Bachelorarbeit

Erweiterung eines E-Learning-Moduls zum Assessment von B-Baum-Datenstrukturen

David Bujok

Themensteller: Prof. Dr. Herbert Kuchen
Betreuer: Dipl.-Wirt.Inform. Claus Alexander Usener
Institut für Wirtschaftsinformatik
Praktische Informatik in der Wirtschaft

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **2**

2.1 Die Lernplattform Moodle 2

2.1.1 Was ist Moodle 2

2.1.2 Moodle als Lernmanagement-System 2

2.1.3 Kommunikationsmethoden in Moodle 4

2.1.4 Modularten 4

2.1.5 Aufbau eines Moduls 5

2.2 E-Assessment 7

2.2.1 Einleitung 7

2.2.2 E-Assessment als Teilbereich des Lernprozesses 8

2.2.3 E-Assessment als Methode des Leistungsnachweises 9

2.3 Die Datenstruktur B-Baum 10

3 Vorstellung des Moodlemoduls EASy-DSBuilder **12**

3.1 Funktionalität aus Benutzersicht 12

3.2 Umsetzung aus technischer Sicht 14

3.2.1 Datenstruktur-Verarbeitungsservice 14

3.2.2 Moodlemodul backendseitig 16

3.2.3 Moodlemodul frontendseitig 18

4 Konzeptionelle Anforderungen an ein E-Learning-Modul zum Assessment von B-Baum-Datenstrukturen **20**

4.1 Neuanforderungen an den EASy-DSBuilder 20

4.2 Spezifizierung der konzeptionellen Anforderungen des Grapheneditors 23

5 Umsetzung der Änderungsanforderungen **26**

5.1 Überblick über Änderungen 26

5.2 Datenstruktur für die Moodlemodul-Webserice-Kommunikation 27

5.3 Datenstrukturverarbeitung backendseitig 28

5.4 Grapheneditor 31

5.4.1 Funktionalität 32

5.4.2 Implementierungsdetails 35

5.4.3 Aufbau des B-Baums im Hintergrund 36

6 Fazit und Ausblick **37**

1 Einleitung

Die steigende Anzahl von Studierenden bei gleichbleibender Zahl der Lehrpersonen und Assistierenden stellt Universitäten vor neue Herausforderungen. Insbesondere die Logistik der steigenden Anzahl von Modulabschlüssen bei knappen Raum- und Personalressourcen wird immer mehr zum Problem. Hierfür bietet E-Assessment mit der Möglichkeit der Verbesserung und Erhöhung der Effizienz im Prüfungsalltag einen möglichen Ausweg. Unter E-Assessment versteht man das Nutzen von Hardware und Software im Prüfungsprozess. So wurden im deutschsprachigem Raum für E-Assessment Synonyme wie Online-Prüfungen oder Computergestützte Prüfungen verwendet [Rue10, S. 11 ff.]. Auf dieser Basis entwickelte der Lehrstuhl Praktische Informatik in der Wirtschaft der WWU Münster das E-Assessment-Tool EASy-DSBuilder zum Assessment von AVL-Baum-Datenstrukturen für die Moodledistribution Learnweb [Use14]. Bei Moodle handelt es sich um ein weltweit anerkanntes Lernmanagement-System [Ger07, S. 33], das Lehrenden, Administratoren und Lernenden eine robuste, sichere und integrierte Plattform bereitstellen soll [mooa]. Weiterhin ist Moodle eine frei verfügbare Open Source Software. Außerdem bietet Sie die Möglichkeit der individuellen Anpassung an spezifische Anwendungssituationen [moog].

Im Zentrum dieser Arbeit steht das Moodlemodul EASy-DSBulder. Insbesondere stellt diese Arbeit die Erweiterung dieses Moduls um die Funktionalität des Assessments von B-Baum-Datenstrukturen vor. Hierzu wird die Arbeit durch ein Grundlagenkapitel (Kapitel 2) eingeleitet, in dem zuerst die wesentlichen Merkmale der Lernplattform Moodle vorgestellt werden (Kapitel 2.1). Der anschließende Teil der Grundlagen ist eine Einleitung in die Grundlagen des E-Assessements (2.2). Das Grundlagenkapitel wird durch die Definition der B-Baum-Datenstruktur abgeschlossen(Kapitel 2.3). Das darauf folgende Kapitel 3 stellt das Moodlemodul EASy-DSBuilder vor. Hierbei wird insbesondere auf die Funktionalität aus Benutzersicht und auf die Struktur aus technischer Sicht eingegangen. Im Kapitel 4 werden die Anforderungen an ein Tool zum Assessment von B-Baum-Datenstrukturen vorgestellt. Insbesondere werden im Kontext des EASy-DSBuilder Änderungsanforderung an dieses Modul zur Umsetzung des Assessments von B-Baum-Datenstrukturen dargelegt. Anschießend wird die Umsetzung der Anforderungen betrachtet. Hierbei werden die entwickelten Funktionalitäten beleuchtet und es wird ein Einblick auf ausgewählte Implementierungsdetails gegeben. Abschießend wir ein Fazit des Ergebnisses dieser Arbeit betrachtet und ein Ausblick auf mögliche weitere Entwicklungen des Moduls gegeben.

2 Grundlagen

Im folgenden Kapitel werden für dies Arbeit wichtige Grundlagen vorgestellt. Insbesondere handelt es sich um Einführungen in die Lernplattform Moodle, das E-Assessment und die B-Baum-Datenstruktur.

2.1 Die Lernplattform Moodle

Der folgende Abschnitt erläutert, um welche Art System es sich bei der Lernplattform Moodle handelt und stellt weitere wichtige Eigenschaften dieser Plattform dar.

2.1.1 Was ist Moodle

Bei Moodle handelt es sich um ein weltweit anerkanntes Lernmanagement-System [Ger07, S. 33], das Lehrenden, Administratoren und Lernenden eine robuste, sichere und integrierte Plattform bereitstellen soll [mooa]. Der Name Moodle leitet sich von der Akronymisierung des Ausdrucks ***M****odular* ***O****bject* ***O****riented* ***D****ynamic* ***L****earning* ***E****nvironment* ab [Ger07, S. 33]. Moodle ist weiterhin eine frei verfügbares Softwarepaket, da es der GNU Public Lizenz unterliegt [SH09]. Software, welche unter einer GNU Public License vertrieben wird, darf kopiert, benutzt und weiterentwickelt werden. Eine einschränkende Bedingung ist, dass Änderungen oder Weiterentwicklungen den eben genannten Pflichten unterliegen, sie folglich auch veröffentlicht und Dritten zur Verfügung gestellt werden müssen [moog]. Die Plattform wird von einer weltweiten Gemeinschaft und von der Moodle Pty. Ltd. laufend weiterentwickelt. Vom australischen Moodle Erfinder Marign Dougiamas wird das Projekt zielgerichtet geleitet. Des weiteren gibt es ein Netzwerk professioneller Partnerunternehmen, welche Support und Beratung leisten [SH09, S. 12].

2.1.2 Moodle als Lernmanagement-System

Unter einem Lernmanagement-System (LMS) versteht man im wesentlichen ein Management-System für die Automatisierung und die Administration von Ausbildung. Insbesondere sollten LMS über folgende Funktionen verfügen [Sch05, S. 14]:

- Eine Benutzerverwaltung (Anmeldung mit Verschlüsselung)
- Eine Kursverwaltung (Kurse, Verwaltung der Inhalte, Dateiverwaltung)
- Eine Rollen- und Rechtevergabe mit differenzierten Rechten

- Kommunikationsmethoden (Chat, Foren) und Werkzeuge für das Lernen (Whiteboard, Notizbuch, Annotationen, Kalender etc.)

Moodle stellt diese Funktionen zur Verfügung. So besteht über die Website-Administration die Möglichkeit der Benutzerverwaltung [Ger07, S. 563 2 ff.] und der Kursverwaltung [Ger07, S. 588 ff.]. Bei der Rollen - und Rechtevergabe bietet Moodle flexible Möglichkeiten der Administration. So verfügt Moodle über vorgefertigte Basisrollen mit bestimmten Rechten, die einen Großteil der Anwendungsfälle abdecken. Für bestimmte Situationen können Rollen jedoch editiert oder neue Rollen erstellt werden [Ger07, S. 191]. Die Basisrollen des Systems sind [Ger07, S. 193]:

- *Kursverwalter:* Wer in einem Kontext *Kursverwalter* ist, kann einen *neuen Kurs erstellen* und in diesem unterrichten, weil er automatisch als *Trainer* eingetragen wird. Zu anderen Kursen im gleichen Kontext hat er aber keinen Zugriff.

- *Trainer*: Wer in einem Kontext *Trainer* ist, ist in sämtlichen Kursen dieses Kontextes als *Trainer* eingetragen und kann diese Bearbeiten

- *Trainer ohne Editorrecht*: ist Trainer in sämtlichen Kursen dieses Kontextes.

- *Teilnehmer/in*: ist Teilnehmer in sämtlichen Kursen dieses Kontextes, kann also auch Kurse mit Zugriffsschlüssel betreten.

Auf die Kommunikationsmethoden, die Moodle zur Verfügung stellt, wird in Kapitel 2.1.3 eingegangen.

Abbildung 1 zeigt die idealtypische Architektur eines LMS. Zu sehen ist, dass ein LMS über drei Schichten verfügt. Bei der untersten Schicht handelt es sich um die Datenbankschicht, in der alle Lernobjekte, Benutzerdaten und andere gehalten werden. Die mittlere Schicht stellt Schnittstellen zur Verfügung. Die oberste Schicht stellt die Sicht bereit, über die über die seitens von Administratoren, Dozenten oder Studierenden auf Inhalte zugegriffen werden kann [Sch05, S. 11]. Im Kontext dieser Arbeit wird insbesondere verstärkt auf die Schnittstellenschicht eingegangen. Das Kapitel **??** wird die Möglichkeit Einbindung von Modulen in Moodle erläutern. Das Kapitel **??** wird hingegen den Teilbereichen Aufgeben und Tests aus dem Bereich Authoring der Ansichtsschicht auseinandersetzen. *Es wird der Forschungsbereich E-Assessment vorgestellt, welcher sich mit Überprüfungen über Onlinemedien auseinandersetzt.*

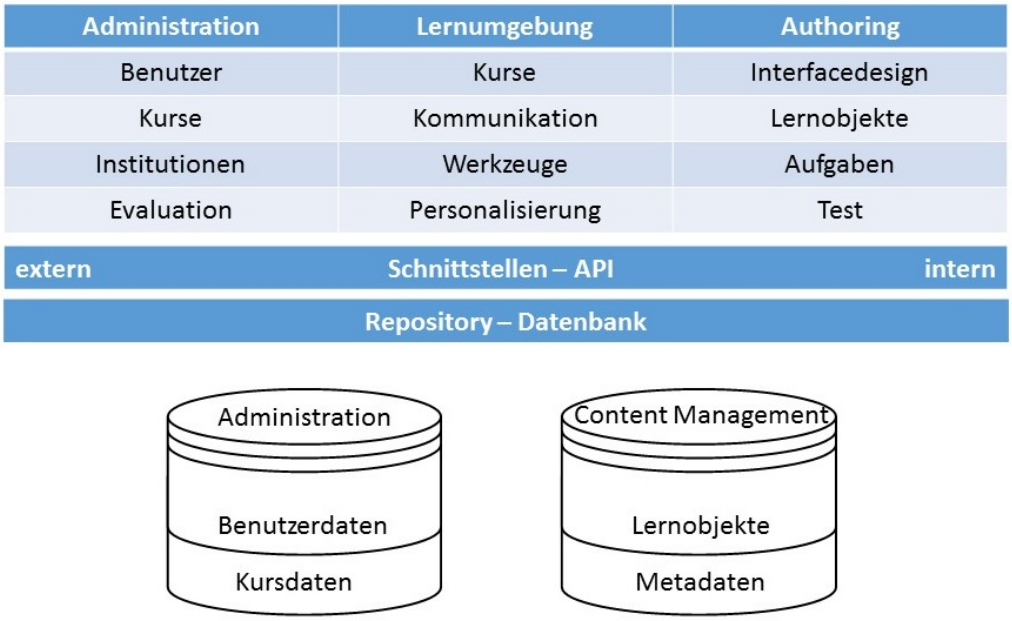

Abbildung 1: Idealtypische Architektur eines LMS [Sch05, S. 12]

2.1.3 Kommunikationsmethoden in Moodle

2.1.4 Modularten

Neben den eingangs beschriebenen Activity modules gibt es noch weitere Pluginarten in Moodle, von denen eine Auswahl nun näher beschrieben wird.

Activity Plugin

Activity Plugins bieten in erster Linie Kursaktivitäten wie Foren, Hausarbeiten, Quizze und Workshops [moob].

Blöcke

Blöcke sind Plugins, welche eine einfache Art Interface besitzen und auf jeder Moodle-Seite hinzugefügt werden können. Zur Platzierung von Blöcken werden zu- meist zwei Spalten auf einer solchen Moodle-Seite angeboten, jeweils eine links und eine rechts vom eigentlichen Inhalt der Seite. Diese Pluginart dient dabei häufig der Darstellung von zusätzlichen Inhalten oder bietet eine weitere Sicht auf schon vorhandene Daten, welche andere Rückschlüsse zulässt. Mithilfe von Blöcken ist es zudem möglich, ein Interface zur Bearbeitung von Daten anzubieten [Mooj].

