01584: Grundpraktikum Programmierung

durchgeführt vom
Lehrgebiet Softwaretechnik und Theorie der Programmierung
im Wintersemester 2020/21

Aufgabenstellung

v1.02

Inhaltsverzeichnis

1. Programmierpraktikum Wintersemester 2020/21

Herzlich willkommen zum Programmierpraktikum im Wintersemester 2020/21.

Das Lehrgebiet Softwaretechnik und Theorie der Programmierung von Prof. Dr. Jörg Desel, das dieses Praktikum durchführt, hat einen seiner Forschungsschwerpunkte im Bereich der Petrinetze. Auch Sie werden sich in diesem Semester eingehend mit Petrinetzen beschäftigen. Ihre Aufgabe wird es sein, ein Anwendungsprogramm in der Programmiersprache Java zu schreiben, mit der vorgegebene Petrinetze angezeigt und Algorithmen zum Berechnen von bestimmten Eigenschaften darauf angewandt werden können.

Für die Umsetzung der in Kapitel 3 beschriebenen Aufgabe haben Sie gut drei Monate Zeit. Da die Zeit nie so schnell verrinnt wie beim Programmieren und Testen, empfehlen wir Ihnen, zügig mit der Bearbeitung zu beginnen.

Wir wünschen Ihnen viel Erfolg und Freude bei der Arbeit!

1.1. Terminplan

- Offizieller Arbeitsbeginn: **1. Oktober 2020**.
- Abgabe des fertigen Programms samt Dokumentation:

  Die Lösungen müssen bis **Sonntag, 10. Januar 2021**, 23.59 Uhr, bei uns eingegangen sein
- Korrektur: **ab dem 11. Januar 2021**
  Die Sichtung der eingegangenen Lösungen und die erste Bewertung dauert circa einen Monat. Das Ergebnis erhalten Sie im Anschluss per Mail.
- Virtuelle Präsenzphase: **zwischen 19. und 26. Februar 2021**
  Ihr persönlicher Termin wird voraussichtlich in diesem Zeitraum stattfinden und eine Dauer von ca. 2 Stunden haben. Bitte geben Sie Terminprobleme, wie beispielsweise durch Klausuren, Hochzeiten und Arbeitseinsätze für Firmen, rechtzeitig bekannt, damit wir eine für alle positive Lösung finden können.

Weitere Informationen zu den Abgabe- und Bewertungsmodalitäten finden Sie in Kapitel 2.

1.2. Betreuung

Während des Semesters stehen Ihnen Herr Dr. Marc Finthammer, Herr Cajus Netzer und Frau Andrea Frank als Betreuer in fachlichen Fragen zur Seite. Für alles Organisatorische und bei sonstigen Problemen können Sie Frau Frank ansprechen.

Mail Die Praktikumsbetreuung ist über die folgende E-Mail-Adresse zu erreichen:

`propra@fernuni-hagen.de`

Fragen, die programmtechnischer Natur sind, werden allerdings nicht per E-Mail beantwortet. Bitte nutzen Sie hierfür die Diskussions-Newsgroup. Meistens haben mehrere Teilnehmer das gleiche Problem und so können sich alle an der Diskussion beteiligen und Hilfe bekommen.

Bitte stellen Sie Ihre Fragen bei Problemen, Unklarheiten, etc. frühzeitig(!) - vor allem vor dem Abgabetermin. Dies ist besser als eine ungültige Abgabe zu riskieren.

Newsgroups Die Betreuung des Praktikums erfolgt überwiegend über die folgenden zwei Newsgroups:

- `feu.informatik.kurs.1584.betreuung.ws`
  In der Betreuungs-Newsgroup werden wir Ankündigungen und allgemeine Informationen wie z.B. Änderungen des organisatorischen Ablaufs bekannt geben. Bitte schauen Sie mindestens einmal pro Woche in diese Newsgroup, um sich über aktuelle Mitteilungen zu informieren.
- `feu.informatik.kurs.1584.diskussion.ws`
  Wenn Sie inhaltliche oder organisatorische Fragen zum Praktikum haben oder anderen Studierenden eine Hilfestellung geben möchten, so gehen Sie bitte in diese Newsgroup. Hier können alle praktikumsbezogenen Probleme diskutiert werden.
  Ihre Mitarbeit in dieser Diskussions-Newsgroup ist wichtig. In den vergangenen Jahren fand hier immer wieder ein intensiver Austausch statt, von dem alle Beteiligten profitieren konnten.

Die beiden Newsgroups können Sie über das Newsportal[1] der FernUniversität erreichen. Als wesentlich komfortablere Alternative zur Bearbeitung der Newsgroups wird jedoch die Benutzung eines Newsreaders wie z.B. Thunderbird oder Outlook empfohlen. Für die Einrichtung eines News-Kontos innerhalb eines solchen News-Clients bietet das ZMI die Broschüre „So richte ich mir ein News-Konto ein“ an[2].

[1] `https://www.fernuni-hagen.de/newsportal`
[2] `https://www.fernuni-hagen.de/imperia/md/content/zmi_2010/upl014.pdf`

2. Abgabe und Bewertung Ihres Programms

Im Rahmen dieses Programmierpraktikums haben wir folgende Anforderungen und Erwartungen an Sie:

1. Sie müssen eigenständig ein Programm implementieren, welches alle Anforderungen erfüllt, die sich aus der nachfolgenden Aufgabenstellung ergeben. Gruppenlösungen sind nicht zulässig.
2. Sie müssen ein Eclipse-Projekt erstellen und dabei folgende Punkte berücksichtigen:
   - Benennen Sie Ihr Eclipse-Projekt exakt wie in Kapitel C.7 beschrieben nach dem Schema:

     `Petrinets_<Matr-Nr>_<Nachname>_<Vorname>`
   - Neben Ihren eigenen Klassen und denen aus der Java-Standard-Bibliothek dürfen Sie noch die von uns zur Verfügung gestellte GraphStream Bibliothek (siehe Anhang B) verwenden. Die Nutzung anderer fremder Klassenbibliotheken ist nicht zulässig. Eine Ausnahme bilden Testframeworks wie z.B. JUnit, die Sie während der Entwicklung Ihres Programms einsetzen dürfen.
   - Erzeugen Sie aus Ihrem Programm eine ausführbare JAR Datei (siehe Kapitel B.5) und legen Sie diese im Hauptverzeichnis ihres Eclipse-Projekts ab. Die JAR Datei Ihres Programms soll sich durch folgenden Aufruf (aus dem Hauptverzeichnis ihres Eclipse-Projekts) direkt ausführen lassen:

     ```
     java -jar dateiname.jar
     ```

     Der Dateiname der JAR Datei soll dem Projektnamen (siehe oben) entsprechen.
   - Ihre Programmdokumentation (siehe Kapitel 4.1) legen Sie bitte als PDF Datei in einem Unterordner des Eclipse-Projekts ab. Dieser Ordner soll den Namen `documentation` tragen.
   - Die generierte Javadoc-Dokumentation (siehe Kapitel 4.2) ist parallel daneben in einem Verzeichnis mit dem Namen `doc` zu erzeugen und ebenfalls abzugeben.
   - Drucken Sie das Dokument `Selbststaendigkeitserklaerung.pdf` (siehe Kapitel C.4) aus, unterschreiben Sie es und legen Sie eine Datei mit einem Scan (oder einem gut lesbaren Foto) des unterschriebenen Dokuments im Hauptverzeichnis Ihres Eclipse-Projekts ab.
3. Ihr Eclipse-Projekt soll (entsprechend den Hinweisen in Kapitel C) so konfiguriert sein, dass es nach dem Auschecken auf einem fremden System sofort lauffähig ist, ohne dass z.B. Verzeichnispfade geändert oder sonstige manuelle Anpassungen vor-

genommen werden müssten.
Sie können das (in guter Näherung) selbst überprüfen, indem Sie in Eclipse einen neuen Workspace anlegen und Ihr Projekt (zusammen mit dem in Kapitel C.5 beschriebenen Basis-Projekt) dort „frisch“ auschecken. Bevor Sie das neu ausgecheckte Projekt testweise starten, sollten Sie noch Ihr ursprüngliches Workspace Verzeichnis temporär umbenennen, um sichergehen zu können, dass in Ihrem Programm keine absoluten Verzeichnispfade verwendet werden.

4. Damit Ihr Programm überhaupt als Abgabe akzeptiert und einer weiteren Bewertung unterzogen werden kann, muss es zudem folgende **Mindestanforderung** erfüllen: Bei Ausführung des in Kapitel 3.9 beschriebenen automatisierten Testlaufs müssen **alle Beispiele**, die von uns zur Verfügung gestellt werden (siehe Kapitel C.5), korrekt berechnet werden. Ob Ihr Programm diese Anforderung erfüllt, können Sie durch einen Vergleich Ihrer Programmausgabe mit den von uns zur Verfügung gestellten Ergebnissen leicht selbst kontrollieren.
5. Die Abgabe Ihrer Lösung als Eclipse-Projekt muss bis zum 10.01.2021 erfolgen. Zu diesem Zeitpunkt muss Ihre Lösung in dem Subversion-Repository, welches für diese Veranstaltung zur Verfügung steht, eingecheckt worden sein. Kennzeichnen Sie das Projekt in Ihrem SVN-Bereich mit dem Tag `Abgabe` (siehe Kapitel D.7).
6. Während der virtuellen Präsenzphase sollen Sie uns zeigen, dass Sie Ihr Programm verstehen und damit umgehen können, indem Sie u.a. den Aufbau Ihres Programms erklären und entsprechende Fragen der Betreuenden beantworten können.

Bitte beachten Sie bei der Implementierung Ihrer Lösung, dass *alle* Anforderungen aus der Aufgabenstellung erfüllt sein müssen. Wir empfehlen daher, dass Sie alle Spiegelpunkte immer wieder durchgehen, prüfen und ggf. abhaken oder – noch besser – zusätzlich eine eigene Check-Liste aus Anforderungen erstellen und konsequent abarbeiten. Ansonsten laufen Sie Gefahr, einige Anforderungen zu übersehen und damit Ihre Abgabe unnötig zu gefährden. Nutzen Sie die bereits vorstrukturierte Form der Aufgabenstellung, um darauf aufbauend Ihre eigene Arbeit an dem Programm entsprechend zu organisieren und noch weiter zu strukturieren.

Die folgenden Punkte schreiben wir nur aus rechtlichen Gründen hinzu. Wir selbst sind uns sicher, nur verantwortungsvolle Programmierer zu haben.

Manipulationsversuche mit den eingesandten Programmen, z.B. durch Aufspielen von Computerviren oder anderen Programmkomponenten, die nicht der Lösung der Programmieraufgabe dienen, führen – unabhängig von Schadensersatzansprüchen seitens der FernUniversität – zu einer nichtbestandenen Prüfungsleistung. Das Kopieren von Sourcecode aus ähnlichen Projekten aus dem Internet ist unter keinen Umständen gestattet und wird mit einem sofortigen Ausschluss geahndet. Wir besitzen sowohl die entsprechenden Werkzeuge, als auch die notwendige Sachkenntnis, um „Refactorings“ zu erkennen.

3. Aufgabenstellung

Der zentrale Punkt der Aufgabenstellung in diesem Praktikum besteht darin, algorithmisch zu überprüfen, ob zu einem gegebenen Petrinetz ein endlicher Erreichbarkeitsgraph existiert. Zudem sollen das Petrinetz, das Schalten im Petrinetz sowie der Erreichbarkeitsgraph graphisch dargestellt werden.

3.1. Petrinetze

Petrinetze modellieren verteilte dynamische Systeme mit besonders einfachen Mitteln. Ihre graphische Repräsentation spiegelt die Verteilung der modellierten Komponenten wider. Das Verhalten eines Petrinetzes basiert auf einer sehr einfachen Schaltregel. Für die Modellierung mit Petrinetzen siehe [DJ01], die Interpretation der Petrinetzelemente ist für dieses Programmierpraktikum allerdings unerheblich.

Ein Petrinetz (siehe Abb. 1) besteht aus ***Stellen*** (dargestellt durch Kreise) und ***Transitionen*** (dargestellt durch Quadrate). Jede Transition hat eine Menge von Stellen in ihrem ***Vorbereich*** (dargestellt durch ***Pfeile*** von diesen Stellen *zu der Transition*) und eine Menge von Stellen in ihrem ***Nachbereich*** (dargestellt durch Pfeile *von der Transition* zu diesen Stellen). Stellen können ***Marken*** tragen (dargestellt durch Punkte in den Stellen). Die ***Markierung*** eines Petrinetzes besteht aus der Information, auf welchen Stellen jeweils wie viele Marken liegen. Eine Markierung ist formal also eine Funktion, die jeder Stelle eine nichtnegative ganze Zahl zuordnet[3]. Zur Angabe eines Petrinetzes gehört immer auch eine ***Anfangsmarkierung***.

Die aktuelle Markierung eines Petrinetzes ändert sich durch das ***Schalten*** von Transitionen. Die ***Schaltregel*** lautet wie folgt: Eine Transition ist unter einer Markierung ***aktiviert***, wenn jede Stelle in ihrem Vorbereich wenigstens eine Marke trägt. Eine aktivierte Transition kann schalten und transformiert dadurch die aktuelle Markierung in eine ***Folgemarkierung***: jede Stelle im Vorbereich der Transition verliert eine Marke und jede Stelle im Nachbereich der Transition gewinnt eine Marke (vgl. Abb. 2). Stellen, die sowohl zum Vor- als auch zum Nachbereich der Transition gehören, müssen markiert sein, damit die Transition schalten kann; sie ändern ihre Markenzahl durch das Schalten der Transition aber nicht. Die Menge der ***erreichbaren Markierungen*** besteht aus der Anfangsmarkierung sowie all denjenigen Markierungen, die von der Anfangsmarkierung aus durch einfaches oder mehrfaches Schalten beliebiger, jeweils aktivierter Transitionen erreicht werden können.

[3] Auch wenn die Begriffe *Marke* und *Markierung* im allgemeinen Sprachgebrauch relativ ähnlich klingen, so müssen sie im Kontext von Petrinetzen sorgsam unterschieden werden: Eine *Marke* ist quasi die kleinste Einheit, die auf einer Stelle liegen kann. Eine *Markierung* hingegen bezieht sich auf das *gesamte Petrinetz* und beschreibt für *jede Stelle*, wie viele Marken auf ihr liegen.

