René Röpke und Ulrik Schroeder (Hrsg.): 21. Fachtagung Bildungstechnologien (DELFI),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 97

Automatisierte Bewertung und Feedback-Generierung für grafische Modellierungen und Diagramme mit FEEDI

Erik Morawetz[1], Nadine Hahm[1] und Andreas Thor[1]

Abstract: Dieser Beitrag präsentiert FEEDI (Feedback im Diagramm-Assessment), ein Web-basiertes System zur automatischen Bewertung und Feedback-Generierung für grafische Modellierungen und Diagramme. FEEDI verfolgt dabei einen generischen Ansatz, in dem es sowohl unterschiedliche Eingabeformate als auch Diagrammtypen prozessiert und Lehrenden die Möglichkeit gibt, Elemente ihrer Musterlösung einfach zu annotieren. Damit ermöglicht FEEDI ein effizientes E-Assessment insbesondere im MINT-Bereich, bei dem Diagramme wichtiger Bestandteil der Hochschullehre sind. Der Beitrag beschreibt die Graph-basierte Repräsentation der Diagramme sowie die Bewertung und Feedback-Generierung unter Verwendung von Graph-Matching. Darüber hinaus skizziert er die prototypische Entwicklung am Beispiel von Entity-Relationship-Diagrammen.

Keywords: E-Assessment, Feedback, Diagramm, Grafische Modellierung

1 Motivation

Grafische Modellierungen und die Erstellung von Diagrammen, wie z.B. UML-Diagramme, Schaltpläne oder Signaldiagramme, spielen eine wichtige Rolle in der Hochschullehre in MINT-Fächern. Sie sind oft Bestandteil von Übungs- und Prüfungsaufgaben, bei denen Studierende insbesondere anwendungsbezogene Kompetenzen nachweisen können. Für das Thema E-Assessment stellen solche Aufgabentypen jedoch eine Herausforderung dar, da die ihre automatisierte Bewertung sehr anspruchsvoll ist. Diagramme visualisieren in der Regel komplexe Zusammenhänge und es existieren meist verschiedene Varianten korrekter Lösungen. Darüber hinaus können Studierende an vielen Stellen kleinere Fehler machen, was eine aufwändige Bewertung mit Teilpunkten notwendig macht.

Es existieren bereits E-Assessment-Lösungen für spezifische Diagrammtypen, wie z.B. für Entity-Relationship- oder UML-Diagramme (u.a. [SG14; SM02]), die dann in der Regel auch mit einer eigenen Anwendung bzw. User Interface für die Erstellung und Bewertung der Diagramme einhergehen. Diese Ansätze können die Spezifika des Diagrammtyps effektiv abbilden und bieten eine sehr gute Unterstützung für die Lernenden, obwohl viele publizierte Arbeiten die technischen Details in den Vordergrund rücken und didaktische Aspekte vermissen lassen [Ul23]. Leider schränken Diagrammtyp-spezifische Ansätze auch die breite Nutzbarkeit ein. Lehrende und Studierende müssen mit einem speziellen

[1] Hochschule für Technik, Wirtschaft und Kultur Leipzig, {erik.morawetz, nadine.hahm, andreas.thor}@htwk-leipzig.de

doi: 10.18420/delfi2023-18

Werkzeug arbeiten und können gewohnte Anwendungen, wie z.B. PowerPoint, nicht nutzen. Darüber hinaus erfordert die Einführung eines neuen Diagrammtyps meist die Entwicklung einer neuen Anwendung *from scratch* und damit einen hohen Aufwand.

Dieser Beitrag präsentiert daher mit FEEDI (Feedback im Diagramm-Assessment) einen generischen Ansatz für ein flexibles E-Assessment von Diagrammen, bei dem sowohl unterschiedliche Eingabeformate als auch Diagrammtypen unterstützt werden. FEEDI kann für E-Assessment-Aufgaben eingesetzt werden, bei denen Studierende ein Diagramm als Lösung einreichen, und gibt automatisiert eine Bewertung sowie Feedback. Dazu transformiert es sowohl die Musterlösung (ML) als auch studentische Lösungen (SL) in eine generische Graphstruktur, aus der dann mittels Graph-Matching ein Mapping berechnet wird. Darüber hinaus können Lehrende ihre ML noch annotieren, um den konkreten Bewertungsmaßstab (Punkte für einzelne Elemente), etwaige alternative Lösungen (Varianten) sowie Feedbacktexte für Elemente zu hinterlegen, die z.B. angezeigt werden können, wenn diese Elemente in der SL nicht oder fehlerhaft vorkommen.

