice");
5 scenario.Definition("ludo is a Game");
6
7 scenario.Given("Alice and Seb play ludo. Alice has dice with
    value 5");
8 scenario.analyse();
9 //Cucumber scenario = Cucumber.createFromFile(file)
```

Quellcode 58: Cucumber createScenario

In dem Beispiel 58 wird im Programmcode ein textuelles Szenario erstellt. Dieses kann allerdings auch wie in Zeile 9 aus einer Textdatei eingelesen werden. Der eigentliche Text ist in Beispiel 57 dargestellt.

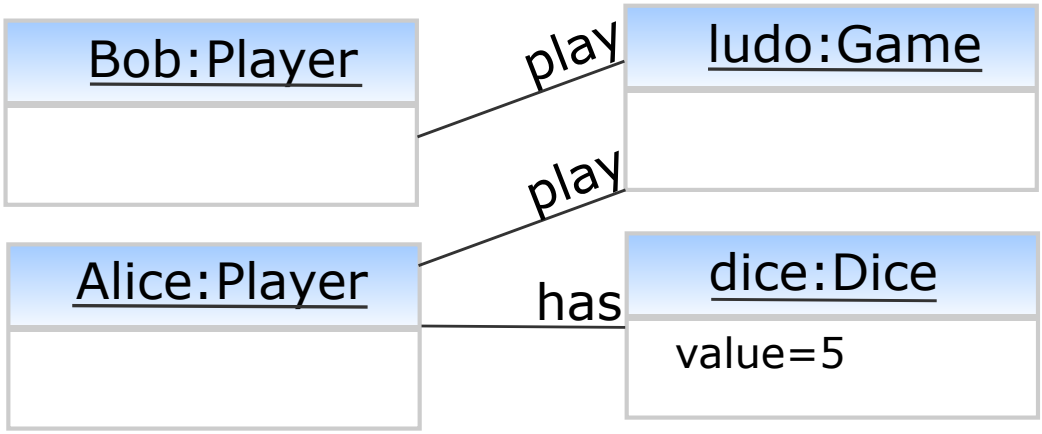

Abbildung 42: Einfaches Cucumberbeispiel als Objecktdiagramm

Anders als in der Dissertation von Jörn Dreyer wurden keine Webdienste wie "vornamen.com" eingebunden. Dieses führt nicht immer zu den erhofften Zielen. So wird Ludo statt als Spiel als Vorname erkannt. [Elt18]

5.20 OCL

[OCL19] Aus der Inspiration von MontiCore [mon18,Rot17] wurde die OCL Sprache in den Grundzügen integriert. Bei dieser UML Sprache handelt es sich um eine textuelle Spezifikationssprache für das Definieren von Randbedingungen für Datenmodelle. Die Klasse "OCLParser"' kann einfache OCL Constraints in die Logikkomponenten vom NetworkParser überführen, so dass es möglich ist Laufzeitobjekte auf ihre Gültigkeit zu überprüfen. Weiterhin bietet OCL die Möglichkeit Programmcode zu erzeugen (Siehe [Rum16] ab Seite 139).

```
1 String item = "context house inv: self.floor > 0";
2 IdMap map = new IdMap().withCreator(new HouseCreator());
3 OCLParser parser = OCLParser.create(item, map);
```

Quellcode 59: OCLParser

6 Ludo

Der Funktionsumfang des NetworkParser Frameworks wird in diesem Kapitel an dem Beispiel des Spiels Ludo erklärt. Als Szenario wird angenommen, dass ein Studierender das Spiel Ludo entwickeln will. Als Erstes macht sich der Entwickler Gedanken, wie das Spiel funktioniert und was er dafür benötigt. Dieses kann auch zusätzlich durch Befragung eines Experten oder durch Internetrecherche geschehen. In Beispiel 60 ist ein einfaches Szenario beschrieben, welches entscheidet, wer bei Ludo anfängt. Das Ziel ist es die Anforderungen in User-Storys zu erfassen, welche im besten Fall von dem Kunden geschrieben werden und somit die Erwartungen an das Programm repräsentieren.

```
Scenario: defining the start player
Definition: Alice is a Player
Definition: Bob is a Player.
Definition: Dice is a Dice. Ludo is a Game. Token is a
   Token. home is a homeField.
Given: Alice and Bob play ludo
Given: the players have tokens on the home.
Given: the players has a name and a color. Alice has a dice
    with value 5.
When: Bob has a dice with 1
Then: Alice is currentplayer from ludo
```

Quellcode 60: Ludo Start

Aus diesem Szenario kann dann manuell oder mit einem Tool ein Objektdiagramm abgeleitet werden. Wenn ein Tool benutzt wird, kann entweder das originale Tool für Cucumber genutzt werden, welches das Szenario als Vorlagentext benutzt und mittels regulären Template-Regeln ein Objektdiagramm erstellen kann. Die Cucumber Regeln können auch mit dem NetworkParser erstellt werden, welcher eine vereinfachte Satzerkennung besitzt und das Diagramm automatisch in ein Objektdiagramm umwandeln kann. Diese Funktionalität ist Bestandteil von dem Story-Feature.

Als Ergebnis wird ein Objektmodell mit dem Inhalt von Abbildung 43 erzeugt.

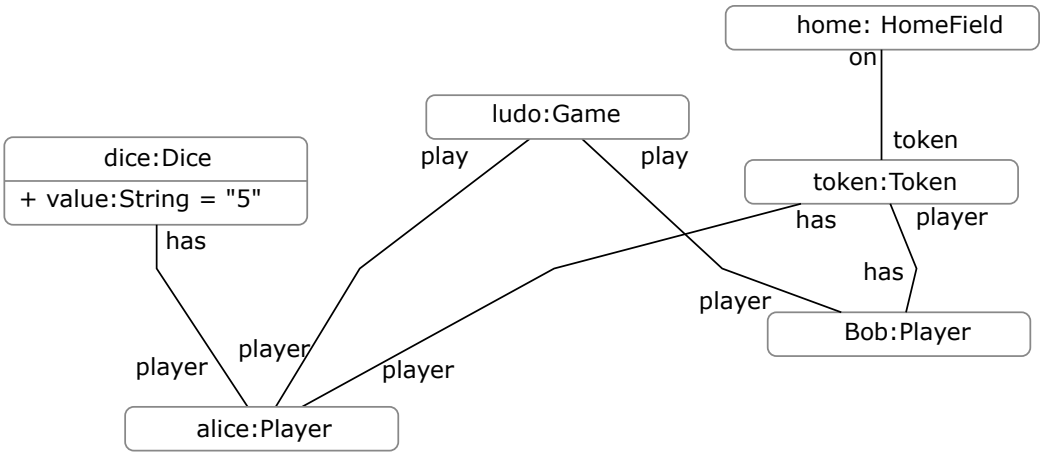

Abbildung 43: Objektdiagramm vom Cucumber Ludo Start

Nach der Startsituation können weitere Szenarien definiert werden. Diese setzen immer das vorherige Voraus. Siehe Quellcode 66, 67, 68, 69 auf den Seiten A - C.

```
1  ClassModel model = new ClassModel();
2  Clazz ludo = model.createClazz("Game");
3  Clazz dice = model.createClazz("Dice");
4  Clazz player = model.createClazz("Player");
5  Clazz token = model.createClazz("Token");
6  Clazz field = model.createClazz("Field");
7  Clazz homeField = model.createClazz("HomeField");
8  Clazz lastField = model.createClazz("LastField");
9  Clazz target = model.createClazz("Target");
10 Clazz startingArea = model.createClazz("StartingArea");
11
12 dice.withAttribute("number", DataType.INT);
13 player.withAttribute("color", DataType.STRING);
14 player.withAttribute("name", DataType.STRING);
15
16 dice.createMethod("throwsMeeple");
17 field.withKidClazzes(homeField, lastField, startingArea, target);
18
19 field.withAssoc(field, "next", ONE, "prev", ONE);
20 player.withAssoc(token,"token",MANY,"player",ONE);
21 player.withAssoc(dice, "dice", ONE, "player",ONE);
22 player.withAssoc(ludo,"currentGame",ONE,"currentPlayer",ONE);
23 player.withAssoc(ludo,"wonGame",ONE,"winner",ONE);
24 player.withAssoc(ludo,"game",ONE,"players",MANY);
25 player.withAssoc(homeField,"home",MANY,"player",ONE);
26 player.withAssoc(startingArea,"start",ONE,"player",ONE);
27 player.withAssoc(target, "target", ONE, "player", MANY);
28
29 token.withAssoc(field, "field",ONE,"token",ONE);
30
31 model.generate("");
```

Quellcode 61: Ludo Classmodel

```
1 model.with("de.uniks.ludo.model");
2 ludo.createMethod("init", Parameter.create("Player..."));
```

Quellcode 62: Ludo Classmodel Erweiterung

Aus den verschiedenen Objektdiagrammen kann der NetworkParser dann Generierungscode für ein Klassendiagramm ableiten. Dieser ist in Quellcode 61 und wird manuell mittels Quellcode 62 erweitert. Die Abbildung 44 zeigt dieses Klassendiagramm. Dieser Programmcode kann in einem Test oder Story verwendet werden und ausgeführt werden. Somit kann aus dem Modell nun Javacode für das Datenmodell generiert werden.

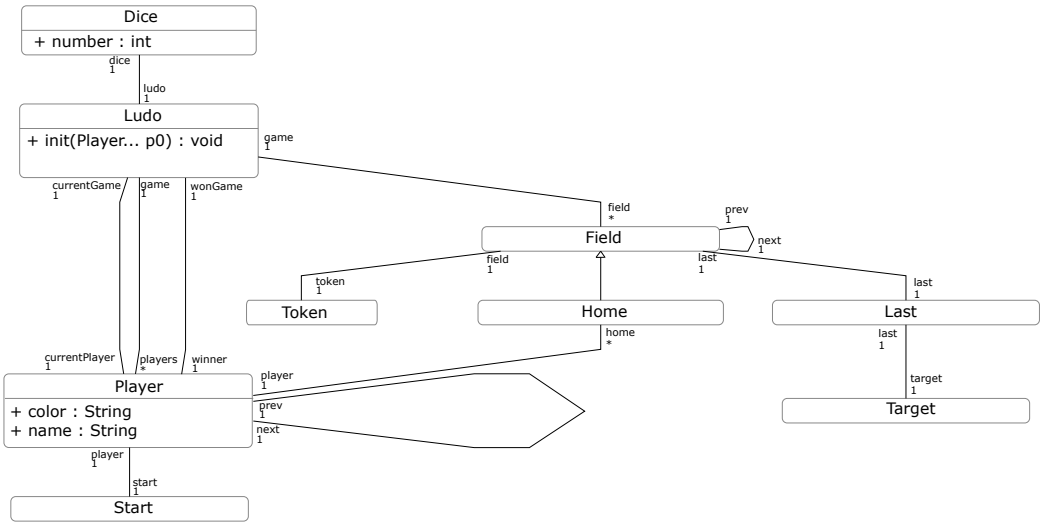

Abbildung 44: Klassenmodell von Ludo

```
1 public boolean moveOut( Home currentHome ) {
2   PatternCondition pattern = PatternCondition.
        createPatternPair(currentHome);
3   HomeSet homePO = (HomeSet) pattern.getRoot();
4   PlayerSet playerPO = homePO.getPlayer();
5   StartSet startPO = playerPO.getStart();
6   TokenSet tokenPO = homePO.getToken();
7   return PatternCondition.setValue(startPO, Start.
