

## Ansätze zur automatischen Generierung von Aufgaben zum Modellverstehen am Beispiel von UML-Sequenzdiagrammen

René Ponto, Tobias Schüler, Michael Striewe<sup>1</sup>

**Abstract:** Um das Verstehen von Diagrammen und Modellen trainieren zu können, ist eine hinreichend große Menge verschiedener Aufgaben notwendig. Die manuelle Erstellung von großen Mengen ähnlicher Aufgaben ist jedoch zeitaufwändig und wenig attraktiv. Dieser Beitrag stellt als Alternative zwei prototypisch implementierte, erweiterbare Generatoren vor, mit denen UML-Diagramme und darauf basierende Aufgaben auf Basis einiger weniger, manueller Vorgaben automatisch erstellt werden können. Die beiden Prototypen zeigen am Beispiel von UML-Sequenzdiagrammen, dass die gewählten Konzepte technisch umsetzbar und in einer hinreichenden Breite einsetzbar sind.

**Keywords:** Modellierung; Modellverstehen; Aufgabengenerierung; Übungsaufgaben

### 1 Einleitung

In der Hochschullehre der Informatik spielt die Modellierung in vielfältiger Weise eine zentrale Rolle im Curriculum [Gl08]. Sowohl auf der Ebene abstrakter Modellkonzepte als auch auf der Ebene konkreter Modellierungssprachen werden in Einführungsveranstaltungen häufig Kompetenzen vermittelt, die wichtige Voraussetzungen für die erfolgreiche Teilnahme an fortgeschrittenen Lehrveranstaltungen sind [Co03] und die auch im nicht-akademischen Berufsleben von hoher Relevanz sind [Da06].

Die Lehre zur Modellierung in einem konkreten Anwendungsbereich muss sich dabei mit mindestens vier disjunkten Aspekten auseinandersetzen: (1) Das konzeptionelle Verständnis für das Subjekt der Modellierung; (2) die Kenntnis der Syntax und Semantik einer konkreten Modellierungssprache, mit der Diagramme erstellt werden können, die das Modell angemessen repräsentieren; (3) die praktische Erstellung eines syntaktisch korrekten Diagramms, das eine geeignete Sicht auf das Modell bietet und dessen Semantik das Subjekt der Modellierung korrekt wiedergibt und (4) die korrekte Interpretation eines gegebenen Diagramms, um Aussagen über das zugrundeliegende Modell und letztlich das modellierte Subjekt zu treffen. Insbesondere der dritte und der vierte Aspekt stehen in unmittelbarer Beziehung zu gängigen Kompetenzmodellen zur informatischen Modellierung und zum Systemverstehen (vgl. [Li13]).

---

<sup>1</sup> Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Gerlingstraße 16, 45127 Essen, Deutschland, michael.striewe@paluno.uni-due.de

Betrachtet man exemplarisch die Modellierung eines Algorithmus', der imperativ und objekt-orientiert implementiert werden soll, lassen sich die vier Aspekte wie folgt illustrieren: (1) Es muss das konzeptionelle Verständnis dafür geschaffen werden, dass Algorithmen in imperativen Programmiersprachen schrittweise entsprechend der Reihenfolge der Anweisungen im Programmcode ausgeführt werden, wobei Schleifen zu Wiederholungen und Fallunterscheidungen zu Auslassungen führen können. Ferner muss das konzeptionelle Verständnis dafür geschaffen werden, dass Objekte über Methodenaufrufe und deren Rückgaben miteinander kommunizieren und dass dadurch ein Aufruf-Stack entstehen kann, bei dem das oberste Element die Kontrolle hat und entweder weitere Aufrufe tätigen oder durch eine Rückgabe die Kontrolle wieder abgeben kann. (2) Man kann die Syntax und Semantik von UML-Sequenzdiagrammen nutzen, um den Ablauf eines Algorithmus auf der Ebene der Methodenaufrufe diagrammatisch zu beschreiben. Dazu müssen neben den Grundkonzepten (Objekte, Lebenslinie, synchrone und asynchrone Nachrichten) auch Fragmente bekannt sein, um Schleifen und Fallunterscheidungen zu modellieren. (3) Eine praktische Modellierungsaufgabe kann auf Basis dieser Kenntnisse einen Algorithmus in Form des Programmcodes oder einer hinreichend präzisen, textuellen Beschreibung vorgeben und die Erstellung eines UML-Sequenzdiagramms (ggf. unter Berücksichtigung bestimmter Aufrufparameter für den Algorithmus) erfordern. (4) Eine Aufgabe zum Modellverstehen kann ein UML-Sequenzdiagramm vorgeben und inhaltliche Fragen zum Ablauf des Algorithmus' oder zur mutmaßlichen Struktur des objekt-orientierten Programms stellen.