Durch FEEDI soll es Lehrenden möglichst einfach gemacht werden, Diagramme als ML zu erstellen, da sie auch bereits existierende Diagramme (z.B. in PowerPoint-Folien ihrer Lehrmaterialien) direkt nutzen können. Auch unterstützt die Architektur von FEEDI die zügige Integration neuer Diagrammtypen, um eine große Verbreitung in unterschiedlichen Fachdisziplinen zu erreichen.

2 FEEDI-Architektur

Abb. 1 illustriert die erweiterbare Architektur von FEEDI. Diagramme können in einem ersten Schritt in unterschiedlichen Formaten eingegeben werden. Dazu bietet FEEDI eine Reihe von **Format-Konvertern**, die jeweils ein bestimmtes Eingabeformat in ein generisches, JSON-basiertes Bildbeschreibungsformat transformieren. Der derzeitige Prototyp unterstützt mit PowerPoint und Diagrams.net zwei häufig zur Erstellung von Diagrammen genutzte Werkzeuge. Damit können insbesondere Lehrende z.B. in PowerPoint gezeichnete Diagramme sowohl in ihren Lehrmaterialien als auch ohne zusätzlichen Aufwand für das E-Assessment mit FEEDI verwenden. Der Format-Konverter für PNG-Bilder, z.B. als eingescannte Stiftzeichnung, ist derzeit noch in Entwicklung.

Alle Format-Konverter erzeugen eine **JSON-basierte Bildbeschreibung**, welche die relevanten Bildelemente, wie z.B. Rechtecke, Pfeile oder Text, in entsprechenden JSON-Objekten repräsentiert. Dazu werden die wesentlichen geometrischen Eigenschaften, u.a. Größe, Position und Rotation, als Attribute gespeichert. Für das E-Assessment irrelevante Information, wie z.B. die Schriftart, werden in diesem Schritt bereits herausgefiltert. Der zentrale Vorteil dieser *generischen* Bildbeschreibung liegt in der Unabhängigkeit sowohl vom Eingabeformat als auch vom Diagrammtyp, wie z.B. Entity-Relationship-Modell o-

der Signaldiagramm. Es ermöglicht dabei als Austauschformat eine flexible Erweiterbarkeit von FEEDI hinsichtlich neuer Eingabeformate sowie Diagrammtypen. Darüber hinaus können Musterlösungen später einfach im JSON-Format annotiert werden.

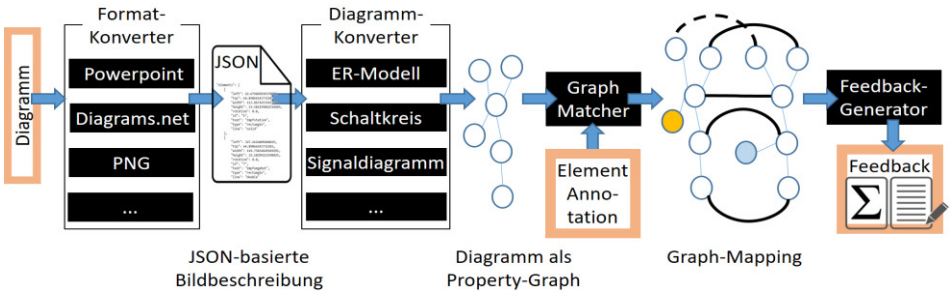


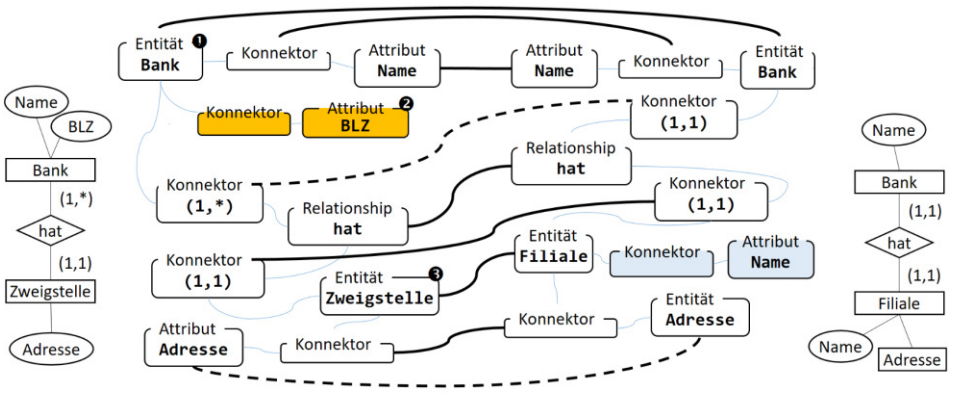
Abb. 1: Architektur und schematischer Workflow von FEEDI.