        PROPERTY_TOKEN, tokenPO, startPO.getToken().size() ==
        0);
8 }
```

Quellcode 63: Programmlogik: MoveOut

```
1  public boolean moveToken( Field currentField ) {
2    int pips = getGame().getDice().getNumber();
3    if(pips<1 || currentField.getToken()==null ||
         currentField.getToken().getPlayer()!=getGame().
         getCurrentPlayer()){
4      return false;
5    }
6    Field target=currentField;
7    for(;pips>0;pips--){
8      if(target==null){
9        return false;
10     }
11     target = target.getNext();
12   }
13   if(target.getToken()==null){
14     currentField.getToken().setField(target);
15     return true;
16   }else if(target.getToken().getPlayer() != this) {
17     //throw out
18     Token opponentToken = target.getToken();
19     for(Home homeField:opponentToken.getPlayer().getHome())
           {
20       if (homeField.getToken() == null) {
21         homeField.setToken(opponentToken);
22         break;
23       }
24     }
25     currentField.getToken().setField(target);
26     return true;
27   }
28   return false;
29 }
```

Quellcode 64: Programmlogik: MoveToken

Die Programmlogik kann entweder wie im Beispiel 63 durch PatternCondition ausgedrückt werden oder manuell mittels eines Editors erstellt werden. Im Beispiel 64 wurde die MoveToken per Hand implementiert.

Die Oberfläche kann dann mittels Gluon Scenebuilder modelliert und danach mittels FXML Loader hinzugefügt werden (siehe Abbildung 45 auf Seite 150).

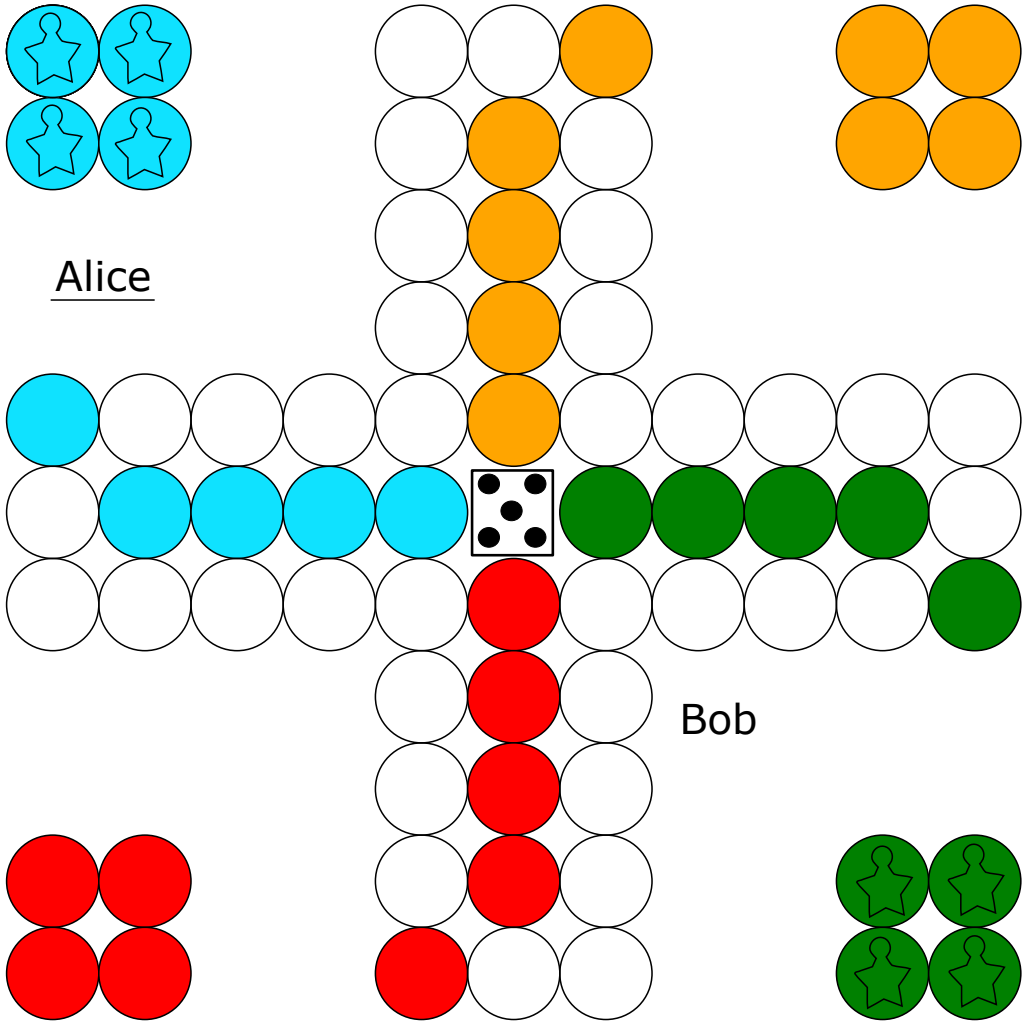

Abbildung 45: FXML von Ludo

Ludo ist spielbar, nachdem die Oberfläche, Businesslogik und Datenmodell miteinander verbunden ist. Das Verbinden ist mittels einer Methode im Umfang von 50 Zeilen geschafft.

Um den aktuellen Zustand des Spiels persistierbar oder das Spiel netzwerkfähig zu machen, müssen folgende Zeilen hinzugefügt werden (siehe Quellcode 65 auf Seite 151).

```
1  // Save
2  IdMap map = LudoSet.createIdMap("");
3  FileBuffer.writeFile("save.json", map.toJsonObject(ludo));
4
5  //Load
6  CharacterBuffer buffer = FileBuffer.readFile("save.json");
7  Ludo ludo = map.decode(buffer);
8
9  // Network initialisieren
10 Space space = new Space();
11 space.withModel(map, ludo);
12 space.with(NodeProxyTCP.createServer(4242), NodeProxyTCP.
       search(4242));
```

Quellcode 65: Ludo Persistierung

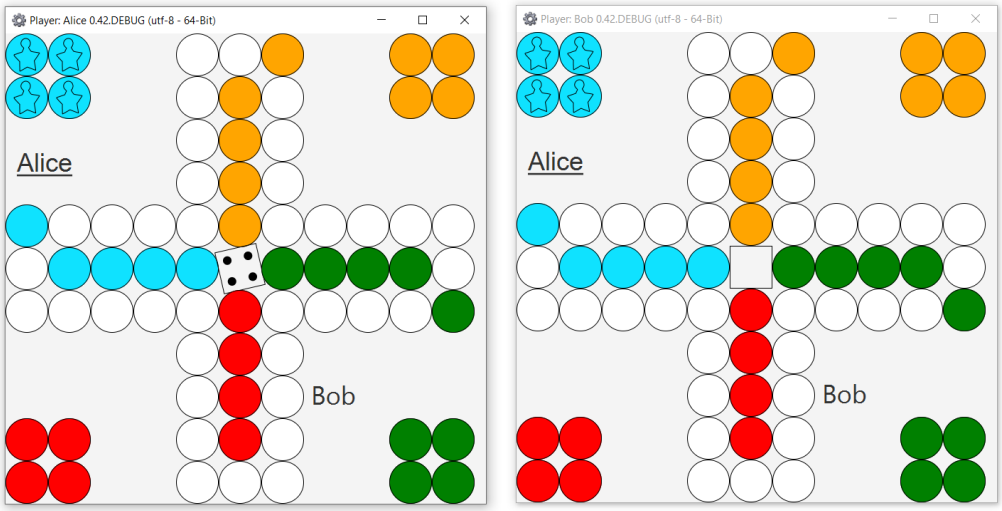

Abbildung 46: Ludo Screendump

7 Fazit

Im Rahmen dieser Arbeit wurden Konzepte und Werkzeuge entwickelt, die bei der Erstellung eines Programms während der einzelnen Entwicklungsschritte einen hilfreichen Beitrag leisten können. Es hat sich bei den Lehrveranstaltungen gezeigt, dass die Studierenden für den Einstieg ein kleines übersichtliches Tool benötigen.
Mit dem NetworkParser ist ein kleines Framework gelungen, welches sich momentan von anderen existierenden Frameworks vom Umfang, Größe und Einsetzbarkeit stark absetzt. Mit seinen 1,4 MB gehört es zu den kleinen Frameworks, umfasst allerdings viel Funktionalität für die einzelnen Entwicklungsschritte.
Es hat sich gezeigt, dass der NetworkParser für die Lehrveranstaltungen von dem Fachgebiet wie 'Programmiermethodik" und "Software Enginering 1" bestens geeignet ist. Gerade die Lehrveranstaltung "Programmiermethodik" umfasst genau die Funktionalität des NetworkParser.

Bei "Software Enginering I" wo die Studierenden ein Client für ein Netzwerkspiel realisieren müssen, kam der NetworkParser zum Einsatz. Die Spielidee wechselt jedes Jahr und die Studierenden müssen gegen eine von uns implementierte Umsetzung des Servers programmieren. Im Sommersemester 2018 wurde der Server mittels NetworkParser realisiert. Alle acht Gruppen nutzen den NetworkParser für die Codegenerierung. Drei der acht Gruppen nutzen auch die Persistierungshilfe.

Der NetworkParser bildet einen guten Grundstein, um zukünftige Entwicklungen zu realisieren. Er bietet mit der Abstraktion der Oberfläche sehr gut die Möglichkeit ein plattformübergreifendes GUI-Framework zu realisieren. Weiterhin können dort diverse Editoren wie der gezeigte EDobs realisiert werden. Der NetworkParser bietet mit den zahlreichen Konvertern für Texte, Binäre Daten und Bilder zahlreiche Einsatzmöglichkeit.