Für alle vier Aspekte können geeignete Übungsaufgaben gestellt werden, die zu einem gewissen Grad auch automatisiert generiert oder bewertet werden können. Insbesondere der dritte Aspekt ist mit verschiedenen Ansätzen zur Generierung (z. B. [Sh16]) und insbesondere zur Bewertung (z. B. [ASI07, CLP18, Le06, Sc12, SG14, TSW08]) Gegenstand umfangreicher Forschungen. Die automatische Bewertung in den anderen drei Aspekten kann zudem zumindest teilweise mit Standardverfahren wie Multiple-Choice oder Lückentexten durchgeführt werden. Im Fokus des vorliegenden Beitrags steht daher die automatisierte Generierung von Aufgaben für den vierten Aspekt, also das Modellverstehen. Auch dazu gibt es bereits Ansätze, beispielsweise für Aufgaben zum Matching von Klassen- und Objektdiagrammen [KSV19]. Es können auch weitere Ansätze zur automatischen Erstellung von Lösungen für Modellierungsaufgaben [MB08] für diese Aufgabe genutzt werden. Dass das Modellverstehen kein einfacher Prozess ist und zahlreichen Einflussfaktoren unterliegt, wird durch zahlreiche Forschungen belegt ([HFL12, Fi17]) und untermauert damit die These, dass eine große Menge von Übungsaufgaben in diesem Bereich hilfreich sein kann.

Im vorliegenden Beitrag wird der Aspekt der Aufgabengenerierung wie folgt näher beleuchtet: Abschnitt 2 differenziert die möglichen Aufgaben zum Modellverstehen genauer hinsichtlich ihrer Komplexität und Anforderungen an die automatische Aufgabengenerierung und Bewertung. Abschnitt 3 stellt ein Konzept zur template-basierten, automatischen Generierung von Aufgaben vor, welches bisher als Prototyp realisiert wurde. Abschnitt 4 geht auf die automatische Generierung von Diagrammen ein, die als Grundlage für Fragen zum Modellverstehen dienen können. Alle drei Abschnitte verwenden wie die Einleitung

UML-Sequenzdiagramme als durchgehendes Beispiel, sind im Kern ihrer Betrachtung aber nicht auf diesen Diagrammtyp beschränkt. Abschnitt 5 zieht ein Fazit und schließt den Beitrag mit einem Ausblick auf weiteren Forschungsbedarf.

## 2 Aufgaben zum Modellverstehen

Der Prozess des Modellverstehens kann auf mehrere, konsekutive Schritte heruntergebrochen werden. Zunächst einmal müssen relevante syntaktische Elemente in einem Diagramm überhaupt erkannt werden. Man kann annehmen, dass dieser Schritt vor allem eine Zeitfrage ist, die wesentlich von der Größe und Übersichtlichkeit des betrachteten Diagramms sowie der Erfahrung der betrachtenden Person abhängt. So kann es bei Anfängern schon zu Schwierigkeiten in diesem Schritt kommen, wenn beispielsweise überprüft werden soll, ob in einem UML-Sequenzdiagramm zwei Objekte A und B enthalten sind, die mindestens eine Nachricht austauschen. In einem zweiten Schritt können gefundene Elemente dann im Sinne der Diagrammsemantik interpretiert werden. Damit kann beispielsweise die Frage geklärt werden, welches Objekt in einem UML-Sequenzdiagramm den zeitlich gesehen letzten Methodenaufwurf tätigt oder welche Nachricht innerhalb einer Schleife verschickt wird. Schließlich können in einem letzten Schritt diese Erkenntnisse auf das Modell und von dort auf das eigentliche Subjekt übertragen werden. Hier kann dann beispielsweise die Frage beantwortet werden, ob eine bestimmte Methode einer bestimmten Klasse rekursiv arbeitet oder ob basierend auf den Diagramminhalten Klasse A ausschließlich von Klasse B genutzt wird.

Praktische Beispiele für entsprechende Übungsaufgaben aus der Hochschullehre sind im Internet verfügbar und über Suchmaschinen leicht auffindbar (z. B. [Mo, Ci] für Aufgaben zum Verständnis von UML-Sequenzdiagrammen). Tabelle 1 zeigt einige Beispiele für mögliche Aufgaben zu UML-Sequenzdiagrammen und kategorisiert diese nach dem Fragetyp. Tiefergehende Fragen zum Modellverständnis sind insbesondere durch die Kombination verschiedener Diagrammtypen möglich, indem beispielsweise fehlende Informationen in einem Diagramm durch Informationen aus dem anderen Diagramm vervollständigt werden sollen. Im vorliegenden Beitrag werden diese Fragentypen jedoch nicht weiter vertieft und es werden lediglich Fragen behandelt, die ausschließlich anhand von Sequenzdiagrammen bearbeitet werden können. Die betrachteten Konzepte sind dabei jedoch nicht auf Sequenzdiagramme beschränkt, sondern können auch auf andere Diagrammtypen übertragen werden.