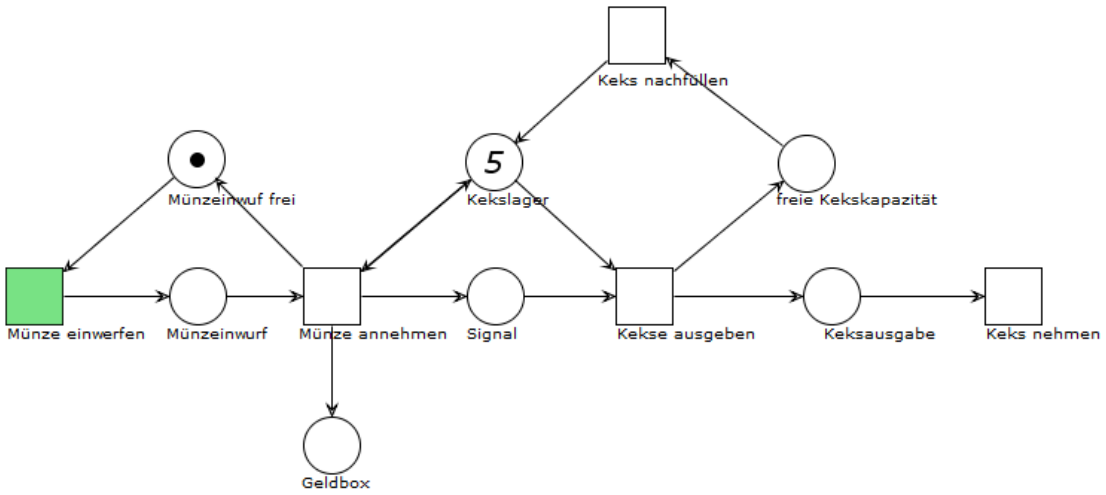


Abbildung 1: Petrinetz „Keksautomat“ mit Anfangsmarkierung (Beispiel aus [Rei13]). Die unter dieser Markierung aktivierte Transition ist grün hervorgehoben. Die Zahl 5 in der Stelle „Kekslager“ steht stellvertretend für 5 Marken.

Da Vor- und Nachbereich einer Transition unterschiedlich groß sein können, kann sich die ***Gesamtmarkenzahl*** eines Netzes durch das Schalten einer Transition verändern und in manchen Fällen sogar beliebig wachsen. So kann sich auch die Markierung einer einzelnen Stelle[4] entweder in einem beschränkten Intervall bewegen, oder diese Stelle kann unbeschränkt sein. Sind alle Stellen beschränkt, so ist die Menge der erreichbaren Markierungen endlich (wir gehen davon aus, dass ein Petrinetz nur endlich viele Stellen und Transitionen besitzt). In diesem Fall nennen wir das Petrinetz ***beschränkt*** (vgl. Abb. 3). Ist aber wenigstens eine Stelle unbeschränkt, so gibt es unendlich viele erreichbare Markierungen dieser Stelle und damit auch des Petrinetzes, das in diesem Fall ***unbeschränkt*** genannt wird (vgl. Abb. 4).

3.2. Erreichbarkeitsgraph eines Petrinetzes

Der ***Erreichbarkeitsgraph*** eines Petrinetzes ist ein gerichteter Graph mit einem ausgezeichneten ***Anfangsknoten***, der die Anfangsmarkierung des Petrinetzes repräsentiert. Jeder Knoten des Erreichbarkeitsgraphen steht für eine erreichbare Markierung des Petrinetzes. Eine gerichtete Kante (v, w) verbindet zwei Knoten v und w, wenn durch das Schalten einer Transition die Ausgangsmarkierung v in die Zielmarkierung w überführt

[4] Die *Markierung einer Stelle* beschreibt die Anzahl an Marken, die auf dieser Stelle liegen. Der Ausdruck *„Markierung einer Stelle“* beschränkt die Betrachtung also auf die Marken einer konkreten Stelle, wohingegen sich der generelle Begriff *Markierung* auf sämtliche Marken aller Stellen des Petrinetzes bezieht.

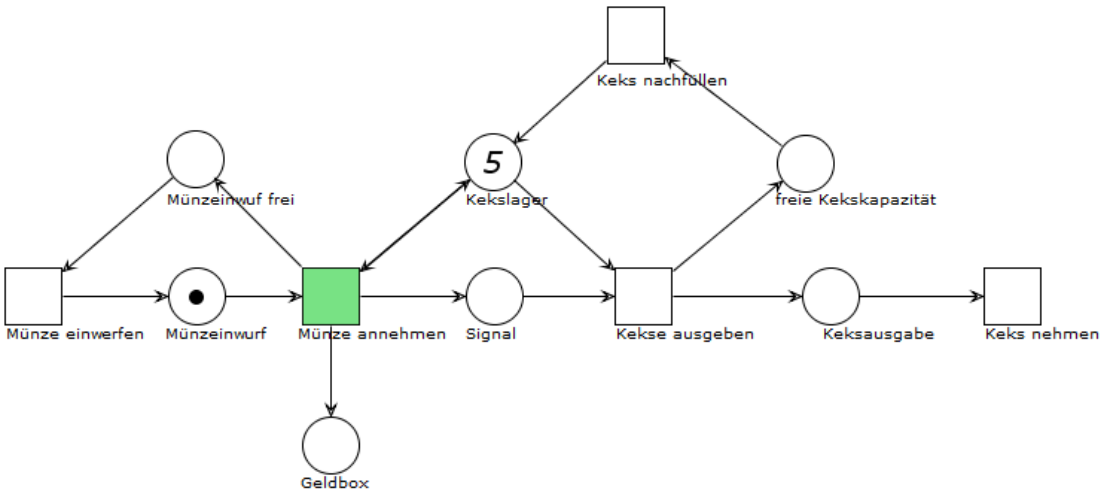


Abbildung 2: Das Petrinetz aus Abb. 1 mit Folgemarkierung, nachdem die Transition „Münze einwerfen“ geschaltet hat. Die nun aktivierte Transition ist wieder grün hervorgehoben.

wird. Die Kante wird mit dem Namen dieser Transition beschriftet. Wenn das Petrinetz beschränkt ist, dann ist der Erreichbarkeitsgraph endlich (vgl. Abb. 5) und somit grundsätzlich generierbar.

Die vollständige Darstellung des Erreichbarkeitsgraphen ist also nur dann möglich, wenn das Petrinetz beschränkt ist. Sie müssen daher algorithmisch herausbekommen, ob dies der Fall ist. Für diese ***Beschränktheitsanalyse*** können Sie folgende Eigenschaft von Petrinetzen verwenden: Ein Petrinetz ist genau dann unbeschränkt, wenn es eine erreichbare Markierung m und eine von m aus erreichbare Markierung m' gibt, mit folgenden Eigenschaften: m' weist jeder Stelle mindestens so viele Marken zu wie m und dabei mindestens einer Stelle sogar mehr Marken als m. Wenn Ihr Programm also während der Konstruktion des Erreichbarkeitsgraphen ein Paar erreichbarer Markierungen m und m' wie oben beschrieben[5] identifiziert, sollte es die Berechnung beenden und zurückmelden, dass das Petrinetz unbeschränkt ist.

Wichtig ist auch zu wissen, dass jeder unendliche gerichtete Pfad (also jede unendliche Folge gerichteter Kanten) in einem Erreichbarkeitsgraphen entweder einen Knoten und damit eine Markierung mehrfach enthält (der Pfad also einen Kreis beinhaltet), oder aber zwei Knoten mit zugehörigen Markierungen m und m' enthält (in dieser Reihenfolge und wie oben beschrieben). Da jeder Knoten nur endlich viele ausgehende Kanten besitzt, können Sie also ohne Bedenken mit der Konstruktion des Erreichbarkeitsgraphen beginnen; sollte er unendlich sein, gibt es einen unendlichen Pfad ohne Kreise (keine Markierung kommt mehrfach vor) und daher mit Markierungen m und m' wie oben beschrieben, die

[5] Eine erreichbare Markierung m kann natürlich auch die Anfangsmarkierung sein.

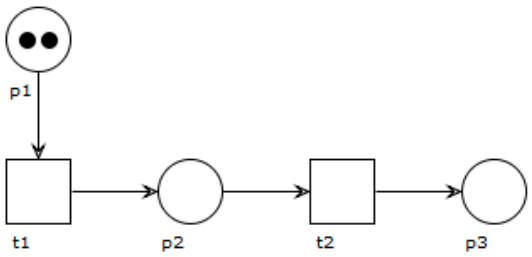


Abbildung 3: Ein beschränktes Petrinetz (mit Anfangsmarkierung). Unabhängig davon, in welcher Reihenfolge und wie oft die Transitionen t1 und t2 schalten, trägt jede der drei Stellen p1, p2 und p3 immer entweder keine, eine oder zwei Marken. Damit ist jede Stelle dieses Petrinetzes und damit auch das Netz selber beschränkt.

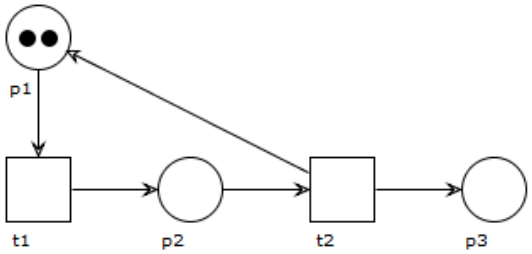


Abbildung 4: Ein unbeschränktes Petrinetz (mit Anfangsmarkierung). Jedes Mal, wenn die Transition t2 schaltet, erhöht sich die Markenanzahl auf der Stelle p3. Da die Transitionen t1 und t2 aufgrund des Zyklus (im Wechsel) unendlich oft schalten können, kann die Markenanzahl auf der Stelle p3 beliebig wachsen. Die Stelle p3 ist somit unbeschränkt und dadurch ist auch das Petrinetz unbeschränkt.

Ihr Programm nur identifizieren muss.

Vorsicht! Auch ein beschränktes Petrinetz kann durchaus Markierungen m und m' wie oben beschrieben besitzen, aber nur dann, wenn m' von m aus *nicht erreichbar* ist. Bei dem Vergleich einer neu konstruierten Markierung m' (bzw. eines entsprechenden Knotens im Erreichbarkeitsgraphen) mit früher konstruierten Markierungen m dürfen Sie daher nur diejenigen Markierungen m berücksichtigen, die auf einem Pfad von der Anfangsmarkierung zur Markierung m' liegen.

Im Rahmen dieser Aufgabenstellung bezeichnen wir einen Erreichbarkeitsgraphen[6], der sich quasi gerade „im Aufbau befindet“, als ***partiellen Erreichbarkeitsgraphen*** (siehe Abb. 6 und 7). Formal bedeutet dies, dass die Struktur eines partiellen Erreichbarkeitsgraphen prinzipiell der eines „vollständigen“ Erreichbarkeitsgraphen entspricht,

[6] Der Erreichbarkeitsgraph kann endlich oder unendlich groß sein.

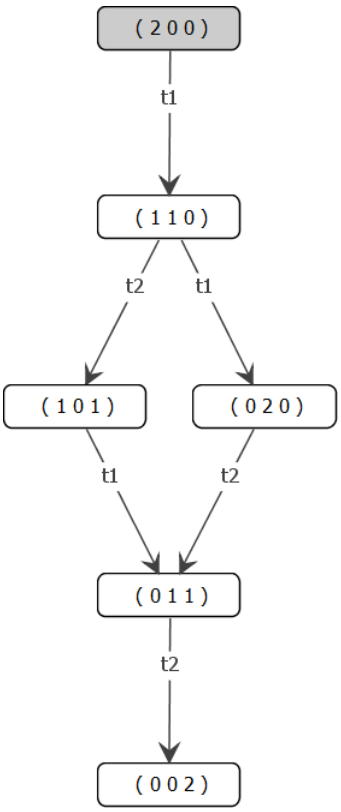


Abbildung 5: Endlicher Erreichbarkeitsgraph des Petrinetzes aus Abb. 3. Der Anfangsknoten ist grau hervorgehoben. Die Beschriftung eines Knotens gibt die Anzahl der Marken auf den Stellen $p1$, $p2$ und $p3$ an.

jedoch mit der Einschränkung, dass nicht zwingend alle erreichbaren Markierungen des Petrinetzes enthalten sein müssen. Allerdings müssen auch in einem partiellen Erreichbarkeitsgraphen alle vorhandenen Markierungen von der Anfangsmarkierung aus erreichbar sein. Ein partieller Erreichbarkeitsgraph stellt somit einen (unter Umständen) noch nicht vollständigen Erreichbarkeitsgraphen dar.

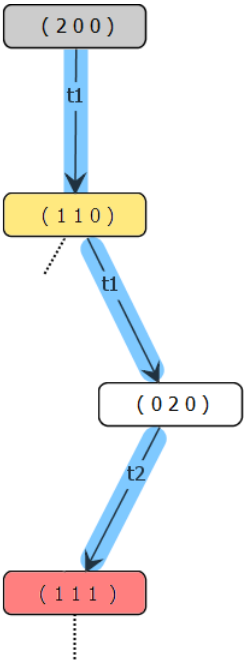


Abbildung 6: Ein partieller Erreichbarkeitsgraph des Petrinetzes aus Abb. 4. Farblich hervorgehoben sind Knoten m (gelb) und m' (rot), die das Petrinetz als unbeschränkt identifizieren, sowie ein dazugehöriger Pfad (mit Länge 3) von der Anfangsmarkierung.

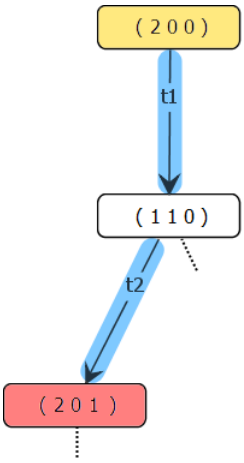


Abbildung 7: Ein weiterer partieller Erreichbarkeitsgraph des Petrinetzes aus Abb. 4 mit anderen Knoten m (gelb) und m' (rot) und einem Pfad (der Länge 2) von der Anfangsmarkierung, die das Petrinetz ebenfalls als unbeschränkt identifizieren.

3.3. Konkrete Aufgabenstellung

Sie sollen ein Programm mit einer graphischen Oberfläche (GUI) entwickeln, welches folgende Kernfunktionalitäten bereitstellt:

- graphische Darstellung eines (aus einer Datei geladenen) Petrinetzes
- Schalten von Transitionen im Petrinetz via Mausklick
- graphische Darstellung von partiellen Erreichbarkeitsgraphen, die sich sukzessive durch das Schalten von Transitionen im Petrinetz aufbauen
- Durchführung der Beschränktheitsanalyse für das Petrinetz und Berechnung des Erreichbarkeitsgraphen
- graphische Darstellung des Ergebnisses der Beschränktheitsanalyse: falls das Petrinetz beschränkt ist, Darstellung des berechneten Erreichbarkeitsgraphen, anderenfalls Visualisierung des Abbruchkriteriums (m, m' und Pfad) in einem partiellen Erreichbarkeitsgraphen
- automatisierte Durchführung der Beschränktheitsanalyse für eine Menge von Petrinetz-Dateien (quasi in Form einer Stapelverarbeitung), ohne graphische Darstellung der Ergebnisse

Bei der Realisierung dieser Funktionalitäten sollen Sie insbesondere folgende Punkte beachten:

- Entwickeln Sie geeignete Datenstrukturen zur Repräsentation eines Petrinetzes. Sie müssen zudem auf Basis dieser Datenstrukturen das Schalten von Transitionen realisieren, so dass Sie u.a. eine entsprechende Schaltfunktion und Datenstrukturen zur Repräsentation von Markierungen implementieren müssen.
- Entwickeln Sie ebenfalls geeignete Datenstrukturen zur Repräsentation der (partiellen) Erreichbarkeitsgraphen von Petrinetzen.
- Realisieren Sie auf Basis des von uns zur Verfügung gestellten Parsers (siehe Kapitel A) das Laden eines Petrinetzes aus einer PNML Datei. Konkret bedeutet dies, dass der vorhandene Parser mittels *Vererbung* von Ihnen so erweitert werden soll, dass er das in der PNML Datei spezifizierte Petrinetz mit den von Ihnen entwickelten Petrinetz-Datenstrukturen aufbaut.
- Zur graphischen Darstellung des Petrinetzes und des Erreichbarkeitsgraphen soll die Graphenvisualisierungsbibliothek ***GraphStream***[7] (siehe Kapitel B) verwendet werden.
- Die graphische Benutzeroberfläche soll mit dem Java Swing GUI-Toolkit implementiert werden (da dieses auch von der GraphStream Bibliothek unterstützt wird).