Das Tool ist bestens für das Industrie Use-Case von ConfNet geeignet. Es wird dort seit September 2011 eingesetzt.

7.1 Evaluation

In diesem Kapitel werden die wesentlichen Use-Cases nochmal kurz beschrieben und auf die Ergebnisse vom NetworkParser eingegangen.

7.1.1 Lehre

In der Lehrveranstaltung "Programmiermethodik" existierte bis 2011 keine Lösung, welche den kompletten Entwicklungsprozess abgedeckt hat. Auch in "Software Engineering 1" mussten immer wieder dieselben Dinge umgesetzt werden. Dieses führte zu einer großen Unzufriedenheit bei den Studierenden, welche die Veranstaltung nicht zu Ende besuchten.
Der NetworkParser wurde seit 2012 bei der Lehrveranstaltung "Programmiermethodik" eingesetzt und dort komplett bei jeder Hausaufgabe eingesetzt. In einer Umfrage wurden jedes Jahr zwei Generierungstools gegenübergestellt und mit der aktuellen Version von SDMLib und NetworkParser verglichen. Heraus kam, dass das eigene Tool im Moment nicht schlechter abschneidet, als bereits etablierte kommerzielle Tools siehe Abbildung 47.

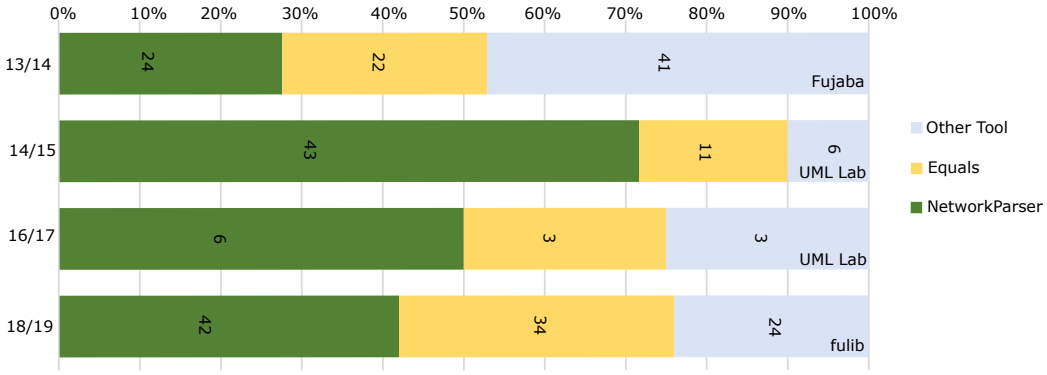

Abbildung 47: PM Befragung

Studierende PM HA	1	2	3	4	5	6	7	8	9
1314	71	71	65	64	63	63	51	51	51
1415	105	104	90	72	64	64	43	42	42
1516	118	108	90	90	77	77	75	69	69
1617	89	88	85	64	52	42	40	31	31
1718	86	78	78	78	64	69	65	65	70

Tabelle 2: Abgaben der Studierenden

Klausur	1	1,3	1,7	2	2,3	2,7	3	3,3	3,7	4	n.b.	Mittelwert
1314	7	1	3	10	3	2	7	3	1	1	13	2,92745098
1415	1	0	3	2	4	1	7	11	2	2	4	3,1
1516	0	1	5	4	16	15	7	9	2	4	6	2,886956522
1617	0	1	0	3	7	4	4	8	1	3	0	2,848387097
1718	7	5	8	7	6	6	5	3	3	2	1	2,266037736

Tabelle 3: Klausurnoten

Anhand der ersten Tabelle (siehe Tabelle 2) sieht man die Anzahl der Abgaben für die Lehrveranstaltung "Programmiermethodik". In der zweiten Tabelle (siehe Tabelle 3) ist der Notenschnitt aufgezeichnet. Der NetworkParser wurde gerade in dem Übungsbetrieb verwendet. Aus dem Mittelwert der Klausur kann man ableiten, dass trotz immer mehr technischen Anforderungen der Lehrstoff immer besser vermittelt werden konnte.

Durch die eigene Umsetzung konnte das Tool exakt auf die Lehrveranstaltung und die Aufgaben der einzelnen Übungsaufgaben angepasst werden. In der wöchentlichen Übung wurde intensiv mit den Studierenden über Probleme diskutiert und so wurde das Tool ständig weiterentwickelt. Somit ist ein ineinandergreifen der einzelnen Aufgaben und ein Wiederverwenden des Teilergebnisse gewährleistet. Das Tool unterstützt zwar die interaktive Weiterentwicklung des Klassenmodells und deren Generierung, jedoch können die Studierende dieses mit dem momentanen Wissensstand nicht nachvollziehen und bezeichnen dieses als "Vodoo". Die Studierenden benutzten das Tool auch im Nachhinein, so wurden einige Projektarbeiten und private Projekte damit realisiert.

7.1.2 ConfNet

Bei ConfNet sollte eine Endanwendung für den PC umgesetzt werden (siehe 5, 6, 7 Seite 30), welche möglichst ausfallsicher und trotzdem einfach durch Ärzte ohne vorhergehende Schulung bedienbar sein sollte. Der NetworkParser kam bei diesem Use-Case an mehreren Stellen zum Einsatz. So wurde der NetworkParser als Persistierungshilfe von Datenmodellen genutzt. Weiterhin wurde die komplette Verteilung des Datenmodells mit dem NetworkParser realisiert. Zum Schluss wurde die Oberfläche dynamisch als Webseite aufgebaut und als eigenständige Fallbackanzeige implementiert. Das bedeutet, die Informationen werden als vollständige HTML Seite gespeichert und mittels Javascript neu eingelesen und übersichtlich dargestellt. So kann das Datenmodell (mit allen Räumen, Tagen und Vorträgen) komplett aus der Ausgabedatei zurück gelesen werden. Die Oberfläche für das Auswählen von Präsentationen ist auch ohne das Java-Programm bedienbar (siehe Abbildung 48). Der Kunde ist sehr zufrieden mit der Anwendung und setzt das Produkt erfolgreich bei sämtlichen Ärztekongresse seit 2012 als Kernsoftware ein. Laut der Firma waren es 2018 über 30 Veranstaltungen bestehend aus Updates (Fortbildungen) und Kongressen. Zu den bestehenden ConfNet haben sich in Laufe der Zeit einige kleinere Applikationen gesellt, wie der "ConfNet Counter" oder "ConfNet Converter".

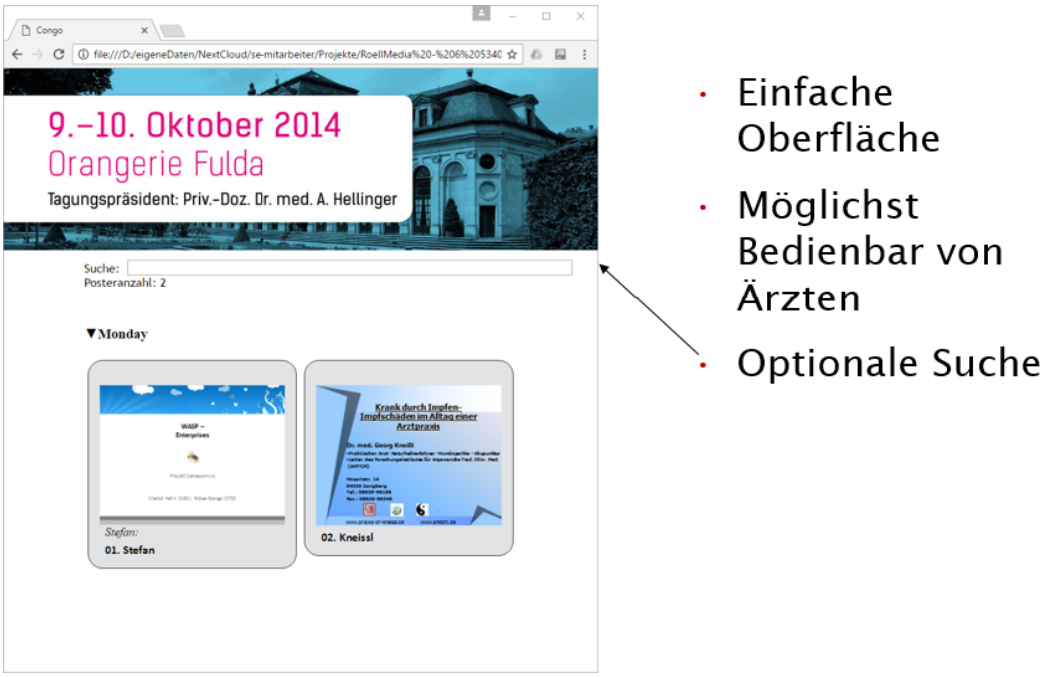

Abbildung 48: ConfNet Web

7.1.3 EONShifting

Bei EONShifting wurde durch einen Zwischenhändler ein Lastenheft entwickelt, welches so umfangreich und nicht Zielführend war, dass das Produkt zum scheiternd verurteilt war. Mit dem NetworkParser und den einfachen USE-Cases ist es gelungen, dieses zu verhindern. Mit dem NetworkParser ist es gelungen eine einfache übersichtliche Oberfläche (siehe Abbildung 49) zu gestalten, welche mittels Inline-Editieren keine weiteren Fenster benötigte. Weiterhin wurde durch die spezielle Collection und durch ein verbessertes Datenmodell die Optimierung der Rufbereitschaft realisiert. Ursprünglich hatte ein Bereichsleiter eine Rufbereitschaft von 40 Personen per Gruppe. Die Gruppen waren nicht disjunkt somit wurden mehrere Mitarbeiter mehrmals eingesetzt. Dieses konnte mit dem Tool auf 12 Mitarbeiter ohne Doppelbelastungen reduziert werden. Das Finden von vier Gruppen für die Rufbereitschaft dauerte bei 1500 Mitarbeiter und 500 Einsatzorten 30-45 Sekunden. Die Anwendung kam bei dem Zwischenhändler und dem Vertreter des Energiekonzerns gut an. Allerdings wurde die Software

aus innenpolitischen Gründen (Bedenken der Gewerkschaft) nicht bei EON eingesetzt.