Bei genauerer Betrachtung der Aufgaben zeigt sich, dass die meisten von ihnen mutmaßlich nur eine kurze Bearbeitungszeit benötigen. Lediglich die allerletzte Aufgabe, in der Programmcode erstellt werden soll, dürfte eine deutlich längere Bearbeitungszeit erfordern. Allen Aufgaben ist zudem gemein, dass sie sich kaum für eine wiederholte Bearbeitung eignen, um den Lerneffekt zu steigern und gelernte Kompetenzen zu vertiefen. Sie unterscheiden sich damit deutlich von Modellierungsaufgaben, bei denen eine wiederholte

<b>Fragetyp</b>	<b>Aufgabe</b>
Kategorie „Syntax“	
Ja/Nein	Gibt es im Diagramm eine Nachricht von Objekt A zu Objekt B mit dem Inhalt C?
Multiple Choice	Welche der folgenden Elemente kommen im Diagramm vor?
Single/Multiple Choice	Welche der im Diagramm markierten Stellen markiert die Aktivierung eines Prozesses?
Single Choice	Wie nennt man das im Diagramm markierte Element?
Single/Multiple Choice	Welche der im Diagramm markierten Nachrichten ist synchron?
Freitext	Geben Sie den Inhalt einer synchronen Nachricht aus dem Diagramm an.
Freitext	Gibt es im Diagramm einen Fehler oder ein fehlendes Element? Falls ja, benennen Sie den Fehler.
Kategorie „Diagramm“	
Multiple Choice	Welche Objekte kommunizieren miteinander, indem sie mindestens eine Nachricht an den jeweiligen Partner senden oder von ihm empfangen?
Freitext oder Single Choice	Welches Objekt ist der Empfänger/Sender der Nachricht X?
Ja/Nein	Ist die Nachricht X die letzte in der zeitlichen Abfolge?
Single Choice	An welcher Stelle in der zeitlichen Abfolge steht die Nachricht X?
Freitext	Wie viele Nachrichten werden maximal/mindestens innerhalb des XY-Fragments verschickt?
Kategorie „Modell“	
Single/Multiple Choice	Welches der im Folgenden angegebenen Klassendiagramme passt zum Sequenzdiagramm?
Single Choice	In welchem der beiden folgenden Diagramme ist der Kontrollfluss zentraler organisiert?
Freitext	Schreiben Sie ein Stück Programmcode, welches das Programmverhalten realisiert, das im Diagramm gegeben ist.

Tab. 1: Beispielaufgaben zum Modellverstehen im Kontext von UML-Sequenzdiagrammen.

Bearbeitung schon alleine deshalb interessant sein kann, weil es mehrere Wege geben kann, denselben Sachverhalt in einem Diagramm auszudrücken.

Wenn eine wiederholte Bearbeitung derselben Aufgabe nicht sinnvoll ist, aber gleichzeitig trotzdem eine große Menge von Übungsmöglichkeiten bereitgestellt werden soll, die nicht in wenigen Minuten zu bearbeiten ist, dann ist dies nur über eine hinreichend große Menge von unterschiedlichen Aufgaben möglich. Die manuelle Erstellung solcher Aufgaben ist jedoch mühsam und bei der Vervielfältigung und Variation von Aufgaben in manueller „Massenproduktion“ kann es leicht zu Flüchtigkeitsfehlern kommen. Es bietet sich daher an,

bei der Erstellung von Aufgaben auf Methoden der automatischen Aufgabengenerierung zurückzugreifen, um schnell eine große Menge gleichartiger Aufgaben zu erzeugen.

### 3 Template-basierte Generierung von Aufgaben

Ein gängiger Ansatz zur automatischen Generierung von Aufgaben basiert auf Aufgabentemplates [Gi13]. Diese bestehen aus einem Aufgabenstamm, der den Aufgabentext sowie Platzhalter enthält. Für jeden dieser Platzhalter gibt es eine Menge von konkreten Werten, die für ihn in den Aufgabentext eingesetzt werden können. Die Platzhalter können dabei voneinander unabhängig oder abhängig sein. Bei unabhängigen Platzhaltern sind beliebige Kombinationen der konkreten Werte möglich. Bei abhängigen Platzhaltern gibt es dagegen nur eine Menge von Tupeln, die gültige Belegungen angeben und eine Teilmenge aller möglichen Kombinationen darstellen. Das Verfahren wurde ursprünglich nur für Multiple-Choice-Aufgaben entwickelt, aber es spricht aus technischer Sicht nichts dagegen, es auch für andere Aufgabentypen einzusetzen.