[7] `https://graphstream-project.org`

Es soll folglich kein JavaFX verwendet werden.

- Es soll ein Algorithmus (im Folgenden ***Beschränktheits-Algorithmus*** genannt) implementiert werden, der die in Kapitel 3.2 beschriebene Beschränktheitsanalyse durchführt. Für ein gegebenes Petrinetz soll der Algorithmus also zurückliefern, ob es beschränkt ist oder nicht. Zusätzlich soll der Algorithmus noch Folgendes zurückliefern:
  - Falls das Petrinetz beschränkt ist, soll der Algorithmus auch den im Zuge der Analyse konstruierten Erreichbarkeitsgraphen zurückliefern.
  - Falls das Petrinetz hingegen unbeschränkt ist, soll der Algorithmus auch den bis zum Abbruchzeitpunkt konstruierten partiellen Erreichbarkeitsgraphen zurückliefern; zudem soll der vom Algorithmus ermittelte Pfad von der Anfangsmarkierung zu den Markierungen m und m' zurückgeliefert werden, der das Petrinetz als unbeschränkt identifiziert hat.
- Die Implementierung des Beschränktheits-Algorithmus soll auf den von Ihnen entwickelten Datenstrukturen basieren, d.h. es sollen innerhalb des Algorithmus keine Klassen der GraphStream Bibliothek verwendet werden. Diese Bibliothek soll also nur zur Visualisierung eines Datenmodells, nicht aber zur strukturellen Repräsentation des Datenmodells verwendet werden. Der Algorithmus muss folglich auch ausführbar sein, wenn die Visualisierung durch die GraphStream Bibliothek nicht zur Verfügung steht; dies ist insbesondere für die oben erwähnte Ausführung des Algorithmus im Rahmen einer Stapelverarbeitung von Bedeutung.
- Ihr Programm sollte so ausgelegt sein, dass Beispiele, die ungefähr in der Größenordnung der von uns zur Verfügung gestellten Beispiele liegen, problemlos verarbeitet werden können. Wir werden bei der Beurteilung Ihres Programms keine Tests mit deutlich größeren Beispielen durchführen.

Zudem sollten Sie bei der Entwicklung Ihres Programms noch folgende Hinweise berücksichtigen:

- Wählen Sie für Ihr Programm einen sinnvoll strukturierten Gesamtaufbau. Bei der gegebenen Aufgabenstellung kann beispielsweise ein Aufbau nach dem Muster Model-View-Controller zu einer geeigneten Architektur des Programms beitragen.
- Berücksichtigen Sie beim Erstellen Ihres Programms die Grundsätze einer objektorientierten Modellierung in angemessener Weise. Setzen Sie beispielsweise – wo es sinnvoll ist – Vererbung ein, strukturieren Sie in angemessener Weise in Klassen und beachten Sie die Sichtbarkeiten von Attributen und Methoden (möglichst wenig `public`).
- Achten Sie generell auf Ihren Code-Stil. Darunter fallen Punkte wie Anordnung von Klammern und Einrückungen, sinnvoll gewählte Bezeichner sowie gute Lesbarkeit

des Codes für Sie selbst und für andere.

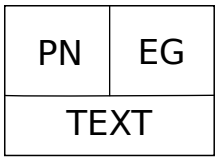
3.4. Aufbau und grundlegende Bedienung der GUI

Im Folgenden listen wir einige Punkte auf, die Sie hinsichtlich des Aufbaus und der Bedienung der GUI beachten sollen. Davon abgesehen haben Sie natürlich alle Freiheiten, Ihre GUI optisch ansprechend und bedienerfreundlich zu gestalten.

- Beim Start des Programms soll das Programmfenster in der Mitte des Bildschirms dargestellt werden. Sie dürfen dabei von einem System mit nur einem Bildschirm und einer Auflösung von mindestens 1920×1080 Pixeln ausgehen.
- Die Größe des Programmfensters soll frei veränderbar sein, wobei die Größe bei Programmstart sinnvoll(!) gewählt sein soll. Das heißt, es sollen alle GUI Elemente sichtbar sein, aber nicht der gesamte Bildschirm (mit einem maximierten Fenster) belegt werden. Eine geeignete Anfangsgröße könnte z.B. 1000×700 Pixel sein.
- In der Titelzeile des Programmfensters sollen Ihr Name und Ihre Matrikelnummer stehen.
- Die GUI soll über eine Menüleiste und eine Toolbar verfügen.
- Die Menüleiste soll ein *Datei-Menü* enthalten, welches (mindestens) über die folgenden Einträge verfügt, um die entsprechenden Aktionen auszuführen:
  - *Öffnen...* : Anzeige eines Dateiauswahl-Dialogs, um die dort ausgewählte PNML Datei anschließend zu laden
  - *Neu Laden*: Erneutes Laden der zuletzt geöffneten PNML Datei (Diese Funktionalität ist hilfreich, um eine evtl. geänderte Anfangsmarkierung (siehe Kapitel 3.5) wieder durch die Anfangsmarkierung aus der PNML Datei zu ersetzen.)
  - *Analyse mehrerer Dateien...* : Durchführung der Beschränktheitsanalyse für mehrere PNML Dateien (siehe dazu Kapitel 3.9)
  - *Beenden*: Beenden des Programms
- In der Toolbar sollen sich Buttons befinden, die in den folgenden Abschnitten noch einzeln erläutert werden. Die Funktionalität jedes Buttons soll in seinem Tool Tip Text kurz erläutert werden.
- Die Auswahl von PNML Dateien soll immer über einen Dateiauswahl-Dialog (d.h. mittels der Java Klasse `JFileChooser`) durchgeführt werden, bei dem ein Filter für die Dateierweiterung PNML gesetzt ist (trotz des Filters sollen Verzeichnisse weiterhin angezeigt werden). Bei der (nach Start des Programms) erstmaligen Anzeige eines Dateiauswahl-Dialogs soll sich dieser standardmäßig in demjenigen Verzeichnis öffnen, in dem sich die (von uns zur Verfügung gestellten) Beispieldateien

befinden. Das Verzeichnis muss dabei mittels des *relativen* Pfades[8] „`../ProPra-WS20-Basis/Beispiele`“ adressiert werden.

- Bei erneuter Benutzung eines Dateiauswahl-Dialogs während des Programmablaufs soll der Dialog wieder im zuletzt verwendeten Verzeichnis geöffnet werden.
- Dialogfenster sollen immer mittig zum Programmfenster platziert werden.
- Die GUI soll (neben Menüleiste und Toolbar) prinzipiell aus drei Teilbereichen bestehen, deren relative Größe zueinander frei veränderbar sein soll und sich der aktuellen Fenstergröße anpasst:



  - Links soll das Petrinetz (PN) dargestellt werden.
  - Rechts soll der Erreichbarkeitsgraph (EG) dargestellt werden.
  - Darunter soll ein mehrzeiliger Textbereich (TEXT) platziert werden.
- Der Textbereich soll zur Ausgabe aller relevanten Meldungen genutzt werden. Lediglich Ausgaben, die für die eigentliche Verwendung des Programms nicht zwingend benötigt werden, wie z.B. Debug-Ausgaben, können auf der Konsole ausgegeben werden. Umfangreiche Debug-Ausgaben sollten sogar auf der Konsole ausgegeben werden, um die Ausgabe im Textbereich übersichtlich zu halten.
- Der Textbereich soll über Scroll-Leisten verfügen und bei Ausgabe einer neuen Textzeile automatisch ans Ende scrollen. Es soll die Möglichkeit bestehen, den Inhalt des Textfeldes zu löschen.
- Der aktuell in der GUI dargestellte (ggf. partielle) Erreichbarkeitsgraph soll gelöscht werden können. In Zuge dessen soll auch die Markierung des Petrinetzes wieder auf die Anfangsmarkierung zurückgesetzt werden. Diese Funktionalität soll durch einen entsprechenden *Lösche-EG-Button* in der Toolbar ausgelöst werden können.
- Die Darstellung eines Graphen (also eines Petrinetzes oder Erreichbarkeitsgraphen) soll bei einer Änderung der Panelgröße automatisch mit angepasst werden (quasi eine Auto-Skalierung). Dies entspricht bereits genau dem Standardverhalten der GraphStream Bibliothek (siehe dazu auch das Demo-Programm in Kapitel B), so dass Sie an dieser Stelle nichts weiter zu tun brauchen, außer darauf zu achten, dass dieses Standardverhalten auch beibehalten wird (und nicht versehentlich von Ihnen modifiziert wird).

[8] Wenn Sie einen absoluten Pfad in Ihrem Programm verwenden, können wir Ihr Programm nicht korrekt ausführen.

- Die in der GraphStream Bibliothek bereits standardmäßig vorgegebenen Tastaturbefehle (siehe Auflistung in Kapitel B) sollen weiterhin zur Verfügung stehen (d.h. diese Tasten sollen nicht mit anderen Funktionen überschrieben werden).

3.5. Darstellung und Funktionalität eines Petrinetzes

In Kapitel 3.3 haben wir bereits ein paar grundlegende Anforderungen bezüglich der Darstellung und Funktionalität von Petrinetzen in Ihrem Programm aufgeführt, die wir nun um einige konkretere Punkte ergänzen:

- Beim Laden eines Petrinetzes sollen die in der PNML Datei vorliegenden Koordinaten genutzt werden, um die Stellen und Transitionen bei der Visualisierung mit GraphStream entsprechend zu positionieren. Die GraphStream Bibliothek kann diese Koordinaten im Prinzip direkt verarbeiten, es muss lediglich der Wert der Y-Ordinate negiert werden (da ansonsten das Petrinetz quasi spiegelverkehrt bzgl. der Y-Achse dargestellt wird).
- Beim Laden eines Petrinetzes muss auch die dazugehörige und in der PNML Datei spezifizierte Anfangsmarkierung mit verarbeitet werden.
- Das Datenmodell des Petrinetzes und die Umsetzung des Schaltens von Transitionen sollen natürlich auf den von Ihnen entwickelten Datenstrukturen basieren; die Klassen der GraphStream Bibliothek sollen nur zur Visualisierung des Petrinetzes eingesetzt werden.
- Die Beschriftung (in GraphStream *label* genannt) einer Stelle oder Transition soll sich aus der ID und dem Namen zusammensetzen, welche beide in der PNML Datei spezifiziert sind. Wenn z.B. die in der PNML Datei spezifizierte ID „`t4`“ und der Name „`send_mail`“ ist, dann soll die Beschriftung wie folgt aussehen:

  ```
  [t4] send_mail
  ```

  Das heißt, die ID soll in eckigen Klammern vor dem Namen stehen.
- Bei jeder Stelle soll zusätzlich immer die Anzahl an Marken am Ende der Beschriftung in spitzen Klammern mit angegeben werden (also selbst dann, wenn die Anzahl 0 ist). Zum Beispiel soll die Beschriftung bei einer Stelle mit *drei* Marken wie folgt aussehen:

  ```
  [s5] mailbox <3>
  ```
- Die Beschriftung einer Kante soll ebenfalls aus ihrer (in eckigen Klammern gesetzten) ID bestehen, also z.B.

  ```
  [a8]
  ```

- Graphische Darstellung der Anzahl von Marken einer Stelle:

  - Wenn eine Stelle *eine bis höchstens neun* Marken trägt, dann soll dies innerhalb der Stelle (also des Kreises) entweder durch eine entsprechende Anzahl von Punkten oder durch die konkrete Zahl dargestellt werden (je nachdem, was Ihnen besser gefällt).
  - Wenn *mehr als neun* Marken auf einer Stelle liegen, soll dies durch die Darstellung „`>9`“ in der Stelle kenntlich gemacht werden.
  - Wenn eine Stelle *keine* Marken trägt, soll sie nicht speziell kennzeichnet werden, d.h. die Zahl 0 soll nicht dargestellt werden.
  - Die GraphStream Bibliothek bietet sogenannte *Sprites* an, welche prinzipiell sehr geeignet für die Darstellung von Marken wären. Leider enthält die GraphStream Bibliothek jedoch einen Fehler, durch den sich die Verwendung von Sprites praktisch verbietet (wenn ein Sprite angeklickt wird, springt es an eine weit entfernte Position des Koordinatensystems).
    Daher empfehlen wir Ihnen, die jeweilige Anzahl an Marken mittels entsprechender PNG-Grafiken darzustellen. Konkret benötigen Sie also neun PNG-Grafiken, die jeweils 1 bis 9 Marken symbolisieren sowie eine zehnte PNG-Grafik für die Darstellung von „`>9`“.

- Die unter der aktuellen Markierung aktivierten Transitionen sollen farblich hervorgehoben werden.
- Das Schalten einer (aktivierten) Transition soll durch einen linken Mausklick erfolgen, woraufhin dann die aktuelle Markierung des Petrinetzes entsprechend der Schaltregel angepasst werden muss.
- Die aktuelle Markierung des Petrinetzes soll editiert werden können, d.h. die Anzahl der Marken einer Stelle soll verändert werden können. Die editierte Markierung gilt anschließend als (neue) Anfangsmarkierung(!) des Petrinetzes, d.h. die bisherige (evtl. aus der PNML Datei gelesene) Anfangsmarkierung wird durch die editierte Markierung ersetzt. Das Editieren der aktuellen Markierung soll mittels eines *Marke-Minus-Button* bzw. *Marke-Plus-Button* in der Toolbar realisiert werden, der die Anzahl der Marken einer ausgewählten Stelle um eins erniedrigt bzw. erhöht. Die zu editierende Stelle soll zuvor mittels linkem Mausklick ausgewählt werden können.
- Die Markierung des Petrinetzes soll auf die (aktuell als solche definierte) Anfangsmarkierung des Petrinetzes zurückgesetzt werden können (dies muss nicht zwingend die Anfangsmarkierung aus der PNML Datei sein, siehe vorherigen Punkt). Diese Funktionalität soll durch einen entsprechenden *Reset-Button* in der Toolbar ausgelöst werden können.

Es steht Ihnen natürlich frei, Ihre GUI (bzw. Ihr Programm) um weitere Bedienelemente und Funktionalitäten zu erweitern. Die von uns aufgestellten Anforderungen geben

dabei zwar einen Rahmen vor, den Sie entsprechend umsetzen müssen, sollen aber nicht als Einschränkung verstanden werden, so dass Sie bei der konkreten Ausgestaltung Ihres Programms noch kreativen Freiraum haben.

3.6. Aufbau eines partiellen Erreichbarkeitsgraphen beim Schalten von Transitionen

Durch das Schalten von Transitionen im Petrinetz (via Mausklick) soll sukzessive ein partieller Erreichbarkeitsgraph (im Folgenden ***pEG*** abgekürzt) aufgebaut und dargestellt werden. Das Datenmodell des pEG soll auf den von Ihnen entwickelten Datenstrukturen für Erreichbarkeitsgraphen basieren und mittels der GraphStream Bibliothek visualisiert werden (d.h., die Klassen der GraphStream Bibliothek dürfen ausschließlich zur Visualisierung, nicht aber zur Repräsentation des Datenmodells genutzt werden).