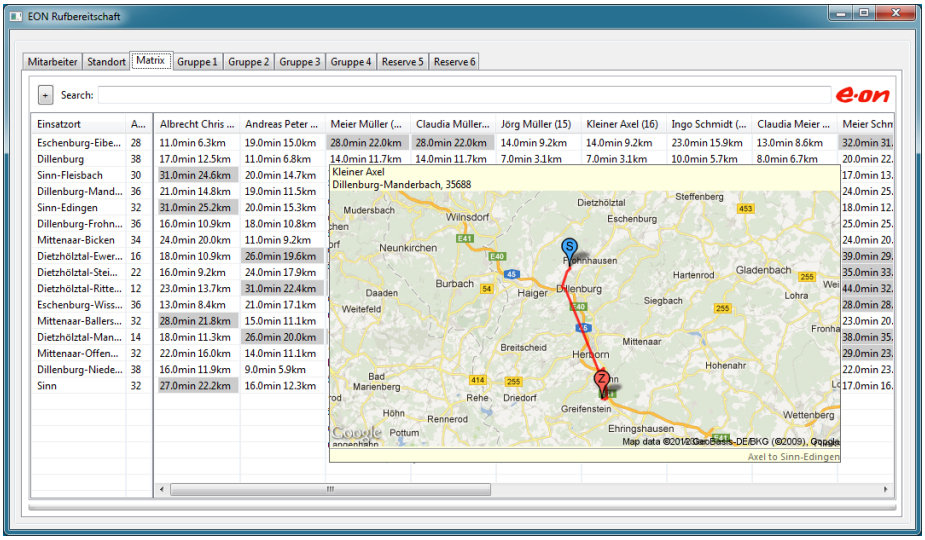

Abbildung 49: EONShifting

7.2 Metriken

Es wurden auch einige Metriken implementiert, um den NetworkParser zu bewerten. So wurde der InBound-Link-Faktor über alle Klassen implementiert.

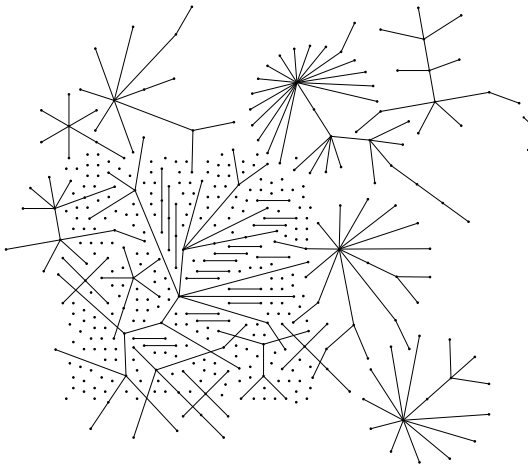

Abbildung 50: Kopplung Klassen

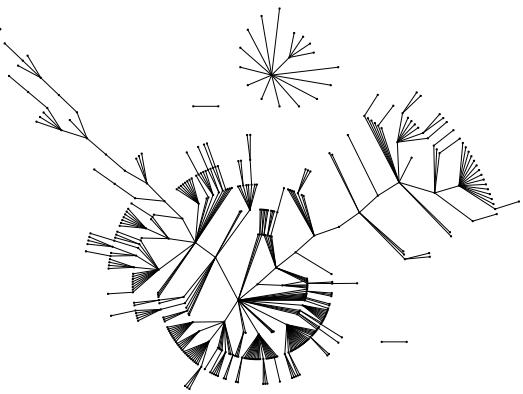

Abbildung 51: Vernetzung Klassen

So sieht man in Abbildung 51, dass die Klassen sehr eng miteinander verzahnt sind, sodass ein Framework entstanden ist, in welchen die Modularisierung sehr stark verwendet wurde.
Für einen guten Kennwert wird in Abbildung 50 ein niedriger Wert verlangt. Dieses ist an sehr vielen Stellen gelungen.

7.3 Analyse

Das komplette Framework wurde unter anderem für den TTC [Afc18] durch das Speicheranalysetool JProfiler [JPr19] und Eclipse MAT [Ecl19] optimiert.

7.4 Ausblick

Der NetworkParser besticht gerade durch seine Kompaktheit und die Vermeidung von Dependencies. Ohne die Dependencies eignet sich der NetworkParser hervorragend, für Lehrveranstaltungen und Tools, die in Resourcenarmen Umgebungen funktionieren müssen. Die einfache Einbindung in bestehende Software durch Gradle oder durch einfaches Hinzufügen einer Jar gewährleistet eine gute Basis für eine starke Verbreitung. Das Framework umfasst 66960 Lines of Code in 440 Dateien.

7.4.1 Story

Der NetworkParser kann auch für die agile Umgebung erweitert werden. Hierfür soll ein Agiles Tool entwickelt werden, welches mittels Scrum Projektmanagement vereinfacht. Zusätzlich soll die HTML-Darstellung erweitert werden um die "Story-Card" wie in Abbildung 52 dargestellt [Kmh14]. Außerdem sollen die Daten des Projektmanagement als GIT-Blobs gespeichert werden und der komplette Ablauf der Task als Git-Branches umgesetzt werden. Somit soll es möglich sein, mit dem NetworkParser Projekte zu verwalten und strukturiert zu bearbeiten (SimpleScrum). In [Lin15, uMS18] sind verschiedene Ansätze gezeigt, welche dann in den NetworkParser überführt und zusammengefasst werden müssen. Bei der Bachelorarbeit von Marcus Schmidt und Felix Dausch wurde eine Webapplikation mit Google Firebase umgesetzt, welche für jeden Task einen eigenen Branch anlegt, jedoch ist hierfür immer eine eigene Firebase Instanz notwendig. Bei dem Projekt wurde basismäßig getestet, dass die Projektmetadaten in dem GIT selbst gespeichert werden kann.

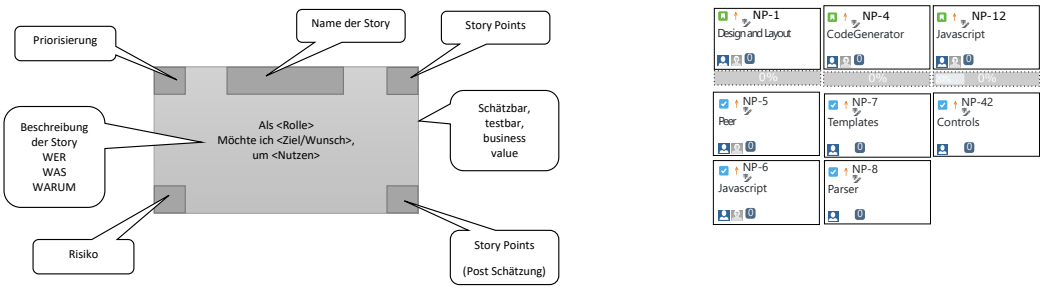

Abbildung 52: StoryCards and User-Story-Map

7.4.2 Mockupdesigner

Es ist weiterhin denkbar ein Webdesigner zu integrieren, welche Mockups erstellen kann. Hierbei können dann interaktive Mockups erstellt werden, welche dynamische Abschnitte und feste Inhalte haben [Gö14]. Dieses ermöglicht ein dynamisches Weiterentwickeln und ein Minimieren des Aufwandes. Das Ergebnis könnte als SVG/PNG Bilder exportiert werden oder in ein dynamisches Format abgelegt werden, um daraus direkt eine Oberfläche generieren zu lassen. Die Bilder können dann in Markdown-Texte eingebunden werden, welche eine Story oder Szenario abbilden.

7.4.3 Testing

Es wurden zwar einige Hilfen für den Entwickler implementiert. Allerdings fehlt noch die Entwicklung von automatischen JUNIT-Tests, welche direkt bei der Sourcecode Generierung erzeugt werden. Mittels dieser Tests kann auf spezielle Anforderungen von Datenmodellen eingegangen werden.

7.4.4 HTML-Darstellung von Oberfläche und Diagrammen

Es wurde ein Framework realisiert, welches mittels Dagre-Algorithmus eine Liste von Knoten und Kanten anordnen und mittels HTML5 oder SVG zeichnen kann. Weiterhin wurden in das Framework zahlreiche Erweiterungen für das Exportieren integriert.

Es bleibt noch zu überprüfen, ob noch bessere
Layout-Algorithmen existieren, wie zum Beispiel der Klay-Algorithmus [Klo12].
Weiterhin fehlen bei dem Klasseneditor noch einige Features. So ist das Bearbeiten von Attributen und Methoden ohne Assistenten noch sehr schwierig. Auch Kantenbeschreibungen können noch nicht richtig bearbeitet werden. Für die Kardinalität von Kanten gibt es noch keine Eingabemöglichkeit. In der Darstellung könnte der Lokalstorage noch integriert werden, um ein verändertes Layout persistent abzulegen. Die Layout-Algorithmen in der diagram.js sind noch zu überarbeiten. Weiterhin sollte überprüft werden, ob alle Knotentypen und deren Kombination korrekt funktionieren.

7.4.5 Code Editor

Was fehlt wäre ein rudimentärer Code Editor. Dieser könnte in Form einer Javascript-Bibliothek wie CodeMirror hinzugefügt werden [Hav18]. Dadurch könnte der CodeEditor mittels JavaBridge hinzugefügt werden.

In der Bachelorarbeit von Sebastian Copei "AyGe Editor – Applikations Generierung nach dem MVC-Pattern" [Cop15] wurde ein Generator vorgestellt, welcher es erlaubt mittels eigener DSL eine Applikation samt Oberfläche zu entwerfen. Eine ähnliche Vorgehensweise wäre auch für den NetworkParser denkbar. Weiterhin wäre ein Storydiagrammeditor ähnlich wie in [ZS+11] denkbar. Allerdings sollte dieser dann als Webeditor umgesetzt werden.

7.4.6 Erreichbarkeitsgraphen

Gerade für die höheren Probleme aus höheren Semestern bietet sich ein Tool für Berechnung von Erreichbarkeitsgraphen an. Dieses kann mittels IdMap und clone-Feature realisiert werden.

7.5 Source und Latex

In der heutigen Zeit ist es nicht mehr notwendig, die Ausarbeitung oder das Software Projekt auf einen Datenträger zu archivieren. Die Dissertation [Lin18c] und das Framework [Lin18e] an sich liegen bei GitHub. Weiterhin liegt die Ausarbeitung auch bei Overleaf [Lin18g] und das Framework bei GitLab [Lin18f]. Das Framework wurde als Release und Snapshot bei Maven veröffentlicht [Lin18d].

A Anhang

Im Anhang findet sich das Stichwortverzeichnis, das Abbildungsverzeichnis und das Literaturverzeichnis.

Archivmedien

Auf dem beigelegten Medium ist die Dissertation als PDF-Datei vorhanden. Weiterhin findet sich dort auch die Git-Repositories vom NetworkParser und DiagramJS und das aktuelle Framework als Jar Datei.