Für die Generierung von Fragen zum Modellverständnis können die Platzhalter in einem Generator allerdings nicht mit festen Werten hinterlegt werden, sondern es müssen Abfragen formuliert werden, mit denen die konkreten Werte aus einem gegebenen Diagramm herausgelesen werden können. Das Vorgehen kann anhand der ersten Frage aus Tabelle 1 wie folgt beschrieben werden, wobei die Frage bereits die Platzhalter A, B und C enthält:

1. Zunächst muss überprüft werden, ob das verwendete Diagramm überhaupt die notwendigen Eigenschaften erfüllt, um die gewünschte Frage zu stellen. Im gewählten Beispiel ist es dazu notwendig zu überprüfen, ob das Diagramm mindestens zwei benannte Objekte enthält, zwischen denen mindestens eine benannte Nachricht verschickt wird. Die meisten Sequenzdiagramme dürften diese Eigenschaft erfüllen, aber es sind auch Beispiele denkbar, in denen es nur anonyme Objekte gibt, die möglicherweise sogar alle derselben Klasse angehören.
2. Sind alle Voraussetzungen erfüllt, kann mit der Sammlung möglicher Werte begonnen werden. Im gewählten Beispiel müssen dazu alle benannten Objekte und alle Nachrichten durchlaufen werden. Für jede Kombination muss dann geprüft werden, ob die Frage mit Ja oder Nein zu beantworten ist. Gegebenenfalls können als zusätzliche Distraktoren in diesem Schritt auch weitere Werte eingefügt werden, die nicht dem Diagramm entnommen sind.
3. Nach diesen beiden Schritten kann die entstandene Aufgabe bei Bedarf manuell auf ihre Sinnhaftigkeit überprüft werden, indem alle erzeugten Belegungen geprüft und einzelne Tupel ggf. manuell ausgeschlossen werden.
4. Anschließend werden alle verbleibenden Tupel dazu verwendet, um konkrete Instanzen der Aufgabe zu erzeugen.

Die ersten beiden Schritte sind deterministisch. Sie beginnen mit der manuellen Eingabe eines Diagramms und es schließt sich die manuelle Inspektion der generierten Werte an. Es bietet sich daher an, diesen Schritt für jedes eingegebene Diagramm nur einmalig und in einem separaten Editor-Werkzeug durchzuführen. Für den vierten Schritt kann die konkrete Erzeugung dann ebenfalls in diesem Werkzeug ablaufen, das dann eine größere Menge von Instanzen exportieren muss. Alternativ exportiert es lediglich eine Aufgabenbeschreibung zum Import in ein geeignetes Übungssystem, welches die Erzeugung von Instanzen auf Basis der ausgewählten Tupel dann selbständig durchführt.

Eine prototypische Umsetzung dieses Konzepts wurde im Rahmen eines studentischen Projektes mit einem Java-basierten Editor und einem Export für das E-Assessment-System JACK [St16] realisiert. Der Editor liest UML-Modelle in Form von XMI-Dateien ein<sup>2</sup> und analysiert anschließend die Anwendbarkeit mehrerer Templates auf das jeweilige Diagramm. Der Editor verfügt derzeit über insgesamt 34 Templates zu Aktivitätsdiagrammen, Klassendiagrammen, Sequenzdiagrammen und Zustandsdiagrammen. Diese decken derzeit vor allem den Bereich „Syntax“ und „Diagramm“ ab und sollen in Zukunft noch weiter im Bereich „Modell“ ausgebaut werden.

Eine erste Erprobung der automatisch erzeugten Aufgaben erfolgt im derzeit laufenden Wintersemester 2019/2020. Dazu wurden zu je einem Diagramm jeden Typs zwei bis fünf Aufgaben automatisch erzeugt, so dass insgesamt 15 der 34 verfügbaren Templates zum Einsatz kommen. Die Aufgaben wurden den Studierenden ergänzend zum regulären Übungsbetrieb einer Lehrveranstaltung (Bachelor, 3. Fachsemester) zur Verfügung gestellt, in der die vier genannten Diagrammtypen zum Lehrstoff gehören. Die Studierenden wurden dabei gebeten, einen Erhebungsbogen auszufüllen, in dem sie die Aufgaben hinsichtlich verschiedener Aspekte auf Likert-Skalen bewerten sowie in einem Freitextfeld Kommentare angeben können.