Beim sukzessiven Aufbau des pEG soll wie folgt vorgegangen werden:

- Da ein Petrinetz immer über eine Anfangsmarkierung verfügt, ist im pEG auch immer mindestens der Anfangsknoten vorhanden. Das heißt, wenn ein Petrinetz in der GUI dargestellt wird, dann muss auch immer ein pEG dargestellt werden, da dieser mindestens aus dem Anfangsknoten besteht. Noch konkreter formuliert: Wenn ein Petrinetz gerade geladen wurde, oder die Markierung des Petrinetzes editiert wurde, oder der Lösche-EG-Button betätigt wurde, dann besteht der darzustellende pEG aus genau dem Anfangsknoten.
- Wenn durch das Schalten einer Transition t eine Ausgangsmarkierung v in eine Zielmarkierung w überführt wird, ...
  - ... für die noch kein entsprechender Knoten w im pEG existiert, dann wird ein neuer Knoten w und die dazugehörige Kante (v, w) in den pEG eingefügt.
  - ... für die im pEG zwar bereits ein entsprechender Knoten w aber noch keine Kante (v, w) existiert, dann wird eine neue Kante (v, w) in den pEG eingefügt.
  - ... für die im pEG bereits eine Kante (v, w) existiert, die allerdings mit einer anderen Transition beschriftet ist, dann soll eine weitere Kante (v, w) eingefügt werden, die mit t beschriftet wird.
- Wenn die Markierung des Petrinetzes auf die Anfangsmarkierung zurückgesetzt wird (siehe Punkt Reset-Button in Kapitel 3.5), dann hat dies keine Auswirkung auf die Struktur des bisher konstruierten pEG.
- Wenn die Markierung des Petrinetzes editiert wird (siehe entsprechenden Punkt in Kapitel 3.5), dann ändert sich dadurch die Anfangsmarkierung des Petrinetzes. Dementsprechend wird der bisher konstruierte pEG ungültig und muss durch einen neuen pEG, der nur aus dem (neuen) Anfangsknoten besteht, ersetzt werden.

- Wenn sich eine Änderung in Ihrem Datenmodell ergeben hat, weil z.B. durch das Schalten einer Transition eine Kante im pEG hinzugefügt wurde, dann soll *nicht* die Visualisierung komplett gelöscht und durch die Darstellung des aktuellen Modells ersetzt werden, sondern die Änderung des Modells soll *inkrementell* mit der Visualisierung abgeglichen werden. Wenn also z.B. eine Kante im Datenmodell des pEG hinzugefügt wurde, dann soll diese Kante auch in der bereits bestehenden Visualisierung hinzugefügt werden. Dieses Vorgehen hat insbesondere bei Verwendung des automatischen Graphen-Layouts (siehe Kapitel 3.7) den Vorteil, dass beim sukzessiven Aufbau des pEG ein irritierendes „Zucken“ in der Darstellung des Graphen vermieden wird (dieses Zucken würde hingegen auftreten, wenn der Graph bei jeder Änderung komplett neu dargestellt und layoutet werden würde).

Wir empfehlen Ihnen nachdrücklich, bei der Umsetzung der Aufgabenstellung zunächst die oben beschriebene Funktionalität für den pEG zu implementieren (sowie die damit einhergehenden Punkte in Kapitel 3.7), bevor Sie sich mit der Implementierung des Beschränktheits-Algorithmus befassen. Auf diese Weise machen Sie sich mit den prinzipiellen Schritten zur Konstruktion eines Erreichbarkeitsgraphen vertraut und setzen dabei bereits die von Ihnen entwickelten Erreichbarkeitsgraph-Datenstrukturen ein. Zudem erhalten Sie durch die graphische Darstellung des sukzessiven Aufbaus des Erreichbarkeitsgraphen ein visuelles Feedback vom aktuellen Zustand Ihres Datenmodells. Für (kleinere) beschränkte Petrinetze können Sie sogar „händisch“ das prinzipielle Vorgehen beim Aufbau des vollständigen Erreichbarkeitsgraphen durchspielen (siehe dazu auch die Anmerkung am Ende von Kapitel 3.7).

3.7. Darstellung des (partiellen) Erreichbarkeitsgraphen und Interaktion mit dem Petrinetz

Die folgenden Punkte sollen bei der Darstellung eines (ggf. partiellen) Erreichbarkeitsgraphen beachtet werden.

- Die Berechnung eines möglichst übersichtlichen Graphen-Layouts (d.h. die Festlegung der Koordinaten für alle Knoten in der graphischen Darstellung) stellt i.Allg. ein komplexes Problem dar. Die GraphStream Bibliothek stellt standardmäßig jedoch eine Funktionalität zur Verfügung, um ein geeignetes Layout für einen Graphen automatisch berechnen zu lassen. Dieses Auto-Layout sollen Sie zur Darstellung des Erreichbarkeitsgraphen einsetzen[9].
- Der Anfangsknoten des Erreichbarkeitsgraphen soll visuell hervorgehoben werden.

[9] Falls Ihnen das Auto-Layout nicht gefällt, steht es Ihnen natürlich frei, die Berechnung eines geeigneten Layouts selbst zu implementieren. Dann müssen Sie in der GUI allerdings auch eine Möglichkeit zur Verfügung stellen, um zwischen dem Auto-Layout und dem alternativen Layout umschalten zu können.

- Die Beschriftung eines Knotens des Erreichbarkeitsgraphen soll in kompakter Form die Markierung wiedergeben, welche dieser Knoten repräsentiert. Die Beschriftung soll wie folgt aufgebaut werden, um eine Markierung quasi in Tupel-Form zu repräsentieren:
  - Die Stellen des Petrinetzes sollen in alphabetischer Reihenfolge gemäß ihrer ID sortiert werden.
  - Gemäß dieser Sortierung soll die Anzahl der Marken jeder Stelle angegeben werden, jeweils getrennt durch ein „|“ Zeichen und insgesamt eingeschlossen in runden Klammern.
  - Das folgende Beispiel illustriert die Beschriftung einer Markierung:
    Das Petrinetz besteht aus den Stellen: s1, s2, s3, s4, s5, s6, s7.
    Die Markierung m weist den Stellen die folgende Anzahl an Marken zu:
    m(s1)=4, m(s2)=0, m(s3)=5, m(s4)=1, m(s5)=0, m(s6)=2, m(s7)=5.
    Dann muss die Beschriftung des Knotens m wie folgt lauten:
    $$(4|0|5|1|0|2|5)$$
- Die Beschriftung einer Kante (v, w) des Erreichbarkeitsgraphen soll die Transition referenzieren, deren Schalten die Ausgangsmarkierung v in die Zielmarkierung w überführt. Dementsprechend soll sich die Beschriftung aus der ID und dem Namen dieser Transition wie in Kapitel 3.5 beschrieben zusammensetzen.

Ausgelöst durch Mausklicks soll folgende Interaktion zwischen dem Petrinetz und dem (ggf. partiellen) Erreichbarkeitsgraphen stattfinden:

- Wenn im Petrinetz (per linkem Mausklick) eine Transition t geschaltet wird, welche eine Ausgangsmarkierung v in eine Zielmarkierung w überführt, dann sollen im Erreichbarkeitsgraphen der Knoten w (welcher der Zielmarkierung entspricht) sowie die Kante (v, w) (welche der geschalteten Transition t entspricht) visuell hervorgehoben werden.
- Wenn im Erreichbarkeitsgraphen ein linker Mausklick auf einen Knoten v erfolgt, dann soll die entsprechende Markierung im Petrinetz gesetzt werden.

Die letztgenannte Funktionalität ermöglicht es Ihnen, im Petrinetz zu einer beliebigen, während des Schaltens schon einmal erreichten Markierung zurückzuspringen, um von dort aus „einen anderen Weg“ einschlagen zu können. Auf diese Weise können Sie „händisch“ alle von der Anfangsmarkierung aus erreichbaren Markierungen eines Petrinetzes ermitteln und – falls das Petrinetz beschränkt ist – prinzipiell sogar den vollständigen Erreichbarkeitsgraphen „von Hand“ erzeugen. Diese quasi manuelle Vorgehensweise ist natürlich nur für relativ kleine Beispiele praktikabel, da Sie selbst darauf achten müssen, ob Sie tatsächlich alle erreichbaren Markierungen erzeugt haben. Nichtsdestotrotz kann Ihnen das manuelle Schalten von Transitionen und der damit einhergehende schrittweise

Aufbau des Erreichbarkeitsgraphen ein gutes Gespür dafür vermitteln, welche Abläufe bei der Implementierung des Beschränktheits-Algorithmus umzusetzen sind.

3.8. Durchführung der Beschränktheitsanalyse für ein Petrinetz

Folgende Punkte sollen bei der Beschränktheitsanalyse für ein Petrinetz bzw. der Ausführung des Beschränktheits-Algorithmus beachtet werden:

- Die Beschränktheitsanalyse soll für das in der GUI dargestellte Petrinetz (mit der dazugehörigen Anfangsmarkierung) mittels eines *Analyse-Buttons* in der Toolbar gestartet werden können.
- Ein bereits vorhandener (ggf. partieller) Erreichbarkeitsgraph muss vor dem Start des Beschränktheits-Algorithmus gelöscht werden. Zudem muss die Markierung des Petrinetzes wieder auf die Anfangsmarkierung zurückgesetzt werden.
- Nach Beendigung der Ausführung des Beschränktheits-Algorithmus soll ein Dialogfenster über das Ergebnis informieren, indem es entweder „Das Petrinetz ist beschränkt“ oder „Das Petrinetz ist unbeschränkt“ meldet. Zusätzlich soll das Ergebnis, ggf. ergänzt um weitere relevante Informationen, im Textbereich der GUI ausgegeben werden.
- Wenn das Petrinetz beschränkt ist, soll der vom Algorithmus zurückgelieferte Erreichbarkeitsgraph in der GUI dargestellt werden.
- Wenn das Petrinetz unbeschränkt ist, soll der vom Algorithmus zurückgelieferte partielle Erreichbarkeitsgraph in der GUI dargestellt werden (vgl. den entsprechenden Punkt in Kapitel 3.3). In diesem Graphen soll zudem der vom Algorithmus ermittelte Grund für die Unbeschränktheit kenntlich gemacht werden, nämlich der vom Algorithmus zurückgelieferte Pfad von der Anfangsmarkierung zu Markierungen m und m'. Sowohl der Pfad als auch die Markierungen m und m' (also die beiden entsprechenden Knoten) sollen im partiellen Erreichbarkeitsgraphen jeweils visuell hervorgehoben werden. Auch dieser partielle Erreichbarkeitsgraph soll sich (wie üblich) durch das Schalten entsprechender Transitionen im Petrinetz sukzessive erweitern lassen (siehe Kapitel 3.6).

3.9. Durchführung der Beschränktheitsanalyse für eine Menge von Petrinetz Dateien

Ihr Programm soll die Beschränktheitsanalyse automatisiert für eine Menge von Petrinetz Dateien durchführen können. Diese Art der Stapelverarbeitung soll wie folgt realisiert werden:

- Bei Auswahl des Eintrags „*Analyse mehrerer Dateien...*“ im Datei-Menü soll ein

Dateiauswahl-Dialog angezeigt werden, welcher die Auswahl *mehrerer* PNML Dateien ermöglicht. Beachten Sie hierbei auch die Hinweise zum Dateiauswahl-Dialog in Kapitel 3.4; insbesondere soll sich auch hier der Dialog standardmäßig in dem Verzeichnis mit den Beispiel-PNML Dateien öffnen.

- Die ausgewählten PNML Dateien sollen nacheinander in alphabetischer(!) Reihenfolge verarbeitet werden. Über die als nächstes zu verarbeitende PNML Datei (also den Dateinamen) soll im Textbereich informiert werden.
- Auf jede PNML Datei bzw. das darin spezifizierte Petrinetz soll der Beschränktheits-Algorithmus angewandt werden.
- Dabei sollen (wie sonst auch) die Meldungen des Algorithmus und das Ergebnis (d.h. beschränkt oder unbeschränkt) im Textbereich ausgegeben werden. Allerdings soll *kein* Ergebnis-Dialogfenster angezeigt werden und es soll *keine* graphische Anzeige des (ggf. partiellen) Erreichbarkeitsgraphen erfolgen.
- Nachdem alle PNML Dateien verarbeitet wurden, soll das Gesamtergebnis zeilenweise im Textbereich ausgegeben werden (auch hier wieder in alphabetischer Reihenfolge). Wenn das Petrinetz beschränkt ist, soll die Anzahl der Knoten und Kanten des EG mit ausgegeben werden. Wenn das Petrinetz hingegen unbeschränkt ist, soll das vom Beschränktheits-Algorithmus zurückgelieferte Ergebnis mit ausgegeben werden, nämlich die Länge des Pfades und der Pfad selbst sowie die Knoten m und m' (in Tupel-Form, siehe S. 22).
  Eine Ausgabezeile soll prinzipiell folgende Form haben:

```
<Datei> <beschränkt>     <Anzahl_Knoten> / <Anzahl_Kanten>
                       | <Pfadlänge>:<Pfad>; <m>, <m'>
```

  Die konkrete Textausgabe sollte dementsprechend wie folgt aussehen (hier exemplarisch illustriert mit vier fiktiven PNML Dateien):

```
                     |            | Knoten / Kanten bzw.
Dateiname            | beschränkt | Pfadlänge:Pfad; m, m'
---------------------|------------|-----------------------------------
EinPetrinetz.pnml    | ja         | 8 / 14
ErstesPetrinetz.pnml | nein       | 4:(t2,t8,t3,t1); (1|0|2),   (1|1|0)
MeinPetrinetz.pnml   | ja         | 6 / 13
NochEinNetz.pnml     | nein       | 3:(t5,t1,t9);    (1|1|0|2), (2|1|2|3)
```

  Beachten Sie bei der Darstellung der Ausgabe insbesondere folgende Punkte:

  – In dem PDF „Dokumentation der Beispiele“ ist eine korrekte Ausgabe der Stapelverarbeitung bei Anwendung auf alle Beispieldateien dargestellt. Die Ausgabe Ihres Programms sollte also genau dieser Darstellung entsprechen (bis auf alternative Knoten und Pfade, siehe letzter Punkt).

- Wie oben beispielhaft illustriert, soll *nicht* der gesamte Pfad der Datei, sondern nur der Dateiname (inkl. Dateiendung) ausgegeben werden.
- Die Ausgabe soll erfolgen, *nachdem* alle PNML Dateien verarbeitet wurden, d.h. das Gesamtergebnis soll quasi „am Stück“ ausgegeben werden, ohne von anderen Ausgaben unterbrochen zu werden.
- Die Ausgabe soll übersichtlich dargestellt werden, d.h. die jeweiligen Angaben sollen exakt untereinander stehen. Daher sollte eine Schriftart mit fixer Zeichenbreite verwendet werden.
- Mit Hilfe der Java-Klasse `Formatter` lassen sich einzelne Strings z.B. so formatieren, dass sie eine feste Mindestlänge (width) haben. Damit lässt sich relativ einfach eine übersichtliche Ausgabe erzeugen. Von der Verwendung von Tabs raten wir hingegen ab, da deren korrekter Einsatz relativ kompliziert und fehleranfällig ist.
- Die Ausgabe ist bzgl. der beschränkten Petrinetze eindeutig. Bei den unbeschränkten Petrinetzen kann es allerdings (in einigen Beispielen) mehrere mögliche Pfade geben, auf denen dann entsprechende Knoten m und m' liegen. Die Ausgabe ist also für manche Beispiele bzgl. der Knoten und Pfade nicht eindeutig, so dass Sie sich frei für eine der möglichen Knoten-Pfad-Alternativen entscheiden können.