Cucumber

Einige textuelle User-Stories vom Ludo im Cucumberformat.

```
Scenario: Move a piece from the starting circle.
Definition: homeField and startingArea and lastField and
    target are a Field.
Definition: All fields have token.
Definition: a field has a next and prev field.
Definition: a lastField has a target field.

Given: Alice and Bob are playing Ludo.
       Alice has three tokens in the startingArea.
Given: one token on field_5.
       Bob has three tokens in the start area and one token
       on field_3.
Given: Alice is currentplayer of Ludo. Alice has thrown the
       dice and it shows a 6.
When:  Alice move her second token on field_1.
Then:  Alice has now two tokens in the start area one token
       on field_1 and the second token on field_5.
Then:  Bob has three token in a start area and one token on
       field 3.
Then:  Its Alice's turn again because she has thrown a 6.
```

Quellcode 66: Ludo Starting Circle

```
Scenario: Another turn.
Given: Alice and Bob are playing Ludo. Alice has three
       tokens in the home column and one token on field_20.
Given: Bob has has 3 tokens in the home column and one
   token
       on field_5.
Given: It's Alice's turn. Alice has thrown the dice and it
       shows a 6.
When:  Alice moves her token on field_20 to field_26.
Then:  Alice has now three tokens in the home column and
       one token on field_26.
Then:  Bob has three tokens in the home column and one
       token on field_5. It`s Alices turn again because
       she has thrown a 6.
```

Quellcode 67: Ludo Another turn

```
Scenario: Winning game
Given: Alice and Bob are playing Ludo. Alice has three
       tokens in the home column and one token on field_38.
Given: Bob has three tokens in the home column and one
       token on field_10.
Given: It's Alice's turn. Alice has thrown the dice with 3.
When:  Alice moves her token on field_38 in to the home
       column.
Then:  Alice has now four tokens in the home column. Alice
       has won the game.
```

Quellcode 68: Ludo Winning gamer

```
Scenario: throws a Token
Given: The players have three tokens on the field.
Given: Alice tokens are on the fields 3, 9 and 14. Bobs
       tokens are on the fields 7, 12 and 30.
Given: It's Alice turn. Alice has thrown a 3 with the dice.
When: Alice throws Bob's token on field 12.
Then: Alice has placed her token on field 12.
Then: Her other tokens are on the fields 3 and 14. Bobs
      token from field 12 is back in the start area.
```

Quellcode 69: Ludo throws a Token

B Stichwortverzeichnis

C Abbildungsverzeichnis

D Literatur

[Ack18] Johannes Ackermann. Einsatz von Vorgehensmodellen in Deutschland, 2018.

Online 11.08.18:

https://johannes-ackermann.de/2016/02/einsatz-von-vorgehensmodellen-in-deutschland/.

[ADH+09] Nina Aschenbrenner, Jorn Dreyer, Marcel Hahn, Ruben Jubeh, Christian Schneider, and Albert Zundorf. Building distributed web applications based on model versioning with CoObRa: An experience

report. In *Comparison and Versioning of Software Models, 2009. CVSM'09. ICSE Workshop on*, pages 19–24. IEEE, 2009.

[Afc18] Applications and Foundations (STAF) federated conferences. transformation tool contest (ttc), 2018.

Online 11.08.18: http://www.transformation-tool-contest.eu/.

[AG17] David Aurelio and yeebase GmbH. Plattformübergreifende Apps mit Web-Technologien entwickeln: Eine für alle, 2017.

Online 14.12.17:

http://t3n.de/magazin/plattformubergreifende-apps-web-technologien-entwickeln-226225/3.

[Alm17] Andres Almiray. Json-Lib, 2017.

Online 14.10.17: https://github.com/aalmiray/Json-lib .

[AV18] Andreas Stavropoulos Andy Vitus. continuous integration with github, 2018.

Online 11.08.18: https://circleci.com/integrations/github/.

[BD09] Christian Borowski and Ira Diethelm. Kinder auf dem Wege zur Informatik: Programmieren in der Grundschule. *Informatische Bildung in Theorie und Praxis. Beiträge zur INFOS*, 13, 2009.

[BG00] Kent Beck and Erich Gamma. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[Bid] Defines a map that allows bidirectional lookup between key and values.

10.01.19: https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/

collections4/BidiMap.html.

[Blu18] Blurb. Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software., 2018.

Online 11.08.18: https://jenkins.io.

[Bos15] Mike Bostock. D3.js - Data-Driven Documents, 2015.

Online 01.09.2015: http://d3js.org/.

[Bra18] Sergej Brataschow. UML Editor - Implementierung eines offline und plattformunabhängigem Klassendiagramm-Editor mit TypeScript, HTML5, CSS Und SVG., 2018.

[Bro02] David Brownell. *SAX2*. O'Reilly, 2002.

[Cam18] Alberto Camacho. what is rest? | codecademy, 2018.
Online 15.08.18: https://www.codecademy.com/articles/what-is-rest.

[Cau18] Andrew Caudwell. gource - a software version control visualization tool, 2018.
Online 11.08.18: http://gource.io/.

[Col18] Brian Cole, 2018. Online 23.03.18: https://spotbugs.github.io/.

[Cop15] Sebastian Copei. AyGe Editor – Applikations Generierung nach dem MVC-Pattern, 2015.

[Cor18] Oliver Corff. SGML-Einführung: Einführung: Was ist sgml, 2018.
Online 27.08.18: http://userpage.fu-berlin.de/corff/SGML/SGML-Einfuehrung-1.html.

[cpe15] cpettitt. cpettitt/dagre, 2015.
Online 01.09.2015: https://github.com/cpettitt/dagre.

[Cro06] Douglas Crockford. JavaScript Object Notation. *json.org*, 12 2006.
Online 07.10.18: http://www.json.org/.

[Cro10] Douglas Crockford. Jslint: The javascript code quality tool. *jslint.com*, 95, 2010.

[CS14] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[CSFP06] Ben Collins-Sussman, Brian W Fitzpatrick, and C Michael Pilato. *Versionskontrolle mit Subversion*. O'Reilly Germany, 2006.

[Cur18] Edward Curry. java message service, 2018.
Online 15.08.18: https://en.wikipedia.org/wiki/Java_Message_Service.

[CW18] Scott Chacon und Tom Preston-Werner Chris Wanstrath, PJ Hyett. build software better, together, 2018.
Online 11.08.18: https://github.com/.

[Dü18] Sven Bunge / Carl Düvel. Builddreikampf: Ant, Maven und Gradle, 2018.

23.11.18: http://alt.java-forum-stuttgart.de/jfs/2012/folien/F5.pdf.

[DAT17] DATACOM. Nexus, 2017.

Online 16.07.17: http://www.itwissen.info/Nexus.html,.

[DeR17] John DeRegnaucourt. jdereg/json-io, 2017.

Online 16.12.17: https://github.com/jdereg/json-io.

[Dew07] Ryan Dewsbury. *Google web toolkit applications*. Pearson Education, 2007.

[Die07] Stephan Diehl. *Software visualization*. Springer, 2007.

[Dil18] Martin Dilger. agile - effective trainings & consulting, 2018.

Online 05.10.18: http://www.effectivetrainings.de/blog/category/agile/.

[DJKZ07] Ira Diethelm, Ruben Jubeh, Andreas Koch, and Albert Zündorf. Whitesocks-a simple GUI framework for Fujaba. *Volume Editors*, page 30, 2007.

[Doc15a] Docs.oracle.com. ArrayList (Java Platform SE 8), 2015.

Online 03.08.2015: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.

[Doc15b] Docs.oracle.com. HashSet (Java Platform SE 8), 2015.

Online 03.08.2015: https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html.

[Doc15c] Docs.oracle.com. LinkedHashSet (Java Platform SE 8), 2015.

Online 03.08.2015: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html.

[Doc15d] Docs.oracle.com. LinkedList (Java Platform SE 8), 2015.

Online 03.08.2015: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html.

[dou10] douglascrockford. A reference implementation of a JSON package in Java., 2010.

Online 30.01.19: https://github.com/stleary/JSON-java.

[Dre15] Jörn Friedrich Dreyer. *Instant storyboarding*. Dreyer, Jörn Friedrich, 2015.

[dro18] dropbox.com. DropBox, 2018. Online 16.01.18: https://www.dropbox.com/.

[DW06] Stijn Dekeyser and Richard Watson. Extending google docs to collaborate on research papers. *University of Southern Queensland, Australia*, 23:2008, 2006.

[Ecl19] Eclipse MAT, 2019.
Online 02.02.19: https://www.eclipse.org/mat/.

[ECM11] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification.* ECMA International, 5.1 edition, June 2011.

[Ed.17] Ed.merks.gmail.com. Modeling for Programmers and Programming for Modelers, 2017.
Online 22.12.17: https://wiki.eclipse.org/Xcore.

[Edl02] Stefan Edlich. *Ant: kurz & gut.* O'Reilly Germany, 2002.

[EGK+02] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer, 2002.

[Elt18] Eltern.de. Vorname.com, 2018.
Online 04.12.18: https://www.vorname.com/name,Ludo.html.

[ES18] Sven Efftinge and Miro Spoenemann. Xtext - Language Engineering Made Easy!, 2018.
Online 03.10.18: https://www.eclipse.org/Xtext/.

[fan17] fangyidong. fangyidong/json-simple, 2017.
Online 03.10.17: https://github.com/fangyidong/json-simple.

[Fel15] Melanie Feldmann. JSweet: Aus Java einfach JavaScript machen - JAXenter. *JAXenter*, 2015.
Online 03.10.17: https://jaxenter.de/jsweet-aus-java-einfach-javascript-machen-32572.

[FGM+99] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–HTTP/1.1. Technical report, Hypertext transfer protocol, 1999.

[Fla06] David Flanagan. *JavaScript: the definitive guide*. Ö'Reilly Media, Inc.", 2006.

[FNTZ98] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *TAGT*, volume 1764 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 1998.

[Fou] Free Software Foundation. GNU General Public License. Online 01.06.16: http://www.gnu.org/licenses/gpl-2.0.html.

[Fou15] Eclipse Foundation. Eclipse Modeling Framework (EMF), 2015. Online 02.03.2015: https://www.eclipse.org/modeling/emf/.

[Fou18] Node.js Foundation, 2018. Online 11.08.18: https://nodejs.org/.

[Fra16] Nicolas Fraenkel. Performance cost of reflection, 2016. Online 30.01.19: https://blog.frankel.ch/performance-cost-of-reflection/.