Im nächsten Schritt wird die generische Bildbeschreibung des Diagramms in einen **Property-Graphen** überführt, d.h. die Bildelemente werden in Knoten sowie Kanten inkl. Attribute (Properties) transformiert. Diese Transformation ist *spezifisch* für jeden Diagramm-Typ und wird von einem **Diagramm-Konverter** realisiert. Für den Diagrammtyp ER-Modell werden z.B. Rechtecke zu Knoten vom Typ `Entität` und erhalten ein entsprechendes Label – entsprechend dem Text innerhalb des Rechtecks. Abb. 2 zeigt die Transformation für ein Beispiel, das in Kapitel 3 näher diskutiert wird. Die generische Property-Graph-Struktur erlaubt es, die weiteren Verarbeitungsschritte (Graph-Matching sowie Feedback-Generierung) zu vereinheitlichen.

Die bisher skizzierten Verarbeitungsschritte, d.h. die sukzessive Konvertierung von Diagrammen in einen Property-Graphen, werden im Rahmen des E-Assessments sowohl für die von Lehrenden spezifizierte Musterlösung (ML) als auch für eingereichte studentische Lösung (SL) durchgeführt. Beide Graphen sind im Folgenden Eingabe für das **Graph-Matching**, bei dem eine bestmögliche Abbildung zwischen den beiden Graphen gesucht wird. Das Matching wird dabei durch hinzugefügte Kanten zwischen den Knoten aus ML und SL repräsentiert, so dass ein zusammenhängender Graph resultiert. Zusätzliche Eingabe für den Graph-Matcher ist auch eine **Element-Annotation**, mit der Lehrende etwaige Varianten der Lösung spezifizieren, sowie die Bepunktung und Feedback-Texte für die Elemente angeben. Letzteres wird vom **Feedback-Generator** verwendet, um sowohl eine Bewertung (Punktzahl) als auch ein Feedback bzgl. Fehler in der SL zu geben.

3 Automatische Bewertung und Feedback-Generierung

Nachdem alle Diagramme, d.h. sowohl die Musterlösung (ML) als auch die studentische Lösung (SL), jeweils in einen Property-Graphen transformiert wurden, wird der automatische Abgleich auf ein Graph-Matching-Problem reduziert. Abb. 2 zeigt ein einfaches



❶ 2 Punkte, "Es gibt mehrere Banken und sie sind eindeutig identifizierbar, daher eine Entität."
❷ 1 Punkt, "Das Attribut BLZ ist relevant zur Beschreibung einer Bank.", `changeValue` (Label, [Bankleitzahl])
❸ 2 Punkte, "Zweigstelle hat eigene Attribute, daher eine Entität.", `changeValue` (Label, [Filiale, Niederlassung, Standort])

Abb. 2: Beispiel für eine Musterlösung (links) und eine zugehörige studentische Lösung (rechts), jeweils dargestellt als ER-Diagramm (außen) und Property-Graph (innen). Zusätzlich ist das Mapping zwischen den Graphen dargestellt sowie die Annotation (Punktzahl, Feedbacktext, Variation) für drei ausgewählte Elemente der Musterlösung. Die farblich hinterlegten Graph-Knoten haben keinen Mapping-Partner.