Diese Form der Stapelverarbeitung können Sie bereits während der Entwicklung Ihres Programms sinnvoll einsetzen, denn damit können Sie schnell und leicht reproduzierbar für eine größere Anzahl an Petrinetzen (z.B. die von uns zur Verfügung gestellten Beispieldateien) überprüfen, ob Ihre Implementierung des Beschränktheits-Algorithmus die korrekte Entscheidung trifft.

4. Dokumentationsrichtlinien

Zu einem Software-Projekt gehört immer auch eine Dokumentation. Daher ist auch die Dokumentation Ihres Projekts ein Bestandteil der Prüfungsleitung und fließt mit in die Gesamtnote ein. Neben dem eigentlichen Programm werden wir uns also auch ausführlich Ihrer Dokumentation widmen. Sie sollten daher bei der Gestaltung Ihrer Programmdokumentation die gleichen Maßstäbe ansetzen, nach denen Sie beispielsweise eine schriftliche Seminarausarbeitung erstellen würden. Dennoch ist die Dokumentation nur einer von mehreren Bestandteilen Ihrer Gesamtleistung, das ausführbare Programm bildet weiterhin den Hauptteil. Orientieren Sie sich daher an der folgenden Empfehlung: Wenden Sie die Zeit und Mühe auf, die für eine Dokumentation nach den im Folgenden beschriebenen Richtlinien erforderlich ist; aber verlieren Sie dabei nicht aus dem Auge, dass die Dokumentation nicht den Kern des Projekts darstellt.

Ihre Dokumentation besteht aus drei Teilen: einer kurzen Einführung in das Programm (PDF-Dokumentation), der Javadoc-Dokumentation und dem kommentierten Programmcode.

4.1. Einführung in das Programm (PDF-Dokumentation)

Beschreiben Sie den grundlegenden Aufbau Ihres Programms (welches sind die wichtigsten benutzten Datenstrukturen und welche Klassen erledigen welche Aufgaben) und begründen Sie Ihre Designentscheidungen. Gehen Sie an dieser Stelle nicht auf Implementierungsdetails ein. Ihre Einführung sollte beispielsweise auch von einem Leser nachvollziehbar sein, der zwar grundlegende Programmiererfahrungen besitzt, aber die Programmiersprache Java nicht beherrscht.

Ein weiterer wichtiger Punkt der Dokumentation ist die Beschreibung des Beschränktheits-Algorithmus. Stellen Sie dazu den von Ihnen implementierten Algorithmus in Pseudocode dar und erläutern Sie die Abläufe anhand des Pseudocodes.

Die in der Aufgabenstellung beschriebene, geforderte Funktionalität soll nicht nochmals gesondert erläutert werden. Nur falls Ihr Programm über weitere optionale Funktionalität verfügt, erläutern Sie bitte deren Bedienung.

Die gesamte PDF-Dokumentation soll in Deutsch geschrieben werden und sollte maximal 10 DIN-A4-Seiten umfassen. Folgende Gliederungspunkte erwarten wir:

1. Einleitung
2. ggf. Bedienungsanleitung (nur wenn zusätzliche Funktionalitäten implementiert wurden)
3. Beschreibung der Programmstruktur

4. Beschreibung des Beschränktheits-Algorithmus (Pseudocode und Erläuterungen)

4.2. Javadoc-Dokumentation

Sämtliche nichtprivate Elemente (Klassen, Methoden, Attribute,...) sind über Javadoc zu dokumentieren (private Elemente benötigen nicht zwingend einen Kommentar; manchmal kann dies jedoch praktisch sein). Erläutern Sie in je zwei bis drei prägnanten Sätzen, welche Aufgabe von der betreffenden Klasse bzw. Methode erfüllt wird, sowie die Einordnung der Klasse/Methode in den Gesamtkontext des Programms. Denken Sie daran, bei Methoden auch die Parameter und Rückgabewerte zu beschreiben.

Zur Erstellung der Javadoc-Dokumentation benutzen Sie das Javadoc-Werkzeug und erzeugen Sie Javadoc für alle Elemente (Sichtbarkeit „package“). Die generierte HTML-Dokumentation ist ebenfalls abzugeben. Die Javadoc-Dokumentation kann wahlweise in Deutsch oder in Englisch verfasst werden.

4.3. Kommentierung des Programmcodes

Kommentieren Sie zudem auch Ihren Programmcode (wahlweise in Deutsch oder Englisch). Das soll nicht heißen, dass Sie zu jeder Anweisung einen Kommentar schreiben müssen, aber Ihr Programm muss mit Hilfe der Kommentare soweit verständlich sein, dass Leser Ihre Lösung ohne ein langwieriges Hineindenken in Ihre Java-Konstrukte nachvollziehen können.

A. PNML Dateiformat

Für das Programmierpraktikum verwenden wir eine stark vereinfachte Version des PNML (Petri Net Markup Language) Dateiformats (Version 1.3.2). Die enthaltenen Informationen sind jedoch mit dem vollständigen PNML Dateiformat, das z.B. von Werkzeugen wie *WoPeD*[10] verwendet wird, kompatibel. Somit können Petrinetze, die mit WoPeD oder kompatiblen Werkzeugen erstellt wurden, von Ihrem Anwendungsprogramm geöffnet werden[11].

Der von uns zur Verfügung gestellte PNML Parser (Java Klasse `PNMLWopedParser`) verarbeitet nur diejenigen Elemente einer PNML Datei, die für die Aufgabenstellung dieses Programmierpraktikums relevant sind; alle anderen Elemente werden ignoriert.

In Abb. 8 ist ein sehr einfaches Petrinetz dargestellt. Das Listing 1 auf der folgenden Seite zeigt die dazugehörige PNML Datei in dem Format, das der zur Verfügung gestellte PNML Parser verarbeiten kann.

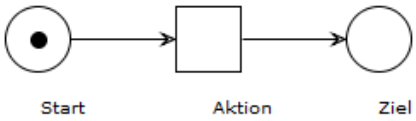


Abbildung 8: Petrinetz, das durch die PNML Datei in Listing 1 spezifiziert wird

[10] `http://www.woped.org`
[11] Ein Großteil der von uns zur Verfügung gestellten Beispiele wurde mit WoPeD v3.7.1 erstellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
  <net>
    <place id="p1">
      <name>
        <text>Start</text>
      </name>
      <initialMarking>
          <text>1</text>
      </initialMarking>
      <graphics>
        <position x="200" y="150"/>
      </graphics>
    </place>
    <place id="p2">
      <name>
        <text>Ziel</text>
      </name>
      <initialMarking>
          <text>0</text>
      </initialMarking>
      <graphics>
        <position x="400" y="150"/>
      </graphics>
    </place>
    <transition id="t1">
      <name>
        <text>Aktion</text>
      </name>
      <graphics>
        <position x="300" y="150"/>
      </graphics>
    </transition>
    <arc id="a1" source="p1" target="t1"/>
    <arc id="a2" source="t1" target="p2"/>
  </net>
</pnml>
```

Listing 1: PNML Datei, die das Petrinetz in Abb. 8 spezifiziert

Das beschriebene Netz enthält zwei Stellen (*place*), die mit `Start` und `Ziel` beschriftet sind, und eine Transition (*transition*), die mit `Aktion` beschriftet ist. Von der mit `Start` beschrifteten Stelle führt eine gerichtete Kante (*arc*) zur Transition `Aktion`, und von dieser Transition führt eine zweite Kante zur Stelle, die mit `Ziel` beschriftet ist.

Der XML Baum eines PNML Dokuments beginnt mit dem XML Element `pnml`. In-

nerhalb dieses Elements können mehrere Netze mit Hilfe von `net` Elementen definiert werden. Für Ihre Implementierung dürfen Sie annehmen, dass jedes Dokument nur ein Petrinetz enthält. Das `net` Element enthält die Beschreibungen von Transitionen, Stellen und Kanten in beliebiger Reihenfolge.

Jede Stelle wird durch ein `place` Element beschrieben. Dieses Element enthält ein Attribut `id` mit einem String als Wert, der die Stelle eindeutig identifiziert und in Kantenbeschreibungen verwendet wird. Jedes `place` Element hat die Kindelemente `name`, `initialMarking` und `graphics`. Das `name` Element hat ein Kindelement `text`, das als XML Textknoten den Beschriftungstext der Stelle enthält. Das `initialMarking` Element hat ein Kindelement `text`, das als XML Textknoten die Markenanzahl der Stelle in der Anfangsmarkierung enthält. Das `graphics` Element hat ein Kindelement `position`, dass über Attribute `x` und `y` verfügt, welche die Mittelpunkt-Koordinaten der Stelle für die graphische Darstellung angeben.

```
<place id="p1">
  <name>
    <text>Start</text>
  </name>
  <initialMarking>
      <text>1</text>
  </initialMarking>
  <graphics>
    <position x="200" y="150"/>
  </graphics>
</place>
```

Listing 2: Definition einer Stelle in PNML

Jede Transition wird durch ein `transition` Element beschrieben. Der Aufbau eines `transition` Elements und seiner Kindelemente entspricht prinzipiell dem eines `place` Elements, mit dem einzigen Unterschied, dass kein `initialMarking` Element vorhanden ist (da eine Transition keine Marken tragen kann).

```
<transition id="t1">
  <name>
    <text>Aktion</text>
  </name>
  <graphics>
    <position x="300" y="150"/>
  </graphics>
</transition>
```

Listing 3: Definition einer Transition in PNML

Jede Kante wird durch ein `arc` Element beschrieben. Jedes `arc` Element hat die Attribute `id`, `source` und `target`. Das Attribut `id` hat einen String als Wert, der die Kante

eindeutig identifiziert. Das Attribut `source` enthält die String-ID des Startelements der Kante und das Attribut `target` enthält die String-ID des Endelements der Kante.

```
<arc id="a1" source="p1" target="t1"/>
```

Listing 4: Definition einer Kante in PNML

Alle `id` Elemente innerhalb einer PNML Datei müssen eindeutig sein.

B. GraphStream Bibliothek

In diesem Kapitel geben wir Ihnen einige Hinweise und Tipps zur GraphStream Bibliothek.

B.1. Version und JAR Dateien

Es wird die GraphStream Bibliothek in der **Version 1.3** verwendet.

Hinweis: Vor wenigen Tagen wurde Version 2.0 der GraphStream Bibliothek veröffentlicht. Die Dokumentation auf den GraphStream Web-Seiten wurde z.T. hinsichtlich Version 2.0 angepasst. Am Ende jeder betroffenen Seite befindet sich allerdings im Abschnitt „Other version of this document" noch ein Link auf die entsprechende Dokumentation für Version 1.3. Achten Sie daher bitte beim Lesen der GraphStream Web-Seiten darauf, ob sich die Informationen auf die im Programmierpraktikum zu verwendende Version 1.3 beziehen.

Wir stellen Ihnen alle benötigten GraphStream JAR Dateien mittels unseres Basis-Projekts zur Verfügung (siehe Kapitel C.5). Im Unterverzeichnis `libs/GraphStream` des Basis-Projekts befinden sich drei weitere Unterverzeichnisse mit den entsprechenden Komponenten der GraphStream Bibliothek (`core`, `ui` und `algo`). Jedes Unterverzeichnis beinhaltet die JAR Dateien mit der entsprechenden Komponente der GraphStream API, dem dazugehörigen Quellcode sowie die Javadoc Dokumentation. In Kapitel C.7.4 wird beschrieben, wie Sie die GraphStream API in Ihr Programm einbinden. Der dazugehörigen Quellcode und das Javadoc werden dabei automatisch mit eingebunden, da dies im Basis-Projekt bereits entsprechend konfiguriert ist.

B.2. Links zu wichtigen Web-Seiten

Homepage der GraphStream Bibliothek:
`https://graphstream-project.org`

Dokumentation (Übersicht):
`https://graphstream-project.org/doc/`

wichtige Tutorials:
`https://graphstream-project.org/doc/Tutorials/Getting-Started/1.3/`
`https://graphstream-project.org/doc/Tutorials/Storing-retrieving-and-displaying-data-in-graphs/`
`https://graphstream-project.org/doc/Tutorials/Graph-Visualisation/1.3/`

FAQ (relativ kurz):
`https://graphstream-project.org/doc/FAQ/`

Javadoc API Dokumentation:
`https://data.graphstream-project.org/api/gs-core/1.3/`
`https://data.graphstream-project.org/api/gs-algo/1.3/`
`https://data.graphstream-project.org/api/gs-ui/`

CSS Referenz (mit Links zu Beispiel-Code unterhalb der Abbildungen)
`https://graphstream-project.org/doc/Advanced-Concepts/GraphStream-CSS-Reference/1.3/`

B.3. Tastaturbefehle

Folgende Tastaturbefehle funktionieren standardmäßig in jedem GraphStream Panel.

Tasten	**Funktion**
$\leftarrow, \rightarrow, \uparrow, \downarrow$	Ansicht scrollen
Umschalten + $\leftarrow, \rightarrow, \uparrow, \downarrow$	Ansicht ein größeres Stück scrollen
Bild hoch, Bild runter	Skalierung vergrößern bzw. verkleinern
Umschalten + R	Skalierung zurücksetzen
Alt + $\leftarrow, \rightarrow$	Graph gegen bzw. im Uhrzeigersinn drehen

Die Implementierung dieser Befehle befindet sich in der Klasse:
`org.graphstream.ui.view.util.DefaultShortcutManager`

B.4. ProPra Demo-Programm

Wir stellen Ihnen ein kleines Demo-Programm in Form eines Eclipse Projekts zur Verfügung (siehe Kapitel C.6), das die Einbindung von GraphStream in eine Swing-GUI demonstriert. Gestartet wird das Demo-Programm mittels der main-Methode in:

`mypackage.ProPra_WS20_Demo`

Dieses Demo-Programm illustriert insbesondere solche Punkte, die in den vorhandenen GraphStream Tutorials nicht oder nur unzureichend erklärt werden. Sie können sich von diesem Demo-Programm prinzipiell viele Punkte für Ihre eigene Programmentwicklung abgucken; beachten Sie jedoch, dass Sie den einfachen Aufbau des Demo-Programs evtl. nicht in allen Aspekten eins-zu-eins auf Ihr deutlich komplexeres Programm übertragen können.

Das Demo-Programm illustriert u.a. die folgenden Punkte der GraphStream API:

- Erzeugung eines Graphen (konkret eines Multigraphen, der mehrere Kanten zwischen zwei Knoten beinhalten kann)
- Hinzufügen von Knoten und Kanten zum Graphen
- Hinzufügen von Attributen zu Graph-Elementen
- Abfragen und Ändern der Attribute von Graph-Elementen
- Verwendung einer CSS Style Sheet Datei, welche das optische Erscheinungsbild des Graphen definiert (Farben, Formen, etc.)
- Einbindung des Graphen in ein JPanel
- Auswahl des passenden Renderers mit Unterstützung für Multigraphen und CSS Styling
- Auswahl des passenden Threading-Modells
- Aktivieren bzw. Deaktivieren des Auto-Layouts
- Verarbeitung von linken Mausklicks auf Knoten inkl. Zurücklieferung der String-ID des angeklickten Knotens

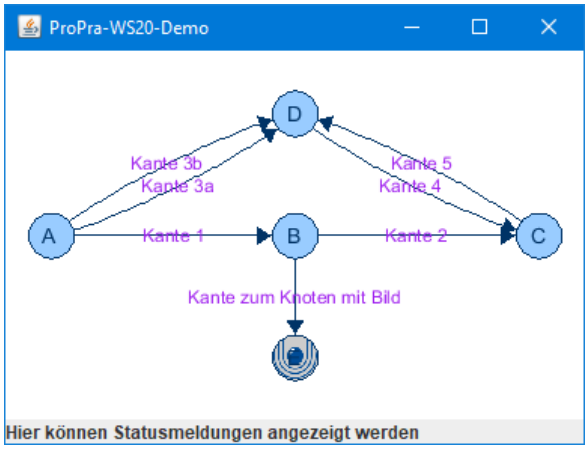


Abbildung 9: ProPra Demo-Programm

B.5. Erzeugung einer ausführbaren JAR Datei

Das ProPra Demo-Projekt ist bereits so konfiguriert, dass sich daraus direkt eine ausführbare JAR Datei[12] (Runnable JAR) erzeugen lässt, welche alle benötigten Dateien

[12]Eine JAR Datei entspricht im Prinzip einem ZIP Archiv und lässt sich daher auch mit jedem ZIP-Programm öffnen, um z.B. den Inhalt zu inspizieren.

enthält (auch die GraphStream CSS Datei und z.B. Grafik Dateien).

Somit können Sie anhand des Demo-Projekts direkt nachvollziehen, wie Sie Ihr eigenes Projekt konfigurieren müssen, damit Sie es (inkl. aller Ressourcen wie Libraries, CSS Datei und Grafik Dateien) in Form eines Runnable JAR exportieren können. Das heißt, Ihr Programm besteht nach dem Export nur noch aus einer einzigen JAR Datei, die sämtliche benötigte Dateien enthält, um „stand-alone“ auf einem beliebigen System gestartet werden zu können.

Bitte beachten Sie noch folgende hilfreiche Hinweise:

- Die JAR Datei soll natürlich *nicht* die Beispiel-Dateien enthalten. Um sicherzustellen, dass die Beispiele bei Ausführung der JAR Datei wie gewohnt (d.h. wie beim Run des Projekts in Eclipse) gefunden werden können, muss die JAR Datei direkt im Hauptverzeichnis des Eclipse-Projekts liegen.
- Der im Demo-Programm konstruierte Graph enthält einen Knoten, der mit einem Bild versehen ist, das aus einer Grafik Datei geladen wird. Dies dient der Demonstration des Ladens einer Ressource aus der JAR Datei.
- Das Demo-Programm gibt zudem auf der Konsole eine Meldung aus, ob der relative Pfad zum Beispiel-Verzeichnis korrekt ermittelt werden konnte.
- Damit die CSS Datei auch beim Start des Programms mittels JAR Datei gefunden wird, muss der Pfad zur CSS Datei so angegeben werden, wie es das Demo-Projekt illustriert:

  ```
  private static String CSS_FILE = "url("
   + MyGraph.class.getResource("/graph.css") + ")";
  ```

- Darüber hinaus ist es wichtig, innerhalb der CSS Datei den Pfad zu einer Grafikdatei passend anzugeben. Beispielsweise muss der Pfad für die Datei `ProPra-WS20-Demo/resources/images/logo_grau.png` wie folgt angegeben werden:

  ```
  fill-image: url('images/logo_grau.png');
  ```

  Das Verzeichnis `resources` wird also nicht mit angegeben (siehe nächsten Punkt).
- In der Projektkonfiguration des Demo-Projekts im Abschnitt Properties → Java Build Path → Source ist – neben dem obligatorischen Verzeichnis `src` – zusätzlich das Verzeichnis `resources` mit angegeben. Dadurch werden die in diesem Verzeichnis abgelegten Dateien (CSS Datei und Grafik Dateien) auch dann problemlos gefunden, wenn alle Dateien des Projekts in einer JAR Datei gespeichert sind.
- Wenn Sie das Runnable JAR in Eclipse mittels Export erzeugen, wählen Sie die Option *Package required libraries into generated JAR* aus, damit auch die JAR Dateien der Libraries direkt mit in das Runnable JAR gepackt werden.

B.6. Weitere Hinweise und Tipps

IDs in PNML und GraphStream GraphStream verwendet String-IDs zur eindeutigen Identifikation von Elementen des Graphen. Da einige Methoden, wie z.B. `buttonPushed()`, keine Objekt-Referenzen sondern nur String-IDs liefern, muss man sich häufig mit diesen IDs auseinandersetzen. In den PNML Dateien werden ebenfalls eindeutige String-IDs für alle Elemente eines Petrinetzes definiert. Daher empfehlen wir Ihnen nachdrücklich, beim Aufbau eines Petrinetz-Graphen die eingelesenen PNML-IDs auch direkt als GraphStream-IDs zu verwenden. Dadurch vermeiden Sie unnötige Verwechslungen und können bei Problemen mit der GraphStream Datenstruktur direkt auf das entsprechende Element in der PNML Datei rückschließen, wodurch die Fehlerfindung deutlich vereinfacht werden kann.

Gewünschter Informationsaustausch in der Newsgroup Sie können in der Newsgroup gerne Informationen austauschen, die sich mit der Verwendung der GraphStream Bibliothek befassen. Da die GraphStream Dokumentation an einigen Stellen relativ kurz gehalten, unvollständig oder ungenau ist, können Sie alle von so einem Informationsaustausch profitieren. Wir würden es daher *sehr begrüßen*, wenn Sie Tipps und Hinweise zu GraphStream rege in der Newsgroup austauschen würden. Wenn es hilfreich erscheint, dürfen Sie auch (auf das Wesentliche reduzierte) Mini-Programme in der Newsgroup zur Verfügung stellen, um bestimmte Funktionalitäten von GraphStream geeignet zu demonstrieren. Natürlich dürfen Sie nicht den sonstigen (d.h. GraphStream unabhängigen) Quellcode Ihres Programms mit anderen teilen; jeder soll ja prinzipiell sein Programm eigenständig entwickeln. Grundsätzlich gilt also folgende Prämisse: Die GraphStream Bibliothek stellt (nur) ein mächtiges Hilfsmittel dar, mit dem Sie zentrale Anforderungen an Ihr Programm relativ einfach umsetzen können; und da die zentralen Anforderungen (und nicht GraphStream) im Vordergrund des Programmierpraktikums stehen, können und sollen Sie sich gerne intensiv über den Einsatz dieses Hilfsmittels austauschen.

Erweiterte Behandlung von Maus-Ereignissen Das von uns zur Verfügung gestellte Demo-Programm sieht nur die Behandlung von (einfachen) linken Mausklicks vor. Zur Umsetzung der Aufgabenstellung benötigen Sie auch *keinerlei* darüber hinausgehende Maus-Funktionalitäten (wie z.B. Rechtsklick, Doppelklick, etc.). Falls Sie sich dennoch intensiver mit der Behandlung von Maus-Ereignissen in GraphStream beschäftigen möchten, stellt die folgende Klasse einen interessanten Einstiegspunkt dar:
`org.graphstream.ui.view.util.DefaultMouseManager`

C. Programmierumgebung

Für das Praktikum benötigen Sie einen Computer mit Internetanbindung. Als Betriebssysteme sind Windows, Linux (diverse Distributionen) und macOS (OS X) geeignet.

Wie Sie die Programmierumgebung einrichten können, wird in den folgenden Abschnitten erläutert.

C.1. Java

Die zu verwendende Programmiersprache ist Java. Gute Kenntnisse in der Java-Programmierung sind daher unbedingt erforderlich. Aus Kompatibilitätsgründen muss Java in der Version 8 verwendet werden. Wenn Sie noch kein Java 8 auf Ihrem Computer installiert haben, dann installieren Sie bitte von

`https://www.oracle.com/technetwork/java/javase/downloads/#JDK8`

das aktuelle Release (8u261, Stand September 2020) des Java SE 8 Development Kits (JDK). Falls die Lizenzbedingungen von Oracle für Sie problematisch sind, können Sie alternativ auch die Open Source Variante von

`https://adoptopenjdk.net/releases.html?variant=openjdk8&jvmVariant=hotspot`

installieren, wobei Sie hier das aktuellen Release (jdk8u265-b01, Stand September 2020) des OpenJDK 8 (LTS) mit JVM Hotspot wählen sollten. Achten Sie bitte jeweils darauf, das Java Development Kit (JDK) zu verwenden, und nicht (nur) das Java Runtime Environment (JRE).

Bei der Implementierung Ihres Programms dürfen Sie nur eigene Klassen und solche aus der Java-Standard-Bibliothek verwenden. Des Weiteren sind die zur Verfügung gestellten GraphStream Bibliotheken (siehe Kapitel B und C.7.4) einzubinden. Die Nutzung anderer fremder Klassenbibliotheken ist nicht zulässig.

C.2. Eclipse

Als Entwicklungsumgebung sollen Sie Eclipse einsetzen. Eclipse kann von der Webseite

`https://www.eclipse.org/downloads/eclipse-packages/`

heruntergeladen werden. Empfohlen wird, die aktuelle Version (2020-06 R, Stand September 2020) von „Eclipse IDE for Java Developers“ zu benutzen.

Es besteht die Möglichkeit, dass Eclipse nach der Installation das JDK nicht automatisch erkannt hat. Prüfen Sie unter Window → Preferences → Java → Installed JREs, ob das von Ihnen installierte JDK (siehe C.1) in der Tabelle „Installed JREs“ aufgelistet und

ausgewählt ist. Fügen Sie es andernfalls hier hinzu und entfernen Sie bei der Gelegenheit das „einfache“ JRE am besten gleich aus der Liste (vergleiche Abb. 10).

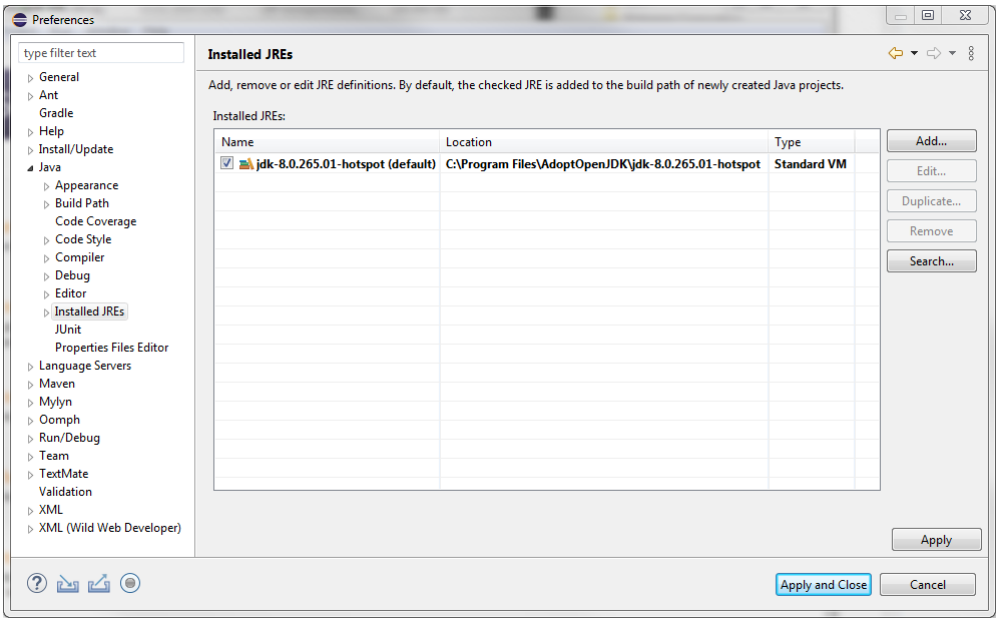


Abbildung 10: Registrierung des JDK in Eclipse (dargestellte Java Version ist ggf. veraltet)

C.3. Subversion (SVN)

Des Weiteren wird das Versionskontrollsystem *Subversion (SVN)* eingesetzt. Ein Versionskontrollsystem gehört heute zu den notwendigen Werkzeugen bei der Erstellung größerer Programmierprojekte. Durch die Verwendung können Dateien (insbesondere Dateien, die Programm-Quelltext enthalten) in verschiedenen Versionen gespeichert werden. Arbeiten mehrere Entwickler an ein und demselben Projekt, ermöglicht die Versionsverwaltung den Zugriff auf den Quelltext.

Um innerhalb von Eclipse SVN benutzen zu können, sollten Sie das Subversive-Plugin installieren. Dazu wählen Sie in Eclipse den Menü-Befehl Help → Install New Software... und fügen mittels des Add... Buttons die folgende Location hinzu, welche Sie anschließend als *Work-with* Site auswählen können:

```
https://download.eclipse.org/technology/subversive/4.0/update-site/
```

In der Abteilung *Subversive SVN Team Provider Plugin* wählen Sie dann *Subversive SVN Team Provider* aus (siehe Abb. 11).

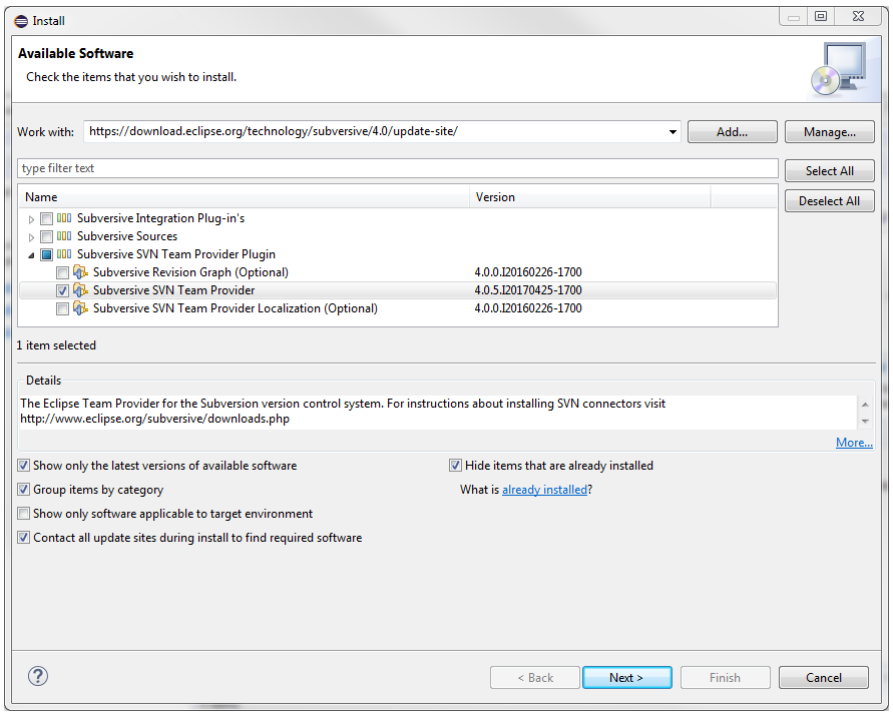


Abbildung 11: Installation Subversive

Danach müssen Sie noch einen sogenannten SVN Connector installieren, empfohlen wird *SVN Kit 1.8.14*. Rufen Sie dazu gleich die Einstellung Window → Preferences → Team → SVN auf und wählen unter *SVN Connector* den passenden aus (siehe Abb. 12).

C.4. Einrichten des SVN Repositories

Nun müssen Sie die benötigten SVN Repository Locations einrichten. In einigen der folgenden URLs müssen Sie die hier exemplarisch verwendete Matrikelnummer `1234567` noch durch Ihre eigene Matrikelnummer ersetzen.

- `https://propra1.fernuni-hagen.de/propra_ws20/public`
  Das Unterverzeichnis `documentation` enthält diese Aufgabenstellung, die Dokumentation der Beispiele, ein Paper zu Petrinetzen [DJ01] sowie die Selbstständigkeitserklärung.

  In `resources` finden Sie - jeweils in einem eigenen Projekt - das Basis-Projekt (siehe Kapitel C.5), das die Ressourcen enthält, auf die Sie in Ihrem eigenen Projekt zugreifen können sollen, sowie ein Demo-Projekt (siehe Kapitel C.6).
- `https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567`

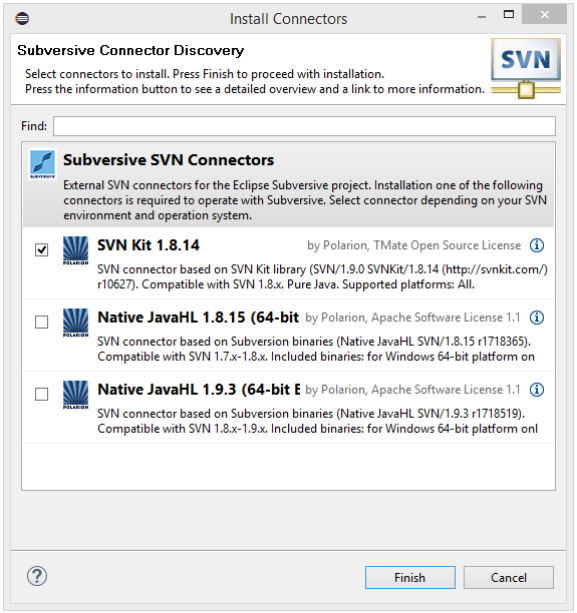


Abbildung 12: Installation SVN Connector

Dies ist Ihr persönliches, nur für Sie selbst und die Betreuer einsehbares Verzeichnis, in das Sie (am besten vom ersten Tag an) Ihre Implementierung unter folgendem Projektnamen ablegen sollten (beachten Sie dabei unbedingt die Hinweise zum Namensschema in Kapitel C.7 sowie die Hinweise in Kapitel C.7.6):

```
Petrinets_<Matr-Nr>_<Nachname>_<Vorname>
```

Wechseln Sie dafür in die *SVN Repository Exploring Perspective* (Window → Perspective → Open Perspective → Other... → SVN Repository Exploring). In dem View *SVN Repositories* wählen Sie über einen Rechtsklick den Befehl New → Repository Location.

Füllen Sie den sich öffnenden Dialog wie in Abb. 13 aus. Verwenden Sie für die Authentifizierung Ihren FernUni-Account (q<Matr-Nr> / <Ihr FernUni-LDAP-Kennwort>).

Verfahren Sie auf dieselbe Weise für den zweiten Link, also für die SVN Repository Location, die Ihr persönliches Verzeichnis enthält.

C.5. ProPra-WS20-Basis

Für die Implementierung Ihres Programms benötigen Sie unbedingt noch unser vorbereitetes Basis-Projekt aus dem SVN Repository als eigenständiges Projekt in Ihrem Eclipse-Workspace. Das Basis-Projekt enthält die GraphStream Bibliothek (siehe Kapitel B), den

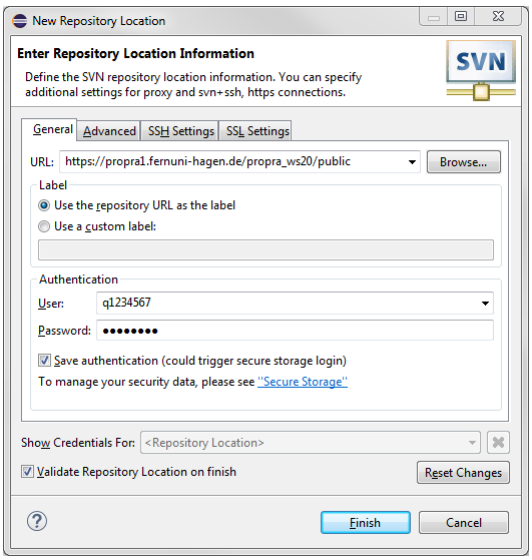


Abbildung 13: Einrichten einer SVN Repository Location

PNML Parser[13] (Klasse `PNMLWopedParser`, siehe Kapitel A) sowie ein Verzeichnis mit den von uns in Form von PNML Dateien zur Verfügung gestellten Beispiel-Petrinetzen (siehe dazu das PDF „Dokumentation der Beispiele“).

Um das Basis-Projekt zu erhalten, wechseln Sie in die *SVN Repository Exploring Perspective* (Window → Perspective → Open Perspective → Other... → SVN Repository Exploring). Klappen Sie in dem View *SVN Repositories* die Ordnerstruktur des Public-Repositories auf (siehe Abb. 14) und wählen Sie in dem Verzeichnis `resources/ProPra-WS20-Basis` mittels Rechtsklick den Befehl Find/Check Out As... aus. Bestätigen Sie den folgenden Dialog, ohne den vorgeschlagenen Projektnamen (ProPra-WS20-Basis) zu verändern.

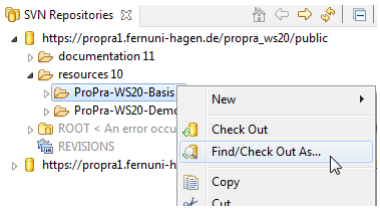


Abbildung 14: Herunterladen des Projekts ProPra-WS20-Basis

[13] Neben dem Parser ist in dem Projekt auch noch ein entsprechender Writer (Klasse `PNMLWopedWriter`) enthalten, welcher mittels Vererbung so erweitert werden kann, dass er das Datenmodell eines Petrinetzes in eine PNML Datei schreibt. Dieser Writer wird für die Umsetzung der Aufgabenstellung jedoch *nicht* benötigt.

C.6. ProPra-WS20-Demo

Auf dieselbe Weise, wie Sie das Basis-Projekt ausgecheckt haben, können Sie auch unser Demo-Projekt im Verzeichnis `resources/ProPra-WS20-Demo` aus dem SVN auschecken.

In diesem Demo-Projekt ist die Implementierung eines kleinen GUI-Anwendungsprogramms enthalten, das die Einbindung der GraphStream Bibliothek veranschaulicht (siehe Kapitel B.4). Nachdem Sie das Basis-Projekt und das Demo-Projekt ausgecheckt haben, können Sie das Demo-Programm direkt starten, da in dem Demo-Projekt der Buildpath bereits passend konfiguriert ist.

C.7. Ihr Java-Projekt

Zu Beginn sollten Sie ein neues Java-Projekt erzeugen, dessen Name dem folgenden Schema entsprechen muss:

```
Petrinets_<Matr-Nr>_<Nachname>_<Vorname>
```

Die Platzhalter `<Matr-Nr>`, `<Nachname>` und `<Vorname>` sind entsprechend zu ersetzen, wobei folgende Punkte zu beachten sind:

- Die Matrikelnummer soll nur aus Ziffern bestehen (also nicht das `Q` enthalten).
- Wenn mehrere Vornamen angegeben werden, sollen diese mit - bzw. _ getrennt werden (also keine Leerzeichen verwenden).
- Der Nachname und der Vorname sollen mit Großbuchstaben beginnen.
- Es sollen keine Umlaute, Akzentzeichen, Sonderzeichen, etc. verwendet werden (also nur die 26 Buchstaben des Alphabets, `ä` wird wie üblich durch `ae` ersetzt, usw.).
- Gültige Projektnamen wären z.B.:
  - `Petrinets_1234567_Mueller_Rene_Joerg`
  - `Petrinets_9876543_Meier-Schmidt_Francois`
  - `Petrinets_815123_Von_Wohlgemuth_Kaethe-Luise`

Die **Quellcodedatei**, welche die `main` Methode zum Starten Ihres Programms enthält, muss ebenfalls dem obigen Namensschema entsprechen, also:

```
Petrinets_<Matr-Nr>_<Nachname>_<Vorname>.java
```

Wir empfehlen Ihnen, auch bei allen weiteren Dateinamen innerhalb Ihres Projekts möglichst auf die Verwendung von potentiell problematischen Sonderzeichen zu verzichten. Insbesondere dürfen Dateinamen keine Sonderzeichen wie z.B. < und > enthalten, die zwar unter Linux, aber nicht unter Windows zulässig sind.

Noch ein wichtiger Hinweis zur **Groß-/Kleinschreibung von Dateinamen**: Wenn

Sie innerhalb Ihres Programms mittels eines Strings einen Dateinamen referenzieren, dann achten Sie unbedingt darauf, dass der String und die referenzierte Datei im Dateisystem in Punkto Groß-/Kleinschreibung exakt übereinstimmen. Ansonsten kann es passieren, dass Ihr Programm zwar (bei Ihnen) unter Windows korrekt funktioniert[14], das Programm (bei uns) unter Linux jedoch nicht ausgeführt werden kann, da eine wichtige Datei auf Grund von Groß-/Kleinschreibfehlern nicht gefunden werden kann.

C.7.1. Text File Encoding

Editieren Sie die Projekteinstellungen, indem Sie mit einem Rechtsklick auf das Projekt den Befehl Properties → Resource aufrufen. Stellen Sie das *Text file encoding* für dieses Projekt auf *UTF-8* ein. Verwenden Sie hier nicht die Einstellung *Inherited from Container*. Stattdessen sollen Sie das Encoding mittels *Other* explizit einstellen, denn nur so wird diese Einstellung direkt zum Projekt gespeichert und ist unabhängig von Veränderungen, die in der globalen Konfiguration vorgenommen werden. So können Schwierigkeiten bei der Verwendung auf verschiedenen Systemen (Studierende bzw. Betreuer) vermieden werden.

C.7.2. Compiler Compliance Level

Editieren Sie einen weiteren Punkt der Projekteinstellungen, indem Sie mit einem Rechtsklick auf das Projekt den Befehl Properties → Java Compiler aufrufen. Stellen Sie hier den *Compiler compliance level* für dieses Projekt auf *1.8* ein. Damit wird sichergestellt, dass innerhalb dieses Projekts nur der Sprachumfang von Java 1.8 genutzt werden kann.

C.7.3. Buildpath - JRE System Library

Überprüfen Sie die eingestellte JRE System Library über einen Rechtsklick auf Ihr Projekt und dem Befehl Properties → Java Build Path → Libraries. Markieren Sie hier *JRE System Library* und werfen Sie über Edit... einen Blick in die Konfiguration. Wählen Sie in dem Dialog in jedem Fall *Workspace Default JRE (jdk-8.0...)* (siehe Abb. 15). So können Schwierigkeiten bei der Verwendung auf verschiedenen Systemen (Studierende bzw. Betreuer) vermieden werden.

C.7.4. Buildpath einstellen (GraphStream Bibliothek)

Um die GraphStream Bibliothek zu verwenden, müssen Sie den Buildpath erweitern. Dazu führen Sie (wie oben) per Rechtsklick auf Ihr Projekt den Befehl Properties → Java

[14] Windows nimmt es mit der Groß-/Kleinschreibung bzgl. Dateinamen nicht immer so genau.

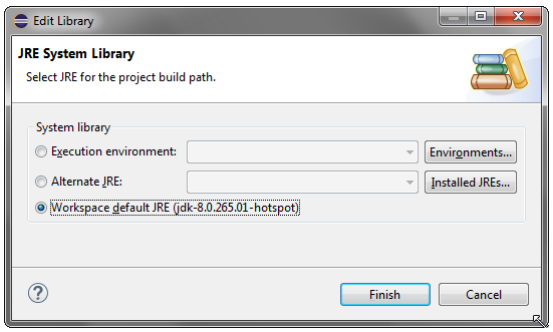


Abbildung 15: im Buildpath eingestellte JRE System Library (die angezeigte Versionsnummer kann ggf. abweichen)

Build Path → Libraries aus. Die entsprechenden JAR Dateien finden Sie in dem Basis-Projekt (siehe Kapitel C.5). Die drei Java-Bibliotheken `gs-algo-1.3.jar`, `gs-core-1.3.jar` und `gs-ui-1.3.jar` aus den entsprechenden Unterverzeichnissen unter `libs/GraphStream` müssen Sie Ihrem Buildpath über die Schaltfläche Add JARs... hinzufügen (siehe Abb. 16).

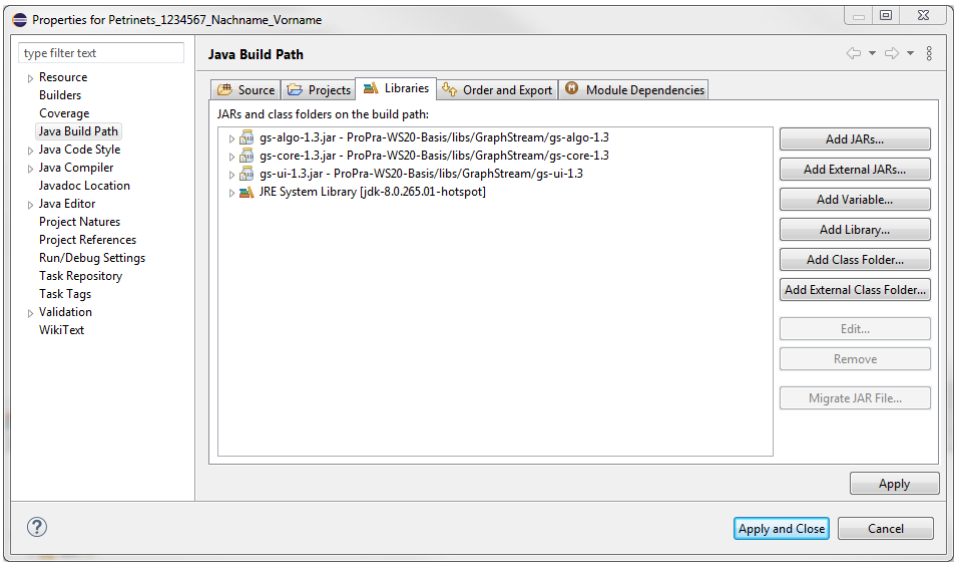


Abbildung 16: Im Buildpath eingestellte Graph Stream Libraries

C.7.5. Buildpath einstellen (PNML Parser)

Jetzt müssen Sie Ihrem Buildpath noch das Projekt ProPra-WS20-Basis hinzufügen, um den von uns zur Verfügung gestellten Parser (Klasse `PNMLWopedParser`) verwenden zu können. Dazu wählen Sie in dem obigen Dialog den Reiter Projects aus und fügen über die Schaltfläche Add... das Basis-Projekt hinzu (siehe Abb. 17).

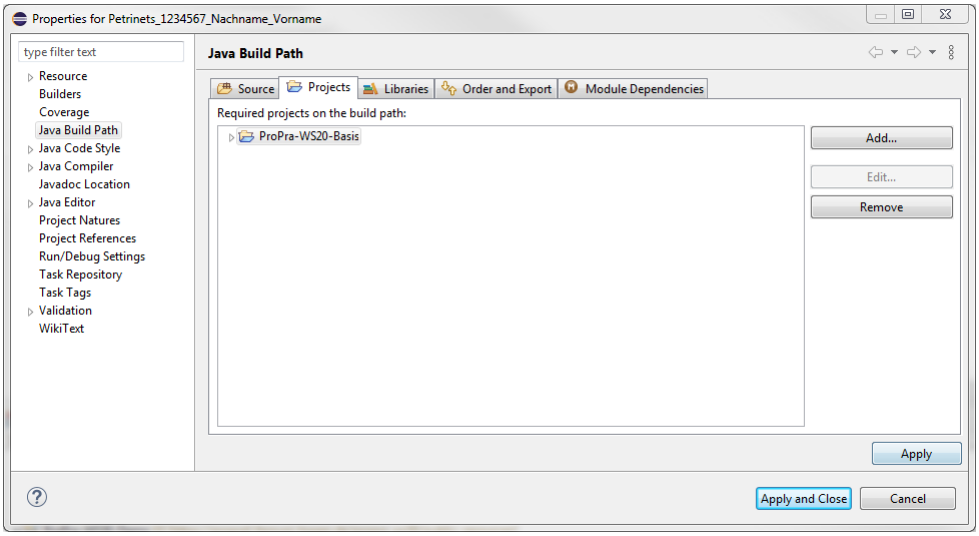


Abbildung 17: Im Buildpath verknüpftes Projekt ProPra-WS20-Basis

Sie können später eine eigene Unterklasse des Parsers erstellen, in der Sie das Einlesen von Petrinetzen aus PNML Dateien realisieren.