[Fre18a] Ulli Freiberger. Java Karol - mebis, 2018. Online 11.01.18: https://www.mebis.bayern.de/infoportal/faecher/mint/inf/java-karol/.

[Fre18b] Ulli Freiberger. Robot Karol, 2018. Online 11.01.18: https://de.wikipedia.org/wiki/Robot_Karol.

[Fri16] Jeff Friesen. *Java XML and JSON*. Apress, 2016.

[Fri17] Uwe Friedrichsen. resilient software design - robuste software entwickeln, 2017. Online 05.11.17: https://www.informatik-aktuell.de/entwicklung/methoden/resilient-software-design-robuste-software-entwickeln.html.

[Gö14] Stephan Göbel. *Muse-Editor zum Erstellen von GUI Mockup Szenarien mit Prototyp Generierung*. PhD thesis, Uni Kassel, 2014.

[Gei04] Leif Geiger. *Automatische JUnit Testgenerierung aus UML-Szenarien mit Fujaba*. PhD thesis, Diploma Thesis. University of Braunschweig, 2004.

[Git13] GitHub. ahwolf/svgToPdf.js, 2013.
Online 01.09.2015: https://github.com/ahwolf/svgToPdf.js.

[Git15] GitHub. google Guava, 2015. Online 08.08.2015: https://github.com/google/guava.

[GKNV93] Emden R Gansner, Eleftherios Koutsofios, Stephen C North, and Gem-Phong Vo. A technique for drawing directed graphs. *Software Engineering, IEEE Transactions on*, 19(3):214–230, 1993.

[Glu16] Gluon. Scene Builder, February 2016.
Online 02.02.16: http://gluonhq.com/products/scene-builder/.

[Gmb17a] URGE IO GmbH. angularjs vs react, 2017.

[Gmb17b] Yatta Solutions GmbH. uml lab von yatta solutions 2017, 2017.
Online 22.12.17: https://www.uml-lab.com/de/uml-lab/.

[Gmb18] Mountainminds GmbH. ECLEmma - jacoco java code coverage library, 2018.
Online 11.08.18: https://www.jacoco.org/jacoco/.

[Goo12] Google. Google Cloud Messaging for Android, 2012.
Online 03.09.2014: http://developer.android.com/google/gcm/index.html.

[Gos00] James Gosling. *The Java language specification*. Addison-Wesley Professional, 2000.

[Gro18] MIT Media Lab Lifelong Kindergarten Group. scratch (programming language), 2018.
Online 10.11.18: https://en.wikipedia.org/wiki/Scratch_%28programming_language%29.

[GZ06a] Leif Geiger and Albert Zündorf. eDOBS - Graphical Debugging for Eclipse. *ECEASST*, 1, 2006.

[GZ06b] Leif Geiger and Albert Zündorf. eDOBS - Graphical Debugging for Eclipse. In *3rd International Workshop on Graph-Based Tools (GraBaTs) ICGT Workshop*, Natal, Brasil, September 2006.

[GZ06c] Leif Geiger and Albert Zündorf. eDOBS - Graphical Debugging for Eclipse. *ECEASST*, 1, 2006.

[Hé18] Philippe Le Hégaret. w3c document object model, 2018.
Online 15.11.18: https://www.w3.org/DOM/.

[Ham18] Graham Hamilton. the java community process(sm) program - jsrs: java specification requests, 2018.
Online 04.09.18: https://jcp.org/en/jsr/detail?id=295.

[Han13] Robert Hanmer. *Patterns for Fault Tolerant Software*. John Wiley and Sons, 2013.

[Han16] Tam Hanna. CI-Server im Vergleich: Jenkins vs. CruiseControl vs. Travis, 2016.
Online 07.04.16:
https://jaxenter.de/ci-server-im-vergleich-jenkins-vs-cruisecontrol-vs-travis-38081.

[Hav18] Marijn Haverbeke. Codemirror, 2018.
Online 13.10.18: http://codemirror.net/.

[HBS+02] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. Java message service. *Sun Microsystems Inc., Santa Clara, CA*, page 9, 2002.

[Hee14] Florian Heerdegen. MODIFy - Identifizierung von Modelländerungen durch generischen Modellvergleich. Master's thesis, University Kassel, 01 2014.

[Hei18] Steffen Heinzl. Verteilte Objekte durch RMI, 2018.
Online 15.08.18: http://www.informatik.uni-marburg.de/~mathes/download/k6.pdf.

[Hel18] Aslak Hellesøy. Cucumber, 2018.
Online 11.10.18: https://cucumber.io/.

[HFBO05] Kien Kim Huynh, Jane Fung, F v Breugel, and Bill O'Farrell. *Analysis through reflection: walking the EMF model of BPEL4WS*. York University, 2005.

[Hof17] Moritz Hoffmann. gitlab vs. github: wer bietet mehr?, 2017.
Online 01.12.17: https://jaxenter.de/gitlab-vs-github-wer-bietet-mehr-23062.

[Hog11] Brian P Hogan. *HTML 5 & CSS 3*. O'Reilly Germany, 2011.

[Inc17] Google Inc. Android Studio - The Official IDE for Android, 2017.

[Inc18a] Google Inc., 2018.
Online 11.08.18: https://firebase.google.com/products/.

[Inc18b] Google Inc. google drive - cloud storage and file backup for photos, docs and more, 2018. Online 16.01.18: https://www.google.com/drive/.

[Inc18c] Google Inc. researchgate.net: Google Cloud to Device Message, 2018.
Online 07.10.18: https://www.researchgate.net/publication/281858140_Framework_fur_verteilte_mobile_Anwendungen_mittels_XMPP_und_'Google_Cloud_to_Device_Message'-Dienst.

[ind17a] inder123. Google GSON, 2017.
Online 01.12.17: https://github.com/google/gson/.

[ind17b] inder123. GraphAdapterBuilder, 2017.
Online 14.06.16:
https://github.com/google/gson/blob/1d9e86e27c97cd85d898104b4ac42bb487d0d7d0/extras/src/main/java/com/google/gson/graph/GraphAdapterBuilder.java.

[Int11] IDEA IntelliJ. the most intelligent Java IDE. *JetBrains[online].[cit. 2016-02-23]. Dostupné z: https://www. jetbrains. com/idea/# chooseYourEdition*, 2011.

[Int16] Ecma International. Ecma-262, jun 2016.

[Iva18] Roman Ivanov. checkstyle, 2018.
Online 23.03.18: http://checkstyle.sourceforge.net/.

[IZSn18] Inc. Isaac Z. Schlueter / npm. node.js npm, 2018.
Online 11.08.18: https://www.w3schools.com/nodejs/nodejs_npm.asp.

[Jan18] Brock Janiczak. eclemma - java code coverage for eclipse, 2018.
Online 24.08.18: https://www.eclemma.org/.

[jav12] javafx4you. Communicating between JavaScript and JavaFX with WebEngine, February 2012.
Online 01.02.2012:
https://blogs.oracle.com/javafx/entry/communicating_between_javascript_and_javafx.

[Jav15] Javamex.com. Java development/profiling tool: Classmexer agent, 2015.

Online 07.08.2015: http://www.javamex.com/classmexer/.

[jav17] Enno Boland - java2js, 2017.

[Joe17] Joe Rossignol. BlackBerry Hits '0Launched, 2017.

Online 15.02.17: https://www.macrumors.com/2017/02/15/blackberry-hits-zero-market-share/.

[Joi15] Jointjs.com. JointJS - JavaScript diagramming library, 2015.

Online 01.09.2015: http://jointjs.com/demos/umlcd.

[JPr19] JProfiler, 2019.

Online 02.02.19: https://www.ej-technologies.com/products/jprofiler/overview.html.

[Jub17] Ruben Jubeh. *Story Driven Modeling als agile Vorgehensmethode für das Internet der Dinge in Lehre und Praxis.* PhD thesis, Uni Kassel, 2017.

[Kat18] Yehuda Katz. handlebars.js: minimal templating on steroids, 2018.

Online 26.08.18: https://handlebarsjs.com/.

[KE04] Alfons Kemper and Andre' Eickler. *Datenbanksysteme.* Oldenbourg, 2004.

[kla18] klauskuenen. user stories vs. use cases - klauskuenen.de | innovation, ecommerce, strategie, 2018.

Online 04.10.18: https://www.klauskuenen.de/user_stories_use_cases/.

[Klo12] Paul Klose. *A generic framework for the topology-shapemetrics based layout.* PhD thesis, Master's thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, 2012.

[Kmh14] Kmheide. User Story, 2014.

Online 06.11.18: https://de.wikipedia.org/wiki/User_Story.

[Kor17] Eugeniya Korotya. 5 best javascript frameworks in 2017, 2017.

Online 22.12.17: https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282.

[Kri18] Anders Kristensen. JSR 116: SIP Servlet API, 2018.

Online 15.08.18: https://www.jcp.org/en/jsr/detail?id=116.

[KZ15] Andreas Koch and Albert Zündorf. Graphical Debugging of Distributed Applications - Using UML Object Diagrams to Visualize the State of Distributed Applications at Runtime. In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*, pages 223–230, 2015.

[LBB+16] Alessandro Lacava, Janek Bogucki, Aliaksandr Bedrytski, Matthew de Detrich, and Benjamin Neil. Functional Programming. *Professional Scala*, pages 19–36, 2016.

[LG10] Christy Pettey Laurence Goasduff. Gartner News, 2010.
Online 18.05.2010: http:/www.gartner.com/it/page.jsp?id=1371425.

[LG11] Christy Pettey Laurence Goasduff. Gartner News, 2011.
Online 08.02.2011: http:/www.gartner.com/it/page.jsp?id=1923316.

[LG12] Christy Pettey Laurence Goasduff. Gartner News, 2012.
Online 15.02.2012: http://www.gartner.com/it/page.jsp?id=1924314.

[LG18a] Christy Pettey Laurence Goasduff. Gartner News, 2018.
Online 03.10.18: https://www.gartner.com/en/newsroom/press-releases/2018-08-28-gartner-says-huawei-secured-no-2-worldwide-smartphone-vendor-spot-surpassing-apple-in-second-quarter.

[LG18b] Christy Pettey Laurence Goasduff. Gartner News, 2018.
Online 03.10.18:
https://www.malaysianwireless.com/2017/12/gartner-top-smartphone-vendors-q317/.

[Lin12] Stefan Lindel. Framework für verteilte mobile Anwendungen mittels XMPP und 'Google Cloud to Device Message'-Dienst. Master's thesis, University Kassel, 05 2012.