## 4 Parametrisierte Generierung von Diagrammen

Die automatische Generierung von Aufgaben auf Basis eines gegebenen Diagramms reduziert den Aufwand für die Erstellung von Aufgaben erheblich, aber trotzdem nur teilweise. Da es nicht immer sinnvoll ist, ein möglichst großes, vielfältiges Diagramm in einer Aufgabe zu verwenden, ist es insbesondere notwendig, viele kleine Diagramme zur Verfügung zu haben. Diese sollten in ihrer Gesamtheit alle relevanten Kombinationen von Diagrammelementen enthalten, so dass alle gewünschten Fragen auch tatsächlich gestellt werden können. Der Zugriff auf Online-Repositoryn mit realen Diagrammen (z. B. [KC13, Ge]) ist dabei nicht immer zielführend, da die dort enthaltenen Diagramme in der Regel unter anderen Gesichtspunkten ausgewählt oder erstellt wurden und sich eine Verschlagwortung nicht an den Bedürfnissen der Aufgabengenerierung orientiert. Ein

---

<sup>2</sup> Um das Problem der automatische Erzeugung von Diagrammlayouts auszuklammern, muss zusätzliche eine Bilddatei mit dem gewünschten Diagramm eingelesen werden. Diese wird jedoch nicht weiter analysiert, sondern lediglich für die Anzeige in der Aufgabe verwendet.

alternativer Lösungsansatz besteht darin, auch die Diagramme aufgrund von vorgegebenen Parametern automatisiert so zu erzeugen, dass sie die gewünschten Eigenschaften erfüllen.

Für das Beispiel der UML-Sequenzdiagramme können die folgenden Anforderungen an eine Parametrisierung gestellt werden:

- **Kommunikationspartner:** Die Menge der im Diagramm enthaltenen Objekte soll festgelegt werden können. Als weitergehende Option soll auch die genaue Benennung der Objekte festgelegt werden können.
- **Nachrichten:** Die Menge der im Diagramm enthaltenen Nachrichten soll festgelegt werden können. Als weitergehende Option soll auch eine Vorgabe von Inhalten oder sonstigen Merkmalen der Nachrichten möglich sein. Sofern die Benennung von Objekten festgelegt wurde, soll auch der Nachrichtenaustausch zwischen bestimmten Objekten festgelegt werden können. Schließlich soll es auch möglich sein, die Nachrichten „anonym“ durchnummerieren, um Aufgaben zu vereinfachen, die sich nur auf die Syntax oder Position von Nachrichten beziehen und nicht auf deren Inhalt.
- **Fragmente:** Es soll festgelegt werden können, welche Fragmente im Diagramm auftauchen dürfen oder müssen. Sofern bestimmte Nachrichten zwischen Objekten festgelegt wurden, soll es auch möglich sein, diese bestimmten Fragmenten zuzuordnen.

Im Rahmen dieser Anforderungen ist es möglich, ein Diagramm komplett zufällig generieren zu lassen, indem keine Vorgaben zu Kommunikationspartnern und Nachrichten gemacht werden und alle Fragmente erlaubt (aber keines zwingend gefordert) werden. Ebenso ist es möglich, ein komplett festgelegtes Diagramm erzeugen zu lassen, indem alle Möglichkeiten der Vorgabe vollständig ausgenutzt werden. Zu beachten ist, dass der Generator im Rahmen der genannten Vorgaben nicht in der Lage sein kann, spezifische Objektamen oder Nachrichteninhalte zu erzeugen, die einer vorgegebenen Modellsemantik folgen. Alle Labels müssen stattdessen aus einer generischen, universell verwendbaren Menge zufällig gezogen werden, sofern keine weiteren Anforderungen an die Parametrisierung festgelegt werden, die sich auf die Einhaltung einer Modellsemantik oder zumindest die Konformität der Bezeichnungen zu einer bestimmten fachlichen Domäne beziehen.

Auch für dieses Konzept wurde eine prototypische Umsetzung im Rahmen eines studentischen Projektes realisiert. Der Java-basierte Generator stellt dazu eine Benutzerschnittstelle bereit (siehe Abbildung 1), in der schrittweise die oben genannten Einstellung für die Erzeugung eines Sequenzdiagramms gemacht werden können. Anschließend wird ein passendes Diagramm generiert und in Form einer Grafikdatei und eines passenden Modells im XMI-Format zum Export zur Verfügung gestellt, so dass die Ausgabe nahtlos als Eingabe für den Aufgabengenerator aus Abschnitt 3 verwendet werden kann. Das oben angesprochene Problem der Benennung von Objekten und Nachrichten wurde dabei so gelöst, dass Objekte im Stile klassischer Aufgaben der Nachrichtentheorie („Alice-und-Bob-Aufgaben“)