Beispiel. Auf der linken Seite ist ein vereinfachtes[2] Entity-Relationship-Diagramm als ML dargestellt, daneben die interne Repräsentation als Property-Graph. Wie in Kapitel 2 ausgeführt, ist die Konvertierung spezifisch für den Diagrammtyp. Für ER-Diagramme werden u.a. Rechtecke (Entitäten), Rauten (Beziehungen), Ellipsen (Attribute) und die verbindenden Striche (Konnektoren) als relevante Bildelemente in Knoten überführt und mit Kanten verbunden. Informationen zur bildlichen Position werden ignoriert, da sie für die Modellierung keine Rolle spielen – im Gegensatz etwa zu Signaldiagrammen, wo die Positionierung z.B. von Datenpunkten einen Informationsgehalt hat. Für jeden Graph-Knoten (abgerundetes Rechteck) sind in Abb. 2 als Properties der `type` (oben) sowie – sofern vorhanden – das `label` im Rechteck notiert. Weitere Properties, z.B. ob es sich bei einem Attribut um ein Primärattribut handelt, können entsprechend ergänzt werden. Auf der rechten Seite sind Diagramm und Graph einer SL dargestellt.

Das **Graph-Matching** erhält nun als Input die zwei Property-Graphen ML und SL und liefert Korrespondenzen (Zuordnungen) zwischen den Knoten der beiden Graphen. Korrespondierende Knoten sind dabei im einfachsten Fall äquivalent, d.h. besitzen die gleichen Werte für alle Properties. Häufig gibt es *ähnliche* Knotenpaare, bei denen Abweichungen in den Property-Werten und benachbarten Knoten auftreten. Zur Quantifizierung wird eine Ähnlichkeitsfunktion verwendet, welche die Knoten bzgl. der Property-Werte

[2] Aus Gründen der besseren Lesbarkeit wurde auf die Angabe der Primärschlüssel und Datentypen verzichtet.

und ggf. der benachbarten Knoten miteinander vergleicht und einen Ähnlichkeitswert zwischen 0 (maximale Unähnlichkeit) und 1 (Gleichheit) ermittelt. Für die Entität Bank würde sich z.B. ein Ähnlichkeitswert von 1 ergeben; für den Konnektor zwischen Bank und hat z.B. ein Wert von 0.7, weil es eine kleine Abweichung im label gibt ((1,*) vs. (1,1)).

Das Ergebnis eines Graph-Matching wird als **Mapping** M dargestellt, das alle Knotenpaare (a, b) mit $a \in$ ML und $b \in$ SL enthält, für die eine Korrespondenz mit einer Mindestähnlichkeit $sim \geq sim_{min}$ existiert. Das Mapping kann als bipartiter Graph interpretiert werden, d.h. Knoten von ML und SL werden durch gewichtete Kanten (Gewicht=Ähnlichkeit) verbunden. In Abb. 2 sind die Mapping-Kanten den beiden Graphen ML und SL hinzugefügt worden, so dass ein gemeinsamer (zusammenhängender) Graph entsteht. Zur vereinfachten Darstellung entsprechen durchgezogene Kanten einem Gewicht von 1, die gestrichelten einem Gewicht kleiner als 1. Darüber hinaus bezeichnet $N_{\mathrm{ML}} = \{a | \nexists b : (a, b) \in M\}$ alle Knoten, die keinen Match-Partner in SL gefunden haben, d.h. diese Diagrammelemente der ML konnten der SL nicht zugeordnet werden (gelb hinterlegt in Abb. 2). Analog ist N_{SL} die Menge der Knoten, die keine Korrespondenz zu ML haben, d.h. Elemente der SL, die nicht in der ML auftreten (hellblau).

Aus den so erhaltenen Mengen M, N_{SL} und N_{ML} lassen sich nun gemeinsam mit den Annotationen der ML (siehe unten) eine Punktzahl und spezifisches Feedback ableiten. Für die **Punktzahl** werden nur diejenigen Knoten der ML, die in M enthalten sind, d.h. $\{a | \exists b : (a, b) \in M\}$, betrachtet und die annotierte Punktzahl jeweils mit der Ähnlichkeit sim multipliziert und aufsummiert. Abweichungen bzw. Ungenauigkeiten, die sich in einem Ähnlichkeitswert kleiner als 1 widerspiegeln, entspricht das einer Teilpunktbewertung. **Textuelles Feedback** kann zunächst für alle Knoten aus N_{ML} produziert werden, da diese Elemente in der SL erwartet, aber nicht gefunden wurden. Der Annotationstext dieser Elemente kann als allgemeines Feedback à la „Folgende Elemente fehlen“ ausgegeben werden. Darüber hinaus kann für die zugeordneten Knoten in SL, die allerdings einen Ähnlichkeitswert $sim < 1$ haben, ein spezifisches Feedback gegeben werden. Der Feedbacktext des zugehörigen Elements in der ML kann als Hinweis für eine Korrektur an die SL ergänzt werden. Für Knoten aus N_{SL} kann es kein textuelles Feedback geben, da die Annotationen nur für die ML vorliegen. Hier bietet FEEDI ein **grafisches Feedback**, d.h. diese Elemente werden in der SL farblich hinterlegt, um anzuzeigen, dass sie in keinem Bezug zur ML stehen. Auch die ML selbst kann als Feedback angezeigt werden.