[Lin15] Stefan Lindel. SimpleScrum, 2015.

[Lin17a] Stefan Lindel. PM WS 13/14, 2017.
Online 23.12.17:
https://docs.google.com/open?id=1GDeNVjakjseRam64fPqscck-Gc-tULuRAhvUZ-2TsZM.

[Lin17b] Stefan Lindel. PM WS 14/15, 2017.
Online 23.12.17:
https://docs.google.com/open?id=1TupmbJKlvPSTNPrS3Q96bn2uO-CZbf_bDv3DwqHzzqo.

[Lin17c] Stefan Lindel. PM WS 16/17, 2017.
Online 23.12.17:
https://docs.google.com/open?id=154kNSUHfQXHMNhYUnkBx1Ha_i1ExBFbrbii78YRUenQ.

[Lin18a] Stefan Lindel. DiagramJS SourceCode, 2018.
10. März 2019: https://github.com/StefanLindel/DiagramJS.

[Lin18b] Stefan Lindel. DiagramJS Webseite, 2018.
10. März 2019: https://stefanlindel.github.io/DiagramJS/.

[Lin18c] Stefan Lindel. GIT Source Dissertation Latex Text, 2018.
Online 10. März 2019:
https://github.com/StefanLindel/PeTaF-Framework---Peer-TaskFlow-Framework-.

[Lin18d] Stefan Lindel. NetworkParser Maven, 2018.
Online 10. März 2019: https://search.maven.org/artifact/de.uniks/NetworkParser/.

[Lin18e] Stefan Lindel. NetworkParser SourceCode, 2018.
Online 10. März 2019: https://github.com/fujaba/NetworkParser.

[Lin18f] Stefan Lindel. NetworkParser SourceCode, 2018.
10. März 2019: https://gitlab.com/StefanLindel/NetworkParser/.

[Lin18g] Stefan Lindel. Overleaf Latex Dissertation, 2018.
Online 10. März 2019: https://de.overleaf.com/5673883922fnczzjsjbnyk.

[Lin18h] Stefan Lindel. PM WS 18/19, 2018.
Online 23.11.18:
https://docs.google.com/forms/d/1Rsa20kMz2g9nZ2YCF8QbkPDpYjN4lPgZbTjsWYHnvOM.

[Lin19a] Stefan Lindel. DiagramJS, 2019.
Online 13.02.19: https://www.npmjs.com/package/diagramjs.

[Lin19b] Stefan Lindel. Swaggerhub, 2019.
Online 04.03.19: https://app.swaggerhub.com/apis/Fulib/NetworkParser/1.0.0.

[Lon17] King's College London. A free Java Development Environment designed for beginners, 2017.
Online 27.12.17: https://bluej.org/.

[Ltd18] Enterprise Architects Pty Ltd. homepage - enterprise architects, 2018.

Online 24.08.18: http://enterprisearchitects.com/.

[LVc18] 2016 vogella GmbH Lars Vogel (c) 2007. Eclipse Modeling Framework (EMF) - Persisting models via XMI - Tutorial, 2018.

Online 18.11.18: http://www.vogella.com/articles/EclipseEMFPersistence/article.html.

[LZ12] Stefan Lindel and Albert Zündorf. Online synchronization of replicated models. *Softwaretechnik-Trends*, 32(4):40–42, 2012.

[Mae12] Kazuaki Maeda. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pages 177–182. IEEE, 2012.

[Mar14] Azat Mardan. *Practical Node.js*. Apress, 2014.

[Mar15] Marvl.infotech.monash.edu. cola.js: Constraint-based Layout in the Browser, 2015.

Online 01.09.2015: http://marvl.infotech.monash.edu/webcola/.

[Mau15a] Thomas Mauch. Magicwerk.org: BigList, 2015.

Online 03.08.2015: http://www.magicwerk.org/res/collections/javadoc/org/magicwerk/brownies/collections/BigList.html.

[Mau15b] "Thomas Mauch". Magicwerk.org: GapList, 2015.

Online 03.08.2015: http://www.magicwerk.org/res/collections/javadoc/org/magicwerk/brownies/collections/GapList.html.

[MKF06] Gail C Murphy, Mik Kersten, and Leah Findlater. How are Java software developers using the Elipse IDE? *IEEE software*, 23(4):76–83, 2006.

[MM18] Bertil Muth and Bertil Muth. Use Cases und User Stories - Verbündete oder Feinde? - HOOD Blog, 2018.

Online 11.01.18: https://www.mebis.bayern.de/infoportal/faecher/mint/inf/java-karol/.

[Moh18] Dominik Mohilo. 8 build-tools im vergleich: ant,buildr,maven,bazel,buck,gradle,pants,sbt - jaxenter, 2018.
Online 23.11.18: https://jaxenter.de/8-build-tools-im-vergleich-ant-buildr-maven-bazel-buck-gradle-pants-sbt-41627.

[mon18] MontiCore is a language workbench for an efficient development of domain-specific languages. This is a mirror only., 2018.
Online 22.12.18: https://github.com/MontiCore/monticore.

[Mos17a] Brian Moschel. Longevity (or Lack Thereof) in JavaScript Frameworks, 2017.
Online 01.09.17: https://www.bitovi.com/blog/longevity-or-lack-thereof-in-javascript-frameworks.

[Mos17b] Mosea3. Android Development Tools, 2017.
Online 01.09.17: https://de.wikipedia.org/wiki/Android_Development_Tools.

[MR98] Thomas J Mowbray and William A Ruh. *Inside CORBA*. Addison-Wesley, 1998.

[Mus14] Benjamin Muschko. *Gradle in action*. Manning Publications Co., 2014.

[Nie18] Oscar Nierstrasz. MSE and FAMIX 3.0: an Interexchange Format and Source Code Model Family, 2018.
Online 26.10.18: http://www.academia.edu/2665064/MSE_and_FAMIX_3.0_an_Interexchange_Format_and_Source_Code_Model_Family, https://rmod.inria.fr/archives/reports/Duca11c-Cutter-deliverable22-MSE-FAMIX30.pdf, .

[Nit18] Niteowlneils. Distributed object, 2018.
Online 15.08.18: https://en.wikipedia.org/wiki/Distributed_object.

[NJZ13] Ulrich Norbisrath, Ruben Jubeh, and Albert Zündorf. *Story Driven Modeling*. CreateSpace Independent Publishing Platform, 2013.

[NNWZ00] Ulrich A Nickel, Jörg Niere, Jörg P Wadsack, and Albert Zündorf. Roundtrip engineering with FUJABA. In *Proc of 2nd Workshop on Software-Reengineering (WSR), Bad Honnef, Germany*, 2000.

[NNZ00a] Ulrich Nickel, Jörg Niere, and Albert Zündorf. The FUJABA environment. In *Proceedings of the 22nd international conference on Software engineering*, pages 742–745. ACM, 2000.

[NNZ00b] Ulrich Nickel, Jörg Niere, and Albert Zündorf. The Fujaba Environment. In *ICSE 2000*, pages 742–745, Limmerick, Ireland, June 2000. acm press.

[Nor18] Patrik Nordwall. akka documentation, 2018.
Online 08.10.18: https://doc.akka.io/docs/akka/current/guide/tutorial_2.html.

[npm18] Inc. npm, 2018. Online 11.08.18: https://www.npmjs.com/.

[OCL19] about the object constraint language specification, 2019.

[Ope15a] Openjdk.java.net. OpenJDK: jmh, 2015.
Online 08.08.2015: http://openjdk.java.net/projects/code-tools/jmh/.

[Ope15b] OpenKieler. OpenKieler/klayjs-d3, 2015.
Online 01.09.2015: https://github.com/OpenKieler/klayjs-d3.

[Ora18a] Oracle.com. java servlet technology, 2018.
Online 15.08.18: http://www.oracle.com/technetwork/java/index-jsp-135475.html.

[Ora18b] Oracle.com. javaserver faces technology, 2018.
Online 19.08.18: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.

[Ora18c] Oracle.com. oracle, 2018.
Online 03.09.18: https://www.oracle.com/de/index.HTML.

[Pan18] Dhiru Pandey. JSR 267: JSP Tag Library for Web Services, 2018.
Online 15.08.18: https://www.jcp.org/en/jsr/detail?id=267.

[Par18] Terence Parr. ANother Tool for Language Recognition, 2018.
Online 24.08.18: http://www.antlr.org/.

[Paw17] Renaud Pawlak. JSweet: how does it compare to TypeScript? *JSweet*, 2017.
Article 22.12.17: http://www.jsweet.org/wp-content/uploads/2015/11/
JSweet-How-does-it-compare-to-TypeScript-Renaud-Pawlak.pdf.

[PB18] Ed. P. Bryan. rfc 6902 - javascript object notation (json-patch), 2018.
Online 14.08.18: https://tools.ietf.org/html/rfc6902.

[Pip18] Andy Piper. MQTT is a machine-to-machine (M2M), 2018.
Online 15.08.18: http://mqtt.org/.

[PL18] Bill Pugh and Andrey Loskutov. findbugs - find bugs in java programs, 2018.
Online 23.03.18: http://findbugs.sourceforge.net/.

[Poi18] Tutorials Point. jsp tutorial, 2018.
Online 15.08.18: https://www.tutorialspoint.com/jsp/.

[qiCT17] By quadrim in Cloud Technologies. apache kafka vs rabbitmq message queue comparison. *Cloud Hack*, 2017.
Article 29.02.16: http://www.cloudhack.in/2016/02/29/apache-kafka-vs-rabbitmq/.

[Rö18] Thomas Röll. röll media event-, medien- und streamtechnik, 2018.
Online 11.08.18: https://roellmedia.de/de/roellmedia-gmbh-veranstaltungstechnik/.

[Res08] Eric Rescorla. *SSL and TLS*. Addison-Wesley, 2008.

[Res18] Mitch Resnick. scratch - imagine, program, share, 2018.
Online 10.11.18: https://scratch.mit.edu/.

[rfc19] rfc5246 - the transport layer security (tls) protocol, 2019.

[Rie18] Yannik Ries. Programmierumgebung fuer den Informatikunterricht an Gymnasien, 2018.
Online 18.02.19: http://ad-publications.informatik.uni-freiburg.de/theses/Zulassungsarbeit_Yannick_Ries_2018.pdf.

[rit18] ritchie@gmx.at. jenesis java code generator, 2018.
Online 24.08.18: http://jenesis4java.sourceforge.net/.