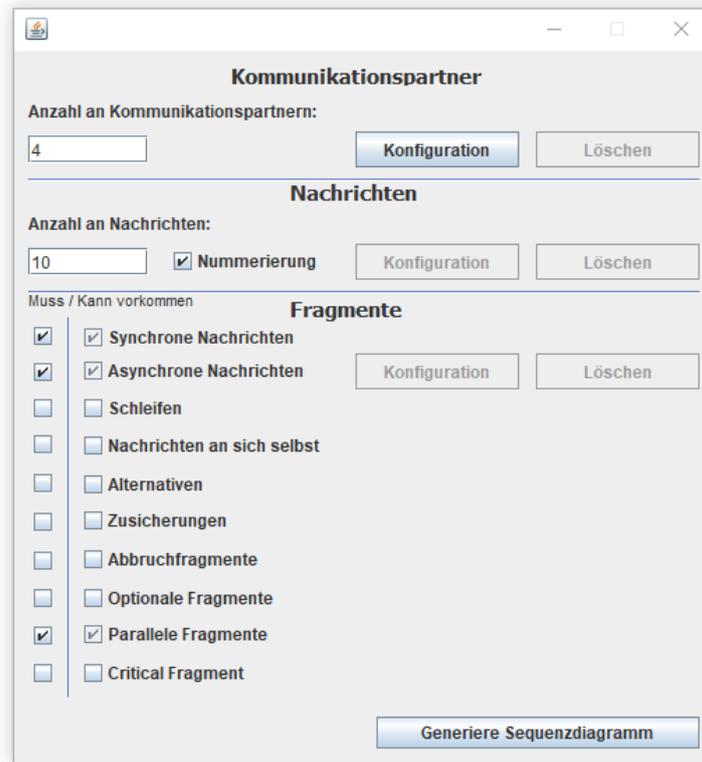


Abb. 1: Benutzeroberfläche des Diagrammgenerators mit einigen exemplarisch ausgewählten Einstellungen.

mit Personennamen versehen werden und Nachrichten generische Inhalte haben, die den Charakter der Nachricht wiedergeben.

Als besondere Herausforderung erwies sich bei der Entwicklung, eine Java-Bibliothek zu finden, die eine API zum Erstellen von Grafiken zu einem Sequenzdiagramm zur Verfügung stellt. Die bisher gefundene Lösung ist noch nicht optimal und soll im Idealfall in Zukunft durch eine bessere Bibliothek ersetzt werden, die mehr Optionen bei der Erzeugung von Objekten und Nachrichten und eine präzisere Kontrolle über die Aktivierungslinien bietet. Durch den Export im XMI-Format steht allerdings schon jetzt die Möglichkeit offen, das exportierte Modell manuell in einem UML-Werkzeug zu öffnen und ein geeignetes Diagramm auf diesem Wege zu erstellen. Eine automatische Kopplung mit dem Aufgabengenerator aus Abschnitt 3 ist dadurch dann allerdings nicht möglich. Der Prototyp ist bisher auf die Generierung von Sequenzdiagrammen beschränkt. Ein analoges Vorgehen für andere Diagrammtypen ist möglich, wirft aber noch stärker die Frage der automatischen Erzeugung

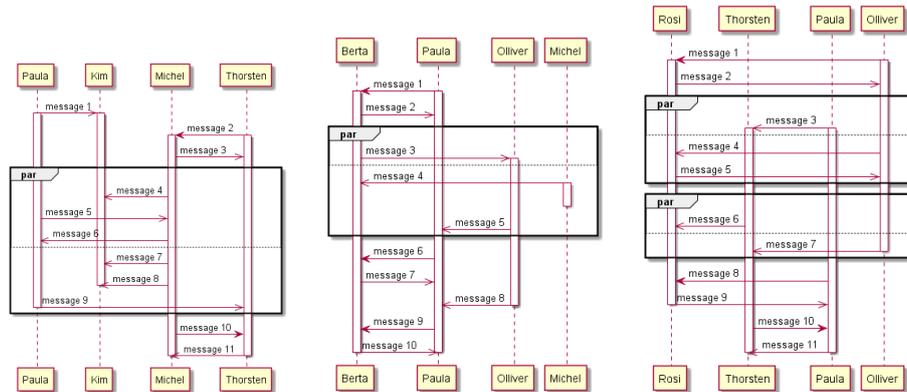


Abb. 2: Drei Beispiele für Sequenzdiagramme, die automatisch auf Basis der Einstellung aus Abbildung 1 erzeugt wurden.

von Grafiken auf, da beispielsweise bei Klassendiagrammen die Berechnung eines guten Layouts noch aufwendiger ist als bei Sequenzdiagrammen.