Themes

Sogenannte *themes* bilden eine weitere Pluginart, mithilfe derer sich die Darstellung und das Aussehen von Moodle auf verschiedenen Ebenen anpassen lässt. Bei diesen

Ebenen, auf denen ein *theme* angewendet wird, kann es sich sowohl um einzelne Moodle- oder Kursseiten als auch um alle Kursseiten einer Kurskategorie handeln. Durch diese Art von Plugin schafft Moodle die Trennung der Präsentationsschicht von der funktionalen Schicht [Mooj].

Course formats
Mithilfe von *course format* Plugins lässt sich die Struktur von Kursen anpassen. Diese Plugins legen fest, wie die Hauptseite eines Kurses aufgebaut ist, wie der Navigationsbaum innerhalb eines Kurses erstellt wird und wie die einzelnen Kursbereiche strukturiert sind [Mooh].

General plugins
Eine weitere wichtige Pluginart, sind die sogenanten *general plugins*, oder auch *local Plugins*. Diese Art von Plugin kommt immer dann zum Einsatz, wenn keine der sonstigen Pluginarten passt. Dies kann unter anderem dann der Fall sein, wenn der Navigationsblock um ein eigenes Menü erweitert werden soll. Auch für Webservices und die Einbindung externer Systeme sind *local-Plugins* die erste Wahl [Mooi].

2.1.5 Aufbau eines Moduls

Für jedes Plugin in Moodle muss eine bestimmte Datenstruktur implementiert werden. Diese besteht aus separaten Unterverzeichnissen und verpflichtenden Dateien. Des weiteren haben Entwickler die Möglichkeit weitere Dateien selbst zu gestalten [mooc].

`/<modname>/backup`
Dieser Ordner dient zur Ablage aller Dateien, welche definieren, wie sich das Modul bei einem Backup oder einer Wiederherstellung verhalten soll [mooc].

`/<modname>/db`

- **`/access.php`** In dieser Datei werden die so genannten *capabilities* für das Plugin definiert. *capabilities* beschreiben die Berechtigungen, welche eine Rolle in diesem Plugin zugeordnet bekommt. Eine Berechtigung ist beispielsweise das hinzufügen einer neuen Instanz diese Plugins zu einem Kurs [mooc].
- **`/install.xml`** Diese Datei wird bei der Installation des Moduls benutzt. Sie definiert, welche Datenbanktabellen und -felder erstellt werden. Hierfür wird

das XML-Format verwendet. Braucht das Modul keine weiteren Tabellen oder Spalten, so kann auf diese Datei verzichtet werden [mooc].

- **upgrade.php** Auf Grund dessen, dass die Datei **install.xml** nur einmal während der Installation aufgeführt wird, braucht es eine Methode um die Datenbank nachträglich um Tabellen oder Spalten zu erweitern. Diese Funktionalität wird von dieser Datei bereitgestellt und kommt bei einem Update des Moduls zum Einsatz [mooc].

/<modname>/lang

In diesem Ordner können alle *Strings* gespeichert werden, die im Modul benutzt werden sollen. Jede Sprache hat hierbei einen spezifischen Ordnernamen ('**/lang/de**' beispielsweise für die Sprache Deutsch). Die in diesem Ordner gespeicherte Datei muss in der Form **<modname>.php** benannt sein [mooc].

/<modname>/pix

Dieser Ordner dient dazu das Logo des Moduls zu speichern, welches neben dem Modulname erscheint. Der Name des Logos muss **icon.gif** lauten. Weiterhin besteht die Möglichkeit weitere Bilder in diesem Ordner zu speichern [mooc].

/<modname>

- **/lib.php** Diese Datei bietet eine Schnittstelle für die zu implementierenden Kernfunktionen. Kernfunktionen werden dazu benötigt, damit das Modul in Moodle integriert arbeiten kann. Diese Schnittstellen-Funktionen werden von Moodle nach einem bestimmten Ereignis im Prozessablauf aufgerufen, sofern diese vom Modul in der Datei **/lib.php** definiert wurden. Dabei ist jeder dieser Funktionen zunächst der Name des Moduls vorangestellt, gefolgt von einem Unterstrich und dem Funktionsnamen (**<pluginname>_core_function**). Diese Konvention ist deshalb so wichtig, da die Datei **/lib.php** keine Klasse definiert, welche Namenskonflikte verhindern. Es wird geraten Funktionalitäten, welche einen hohen Codeumfang haben, der Übersichtlichkeit halber in eine Datei namens **locallib.php** auszulagern. würde. [mooe]

- **/mod_form.php** Diese Datei wird beim Hinzufügen oder Bearbeiten eines Kurses genutzt. Es enthält die Elemente welche im Editiermenü des Moduls zu sehen sind. Die in dieser Datei enthaltende Klasse muss der Namenskonvention

nach in der Form **mod_<modname>_mod_form** benannt sein.

- **/index.php** Diese Datei wird von Moodle dazu genutzt, um auf Aktivitäten bei allen Instanzen dieses Moduls, welche einem bestimmten Kurs übergeben wurden, zu hören. Diese Datei ist spezifisch für diese Modulart *Activity Module*.

- **/view.php** Diese Datei wird bei der Erzeugung der Anzeige benötigt. Beim Aufrufen eines Moduls über die Kurssicht wird auf diese Datei verwiesen. Dabei wird dieser Datei die Instanz-ID übergeben, anhand welcher die Daten der Instanz ausgewählt und angezeigt werden können. Diese Datei ist spezifisch für diese Modulart *Activity Module*.

- **/version.php** Diese Datei enthält die aktuelle Versionsnummer dieses Moduls. Außerdem enthält diese Datei weitere Attribute wie beispielsweise die Mindestanforderungen hinsichtlich der Moodleplattform.

2.2 E-Assessment

2.2.1 Einleitung

Auf Grund des Bologna-Prozesses, Sparmaßnahmen, Diskussionen über die Verwendung von Studiengebühren und steigende Studierendenzahlen wird im Bereich der Übungsangebote der Einsatz von E-Assessments vermehrt in Betracht gezogen [KP10, S. 9]. Unter e-Assessment versteht man das Spektrum der auf den neuen (elektronischen) IKT basierenden Verfahren der lehrzielbezogenen Bestimmung, Beurteilung, Bewertung, Dokumentation und Rückmeldung der jeweiligen Lernvoraussetzungen, des aktuellen Lernstandes oder der erreichten Lernergebnisse/ -leistungen vor, während (?Assessment für das Lernen?) oder nach Abschluss (?Assessment des Lernens?) einer spezifischen Lehr-Lernperiode [Blo06, S. 6]. E-Assessment-Systeme können des Weiteren Nutzen für Lehrende und Lernende bieten. Cook und Jenkins identifizierten neuen Vorteile. Bei den Wichtigsten handelt es sich um die Möglichkeiten des direkten Feedbacks und der sofortigen und objektiven Benotung. Außerdem bieten E-Assessment-Systeme einfache Skalierbarkeit und Wiederverwendbarkeit [CJ10, S. 3]. E-Assessments können hinsichtlich ihrer Hauptaufgabe in drei Typen unterteilt werden [CJ10, S. 8]:

- **Diagnostische Assessments** finden normalerweise zu beginn einer Lehrveranstaltung statt um mögliche Wissenslücken bei Teilnehmern aufzudecken und gegebenenfalls ein Nachbesserungsangebot zu schaffen.

- **Formative Assessments** ermöglicht Studierenden und Lehrenden einen Überblick über den aktuellen Lernstand zu erhalten. E-Assessment ermöglicht Studierenden weiterhin ein direktes Feedback.
- **Summative Assessments** bieten eine Bewertungsgrundlage über den Lernfortschritt eines Studierenden. Bei diesem Typen steht im Gegensatz des Feedbacks die Notengebung im Vordergrund.

So wird in den folgenden Abschnitten auf formative Assessment eingegangen, indem E-Assessments als Teilbereich des Lernprozesses vorgestellt werden, und anschließend summative Assessments behandelt werden, indem tiefgehender auf den Aspekt des Leistungsnachweises im Zusammenhang mit E-Assessment eingegangen wird.

2.2.2 E-Assessment als Teilbereich des Lernprozesses

In der Hochschullehre nehmen Leistungsüberprüfungen einen integralen Bestandteil in Lehr- und Lernprozessen ein. Ein Augenmerk liegt hierbei auf der Identifizierung und Bewertung individueller Lernfortschritte [KG10, S. 23 f.]. Darunter wird insbesondere das Prüfen und Bewerten einzelner Übungen als Teil einer pädagogischen Handlungseinheit verstanden. Dabei können die verschiedenen Übungsphasen und -iterationen unterschiedliche Längen haben und inhaltliche Abhängigkeiten haben [KG10, S. 25]. Abbildung 2 zeigt den Lehr-Lern-Prozess eines Übungsbetriebs.

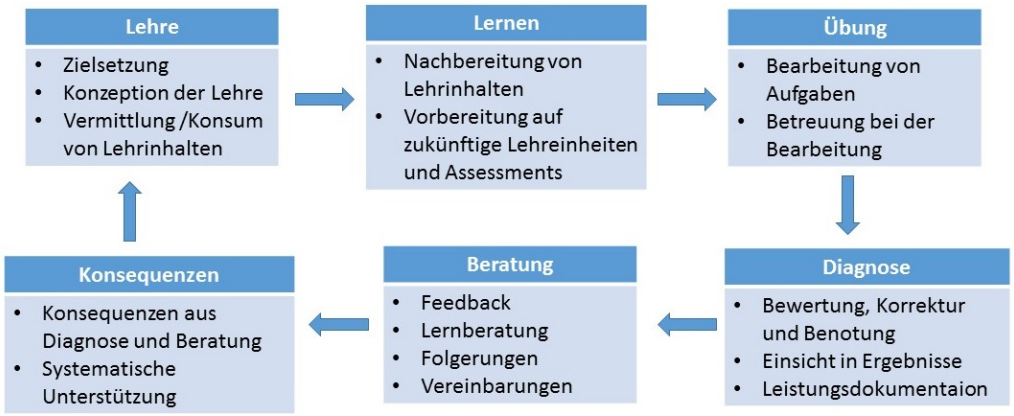

Abbildung 2: Iterative Lehr-Lern-Prozesse eines Übungsbetriebs

Im Regelfall beginnt ein Übungszyklus mit der Phase der Lehre. In dieser Phase vermittelt der Lehrende dem Studierenden die entsprechenden Inhalte. Anschließend hat der Studierende in der Lernphase die Möglichkeit das vermittelte Wissen im Selbststudium zu vertiefen. In einer darauf folgenden Übung kann der Studierende sein theoretisches Wissen anhand geeigneter praktischer Übungen ausprobieren

und festigen. Im Anschluss folgt die Diagnose der Übung, welche Korrektur und Benotung beinhaltet. Dies geschieht im Regelfall durch DozentInnen oder TutorInnen, kann aber auch durch KommilitonInnen erfolgen (Peer Review). Es sollte ein Feedback über die erbrachte Leistung und Ratschläge folgen. Optional kann aus den Ratschlägen eine Leistungsvereinbarung abgeleitet werden [KG10, S. 25 f.].

Weiterhin sind Leistungsüberprüfungen ein integraler Bestandteil der Lehr- und Lernprozesse. So sollen wöchentliche Übungszettel dazu beitragen, dass das von Studierenden erlernte theoretische Wissen durch Bearbeitung geeigneter Aufgaben reflektiert und verinnerlicht wird. Insbesondere gestaltet sich das Halten von klassischen Präsenzübungen bei knappen finanziellen Ressourcen schwer. Eine Möglichkeit trotz Reduktion des Aufwands eine langfristige hohe Qualität des Übungsbetriebs zu gewährleisten ist in E-Assessment zu sehen. [KG10, S. 24]. In den meisten Systemen, welche E-Assessment anbieten werden jedoch nur einfache Aufgabentypen wie Multiple-Choice-Aufgaben oder Lückentexte unterstützt. Über solche Aufgabentypen können Wissenszuwächse gut geprüft werden, kognitive Fähigkeiten und Methodenwissen jedoch nur bedingt. Insbesondere in naturwissenschaftlichen oder technischen Disziplinen wie der Informatik ist die Kontrolle und Überprüfung von Aufgaben, bei denen die Anwendung analytischer, kreativer und konstruktiver Fähigkeiten gefordert ist besonders wichtig. So entwickelte beispielsweise die WWU Münster das E-Assessment-System EASy für das Informatikstudium, welches durch die Bereitstellung von einfachen Aufgabentypen wie dem Multiple-Choice, aber auch anspruchsvoller Aufgabentypen zu mathematischen Beweisen und Programmierung[KG10, S. 24]. So zeigte die Evaluation dieses Systems im Praxiseinsatz, dass der Lernprozess unterstützt wurde und insbesondere der Übungsbetrieb von diesem System profitiert hat. Weiterhin stellte die Evaluation in Ausblick, dass solche Systeme auch auf eine Vielzahl von anderen Bildungseinrichtungen übertragen werden kann [KG10, S. 34].