Wie bereits erwähnt, können Lehrende durch die Annotation ihrer ML Einfluss auf die Bewertung und Feedback-Generierung nehmen (siehe auch Abb. 1). Über ein einfaches GUI können Lehrende die Graph-Darstellung ihrer ML erweitern, in dem sie den Elementen zusätzliche Feedack-Properties geben. In Abb. 2 sind dazu Punktzahl und Feedbacktext exemplarisch für die Entitäten Bank und Zweigstelle sowie für das Attribut BLZ dargestellt. Sollten mehrere Elemente des Diagramms in der Feedbackgenerierung als geschlossene Komponente betrachtet werden, so können diese in Feedbackgruppen zusammengefasst werden. Darüber hinaus besteht die Möglichkeit, **Variationen der Musterlösung** zu spezifizieren, da häufig nicht nur eine Lösung als korrekt angesehen wird. Auch hier nutzt FEEDI die generische Graph-Repräsentation und stellt Variationen als Graph-

Transformationen dar. Ausgehend von der ML können Transformationen wie das Hinzufügen von Knoten (`addNode`, z.B. für optionale Inhalte) oder das Verändern von Werten (`changeValue`, z.B. für synonyme Begriffe) auf den Graphen angewendet werden. Im Beispiel von Abb. 2 gibt es für BLZ und Zweigstelle die Möglichkeit, synonyme Begriffe zu verwenden. Synonyme können in der ML angegeben werden, während kleine Schreibfehler durch Ähnlichkeitsschwellenwerte abgefangen werden. Beim Graph-Matching ermittelt FEEDI dann alle möglichen Variationen mit Hilfe der Graph-Transformationen auf der ML und bestimmt das *beste Mapping*, d.h. dasjenige Mapping, welche die höchste Punktzahl liefert. Sollten mehrere unterschiedliche Lösungen existieren, so können mehrere ML hinterlegt werden. FEEDI wählt dann die Option mit der höchsten Ähnlichkeit.

Die prototypische Implementation von FEEDI beinhaltet derzeit mehrere Format- und Diagramm-Konverter sowie einen Graph-Matching-Algorithmus. Für die Annotation der ML können Punkte, Feedbacktext und einfache Variationen spezifiziert werden. Der Einsatz komplexerer Graph-Transformationen ist gerade in Entwicklung. Die einzelnen Komponenten der FEEDI-Architektur sind in Python implementiert und werden als REST-Webservice publiziert, so dass sie flexibel kombiniert werden können.

4 Zusammenfassung und Ausblick

Dieser Beitrag präsentierte mit FEEDI ein E-Assessment-System zur automatischen Bewertung und Feedback-Generierung für grafische Modellierungen und Diagramme. Es soll insbesondere Studierende in MINT-Fächern unterstützen, da die Erstellung solcher Diagramme eine wichtige zu erlernende Kompetenz darstellt. Durch eine generische Architektur unterstützt FEEDI sowohl unterschiedliche Eingabeformate als auch Diagrammtypen. Zentrale Idee von FEEDI ist dabei die Überführung der Diagramme in eine einheitliche Graph-Struktur, auf der ein Graph-Matching angewendet wird, um Gemeinsamkeiten und Unterschiede zwischen Musterlösung und studentischen Lösungen zu identifizieren. Zukünftige Arbeiten umfassen die Fertigstellung der Implementation sowie den praktischen Einsatz von FEEDI in der Lehre.

Literaturverzeichnis

[SG14] Striewe, M.; Goedicke, M.: Automated Assessment of UML Activity Diagrams. In: Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education. 2014.

[SM02] Suraweera, P.; Mitrovic, A.: KERMIT: A Constraint-Based Tutor for Database Modeling. In: Intelligent Tutoring Systems. 2002.

[Ul23] Ullrich, M. et al.: Automated Assessment of Conceptual Models in Education. Enterprise Modelling and Information Systems Architectures (EMISAJ) 18/2, 2023.