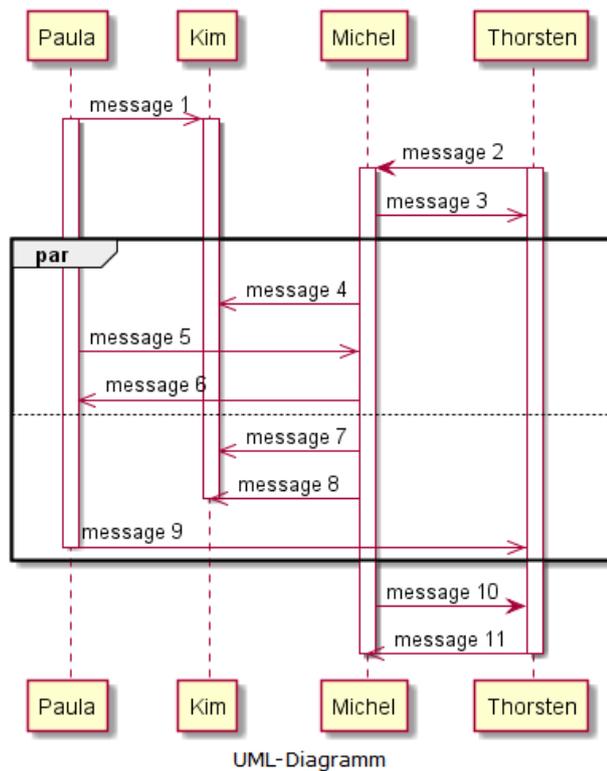
In Abbildung 2 sind drei Beispiele für Sequenzdiagramme angegeben, die mit den Einstellungen wie in Abbildung 1 gezeigt erzeugt wurden. Zwei der Diagramme enthalten mehr als die eingestellten zehn Nachrichten. Dies liegt am derzeit implementierten Verfahren, bei dem es vorkommen kann, dass die letzte generierte Nachricht eine synchrone Nachricht ist, auf die noch eine asynchrone Antwort folgen soll. Diese wird dann als überzählige Nachricht hinzugefügt.

Abbildung 3 zeigt exemplarisch eine Aufgabe, die auf Basis des ersten Diagramms aus Abbildung 2 mit dem Generator aus Abschnitt 3 erzeugt wurde. Generiert wurde eine Multiple-Choice-Frage zum Verständnis des Diagramms.

## 5 Fazit und Ausblick

Wie im Titel dieses Beitrags bereits angedeutet, stellen die bisherigen Arbeiten lediglich Ansätze dar, um die automatische Generierung von Aufgaben zum Modellverstehen zu ermöglichen. Die beiden Prototypen zeigen, dass die gewählten Konzepte technisch umsetzbar sind und auch in einer hinreichenden Breite über verschiedene Diagrammtypen und Aufgabentypen hinweg einsetzbar sind. Durch die Nutzung des XMI-Standards ist zudem eine problemlose Kopplung mit anderen Werkzeugen möglich. Der Export ist derzeit zwar noch spezifisch auf ein bestimmtes E-Assessment-System zugeschnitten, aber die Nutzung für verschiedene Systeme und auch für klassische Papier-Klausuren ist konzeptionell ohne Einschränkung möglich.

Welche Objekte kommunizieren miteinander, indem sie mindestens eine Nachricht an den jeweiligen Partner senden oder von ihm empfangen?



**Antworten:**

- Kim und Paula
- Michel und Paula
- Paula und Thorsten
- Kim und Michel
- Kim und Thorsten
- Michel und Thorsten

Abb. 3: Beispiel für eine Multiple-Choice-Aufgaben, die mit dem Generator aus Abschnitt 3 auf Basis des ersten Diagramms aus Abbildung 2 erzeugt wurde. Die zutreffenden Antworten sind in der Abbildung angekreuzt.

In der inhaltlichen Tiefe gehen jedoch beide Werkzeuge noch nicht weit genug, um das Modellverstehen umfassend mit Übungsaufgaben abdecken zu können. Dazu müssen in kommenden Arbeiten noch weitere Aufgabentemplates im Aufgabengenerator realisiert und weitere Einstellungsmöglichkeiten im Diagrammgenerator geschaffen werden. Letzteres betrifft insbesondere die Erzeugung von Diagrammlabels sowie die Möglichkeit, auf eine vorgegebene Diagrammsemantik Bezug zu nehmen. Zudem muss der Diagrammgenerator erweitert bzw. durch weitere Werkzeuge ergänzt werden, die andere Diagrammtypen behandeln. Idealerweise entsteht daraus ein System, welches verschiedene Diagramme unterschiedlicher Typen zum selben Modell erzeugen kann und darauf aufbauend auch Fragen ermöglicht, die verschiedene Diagrammtypen einbinden.