2.2.3 E-Assessment als Methode des Leistungsnachweises

Neben Leistungsüberprüfungen als Bestandteil des Lehr- und Lernprozesses kann E-Assessment auch für einen Leistungsnachweis herangezogen werden. Als Leistungsnachweis wird die abschließende Leistungserbringung zum Bestehen einer Lehrveranstaltung oder eines Modul angesehen. So werden Leistungsnachweise klassisch durch Klausuren, Referate oder Studienarbeiten repräsentiert. Wir also E-Assessment als Leistungsnachweis eingesetzt, kann man davon sprechen, dass anstatt einer traditionellen Prüfungsform eine vergleichbare elektronische Durchführungsvariante gewählt

wurde [Rue10, S. 18]. Der folgende Abschnitt beschäftigt sich mit der Gegenüberstellung von traditionellen Prüfungsformen und elektronischen Äquivalenten.

Ein klassisches Beispiel für einen Leistungsnachweis ist die schriftliche Prüfung. Ihr elektronisches Äquivalent ist die E-Prüfung bzw. E-Klausur, welche offene und geschlossene Fragen verwendet um Lehrinhalte zu prüfen. Oft wird sie als computerunterstützende Prüfung mit Multiple-Choice-Fragen verstanden, jedoch sind auch offene Fragen sehr effektiv. So kann beispielsweise durch die verbesserte Lesbarkeit die Korrekturzeit verkürzt werden. Weiterhin kann unter Elektronic Submission der automatisierte elektronische Abgabeprozess von schriftlichen Arbeiten wie Protokollen, Seminar- und Hausarbeiten verstanden werden. Hierbei kann beispielsweise in Lernplattformen Ablagemöglichkeiten zur fristgerechten Abgabe bereitgestellt werden. Daneben können Simulationen am Computer komplexe wissenschaftspraktische Tätigkeiten übernehmen. Derweise können Simulationen die Bewertung typischer Labortätigkeiten wie beispielsweise die Handhabung von Mikroskopen oder die Interpretation von Röntgenbildern übernehmen. Außerdem könne beispielsweise Foren als Bewertungsmöglichkeit von mündlicher Mitarbeit dienen. Ebenso kann können Wikis eine Präsentationsmöglichkeit einer erbrachten Gruppenleistung dienen. So kann abschließen E-Assessment als Methode der Leistungserbringung als Substitutionsmodell verschiedener klassischer Beurteilungsmethoden verstanden werden [Rue10, S. 18 ff.].

2.3 Die Datenstruktur B-Baum

Der B-Baums ist eine von R. Bayer und E. McCreight entwickelte ausgeglichene Baumstruktur. Hierbei steht der Name B für balanciert, breit, buschig ober Bayer, nicht jedoch für binär. Im Gegensatz zu Binärbäumen ist die Grundidee eines B-Baums der variierende Verzweigungsgrad, während die Baumhöhe vollständig ausgeglichen ist. Dies bedeutet, dass alle Pfade von der Wurzel zu den Blättern gleich lang sind, Knoten jedoch mehrere Kanten enthalten können. Deswegen fallen B-Bäume auch in die Kategorie Mehrwegbäume [SS14, S. 386].

Ein Baum T ist ein *B-Baum der Ordnung* $k, k \in \mathbb{N}, k \geqslant 2$, falls

1. alle Blätter dieselbe Tiefe,
2. die Wurzel mindestens 2 und jeder andere innere Konten mindestens k Söhne und
3. jeder innere Knoten höchstens $2k - 1$ Söhne

haben [Blu13, S. 21]. Für einen sortierten Baum muss folgendes gelten: Sei u ein

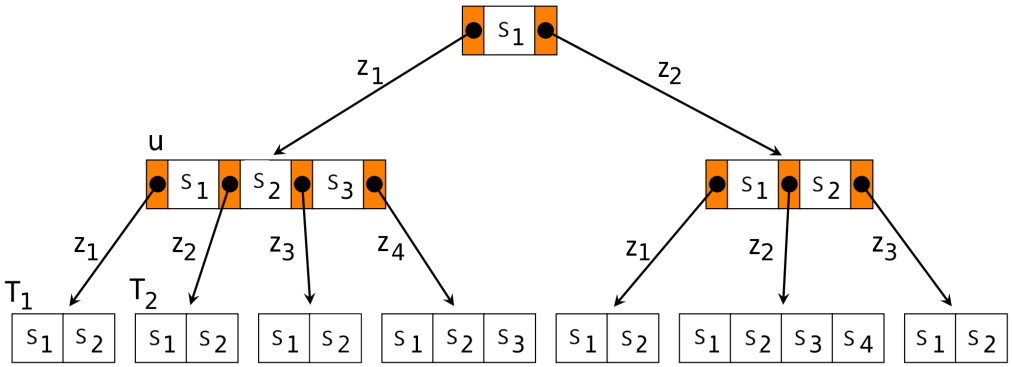

Abbildung 3: Struktur eines B-Baums

innerer Knoten mit l Söhnen. Bezeichne $T_i, 1 \leqslant i \leqslant l$ den i-ten Unterbaum des Teilbaumes mit Wurzel u. Der Knoten u enthält $l-1$ Schlüssel $s_1, s_2, ..., s_{l-1}$ und l Zeiger $z_1, z_2, ..., z_l$. Der Zeiger z_i zeigt auf den i-ten Sohn von u. Für alle Knoten v in T_i gilt für alle Schlüssel s in v [Blu13, S. 22]:

$$\begin{cases} s \leqslant s_i & \text{falls } i = 1 \\ s_{i-1} < s \leqslant s_i & \text{falls } 1 < i < l \\ s_{l-1} < s & \text{falls } i = l \end{cases}$$

3 Vorstellung des Moodlemoduls EASy-DSBuilder

Der EASy-DSBuilder ist ein E-Assessent Tool, welches der Evaluation grundlegender Konzepte über Operationen (z.B. Suchen, Einfügen, und Entfernen) innerhalb der Datenstruktur *Binärbaum* dient [Use14]. Das Tool wurde speziell für die Lernplattform Moodle implementiert.

Diese Kapitel wird das Tool EASy-DSBuilder vorstellen. Hierbei wird zu erst in Kapitel 3.1 auf die Funktionalität aus Benutzersicht eingegangen. Anschließend erfolgt eine Erläuterung der technischen Umsetzung (Kapitel 3.2).

3.1 Funktionalität aus Benutzersicht

Im folgenden Kapitel wird die Funktionalität des Moodlemodul EASy-DSBuilder vorgestellt. Hierbei wird auf die beiden Sichten Studierender und Lehrender eingegangen.

Lehrender

Der Lehrende hat zwei grundlegende Aufgaben. Zum Einen ist er dafür verantwortlich, dass eine Aufgabe erstellt wird, zum Anderen hat er die Möglichkeit, die Ergebnisse einzusehen, um beispielsweise Indikatoren zur Verbesserung der Lehre zu finden [Use14]. Wird eine neue Aufgabe erstellt, hat der Lehrende die Möglichkeit allgemeine Informationen wie den *Titel*, die *Beschreibung* und das *Fälligkeitsdatum* anzugeben. Unter *Source Files* kann der Lehrende über Drag-and-Drop seine eigene Implementierung einer Datenstruktur zu dem Moodlemodul hinzufügen. Hierzu muss er eine Wrapper auf Basis eines Interfaces *DataStructureWrapperInterface<K extends Comparable<K>, D>* implementieren, sodass die verlangten Voraussetzungen erfüllt sind. Diese Wrapperklasse muss anschließend vom Lehrenden als Hauptklasse eingestellt werden. Auf die Funktionalität der Wrapperklasse aus technischer Sicht wird im Kapitel 3.2.1 näher eingegangen. Des weiteren kann der Lehrende eine Feedback aktivieren. Die genau Funktionalität des Feedbacks wird im Absatz der Studentensicht erläutert.

Übersichtsansicht und Ansicht der Ergebnisse

Studierender

Der Studierende verfügt über zwei Ansichten. Zum einen die Übersichtsansicht, zum anderen die Bearbeitungsansicht. Nachdem der Studierende sich in das Modul eingewählt hat, ist Übersichtsansicht über den bisherigen Verlauf des Assessments zu

sehen. In dieser Übersicht ist der Abgabestatus, der Bewertungsstand, der Abgabezeitpunkt und die verbliebene Zeit zu sehen (vergl. Abb. **??**). Über den Button *Aufgabe bearbeiten* gelangt der Studierende zum Editor, in dem die Aufgabe bearbeitet werden kann.

Die Bearbeitungsansicht ist in drei grundlegende Abschnitte unterteilt. Den oberen Teil der Ansicht bildet ein Überblick über den aktuellen Schritt. Dieser Überblick beinhaltet den Fortschritt der Aufgabe, die Nummer des aktuellen Schritts und den aktuellen Arbeitsauftrag. Im mittleren Teil der Sicht befindet sich der Editor, in dem der Studierende die Aufgabe bearbeiten kann. Im oberen linken Bereich des Editor befinden sich drei Knöpfe (vergl. Abb. **??**.1), über welche der Editiermodus ausgewählt werden kann. Der 1. Knopf ermöglicht das verschieben von Konten im Editor, der zweite Knopf ermöglicht das Ziehen von Kanten zwischen zwei Knoten, und der dritte Knopf ermöglicht das Entfernen von Konten.

Der **DragandDropGrafikeditor** enthält zwei bearbeitbare Elemente, die Knoten und die Kanten. Über Manipulation dieser Elemente sollen Studierende den Umgang mit Datenstrukturoperationen erlernen. Hierbei kann der Studierende Operationen wie das Einfügen in oder das Löschen eines Elements aus einer Datenstruktur praktizieren. In der **momentanen** Version des EASyDSBuilders beginnt jeder Schritt mit dem Ergebnisbaum des zuvor eingereichten Schritts, oder einem Initierngsbaum, falls es sich um den ersten Schritt handelt. Auf der linken Seite des Editors wird der einzufügende Knoten bereitgestellt. Die Aufgabe des Studierenden ist es, diesen Knoten an der richtigen Stelle in den Baum einzufügen. Hierbei kann er sich dem Verschieben der Knoten wie dem Löschen oder neu Ziehen von Kanten bedienen. Nachdem der Studierende seine Veränderungen vorgenommen hat, kann er über den Knopf *Syntax prüfen* den Baum ausbalanciert anzeigen lassen. Auf diese Weise kann der Studierende überprüfen, ob die Anwendung den Baum im Sinne des Studierenden verarbeitet hat. Entspricht die überprüfte Struktur nicht der Struktur eines Baumes, **genauere Definiton** bekommt der Studierende eine Fehlermeldung mit einem Hinweis über die Fehlerquelle.

Hat der Lehrende bei der Einrichtung des DSBuilders die Option *direktes Feedback* eingestellt, erscheint im Falle einer falschen Eingabe ein Feedbackfeld unterhalb des Editors. In diesem Feedbackfeld wird zu erst ein Informationstext angezeigt, welches das richtige Vorgehen in dem zuvor eingereichtem Schritt beschreibt. Unterhalb dieses Informationstextes ist der korrekte Baum zu sehen. Die falsch eingeordneten Knoten sind rot markiert.

3.2 Umsetzung aus technischer Sicht

Das gesamte System um den EASy-DSBuilder besteht backendseitig aus zwei separaten Systemen. Zum einen gibt es das eigentliche Moodlemodul, welches in eine bestehenden Moodelplattform integriert werden kann, zum anderen gibt es einen Datenstruktur-Verarbeitungsservice, welcher als Webservice implementiert ist. Das Moodlemodul hat die Möglichkeit über die Moodle-API Daten in einer SQL-Datenbank - beispielsweise einem MySQL-Server - zu hinterlegen. Die Kommunikation zwischen dem Moodlemodul und dem Webservice läuft über das SOAP-Protokoll. Der Webservice ist als WildFly Application Server implementiert und unterliegt somit dem Java-EE7-Standard [Gre]. In Abbildung 4 ist dargestellt, wie die unterschiedlichen Technologien in einander greifen.