[Rot17] Alexander Roth. *Adaptable code generation of consistent and customizable data-centric applications with MontiDEx*. Dissertation, RWTH Aachen University, Aachen, 2017. Dissertation, RWTH Aachen University, 2017.

[Rum16] Bernhard Rumpe. *Modellierung Mit UML*. Springer International Publishing AG, 2016.

[RvdM18] Thomas McCall Rob van der Meulen. Gartner News, 2018. Online 22.02.18: gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017 .

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[Sch07] Christian Schneider. *CoObRA: Eine Plattform zur Verteilung und Replikation komplexer Objektstrukturen mit optimistischen Sperrkonzepten*. PhD thesis, Uni Kassel, 2007.

[Sch14] Enno Schwanke. Generierung von UML Klassendiagrammen aus Java Code in Eclipse. *Bachelor thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science*, 2014.

[Sch17a] Hartmut Schlosser. react versus angular - wer gewinnt, 2017.

[Sch17b] Hartmut Schlosser. technologie trends 2017, 2017. Online 22.02.17: https://jaxenter.de/framework-trends-2017-53153.

[Sch17c] Hartmut Schlosser. Wo steht JavaFX im Vergleich zu anderen UI-Toolkits: Swing, HTML5, SWT?, 2017. Online 01.03.16: https://jaxenter.de/wo-steht-javafx-im-vergleich-zu-anderen-ui-toolkits-swing-html5-swt-35821.

[Sch18] Hartmut Schlosser. maven vs. ant vs. gradle: zwischenbilanz - jaxenter, 2018. Online 23.11.18: https://jaxenter.de/maven-vs-ant-vs-gradle-zwischenbilanz-2-7700.

[SDB11] Bruce Snyder, Rob Davies, and Dejan Bosnanac. *ActiveMQ in Action*. Manning Publications, 2011.

[SE04] Holger Schwichtenberg and Frank Eller. *Programmierung mit der. NET-Klassenbibliothek: Zugriff auf das Windows-Betriebssystem mit Visual Basic. NET und C.* Pearson Deutschland GmbH, 2004.

[See17] Heiko Seeberger. Einführung in die Aktor-Programmierung mit Akka, 2017.

Online 22.12.17: https://www.heise.de/developer/artikel/Einfuehrung-in-die-Aktor-Programmierung-mit-Akka-1921580.html.

[Shi02] Iain Shigeoka. *Instant messaging in Java: the Jabber protocols.* Manning, 2002.

[Sie18] Marvin Siegert. user stories in scrum – widas, 2018.

Online 05.10.18: https://www.widas.de/user-stories-in-scrum-3/.

[Skr18] Neil Skrypuch. Eclipse Modeling - M2T - Home | The Eclipse Foundation, 2018.

Online 03.10.18: https://www.eclipse.org/modeling/m2t/?project=xpand.

[sna19] asomov / snakeyaml, 2019.

Online 29.01.19: https://bitbucket.org/asomov/snakeyaml/.

[Son18a] SonarSource. SonarQube NetworkParser, 2018.

Online 13.12.18: https://sonarcloud.io/dashboard?id=NetworkParser.

[Son18b] SonarSource. SonarQube Webseite, 2018.

Online 13.12.18: https://www.sonarqube.org/.

[son19] Sonatype.org, 2019.

Online 13.02.19: https://oss.sonatype.org/content/repositories/snapshots/de/uniks/NetworkParser/.

[Soy15] Soyatec.com. Soyatec - Open Solution Company: XAML for Java, UML for Eclipse and BPMN designer, 2015.

Online 01.09.2015: http://www.soyatec.com/euml2/.

[Str17] King's College London Strand. Greenfoot - Teach and Learn Java Programming, 2017.

Online 27.12.17: https://www.greenfoot.org/.

[TH10] Linus Torvalds and Junio Hamano. Git: Fast version control system. *URL http://git-scm.com*, 2010.

Online 16.01.18: http://git-scm.com.

[TH14] Louis Rose Tassilo Horn, Christian Krause. Transformation Tool Contest (TTC) 21014 in York, 2014.

Online 04.03.2015: http://www.transformation-tool-contest.eu/2014.

[Tho15] Michael Thomas. Ruhe in Frieden, JavaFX?, November 2015.

Online 01.09.17: https://jaxenter.de/ruhe-in-frieden-javafx-31013.

[TKc18] Johannes Ewald Tobias Koppers, Sean Larkin and Webpack contributors. webpack, 2018.

Online 03.10.18: https://webpack.js.org/.

[Tra19] Travis-Ci.org, 2019.

Online 31.01.19: https://travis-ci.org/.

[Tru17] Trustable. Liste von GUI-Bibliotheken, 2017.

Online 01.12.17: https://de.wikipedia.org/wiki/Liste_von_GUI-Bibliotheken.

[Tuc18] Matt Tucker. Smack, 2018.

Online 20.02.18: https://www.igniterealtime.org/projects/smack/.

[TV10] Stefan Tilkov and Steve Vinoski. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80, 2010.

[TW14] Andrew S. Tanenbaum and David Wetherall. *Computernetzwerke*. Pearson, 2014.

[Tys18] Matthew Tyson. What is the JVM? Introducing the Java virtual machine, 2018.

Online 03.09.18: https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html.

[uMS18] Felix Dausch und Marcus Schmidt. Entwicklung einer Desktopanwendung mit Webtechnologien zur Zusammenführung von agiler Softwareentwicklung nach Scrum und der Versionsverwaltung mit Git, 2018.

[vB18] Danny van Bruggen. Java 9 Parser and Abstract Syntax Tree for Java, 2018.

Online 11.08.18: http://javaparser.org/.

[Vel18] Apache Velocity. the apache velocity project, 2018.

Online 03.10.18: http://velocity.apache.org/.

[Vog18] Lars Vogel, 2018.

Online 18.11.18: http://www.vogella.com/articles/EclipseEMFNotification/article.html.

[Wei18] Alexander Weidt. Entwicklung eines IOT-Frameworks zur Anlagenüberwachung, 2018.

[Wen18] Eric Wendelin. Gradle vs Maven Comparison, 2018.

Online 23.11.18: https://gradle.org/maven-vs-gradle/.

[Wit12] Ludwig Wittgenstein. *Big Typescript: TS 213.* John Wiley & Sons, 2012.

[Wit15] Ingo Witzky. *Weborientierter interaktiver Fragebogen- und Logikdesigner für die Erstellung offline bezogener Assistenten zur Ermittlung individueller Entscheidungshilfen.* PhD thesis, Universität Kassel, 2015.

[WLR11] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: a controlled experiment. In *ICSE '11 Proceedings of the 33rd International Conference on Software Engineering*, 2011.

[WWR+18] Oliver White, Oliver White, Simon Relations, Sigmar engineer, Juri ZeroTurnaround, and Michael Rasmussen. The Ultimate Java Build Tool Comparison: Gradle, Maven, Ant + Ivy | zeroturnaround.com, 2018.

Online 23.11.18: https://zeroturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy/.

[yam19] yaml ain't markup language (yaml), 2019.

Online 29.01.19: https://yaml.org/spec/1.2/spec.html.

[You18] Youtube.com. YouTube Gource, 2018.

Online 05.12.18: https://youtu.be/HXYg0CcovhY.

[Yu18] Xin Yu. Jet Model Robotization, 2018.

Online 24.08.18: http://www.jmr-source.com/doc/en/index.html.

[Yum15] Yuml.me. Create UML diagrams online in seconds, no special tools needed., 2015.

Online 01.09.2015: http://yuml.me/.

[yWo15] the diagramming company yWorks. yFiles for Java - Java Graph Layout and Visualization Library, 2015.

Online 01.09.2015: http://www.yworks.com/en/products_yfiles_about.html.

[Zü15] Albert Zündorf. Story Driven Modeling Library, 2015.

Online 22.12.17: https://github.com/fujaba/SDMLib.

[Zü18] Albert Zündorf. Fujaba library, 2018.

27.11.18: https://github.com/fujaba/fulib.

[Zan16] C. Zangl. the human json (hjson) configuration format, 2016.

Online 13.10.18: https://hjson.org/rfc.html.

[ZE16] Albert Zündorf and Christoph Eickhoff. The SDMLib solution to the Class Responsibility Assignment Case for TTC2016. 2016.

Online 08.07.16:

http://www.transformation-tool-contest.eu/2016/solutions/cra/TTC_2016_paper_7.pdf.

[zen18] Zenju: zenju@freefilesync.org. FreeFileSync Synchronize Files and Folders, 2018.

Online 16.01.18: https://www.freefilesync.org/.

[ZGK15] Albert Zündorf, Tobias George, and Bodo Kraft. Software Stories Guide. *pohl2010*, 2015.

Online 31.07.17: https://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:

34-2017073153163/3/SoftwareStoriesNotationWhitePaper.pdf.

[ZL15] Albert Zündorf and Stefan Lindel. The SDMLib solution to the Model Execution Case for TTC2015. *TTC 2015*, 2015.

Paper 24.07.2015: ceur-ws.org/Vol-1524/paper18.pdf.

[ZLGE14a] Albert Zündorf, Stefan Lindel, Tobias George, and Christoph Eickhoff. The SDMLib Solution to the FIXML Case for TTC 2014. *TTC 2014*, 2014.

Paper 25.07.2014:

http://www.transformation-tool-contest.eu/2014/solutions/fixml/ttc2014_submission_5.pdf.

[ZLGE14b] Albert Zündorf, Stefan Lindel, Tobias George, and Christoph Eickhoff. The SDMLib solution to the MovieDB case for TTC 2014. *TTC 2014*, 2014.

Paper 25.07.2014:

http://www.transformation-tool-contest.eu/2014/solutions/movie/ttc2014_submission_6.pdf.

[Zor12] Zorbedit. ACID, 2012. Online 22.12.17: https://de.wikipedia.org/wiki/ACID.

[ZS+11] Albert Zündorf, Wilhelm Schäfer, et al. *Fehlersuche im Modell-Modellbasiertes Testen und Debuggen.* PhD thesis, 2011.

[ZW17] Albert Zündorf and Alexander Weidt. The SDMLib Solution to the TTC 2017 Families 2 Persons Case. *TTC 2017*, 2017.

Online 21.07.17: http:

//www.transformation-tool-contest.eu/2017/solutions/familiesToPersons/TTC_2017_paper_9.pdf.