## Literaturverzeichnis

- [ASI07] Ali, Noraida Haji; Shukur, Zarina; Idris, Sufian: Assessment System For UML Class Diagram Using Notations Extraction. *IJCSNS International Journal of Computer Science and Network Security*, VOL.7 No.8, August 2007, 7(8):181–187, 2007.
- [Ci] Ciancarini, Paolo: , Exercises on basic UML behaviors. Online [<http://www.cs.unibo.it/cianca/wwwpages/ids/esempi/uml-b.pdf>].
- [CLP18] Correia, Helder; Leal, José Paulo; Paiva, José Carlos: Improving Diagram Assessment in Mooshak. In (Ras, Eric; Roldán, Guerrero; Elena, Ana, Hrsg.): *Technology Enhanced Assessment (TEA 2017)*. Springer International Publishing, Cham, S. 69–82, 2018.
- [Co03] Cowling, A. J.: Modelling: a neglected feature in the software engineering curriculum. In: *Proceedings 16th Conference on Software Engineering Education and Training*, 2003. (CSEE&T 2003). S. 206–215, March 2003.
- [Da06] Davies, Islay; Green, Peter; Rosemann, Michael; Indulska, Marta; Gallo, Stan: How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, 58(3):358 – 380, 2006. Including the special issue : ER 2004.
- [Fi17] Figl, Kathrin: Comprehension of Procedural Visual Business Process Models. *Business & Information Systems Engineering*, 59(1):41–67, Feb 2017.
- [Ge] GenMyModel Repository. Online [<https://repository.genmymodel.com/>].
- [Gi13] Gierl, Mark J.: *Automatic Item Generation*. Routledge, 2013.
- [GI08] Glinz, Martin: Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik-Spektrum*, 31(5):425–434, Oct 2008.
- [HFL12] Houy, Constantin; Fettke, Peter; Loos, Peter: Understanding Understandability of Conceptual Models – What Are We Actually Talking about? In (Atzeni, Paolo; Cheung, David; Ram, Sudha, Hrsg.): *Conceptual Modeling*. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 64–77, 2012.
- [KC13] Karasneh, Bilal; Chaudron, Michel R. V.: Online Img2UML Repository: An Online Repository for UML Models. In: *EESSMOD@MoDELS*. 2013.

- [KSV19] Kafa, Violet; Siegburg, Marcellus; Voigtlander, Janis: Exercise Task Generation for UML Class/Object Diagrams, via Alloy Model Instance Finding. In: SACLA 2019. CCIS 1136, 2019.
- [Le06] Le, Nguyen-Thanh: A Constraint-based Assessment Approach for Free-Form Design of Class Diagrams using UML. In: Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains. S. 92, 2006.
- [Li13] Linck, B.; Ohrndorf, L.; Schubert, S.; Stechert, P.; Magenheimer, J.; Nelles, W.; Neugebauer, J.; Schaper, N.: Competence model for informatics modelling and system comprehension. In: 2013 IEEE Global Engineering Education Conference (EDUCON). S. 85–93, March 2013.
- [MB08] Moritz, Sally; Blank, Glenn: Generating and Evaluating Object-Oriented Designs for Instructors and Novice Students. In: Intelligent Tutoring Systems for Ill-Defined Domains: Assessment and Feedback in Ill-Defined Domains. S. 35, 2008.
- [Mo] Modelling exercises. Online [<http://sce2.umkc.edu/bit/burris/pl/modeling/qanda.html>].
- [Sc12] Schramm, Joachim; Strickroth, Sven; Le, Nguyen-Thanh; Pinkwart, Niels: Teaching UML Skills to Novice Programmers Using a Sample Solution Based Intelligent Tutoring System. In (Youngblood, G. M.; McCarthy, P., Hrsg.): Proceedings of the 25<sup>th</sup> International Conference of the Florida Artificial Intelligence Research Society (FLAIRS). AAAI, Marco Island, FL, USA, S. 472–477, 2012.
- [SG14] Striewe, Michael; Goedicke, Michael: Automated Assessment of UML Activity Diagrams. In: Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE 2014). S. 336, 2014.
- [Sh16] Shenoy, Varun; Aparanji, Ullas; Sripradha, K.; Kumar, Viraj: Generating DFA Construction Problems Automatically. In: International Conference on Learning and Teaching in Computing and Engineering (LaTICE). S. 32–37, 2016.
- [St16] Striewe, Michael: An architecture for modular grading and feedback generation for complex exercises. *Science of Computer Programming*, 129:35–47, 2016.
- [TSW08] Thomas, Pete; Smith, Neil; Waugh, Kevin: Automatically assessing graph-based diagrams. *Learning, Media and Technology*, 33(3):249–267, 2008.