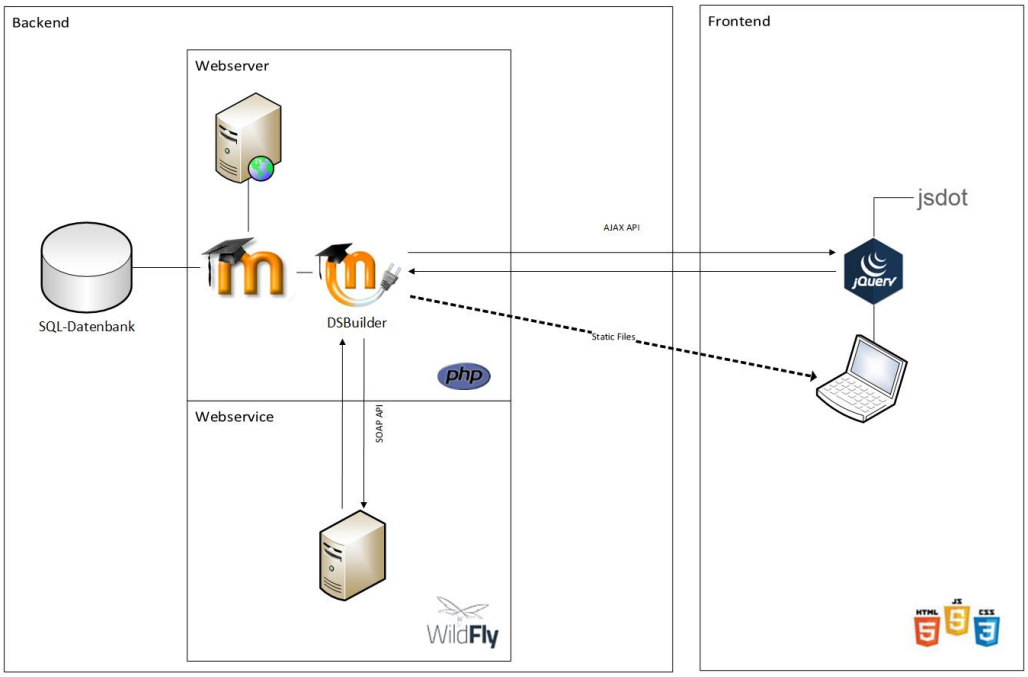

Abbildung 4: Technischer Überblick

Auf Clientseite wird HTML mit CSS und JavaScript verwendet, um das Modul für den Benutzer darstellen zu können. Als JavaScript-Frameworks wird jQuery und und als JavaSrcipt-Applikation wird jsdot eingesetzt. Über jQuery ist die Kommunikation mit dem Moodlemodul über das AJAX-Protokoll organisiert. Jsdot dient als Grapheditor.

3.2.1 Datenstruktur-Verarbeitungsservice

Der Datenstruktur-Verarbeitungsservice hat die Aufgabe Datenstrukturen in Form eines JSON Strings für das Moodlemodul bereitzustellen. Hierfür muss der Lehren-

de zuvor eine Wrapperdatei auf Basis eines vorgegebenen Interfaces (vergl. Anhang Codebeispiel 8) in Java implementieren und der Anwendung zur Verfügung stellen. Diese Wrapperklasse dient als Schnittstelle zwischen dem Webservice und der Implementierung der Datenstruktur. Die wichtigen Funktionen, die durch die Implementierung dieses Interfaces zur Verfügung gestellt werden sollten sind das Hinzufügen und Löschen von Schlüsseln, sowie das Erzeugen eines serialisierten JSON Strings aus der Datenstruktur und das Deserialisieren eines JSON Strings in die Datenstruktur.

Um funktionsfähig zu sein, muss der Datenstruktur-Verarbeitungsservice den vom Lehrenden bereitgestellten Code kompilieren und ausführen. Der Verarbeitungsservice ist als separater Webservice implementiert und wird somit von einem anderen Server aus bereitgestellt. Die Separierung des Systems erfolgt aus den Risiken, dass der Code schädlich sein oder eine schlechte Ausführungsleistung aufweisen kann. Durch die Trennung der beiden Systeme kann in beiden Fällen Zusammen- oder Performanceeinbrüchen der gesamten E-Learning-Plattform vorgebeugt werden. Weiterhin kann so Datendiebstahl verhindert werden, da in der Verarbeitungsumgebung keine nutzerbezogenen Daten verarbeitet werden. Bei Ausfall des Verarbeitungsservices ist jedoch das Aufrufen eines nächsten Schrittes nicht mehr möglich [Use14].

Der Webservice verfügt über die SOAP-Schnittstellen *analyzeSourceCode*, *assignmentInitialize* und *assignmentIsIntialized*, die zur Initialisierung einer neuen DSBuilder-Instanz beötigt werden, ebenso wie die SOAP-Schnittstellen *submissionGetNextInput* und *submissionGetInitialDS*, die zur Verarbeitung einer Datenstruktur benötigt werden. Um den Webservice nutzen zu können müssen ihm Codedateien zu Verfügung gestellt werde. Über den SOAP-Aufruf *analyzeSourceCode* wird die Lesbarkeit der Dateien geprüft. Anschließend kann über den SOAP-Aufruf *assignmentInitialize* der Code compiliert und bereitgestellt werden. Die Ausführung des Codes ist vor jedem Einfügen oder Löschen, das von einem Studierenden durchgeführt wird, notwendig. So kann über den SOAP-Aufruf *submissionGetNextInput* der der nächste Übungsschritt für den Studierenden erhalten werde. Bei der Initialisierung der Übung wird der SOAP-Aufruf *submissionGetInitialDS* benötigt. Die Antwort auf diese beiden Aufrufe beinhaltet den neuen Schlüssel und dessen Wert, die serialisierte Datenstruktur und zusätzliche Hilfe.

3.2.2 Moodlemodul backendseitig

Das backendseitige Moodlemodul besitzt die grundlegende Struktur eines Moodlemoduls, wie sie in Kapitel 2.1.5 dargestellt wurde. Die weiteren, für die Funktionalität des Moduls wichtigen Dateien sind die Dateien **`renderer.php`** und **`renderable.php`**, welchen DOM-Code generieren, die Dateien **`ajax_request.php`** und **`ajax_helper.php`**, welche das Handling von AJAX-Anfragen übernehmen und die Datei **`lib/js_dot_convert.php`**, welche die Funktionen zur Verarbeitung der Datenstrukturen zur Verfügung stellt. Im weiteren Verlauf dieses Abschnittes werden diese drei Funktionalitäten tiefgehender erläutert und Codeausschnitte exemplarisch vorgestellt.

Generierung des DOM-Codes

Die Datei, welche für die Generierung des DOM-Codes verantwortlich ist, ist die **`renderer.php`**. Die Datei **`renderable.php`** implemeniert hingegen ein Interface, welches für die Verwendung des moodleinternen Renderers notwendig ist [moof]. Die in der Datei **`renderer.php`** enthaltende Klasse **`mod_dsbuilder_renderer`** enthält Funktionen zum erstellen der für den DSBuilder benötigten Ansichten. So können Übersichten über laufende oder eingereichte Abgaben oder Notentabellen generiert werden. Ebenso können Ansichten zur Aufgabenbearbeitung generiert werden. Hier-

```
1  // call initialize graph function
2 $this->page->requires->js_init_call('M.mod_dsbuilder.
     init_jsdot_show', array(
3                 $div_id_graph_2,
4                 json_encode($com_object->graph)
5 ), false, self::get_jsdot_module_info());
```

Quellcode 1: Aufruf zur Initialisierung eines JSDot-Graphs

bei übernimmt die Initiierung des Grapheneditors, welche im Quellcode 1 dargestellt wird, eine zentrale Rolle. Es wird eine Hilfsfunktion des Moodle-API [mood] verwendet, welche die frontendseitige JavaScript-Funktion zur Initialisierung des Grapheneditors anstößt. Die frontendseitige Funktionalität wird im Kapitel 3.2.3 vertieft.

Modulinterne AJAX-API

Zur asynchronen Datenübertragung zwischen Browser und Server stellt das Moodlemodul eine AJAX Api zur Verfügung. Der Quellcode 2 zeigt einen Codeausschnitt aus der **`ajax_request.php`**, in dem die möglichen Aktionen definiert sind, die nach einer AJAX-Anfrage durchgeführt werden können. Hierbei handelt es sich um die Funktionen, welche über die Knöpfe unterhalb des Grapheneditors (vergl. Kapitel

3.1, Abschnitt Studierender) angestoßen werden können. Explizit handelt es sich um die Funktionen *Syntax prüfen* (Quellcode 2, Z. 2), *Speichern und weiter* (Quellcode 2, Z. 6) und *Letzten Schritt wiederholen* (Quellcode 2, Z. 10). Die jeweils angestoßenen Funktionen sind in der **ajax_helper.php** definiert. Von dort aus werden weitere

```
1  try {
2      if ($action === DSBUILDER_AJAX_ACTION_CHECK) {
3          $jsdot_graph_raw = required_param('jsdot_graph', PARAM_TEXT
               );
4          $result = $dsbuilder_ajax->action_check_valid_graph(
               $jsdot_graph_raw);
5          $dsbuilder_ajax->add_to_log($action, $step_no);
6      } elseif ($action === DSBUILDER_AJAX_ACTION_NEXT_STEP) {
7          $jsdot_graph_raw = required_param('jsdot_graph', PARAM_TEXT
               );
8          $result = $dsbuilder_ajax->action_submit_current_step(
               $jsdot_graph_raw);
9          $dsbuilder_ajax->add_to_log($action, $step_no);
10     } elseif ($action === DSBUILDER_AJAX_DELETE_LAST_STEP) {
11         $result = $dsbuilder_ajax->action_delete_last_step();
12         $dsbuilder_ajax->add_to_log($action, $step_no);
13     }
14 }
```

Quellcode 2: AJAX API

Funktionen zur Datenstrukturverarbeitung in der Klasse **jsdot_graph** angestoßen. Diese Funktionen werden im nächsten Abschnitt vertiefender behandelt.

Die Datenhaltung

Das Modul DSBuilders benötigt vier Entitäten für seine Datenhaltung. Es handelt sich um die Entitäten *DSBuilder*, *Assigment File*, *Submission* und *Submission Step*. Die Abbildung 5 zeigt das Datenmodell des Moduls. Nachdem das Modul neu in einem Kurs initialisiert worden ist, wird ein neues Datum der Entität *DSBuilder* angelegt. Die der neuen Instanz des Moduls vom Lehrenden zugewiesenen Javadateien werden als Datum der Entität *Assigment File* gespeichert. Sobald Studierende die Instanz nutzen, wird für jeden Studierenden mit Vermerk auf die Instanz ein neues Datum der Entität *Submission* angelegt. Jeder *Submission* werden *Submission Steps* zugeordnet. Sie beinhalten Informationen über die jeweiligen ausgeführten Schritte.

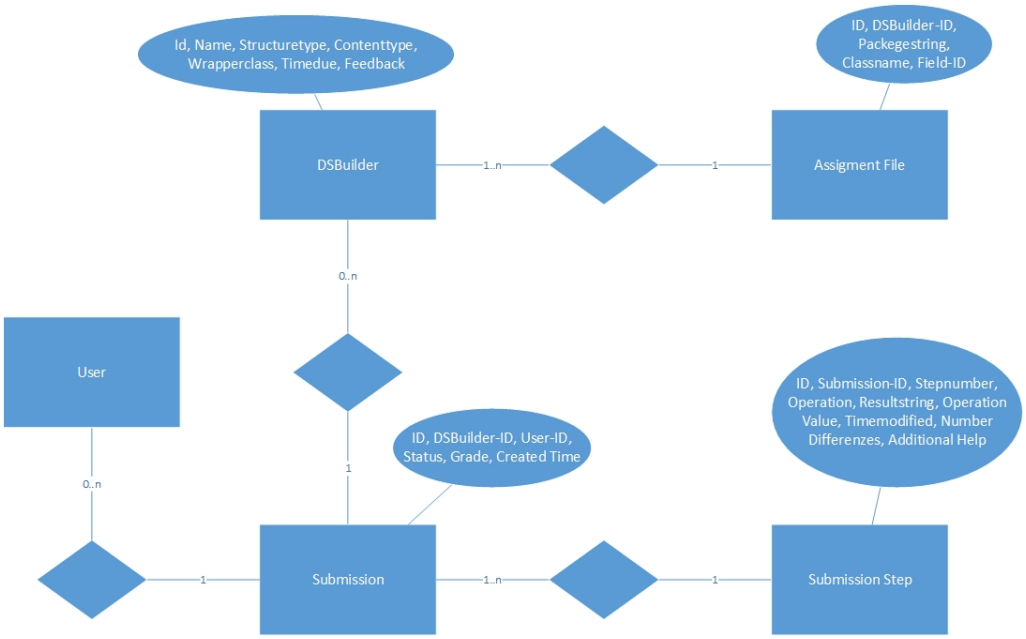

Abbildung 5: Datenmodell des DSBuilders

Die Datenstrukturverarbeitung

In der Datei **`lib/js_dot_convert.php`** liegt die Funktionalität der Datenstrukturverarbeitung. In ihr sind vier Klassen implementiert, von denen die Klasse **`jsdot_graph`** eine Schnittstelle zur Konvertierung einer Datenstruktur zwischen dem Grapheneditor im Frontend und der Datenhaltung im Backend bietet. Abbildung 6 zeigt ein UML-Klassendiagramm der vier Klassen.

3.2.3 Moodlemodul frontendseitig

Die im Vordergrund stehende Funktionalität des Fronends ist die von der JavaScript-Applikation jsdot bereitgestellte Grapheneditor.

Die zentrale Funktionalität zur Verarbeitung der Nutzereingaben wird im Frontend über die Datei **`dsbuilder.js`** Bereitgestellt. Sie organisiert die Kommunikation mit dem Grapheneditor. Der Codeauszug 3 zeigt die Initialisierung eines neuen jsdot-Graphen. Des weiteren stellt diese Datei die AJAX-Kommunikation zur Verfügung.

Der Grapheneditor

Der Grapheneditor wird

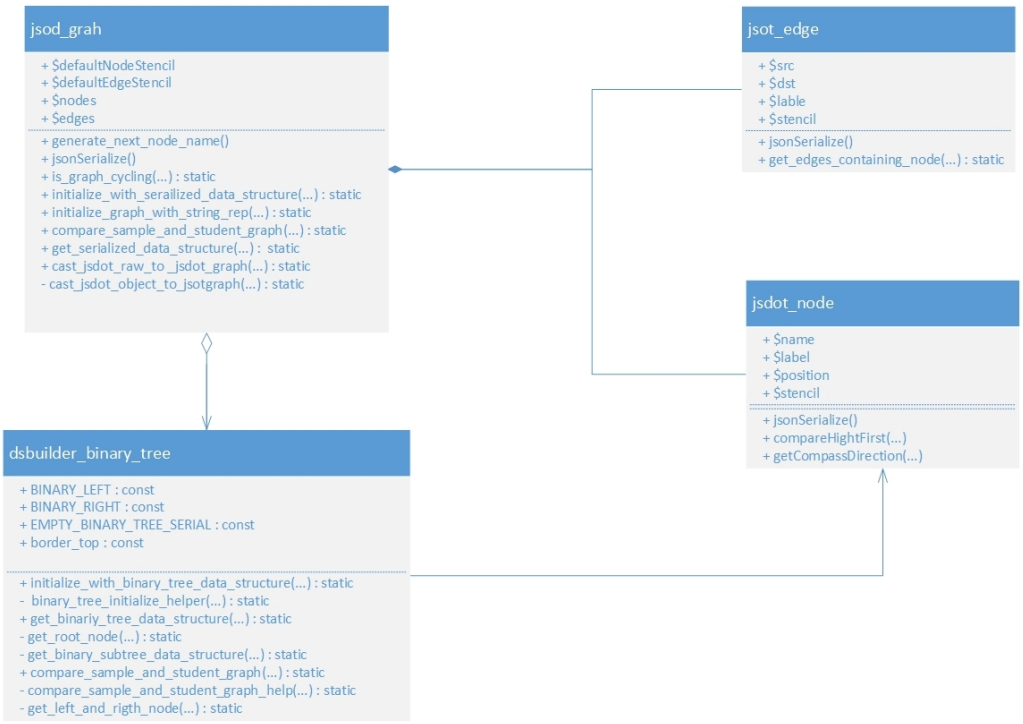

Abbildung 6: UML js_dot_convert

```
init_jsdot_edit : function(e, divname, jsongraph) {
        this.jsdot_graphs[divname] = new JSDot(divname, {
                mode : "editor",
                json : jsongraph
        });
},
```

Quellcode 3: Initiierung eines JsDot-Graphs

4 Konzeptionelle Anforderungen an ein E-Learning-Modul zum Assessment von B-Baum-Datenstrukturen

In diesem Kapitel werden die bestehenden Anforderungen an ein E-Learning-Modul zum Assessment von B-Baum-Datenstrukturen vorgestellt und näher erläutert. Da es sich bei diesem E-Learning-Modul um den bereits existierenden EASy-DSBuilder handelt, welcher lediglich um die Funktion des Assessments von B-Baum-Datenstrukturen erweitert werden muss, stellt das Kapitel 4.1 die Neuanforderungen an den EASy-DSBuilder vor. Anschließend wird in Kapitel 4.2 tiefgehender auf die Anforderungen des Grapheneditors eingegangen.

4.1 Neuanforderungen an den EASy-DSBuilder

In diesem Kapitel werden die Anforderungen an die Erweiterungen des EASy-DSBuilders vorgestellt und näher erläutert. Die Hauptanforderung lautet:

> Der EASy DSBuilder soll um die Datenstruktur B-Baum erweitert werden.

Aus dieser Hauptanforderung lassen sich weitere Unteranforderungen Ableiten.

(1) Dem Studierenden muss ein Assessment einer B-Baum-Datenstruktur durchführen können

> Dem Studierenden wird eine graphische Oberfläche bereitgestellt, auf der ein Assessment einer B-Baum-Datenstruktur möglich ist. Hierfür braucht es insbesondere einen Editor, der eine B-Baum-Grundstruktur mit Kanten und Knoten bestehen. Auf dieser soll es möglich sein Schlüssel zu positionieren.

Mit dieser Anforderung ist die grundlegende Funktionalität des Moduls, die das Assessment von B-Baum-Datenstrukturen ermöglichen soll, begründet. Wie in Kapitel 3.1 erläutert verfügt der EASy-DSBuilder über die Funktionalität des Assessments von AVL-Bäumen. So soll das Assessment von B-Baum-Datenstrukturen den selben Funktionalen Umfang des AVL-Baum-Assessments haben. Hierzu gehören folgende Funktionen. Das Assessment geht über 10 Schritte, in denen der Baumstruktur jeweils ein neuer Schüssel hinzugefügt werden soll. Der Studierende erhält einen Überblick über den aktuellen Stand der Aufgabe, die er bearbeitet. Der Studierende erhält einen Editor zum Assessment von B-Baumstrukturen. Der Studierende

kann die Syntax seiner Eingabe prüfen. Der Studierende kann seine Eingabe abgeben. Der Studierende kann den aktuellen Schritt seiner Eingabe zurücksetzen. Der Studierende kann den vorherigen Schritt seiner Eingabe zurücksetzen. Es soll die Möglichkeit des Feedbacks gegeben werden. Im folgenden Verlauf wird detaillierter auf die soeben genannten Anforderungen eingegangen.

() Das Assessment soll zehnschrittig sein

> Das Assessment soll über 10 Schritte gehen, in denen der Baumstruktur pro Schritt jeweils ein neuer Schüssel hinzugefügt wird.

Ebenso wie beim Assessment des AVL-Baums soll das Assessment des B-Baums über zehn Schritte gehen. Pro Schritt wird dem Studenten hierbei ein neuer Schlüssel zur Verfügung gestellt, die er in den bereits vorhanden B-Baum einfügen können soll. Nachdem die zehn Schritte absolviert sind, soll der Studierende darüber informiert werden, dass er die gesamte Aufgabe absolviert hat.

() Der Studierende soll einen Überblick über den aktuellen Aufgabenstand erhalten

> Der Studierende soll die Möglichkeit erhalten einen Überblick über den aktuellen Stand der Aufgabe, die er bearbeitet, zu bekommen.

Ebenso wie beim Assessment des AVL-Baums soll das Assessment des B-Baums über eine Übersicht über dein aktuellen Stand der Aufgabe bereitstellen. Auf dieser Übersicht erhält der Studierende Informationen darüber, wie viel Prozent er bereits absolviert hat, in welchem Schritt er sich befindet und welcher Schlüssel in diesem Schritt zum Einfügen bereitgestellt wurde.

() Der Studierende soll einen Editor zum Assessment von B-Baumstrukturen erhalten

> Der Studierende soll die Möglichkeit bekommen, eine B-Baum-Datenstruktur innerhalb eines Editors so zu bearbeiten, dass er die Aufgabenstellung jedes Schrittes lösen kann.

Ebenso wie beim Assessment des AVL-Baums soll das Assessment des B-Baums über einen Editor verfügen, in dem man eine B-Baum-Datenstruktur bearbeiten kann. Die Elemente, die durch den Editor bereitgestellt werden sollen, sind Kanten, Knoten und Schlüssel. Es soll möglich sein, aus diesen Elementen eine syntaktisch

korrekten B-Baum zu erstellen. Des weiteren muss der Editor Funktionalitäten bereitstellen, die das Einfügen eines neuen Schlüssels in einen bereits bestehenden B-Baum ermöglichen.

() Der Studierende soll die Syntax seiner Eingabe prüfen können

() Der Studierende soll seine Eingabe abgeben können

() Der Studierende soll den aktuellen Schritt seiner Eingabe zurücksetzen können

() Der Studierende soll den vorherigen Schritt seiner Eingabe zurücksetzen können

() Es soll die Möglichkeit des Feedbacks gegeben werden

> Je nach Einstellung des Lehrenden soll der Studierende bei falscher Eingabe in Feedback bekommen.

Momentan besteht die für den Lehrenden die Möglichkeit ein Feedback für den Studierenden einzustellen. Hat dieser das Feedback aktiviert, so soll während des Assessments von B-Baum-Datenstrukturen bei falscher Eingabe seitens des Studierendem dem Studierenden unterhalb des Editors der B-Baum mit richtig eingefügtem Schlüssel angezeigt werden. Weiterhin sollen die sich in Editor und Feedback unterscheidenden Knoten rot markiert werden.

() Navigationsmöglichkeiten innerhalb einer Aufgabe müssen vorhanden sein

> Der Studierende soll die Möglichkeit haben, innerhalb der einzelnen Schritte einer Aufgabe zu Navigieren. Zur Navigation gehört das Bestätigen des aktuellen Schritts und das Zurücksetzen auf den vorherigen Schritt.

Bisher hat der Studierende die Möglichkeit sich über die Knöpfe *Syntax prüfen*, *Speichern und weiter*, **Letzten* Schritt wiederholen* und **Aktuellen* Schritt zurücksetzen* durch eine Aufgabe zu navigieren. Für das Assessment von B-Baum-Datenstrukturen soll diese Art der Navigation beibehalten werden.

() Aufgabenschritte müssen auf der vorherigen Eingabe basieren

Bis auf den Initialisierungsschritt müssen alle Aufgabenschritte einer Aufgabe auf der letzten Eingabe des Studierenden basieren.

() Der Lehrende soll eine Auswahlmöglichkeit erhalten, zwischen Datenstrukturen wählen zu können

Der Lehrende soll bei Einrichtung einer neuen Instanz des DSBuilders aus einer Liste mit allen verfügbaren Datenstrukturen die Auswahl treffen können, mit welcher Datenstruktur die Anwendung dem Studierenden bereitgestellt wird.

- editirbare graphische Oberfläche zur Erstellung von B-Bäumen
- Funktionalität soll beibehalten werden:
  - Kommunikation Moodle <—> Web-Server
  - Schritte werden gespeichert
  - Eingabe überprüfen
  - Feedback
  - farbige Markierung falscher Knoten
  - Schritt zurück
- Lehrender: Auswahl zwischen Typ
- alles in einer akzeptablen Zeit.

4.2 Spezifizierung der konzeptionellen Anforderungen des Grapheneditors

Wenn man die Anforderungen aus Kapitel 4.1 auf eine Kernanforderung reduziert, so kann sie darin verstanden werden, dass dem Studierenden ein graphischer Editor zur Verfügung gestellt werden soll, über den ein B-Baum erweitert werden kann.

Vergleicht man eine B-Baum mit dem bereits implementierten AVL-Baum, so verfügt der B-Baum über eine weitere editierbare Komponente. Beim AVL-Baum sind Knoten und Schüssel durch ein **eins-zu-eins-Mapping** unveränderbar miteinander verbunden, beim B-Baum kann ein Knoten jedoch mehrere Schlüssel enthalten, welche aber auch nicht fest an diesen Knoten gebunden sind. Die daraus resultierende Herausforderung ist die drei Komponenten Kante, Knoten und Schlüssel dem Studierenden so zur Verfügung zu stellen, dass der Studierende die Komponenten möglichst intuitiv nutzen kann. Die JavaScript-Applikation jsdot soll dabei beibehalten werden. Es ergaben sich die Möglichkeiten, einen B-Baum komplett aus editierbaren jsdot-Elementen zu implementieren, oder eine B-Baumstruktur in den Hintergrund des Editors zu legen, auf dem die Schlüssel richtig eingeordnet werden müssen. Im Folgenden werden Vor- und Nachteile beider Möglichkeiten vorgestellt.

B-Baum aus jsdot-Elementen

Jsdot bietet die Möglichkeit neben runden auch rechteckige Knoten zu erstellen. Aus diesen könnten Knoten generiert werden. Auf diese Weise könnten Studierende eine B-Baumstruktur auf der Basis dieser Knoten und Kanten erstellen, um darauf die jeweiligen Schlüsselelemente richtig zu positionieren.

Jedoch ergeben sich daraus mehrere Probleme. Zum einen ist die Umsetzung der Positionierung der abwärtsgerichteten Kanten ein Problem. Idealtypisch ist die Positionierung der abwärtsgerichteten Kanten am Anfang oder am Ende eines Knotens oder zwischen zwei Schlüsseln (vergl. Abb. 3). Dies ist jedoch ohne einen größeren Eingriff in die jsdot-Implementierung nicht möglich. Alternativ können die abwärtsgerichteten Kanten an die Schlüssel gehängt werden, wobei der erste oder letzte Schlüssel eines Knotens über zwei abwärtsgerichtete Kanten verfügt. Die Kanten können dann wie zuvor an Knoten enden. Jedoch kann nicht unterbunden werden, dass Schlüssel oder Kanten untereinander im nicht beabsichtigten Sinn verbunden werden. Daraus resultiert, dass eine Großzahl an möglichen falschen Eingabemöglichkeiten überprüft werden müsste. Auch könnte diese Vorgabe die Usability beeinträchtigen, da sich die geforderte Benutzung der einzelnen Elemente seitens des Studierenden nicht eindeutig nachvollziehbar herausstellen könnte. Des Weiteren bereitet die Höhenhierarchie der einzelnen jsdot-Elemente Probleme. So ist es auf Grund fehlender Dokumentation bezüglich der Höhenhierarchie einzelner Elemente schwer umsetzbar Schlüssel so zu definieren, dass sie Knoten durchgehend überlappen.

B-Baumstruktur im Hintergrund

Die Baumstruktur im Hintergrund bietet den Vorteil, dass nur noch die einzelnen Schlüssel editierbar sein müssen, der Studierende folglich nur noch die Schlüssel an die richtige Position bringen muss. Dies bietet den Vorteil, dass die Richtigkeit über die absolute Position, und nicht Position relativ zu anderen Schlüsseln, geprüft werden kann. Es ergeben sich zwei Möglichkeiten eines Hintergrundgraphen. Es kann entweder ein statischer Baum, der durchgehend die gesamte Struktur aufweist, oder ein dynamischer Baum, der sich der Eingabe des Studierenden anpasst, implementiert werden. Ein statischer Baum erfordert weniger Implementierungsarbeit, ein dynamischer Baum erhöht jedoch die Übersichtlichkeit, da nur benötigte Teile des Baums angezeigt werden. Außerdem bietet ein dynamischer Baum dem Studierenden die Chance die Entwicklung eines B-Baums besser zu veranschaulichen. implementiert **kann man das so Schreiben −> ist ja nur meine Meinung** Diesen Vorteilen steht jedoch gegenüber, dass der Studierende aus didaktischer Sicht in seinen Bearbeitungsmöglichkeiten eingeschränkt wurde. Eingeschränkt aus didaktischer Sicht bedeutet in diesem Kontext, dass den Studierenden während in der soeben beschriebenen Aufgabenbearbeitung im Verglich zur papierbasierten Aufgabenbearbeitung Bearbeitungsvariationen bei der Lösungsfindung vorenthalten werden. Insbesondere kann der Studierende die Baumstruktur nicht mehr falsch aufbauen, sodass die Hintergrundstruktur des B-Baums zur Reduktion möglicher Fehlerquellen führt.

Ergebnis

Insbesondere der B-Baum aus jsdot-Elementen bringt verschiedene Herausforderungen mit sich. So ist es in dieser Variante auf Grund der Vielzahl von Elementen schwierig eine einfach zu verstehende und zu benutzende graphische Oberfläche zu bieten. Dem gegenüber steht die einfach zu bedienende graphische Oberfläche mit einer B-Baumstruktur im Hintergrund, die dem Studierenden jedoch striktere Vorgaben in der Benutzung vorgibt und somit Fehlerquellen reduziert. Die hier vorliegende Situation beschreibt einen Trade-off zwischen geringer Usability mit einem hohen Maß an Bearbeitungsvariationen und einer hohen Usabiltiy mit einem geringeren Maß an Bearbeitungsvariationen. Wie die Auswertung der Fallstudie über die Nutzung des EASy-DSBuilders zeigte sorgte besonders die gute Usability für hohe Akzeptanz bei den Studierenden[Use14, S. 5]. Auch in der neuen Anwendung soll der Akzeptanz eine hohe Priorität zugeschrieben werden, sodass trotz der Vereinfachung der Aufgabe durch die Reduktion möglicher Fehlerquelle der Editor mit Hintergrundstruktur dem Editor, in dem alle Elemente bewegbar sind, vorzuziehen ist.

5 Umsetzung der Änderungsanforderungen

Dieses Kapitel stellt die Stellen vor, an denen Änderungen vorgenommen werden mussten und erläutert die Ursachen, auf Grund derer die Änderungen vorgenommen werden mussten. Hierzu wird zu erst in Abschnitt 5.1 ein Überblick über die notwendigen Änderungen und Erweiterungen gegeben. In den restlichen Abschnitten wird tiefgehender auf die wichtigen Änderungen und Erweiterungen eingegangen. Dabei wird zuerst die Funktionalität erläutert. Zur Verdeutlichung werden anschließend exemplarisch Ausschnitte aus der Implementierung vorgestellt.

5.1 Überblick über Änderungen

In diesem Abschnitt wird ein grundlegender Überblick über alle Änderungen gegeben, die in dem Moodlemodul EASy-DSBuilder vorgenommen wurden, um die Funktionalität des Assessments von B-Baum-Datenstrukturen.

Für die Verwendung des EASy-DSbuilders ist eine Initialisierung seitens eines Lehrenden nötig. In der alten Version konnten die in Kapitel 3.1 beschriebenen Eigenschaften übergeben werden. Es konnte jedoch noch nicht zwischen Datenstrukturen gewählt werden. So wurde in der alten Version als Datenstrukturtyp defaultmäßig *Tree* übergeben. An dieser Stelle musste eine Auswahlmöglichkeit für Datenstrukturen in der Erstellungsmaske des Moduls implementiert werden. Die Auswahlmöglichkeit wurde als Dropdown-Menü umgesetzt, wobei die in der Datei `dsbaStructureType.php` definierten Konstanten als Datenstrukturtypen zur Auswahl gestellt werden.

Nachdem eine Instanz des EASy-DSBuilders erzeugt wurde, muss dem Modul jedoch zur Durchführung eines Assessments eine Datenstruktur seitens des Datenstrukturverarbeitungs-Webservices bereitgestellt werden. Die zur Kommunikation zwischen Moodlemodul und Webservice verwendete Datenstruktur wird in Abschnitt 5.2 vorgestellt. An der Funktionalität des Datenstrukturverarbeitungs-Webservice sollte nichts geändert werden. Fehler, die während der Entwicklung auftraten, wurden behoben. Zur Endwicklung des Moodlemoduls musste jedoch die Datenstruktur eines B-Baums in Java implementiert werden, damit die Funktionen des Webservices bereitgestellt werden konnten. **Über die B-Baum Implementierung schreiben?**

Die wichtigen Erweiterungen zur Bereitstellung der Funktionalität eines Assessments von B-Baum-Datenstrukturen wurde im Moodlemodul fronend- und backendseitig implementiert. Hierbei steht frontendseitig die Erweiterung des Grapheneditors im Vordergrund, welche ausführlich im Abschnitt 5.4 erläutert wird. Des Wei-

teren war die Implementierung einer Verarbeitung der Datenstruktur nötig, sodass zwischen der vom Webservice stammenden Datenstruktur und der für den Editor gebrachten Datenstruktur gewandelt werden konnte. Diese Funktionalität wird ausführlicher in Abschnitt 5.3 erläutert. Des weiteren wurden noch folgende kleinere Veränderungen vorgenommen. In der Datei `renderer.php`, in der die Ansicht generiert wird, wurde eine Fallentscheidung für die Anzeige des Grapheneditors implementiert. Dies ist damit begründet, dass der B-Baum-Editor, wie in Kapitel 4.2 begründet, zwei übereinander liegende jsdot-Editoren benötigt. Die Überlagerung der zwei Editoren wurde mit *CSS* umgesetzt. Weiterhin wurde die AJAX-Api angepasst. Die minimale Änderung bestand darin, dass eine weitere Information übergeben wird, auf die der B-Baum-Editor aufbaut. Näheres dazu ist in Abschnitt 5.4.2 zu finden.

5.2 Datenstruktur für die Moodlemodul-Webserice-Kommunikation

Die Datenstruktur, die in in der Kommunikation zwischen Moodlemodul und Webservice eingesetzt wird, spielt eine fundamentale Rolle in der Gesamtanwendung EASy-DSBuilder. Diese Datenstruktur muss in einem JSON-kompatiblen Format alle notwendigen Informationen über die zu verarbeitende Baumstruktur beinhalten. Im Falle des B-Baums muss die Datenstruktur als zentrales Element den Knoten mit der jeweiligen Anzahl an Schlüsseln wiedergeben. Des weiteren muss die Datenstruktur noch die Kindsknoten der Wurzelknotens beinhalten, die wiederum richtig positioniert werden müssen. So können die Kindsnoten als linkes oder rechtes Kind des Wurzelknotens positioniert werden. Weiterhin gibt es noch Kindsknoten, die zwischen den einzelnen Schlüsseln des Wurzelknotens positioniert werden (vergl. Kapitel 2.3).

Auf Basis dieser Grundlage wurde zur Abbildung eines B-Baums ein Array als Datenstruktur ausgewählt. Dieser Array ist bei einem B-Baum T mit einer Höhe größer eins wie folgt aufgebaut. Erstes Element des Arrays, welches die oberste Wurzel des B-Baums T repräsentiert, ist der linke Kindsknoten der Wurzel. Anschließend folgt der erste Schlüssel der Wurzel, dem ein weiterer Kindsknoten nachfolgt. Diese Abfolge aus Schüssel und Kindsknoten erfolgt so lange, bis das Array mit dem rechten Kindsknoten der Wurzel abschließt. Die jeweiligen Kindsknoten sind nach der selben Struktur des Wurzelknotens aufgebaut. Eine mögliche Repräsentation eines B-Baums durch diese Datenstruktur kann wie folgt aussehen:

[[[”16”, ”19”], ”31”, [”37”, ”41”]], ”50”, [[”56”], ”86”, [”96”]]]

Die so eben gezeigte Datenstruktur zeigt einen B-Baum der Höhe drei. Der oberste Wurzelknoten enthält den Schüssel ”3”. Die Wurzel hat das linke Kind [[”16”, ”19”], ”31”, [”37”, ”41”]] und das rechte Kind [[”56”], ”86”, [”96”]], welche jeweils separat betrachte einzelne B-Bäume darstellen. Diese Beispiel verdeutlicht anschaulich den rekursiven Aufbau dieser Datenstruktur. Abgeschlossenen wird die Struktur durch kinderlose Knoten, wie sie im vorgestellten B-Baum beispielsweise durch den Knoten [”16”, ”19”] repräsentiert werden. Diese Datenstruktur steht wiederum in einem weiteren Array mit der Länge zwei an erster Position. An zweiter Position wird die Ordnungszahl k (vergl. Kapitel 2.3) abgelegt.

5.3 Datenstrukturverarbeitung backendseitig

Die in diesem Abschnitt vorgestellten Änderungen beziehen sich auf die in Kapitel 3.2.2 unter Abschnitt *Datenverarbeitung* vorgestellte Funktionalität. Bei der Datei, in der Änderungen vorgenommen wurden handelt es sich um die **`lib/js_dot_convert.php`**. In ihr befinden sich in der alten Version des EASy-DSBuilders die Klassen **`jsdot_graph`**, **`jsdot_edge`**, **`jsdot_node`** und **`dsbuilder_binary_tree`**. Die Klasse **`jsdot_graph`** dient als Schnittstelle zum restlichen Modul und verwaltet den jsdot-Graph. Zum Umgang mit übergebenen Datenstrukturen bedient die Klasse sich im Fall von Binärbäumen der Klasse **`dsbuilder_binary_tree`**. Die nun vorgenommene Erweiterung enthält die Implementierung der Klasse **`dsbuilder_b_tree`**. Diese neu implementierte Klasse orientiert sich stark an der Klasse **`dsbuilder_binary_tree`**. Sie stellt die selben Funktionen bereit, nur dass die Funktionen in der Klasse **`dsbuilder_b_tree`** dem Umgang mit B-Baum-Datenstrukturen dienen.

Die drei öffentlichen Funktionen der Klasse **`dsbuilder_b_tree`** sind die statischen Funktionen *initialize_with_b_tree_data_structure(...)*, *get_b_tree_data_structure(...)* und *compare_sample_and_student_graph(...)*. Hierbei ermöglichen die ersten beiden Funktionen die Umwandlung der Datenstruktur, sodass mit der Datenstruktur in den beiden Anwendungsfällen Grapheneditor und Webservices gearbeitet werden kann. Die dritte Funktion vergleicht zwei Graphen und markiert die sich unterscheidenden Knoten. Sie wird für die Feedbackfunktion genutzt. Im folgenden werden die Funktionen tiefgehender vorgestellt.

Die Funktion *initialize_with_b_tree_data_structure(...)* initialisiert einen neuen **`jsdot_graph`** anhand der in Kapitel 5.2 vorgestellten serialisierten Datenstruktur. Der Funktion können drei Paramieter übergeben werden. Die Übergabe einer serialisierte Datenstruktur ist verpflichtend. Ansonsten kann übergeben werden, wie viel Abstand der Baum im Editor auf der x-Achse zum linken Rand und auf der y-

Achse zum oberen Rand haben soll. Standardmäßig werden 50 Pixel für die x-Achse und 25 Pixel für die y-Achse übergeben. Jedoch handelt es sich bei der Funktion *initialize_with_b_tree_data_structure(...)* nur um eine Wrapperfunktion. Die eigentliche Umwandlung einer serialisierten Datenstruktur in einen **jsdot_graph** geschieht in der Hilfsfunktion *b_tree_initialize_helper(...)*. Quellcode 4 zeigt einen Ausschnitt der Hilfsfunktion. Da es sich bei der serialisierten Datenstruktur um

```
1  private static function b_tree_initialize_helper(jsdot_graph
       $jsdot_graph, $json_row, $height, $index_array, $spacing_x,
       $spacing_y, $t) {
2    /* ... */
3    $index_node = 0;
4    foreach($json_row as $e){
5      if(gettype($e) == "string"){
6            $node = new jsdot_node (/* ... */);
7            $jsdot_graph->nodes [] = $node;
8            $index_node++;
9      } else {
10           $left = self::b_tree_initialize_helper (/* ... */);
11     }
12   }
13 }
```

Quellcode 4: Ausschnitt *b_tree_initialize_helper(...)*

eine rekursiv aufgebaute Datenstruktur handelt ist die Verarbeitung dieser Struktur ebenfalls rekursiv aufgebaut. Der Hilfsfunktion wird eine serialisierte B-Baum-Datenstruktur übergeben. Wie in Kapitel 5.2 beschrieben besteht die Datenstruktur einem Array, welches Schlüssel und Kindsknoten-Arrays des Wurzelknotens enthält. Auf Grund dieser Struktur wird jedes Element des Wurzelknoten-Arrays durchlaufen (Z. 4). Hierbei werden die einzelnen Knoten-Arrays des übergebenen B-Baums durch die Variable **$json_row** widergespiegelt. Für jedes Element in dem jeweiligen Array erfolgt eine Fallunterscheidung, in der geprüft wird, ob es sich bei dem jeweiligen Element um ein *String* handelt. Ist das jeweilige Element ein *String* (Z. 5), so muss es sich um einen Schlüssel im Array handeln. In diesem Fall wird ein neues **jsdot_node** initialisiert und den Knoten des **$jsdot_graph** übergeben (Z. 6 f.). Es werden hierbei jeweils Name, Schlüssel und Position an den Knoten übergeben. Wie die Position bestimmt wird, wird in Kapitel **??** erläutert. Handelt es sich nicht um einen *String*, so muss es sich um ein Kindsknoten-Array handeln. In diesem Fall ruft die Hilfsfunktion sich selber auf, damit die Kindsknoten-Arrays auf

die selbe Weise verarbeitet werden können. Quellcode 5 zeigt den rekursiven Aufruf in seiner Gesamtheit. Der erste übergebene Parameter ist wieder der in Funktion

```
1 self::b_tree_initialize_helper ( $jsdot_graph, $e, ($height + 1),
    $index_node + $index_array * $t, $spacing_x, $spacing_y, $t)
```

Quellcode 5: rekursiver Aufruf aus Quellcode 4 in Z. 10

initialize_with_b_tree_data_structure(...) initialisierte **jsdot_graph**. Der zweite Parameter ist das Kindsknoten-Array. Die nächsten beiden Parameter beschreiben die Position des Arrays. Als erstes wird die Höhe übergeben, die eine größer als die momentane Höhe ist. Danach wird die Position innerhalb einer Schicht weitergegeben. **$index_node** sagt dabei aus, um den wievielten Kindsknoten es sich handelt. **$indes_array** zeigt die Position des Wurzelknotens innerhalb der Schicht auf. Diese muss noch mit der Ordnungszahl **$t** multipliziert werden, damit die richtige Position für den Kindsknoten gefunden wird. Die Summe bildet schließlich die Finale Position. Die restlichen Parameter werden unverändert weitergegeben. Zum Abbruch der Funktion dienen die kinderlosen Knoten, da in ihnen keine Kindsknoten-Arrays zum weiteren rekursiven Aufruf vorhanden sind.

Die Funktion *get_b_tree_data_ structure(...)* wandelt wiederum eine **jsdot_graph**-Datenstruktur in eine serialisierte Datenstruktur für den Webservice um. Quellcode 6 zeigt die gesamte Funktion. So wird der **jsdot_graph** als erstes in die Form

```
1 public static function get_b_tree_data_structure(jsdot_graph
    $jsdot_graph, $t) {
2       $nodesPerTier = self::get_b_tree_nodes_per_tier_structure(
          $jsdot_graph, $t);
3       $result = self::b_tree_structure_helper($nodesPerTier
          [0][0], $nodesPerTier, 0, 0, $t);
4       return json_encode ( [$result, $t] );
5 }
```

Quellcode 6: *get_b_tree_data_ structure(...)*

„Knoten pro Schicht“umgewandelt (Z. 2). Da diese Form bereits aus Kapitel 5.4.2 bekannt ist, und die Umwandlung hier äquivalent verläuft, wird an dieser Stelle nicht weiter darauf eingegangen. Wichtig für diese Funktion ist jedoch noch, dass in ihr die Richtigkeit der Syntax des B-Baums geprüft wird. Es wird geprüft, ob alle Schlüssel Teil des Baums sind, ob alle Knoten die maximale Anzahl an Schlüsseln nicht überschreiten und ob der Baum ausbalanciert ist. Anschließend wird die rekursive Hilfsfunktion *b_tree_structure_helper(...)* aufgerufen, die die eigentliche

Umwandlung umsetzt. Die Hilfsfunktion wird mit dem Wurzelknoten des B-Baums initialisiert (**`$nodesPerArray[0][0]`**). Weiterhin wird noch der gesamte B-Baum, die Position des Wurzelknotens und die Ordnungszahl **`$t`** übergeben. Quellcode 7 zeigt die Hilfsfunktion. In der Funktion wird jeder Schlüssel des übergebenden Kno-

```
1  private static function b_tree_structure_helper($nodeList,
       $nodesPerTier, $i, $j, $t){
2      $result = array();
3      $index = 0;
4      foreach ($nodeList as $node){
5              if ($nodesPerTier[$i+1][$j*$t + $index]){
6                      $r = self::b_tree_structure_helper(/* ... */);
7                      array_push($result, $r);
8              }
9              array_push($result,   $node->label->value);
10             if ($nodesPerTier[$i+1][$j*$t + $index+1]  && sizeof(
                   $nodeList)-1 == $index){
11                     $r = self::b_tree_structure_helper(/* ... */);
12                     array_push($result, $r);
13             }
14             $index++;
15     }
16     return $result;
17 }
```

Quellcode 7: *b_tree_structure_helper(...)*

tens (**`$nodeList`**) durchlaufen (Z. 4). Für jedes Element wird zu erst geprüft, ob ein linker Kindsknoten existiert (Z. 5). Ist dies der Fall, ruft die Funktion sich selbst auf, wobei der Kindsknoten als zu bearbeitendes Element übergeben wird (Z. 6). Anschließend wird das Ergebnis dieses rekursiven Aufrufs zum Ergebnis hinzugefügt (Z. 7). Auch jeder Schlüssel wird dem Ergebnis übergeben (Z. 9). Die zweite if-Abfrage (Z. 10) prüft, ob sich um den letzten Schlüssel eines Knotens handelt, und ob dieser einen rechten Kindsknoten hat. Ist dies der Fall, wird für diesen Kindsknoten ein weiterer rekursiver Aufruf initiiert (Z. 11). Als abbrechende Bedingung für die Rekursion dienen die Kinderlosen Knoten.

5.4 Grapheneditor

Dieses Kapitel erläutert die neu entwickelte Funktionalität des Frontends. Insbesondere heißt das, dass auf die Umsetzung des Grapheneditors für die B-Baumstruktur

eingegangen wird. Hierzu wird zu erst die Umsetzung des in Kapitel 4.2 vorgestellten Konzepts des Grapheneditors in Kapitel 5.4.1 vorgestellt. Anschießend wird in Kapitel 5.4.2 tiefgehender auf einzelne, in Kapitel 5.4.1 vorgestellten Funktionen eingegangen.

5.4.1 Funktionalität

Aus dem im Kapitel 4.2 bereits erläuterten Gründen soll der Grapheneditor aus zwei separaten Elementen bestehen. Diese beiden Elemente sind zwei jsdot-Editoren, welche übereinander liegen. Der im Hintergrund liegende Editor ist nicht bearbeitbar und und liefert die B-Baumstruktur, auf der die Schlüssel des B-Baums angeordnet werden können. Die Schlüssel werden von einem im Vordergrund liegendem jsdot-Editor zur Verfügung gestellt und sind verschiebbar. Um eine höhere Interaktivität bereitzustellen entwickelt sich die im Hintergrund liegend Baumstruktur dynamisch. Das bedeutet, dass sobald ein Schlüssel zu der Baumstruktur hinzugefügt wird, sich die Baumstruktur automatisch um neue Knoten vergrößert. Die Anpassung beinhaltet die Erweiterung des Knotens, in den eingefügt wurde, sowie die Kindsknoten, die der erweiterte Knoten bereitgestellt bekommt. Ebenso reduziert sich die Baumstruktur nach Endnahme eines Schlüssels um die jeweiligen Knoten. Sobald ein Schlüssel auf einen Knoten abgelegt wird, erkennt das System dies und richtet den gesamten Knoten mit dem neu eingefügten Schlüssel in Echtzeit aus. Durch die automatische Neuausrichtung der Schlüssel kann der Studierende erkennen, ob das System seine Eingabe erkannt hat.

Abbildung 7 zeigt ein Petri-Netz, welches die Funktionalität der Echtzeitverarbeitung beschreibt. Das Petri-Netz beinhaltet drei Arten von Transaktionen. Zur erste Art von Transaktionen gehören Referenzen auf Funktionen, die eine Kommunikation zwischen dem GUI und dem Nutzer ermöglichen. Es wird hierfür auf die jQuerey-Funktionen *mousemove()*, *mousedown()* und *mouseup* verwiesen. Zur besseren Erkennbarkeit sind auf diese Funktionen verweisenden Referenzen im Petri-Netz kursiv abgebildet. Bei der zweiten Art von Transaktionen handelt es sich um Ergebnisse von Verzweigungen. Die dritte Art von Transaktionen beschreibt umfangreichere Funktionalitäten, welche in Kapitel 5.4.2 tiefgehender beschreiben werden.

Ausgangspunkt ist die Funktion *b_tree_helper()*, welche die Echtzeitverarbeitung initialisiert. Aus der auf diese Funktion referierenden Transaktion heraus entstehen zwei Stellen, welche auf Eingaben des Nutzers warten. Es handelt sich hierbei um die Funktionen *mousemove()* und *mousedown()*. *Mousemove()* schlägt jedes mal aus, wenn der Mauszeiger über den Editor fährt. Aus der Stelle *Mouse selected*

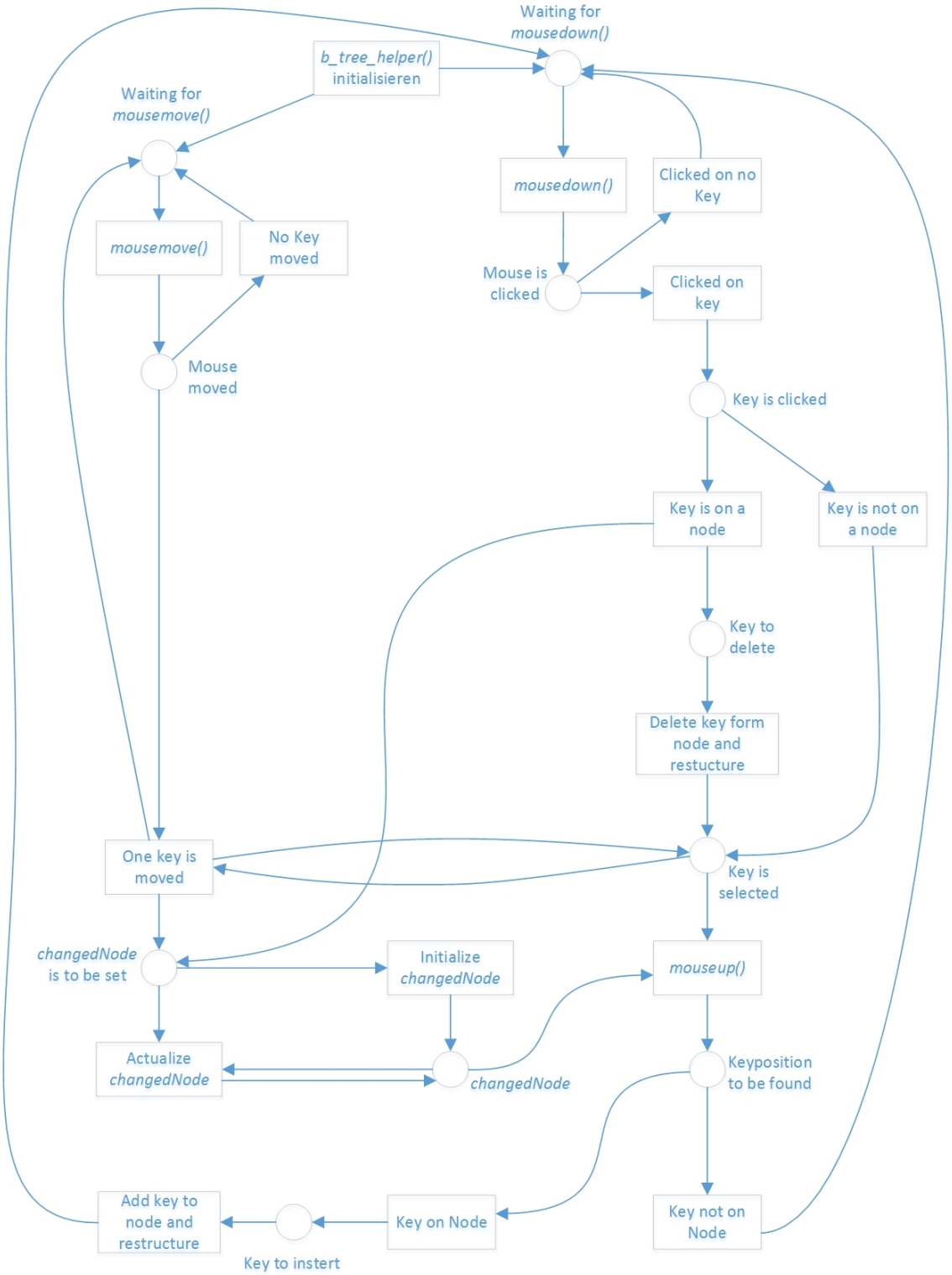

Abbildung 7: Petri-Netz des Grapheneditors

heraus gibt es zwei mögliche Folgezustände. Wurde kein Schlüssel bewegt (*No key moved*) wird der nächste Zustand mit der erneuten Belegung der Stelle *Waiting for mousemove()* erreicht. Für die Transaktion *One key is moved* muss jedoch die Stelle *Key is selected* belegt sein. Für die Belegung dieser Stelle muss zuerst die Transaktion *mousedown()* durchlaufen sein. Anschließend können die Transaktionen *Clicked on no key* und *Clicked on Key* ausgeführt. Der Grund für den direkten Rücksprung zur Stelle *Waiting for mousedown()* über die Transaktion *Clicked on no key* erfolgt gegen Ende des Kapitels. Wurde jedoch auf einen Schlüssel gedrückt (*Clicked on key*), wird im Anschluss die Verzweigung durchlaufen, die zwischen dem Schlüssel auf (*Key is on node*) oder nicht auf einem Knoten (*Key is not on node*) unterscheidet. Ist der Schlüssel nicht auf einem Knoten, wird direkt die Stelle *Key is selected* markiert. Ist der Schlüssel hingegen auf einem Knoten wird zuerst die Stelle *changeNode is to be set* markiert. Währenddessen wird die Transaktion *Delete key from node and restructure* durchlaufen, sodass die Stelle *Key is selected* schließlich auch über diese Verzweigung markiert wurde.

In folgenden Absatz wird das Teilnetz um die Stellen *changedNode is to be set* und *changedNode* vorgestellt. Bei *changedNode* handelt es sich um eine Variable, die den zuletzt geänderten Schlüssel beinhaltet. Enthält diese Variable einen Wert, dann ist die gleichnamige Stelle im Petri-Netz markiert. Um die Stelle zu markieren können zwei Transaktionen durchlaufen werden. Zum einen gibt es die Transaktion *Initialize changedNode*, zum anderen *Actualize changedNode*, wobei bei dieser *changedNode* bereits markiert sein muss. Die für beide Transaktionen ausgehende Stelle ist *changedNode is to be set*. Die zur Markierung dieser Stelle notwendigen Transaktionen sind *One key is moved* und *Key is on a node*. Dies bedeutet, dass die Variabel *changeNode* gesetzt wird, sobald ein Schlüssel, der auf einem Knoten liegt, angeklickt, bzw. ausgewählt wird. Des weiteren wird die Variable gesetzt, sobald ein ausgewählter Knoten – siehe Abhängigkeit zwischen *One key is moved* und *Key is selected* – auf dem Editor bewegt wird.

Eine von der Stelle *Key is selected* nächstmögliche Transaktion ist die Transaktion *mouseup*, welche auf die jQuery-Funktion *mouseup()* referiert. Diese Funktion wird ausgelöst, sobald der Nutzer die Maustaste löst. Das Petri-Netz zeigt, dass die beiden Stellen *changedNode* und *Key is selected* markiert sein müssen, damit die Transaktion *mouseup* durchlaufen werden kann. Insbesondere muss in diesem Fall zwischen der durch die Transaktion *mouseup* repräsentierten Funktionalität und der jQuery-Funkion *mouseup()* differenziert werden. Die jQuery-Funktion *mouseup()* wird ausgelöst, sobald der Nutzer die Maustaste löst, folglich auch, wenn kein Schlüssel ausgewählt ist. Der auftretende Widerspruch, der durch die Forderung des

Petri-Netz nach einen gewählten Knoten für die Ausführung der Transaktion auftritt, wird durch Betrachtung der in der Funktion *mouseup()* steckenden, durch die Transaktion beschriebenen, Funktionalität aufgelöst. Die durch die Transaktion *mouseup* beschriebene Funktionalität enthält die notwendige Bedingung, dass ein veränderter Schlüssel vorhanden sein muss. Wird diese Bedingung nicht erfüllt, wird die Funktion abgebrochen. Auf Basis dieser Argumentation ist die durch die Transaktion *mouseup* beschriebene Funktionalität nur mit dem Vorhandensein einer *changedNode* verfügbar, obwohl die jQuery-Funktion *mousemove()* ausgelöst worden ist. Hiermit ist ebenso das Enden der Transaktion *Clicked on no Key* in der Stelle *Waiting for mousedown()* begründet. Hiernach wird nach dem lösen der Maustaste zwar die jQuery-Funktion *mouseup()* ausgelöst, die aus der Funktion resultierende Funktionalität wird jedoch nicht angesprochen.

5.4.2 Implementierungsdetails

In diesem Kapitel wird die Umsetzung der in Kapitel 5.4.1 beschriebene Funktionalität der Echtzeitverarbeitung des Grapheneditors vorgestellt. Hierfür wird zuerst auf die wichtigen Datenstrukturen eingegangen. Anschließend werden die für die Gesamtfunktionalität wichtigen Teilfunktionalitäten veranschaulicht. Zu diesem Zweck wird insbesondere auf die bereits in Kapitel 5.4.1 vorgestellten Teilfunktionalitäten eingegangen.

Die zentrale Datenstruktur für die Echtzeitverarbeitung der Schlüsselausrichtung liefert das mehrdimensionales Array **`nodesPerTier`**. Der Datenstruktur kann man die Knoten einer Schicht entnehmen, wobei die einzelnen Knoten wiederum über ihre jeweiligen Schlüssel verfügen. Alle Knoten einer Schicht meint in diesem Zusammenhang alle Knoten einer bestimmten Höhe. So verwaltet die höchste Arraydimension die Schichten des B-Baums. Insgesamt ist der mehrdimensionale Array so aufgebaut, dass die oberste Schicht des B-Baums auf Position null in dem Array der obersten Dimension abgebildet wird. Die zweite Dimension des Arrays beinhaltet die Knoten einer Schicht. Das Arrays auf Position Null beinhaltet diesbezüglich den Wurzelknoten. Auf Position Eins beinhaltet der Array der ersten Dimension die nächste Schicht, sodass dort ein Array mit den Kindsknoten der Wurzel abgelegt ist. Die weiteren Positionen enthalten demnach folglich die weiteren Schichten. Die einzelnen Knoten werden wiederum auch durch Arrays dargestellt, welche alle Schlüssel eines Knotens enthalten. Somit handelt es sich bei dem Array **`nodesPerTier`** um ein dreidimensionales Array mit Schichten auf der ersten, Knoten auf der zweiten und Schlüsseln auf der dritten Dimension.

Eine weitere wichtige Datenstruktur ist das Array **`nodeListBounds`**. Diese Datenstruktur enthält die Grenzen, in denen ein Schlüssel als Schlüssel innerhalb eines Knotens erkannt wird. Wird folglich ein Schüssel auf dem Editor innerhalb dieser Grenzen abgelegt, wird ein Positionierungsprozess angestoßen, in dem die Schlüssel eines Knotens neu positioniert werden. Das Array **`nodeListBounds`** ist dem Array **`nodesPerTier`** stark nachempfunden. So handelt es sich ebenfalls um ein dreidimensionales Array, wobei auch hier die Schichten auf der ersten und die Knoten auf der zweiten Dimension abgelegt werden. In der dritten Dimension sind jedoch anstatt von Schlüsseln die Grenzen der einzelnen Knoten gespeichert.

Die Funktion **`b_tree_helper`** ist die Hilfsfunktion, welche eine Echtzeitverarbeitung der Nutzereingabe ermöglicht. Sie bekommt als Parameter die beiden Graphen, den Vordergrund- und den Hintergrundgraphen, und die Ordnung t übergeben.

5.4.3 Aufbau des B-Baums im Hintergrund

Dieser Abschnitt erläutert das Prinzip nach dem die B-Baum-Struktur im Hintergrund des Grapheneditors aufgebaut wird.

6 Fazit und Ausblick

Anhang

```java
package de.wwu.pi.treeBuilder.api;

public interface DataStructureWrapperInterface<K extends Comparable
    <K>, D> {
        /**
         * Initializes the data structure and returns an empty data
             structure object.
         *
         * @param random
         *             a "pseudo" random number that can be used to
            randomly initialize the data structure.
         * @return returns the data structure object
         */
        public DataStructureWrapperInterface<K, D> initializeEmpty(
           double config);

        /**
         * adds an element (key + data) to the data structure.
         *
         * @param key
         *             key to be added to the DS
         * @param data
         *             data to be added to the DS
         */
        public void add(K key, D data);

        /**
         * removes the element with the key 'key' from the DS
         *
         * @param key
         *             key to be removed from DS
         */
        public void remove(K key);

        /**
         * Deserializes data structure and returns a new data
            structure object. The serialized object was created
            with the
         * serialize method of this class. Important recompute
            hidden values (e.g. balance of nodes within AVL tree)
            after
         * deserialization of an object.
```

```
35          *
36          * @param serialDataStructure
37          *            data structure as JSONString, created with
                the serialize method of this class.
38          * @return data structure object, to be manipulated later.
39          */
40         public DataStructureWrapperInterface<K, D> deserialize(
               String serialDataStructure);
41
42         /**
43          * Serializes the data structure as a JSON String. Hidden
                data e.g. balance flag of a node within an AVL tree
                should
44          * be recomputed after deserialization instead of parsing
                it to the serialized string.
45          *
46          * @return JSON string representing the data structure.
47          */
48         public String serialize();
49
50         /**
51          * A generator method to be called to generate the next key
                , that should be inserted by the student.
52          *
53          * @param random
54          *            random value, that should be used to generate
                 the next key. Same random values should lead to the
                same
55          *            key.
56          * @return key to be inserted
57          */
58         public K generateKeyToInsert(double random);
59
60         /**
61          * A generator method to generate Data for a key.
62          *
63          * @param key
64          * @return
65          */
66         public D generateDataToKey(K key);
67
68         /**
69          * serializes a given key to String
70          *
```

```
71          * @param key
72          *             key to be serialized
73          * @return serialized String
74          */
75         public String serializeKey(K key);
76
77         /**
78          * serializes a given data element to String
79          *
80          * @param data
81          *             data object to be serialized.
82          * @return serialized data object
83          */
84         public String serializeData(D data);
85
86         /**
87          * deserializes a given string representation of a key
                object
88          *
89          * @param serialKey
90          *             string representation of the key
91          * @return key object
92          */
93         public K deserializeKey(String serialKey);
94
95         /**
96          * deserializes a given string representation of a data
                object
97          *
98          * @param serialData
99          *             string representation of the data
100         * @return data object
101         */
102        public D deserializeData(String serialData);
103
104        /**
105         * Info Text for a student.
106         * @return helptext that can be used to give detailed info
                on how what to to in an action/ Might be empty.
107         */
108        public String getAdditionalHelpText();
109 }
```

Quellcode 8: Aufruf zur Initialisierung eines JSDot-Graphs

Literatur

[Blo06] E. Bloh. Methodische Formen des E-/Online-Assessment. *Unveröffentlichtes Manuskript*, 2006.

[Blu13] Norbert Blum. *Algorithmen und Datenstrukturen.* Oldenburg Verlag München, 2013.

[CJ10] Julian Cook and Vic Jenkins. Getting started with e-assessment. *University of Bath*, 2010.

[Ger07] Fredi Gertrsch. *Das Moodle 1.8 Praxisbuch.* Addison-Wesely Verlag, 2007.

[Gre] Jason Greene. WildFly News.

[KG10] Herbert Kuchen and Susanne Gruttmann. Computerunterstützter Übungsbetrieb im Informatikstudium. *Zeitschrift für e-lerning*, 1:23–35, 2010.

[KP10] Gerd Kortemeyer and Riegler; Peter. Large-Scale E-Assessments, Prüfungsvor- und Nachbearbeitung. *Zeitschrift für e-lerning*, 1:8–22, 2010.

[mooa] About Moodle.

[moob] Activities.

[mooc] Activity Modules.

[mood] JavaScript usage guide.

[mooe] NEWMODULE Documentation.

[moof] Output renderers.

[moog] Was ist Moodle.

[Mooh] MoodleDocs. Course formats.

[Mooi] MoodleDocs. Local plugins.

[Mooj] MoodleDocs. Moodle architecture.

[Rue10] Cornelia Ruedel. *E-Assessment.* Waxmann Verlag GmbH, 2010.

[Sch05] Rolf Schulmeister. *Lernplattformen für das virtuelle Lernen.* Oldenburg Verlag München Wien, 2. edition, 2005.

[SH09] Christoph Scheb and Ralf Hilgenstock. *moodle einführen*. DIALOGE Verlag, 2009.

[SS14] Gunter Saake and Kai-Uwe Sattler. *Algorithmen und Datenstrukturen*. dpunkt.verlag, 5 edition, 2014.

[Use14] Claus A Usener. EASy-DSBuilder : Automated Assessment of Tree Data Structures in Computer Science Teaching. 2014.

Ich versichere hiermit, dass ich meine Diplomhausarbeit „*Erweiterung eines E-Learning-Moduls zum Assessment von B-Baum-Datenstrukturen*“ selbstständig und ohne fremde Hilfe angefertigt habe, und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Münster, den (Abgabedatum)

David Bujok