```
public class MyPNMLParser extends PNMLWopedParser {
    ...
}
```

Listing 5: Eigene Parser-Klasse zum Einlesen von PNML Dateien

C.7.6. Mit SVN verknüpfen

Um das Java-Projekt mit Ihrem SVN-Bereich zu verknüpfen, wählen Sie nun mit einem Rechtsklick den Befehl Team → Share Project... → SVN und in der anschließenden Auswahl Ihr eigenes Repository aus:

`https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567`

Klicken Sie nun auf Next> und achten Sie bitte dann noch darauf, im folgenden Specify...-Dialog den *Advanced Mode* auszuwählen (nicht *Simple Mode*!) (siehe Abb. 18).

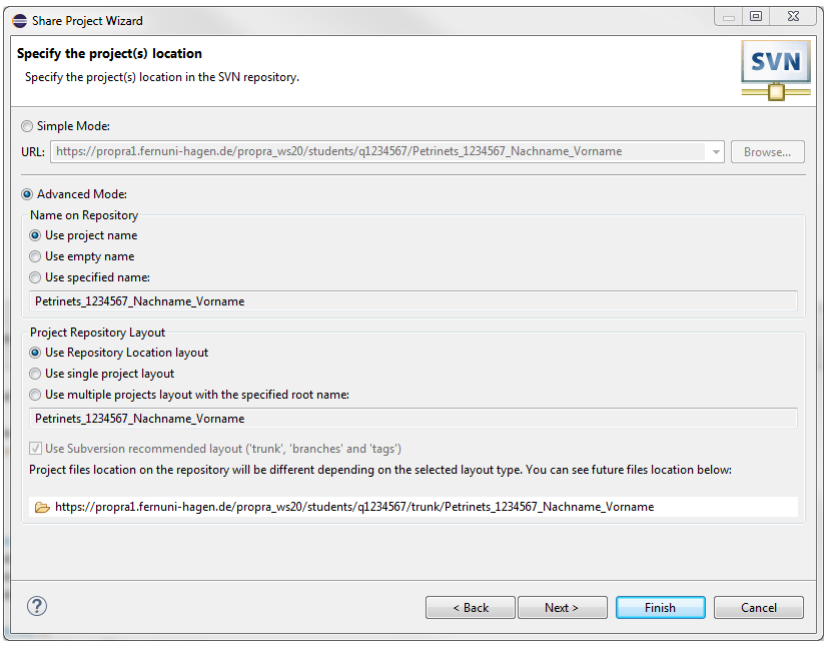


Abbildung 18: Share Project (Advanced Mode)

Achtung: Beachten Sie, dass Ihr Projekt auch nach einem Klick auf Finish noch nicht wirklich auf den Server geladen wurde (die Namensgebung ist hier ein wenig irreführend). Das Projekt wurde lediglich zum Einchecken markiert.

Im Anschluss werden Sie jedoch direkt aufgefordert, den Inhalt Ihres Java-Projekts mit einem sogenannten *Commit* ein erstes Mal ins SVN einzuchecken.

Ob die Verknüpfung mit dem SVN funktioniert hat, können Sie mit Hilfe Ihrer persönlichen URL über einen gewöhnlichen Internet-Browser selbst überprüfen:

```
https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567/trunk
```

Falls Sie bisher keine oder nur geringe Erfahrungen mit dem Versionskontrollsystem SVN sammeln konnten, können Sie im Kapitel D weitere Tipps nachlesen.

D. Subversion (SVN) Versionsverwaltung

In diesem Kapitel werden weitere Schritte bei der Arbeit mit dem SVN beschrieben.

Die Installation des Plugins Subversive sowie das Einrichten eines SVN Repositories, das Herunterladen bereits bestehender Projekte sowie das Hinzufügen eines neuen Projekts wurden bereits in Kapitel C beschrieben und werden daher hier nicht erneut aufgegriffen.

Sie können aber über den Befehl Help → About Eclipse IDE → Installation Details nochmal überprüfen, ob Sie das richtige Plugin installiert haben (Subversive, nicht Subclipse, siehe Abb. 19).

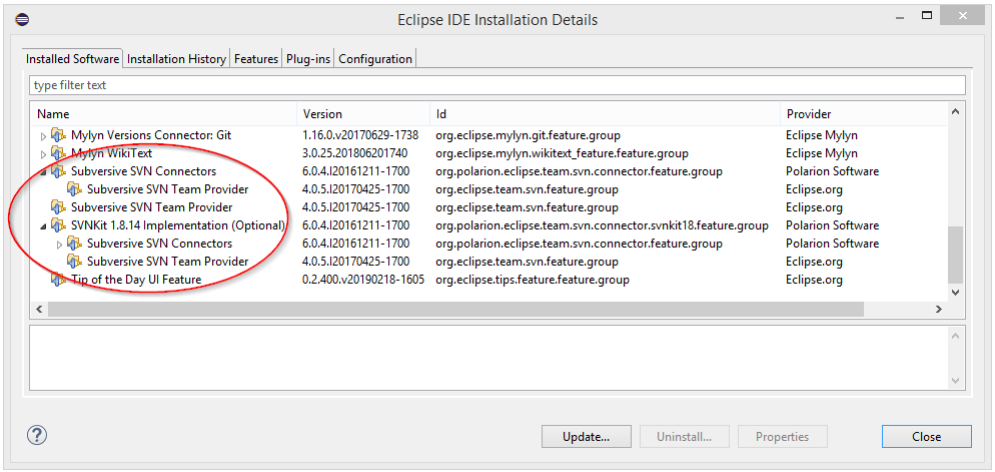


Abbildung 19: Eclipse IDE Installation Details

D.1. Aufbau Ihres persönlichen Repositories

Über den *SVN Repositories*-View können Sie das Repository, das bei diesem Praktikum verwendet wird, einsehen.

In dem Unterverzeichnis `students/q<Ihre Matrikel-Nr>` liegt Ihr persönliches Verzeichnis. In diesem Verzeichnis liegen die Unterordner `trunk`, `tags` und `branches`. Diese Struktur des Verzeichnisses wurde von uns bereits eingerichtet und steht Ihnen zur Verfügung.

Im Verzeichnis `trunk` sollte immer eine aktuelle Arbeitskopie Ihres Projekts liegen. Laden Sie also direkt am Anfang Ihr Projekt wie in Kapitel C.7.6 beschrieben auf den Server. Während der Bearbeitung sichern Sie hier regelmäßig - am besten mindestens einmal täglich - Ihre Arbeit (d.h. Sie machen einen Commit).

In dem Verzeichnis `tags` können Zwischenversionen Ihres Projekts abgelegt werden. Diesem Verzeichnis kommt hier beim Praktikum eine besondere Bedeutung zu. Hier muss die verlangte Abgabe-Version `Abgabe` abgelegt werden. Wie Sie hierzu vorgehen müssen, ist in Kapitel D.7 beschrieben.

Auf das Verzeichnis `branches` wollen wir an dieser Stelle nicht näher eingehen, da es für die Durchführung des Praktikums nicht relevant ist. Hier würde die Möglichkeit bestehen, sogenannte Zweige einzurichten, in denen ein Projekt weiter entwickelt werden kann, ohne die Arbeitskopie im `trunk` zu verändern. Später müsste ein solcher Zweig dann wieder mit der Version im `trunk` zusammengeführt werden.

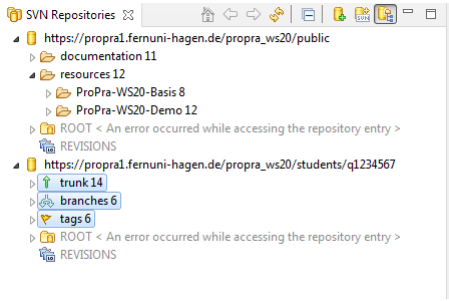


Abbildung 20: Das persönliche Verzeichnis eines Teststudenten im SVN-Repository

D.2. Inhalt des SVN Repositories überprüfen

Auf das von uns zur Verfügung gestellte SVN Repository kann auch über eine Web-Schnittstelle zugegriffen werden. Das heißt, Sie können auch mit einem gewöhnlichen Internet-Browser den aktuellen Stand des gesamten Repositories einsehen. Die Authentifizierung geschieht dabei über Ihren FernUni-Account. Über die angezeigten Links können Sie sich durch den Verzeichnisbaum hangeln (siehe Abb. 21).

```
https://propra1.fernuni-hagen.de/propra_ws20
```

```
https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567
```

D.3. Synchronisieren

Ein Blick in den *Synchronize*-View ist eher interessant, wenn mehrere Entwickler an demselben Projekt arbeiten. Aber auch wenn Sie - wie in diesem Praktikum - alleine eine Versionsverwaltung verwenden, kann dies hilfreich sein. Insbesondere wenn Sie mit mehreren Endgeräten arbeiten sollten, werden Sie an der Verwendung nicht vorbei kommen.

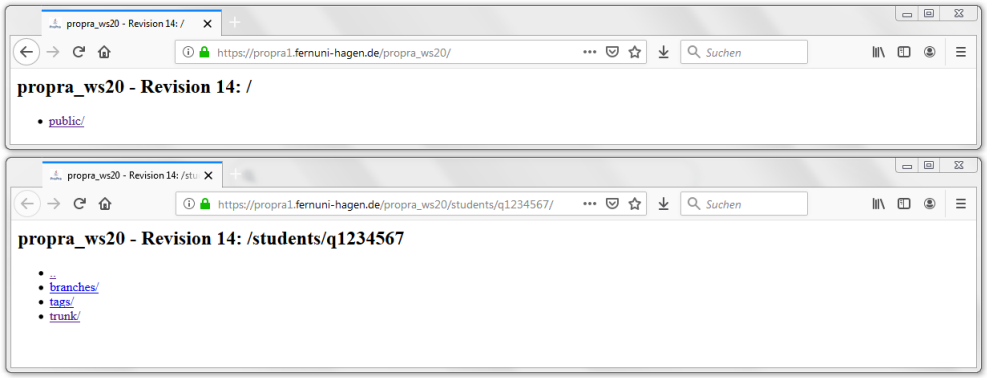


Abbildung 21: Anzeige des Repositories mittels eines Internet-Browsers: öffentlicher Bereich (oben) und persönlicher Bereich (unten)

Über einen Rechtsklick auf ein über SVN verwaltetes Projekt können Sie den Befehl Team → Synchronize with Repository aufrufen. In dem sogenannten *Synchronize*-View werden Ihnen dann alle ein- (blauer Pfeil nach links) und ausgehenden (grauer Pfeil nach rechts) Veränderungen angezeigt. Sollte es zu Konflikten kommen, werden diese mit einem roten Pfeil markiert.

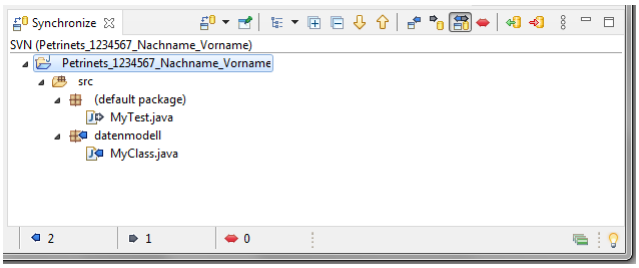


Abbildung 22: Anzeige eines Projekts im *Synchronize*-View

Bei angezeigten Unterschieden zwischen lokaler und entfernter Version können Sie über einen Doppelklick auf den jeweiligen Dateinamen die Unterschiede im sogenannten *Compare Editor* betrachten.

D.4. Update

Um Änderungen aus dem Repository in die eigene Version zu übernehmen, müssen Sie den Befehl Update anwenden. Dies ist an unterschiedlichen Stellen möglich. Zum einen können Sie im *Package Explorer* über einen Rechtsklick mittels des Befehls Team → Update Veränderungen aus dem Repository herunterladen. Aber auch im *Synchronize*-

View können Sie den Befehl Update über das Kontextmenü aufrufen (siehe Abb. 23).

Achten Sie darauf, für welche Auswahl Sie das Herunterladen aufrufen. Dies ist auf dem gesamten Projekt, auf Unterverzeichnissen, auf einzelnen Dateien, sowie auch auf einer Auswahl von Dateien/Verzeichnissen möglich.

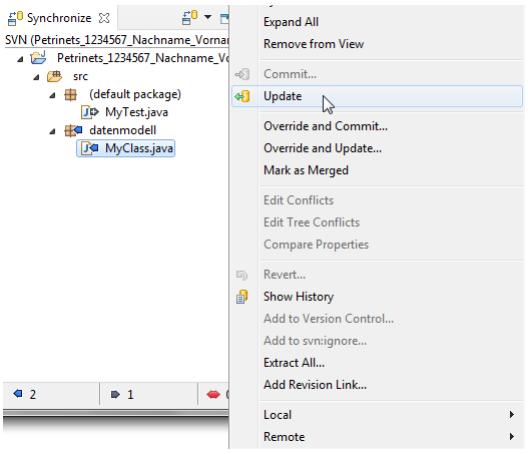


Abbildung 23: Update einer Datei über das Kontextmenü im *Synchronize*-View

Für Ihr eigenes Projekt werden Sie kein Update ausführen müssen, falls Sie dieses nur an einer Arbeitsstation bearbeiten. Allerdings werden Sie für die von uns zur Verfügung gestellten Projekte zu gegebener Zeit Updates durchführen müssen, z.B. wenn wir über das Basis-Projekt weitere Beispieldateien veröffentlichen.

D.5. Commit

Änderungen, die Sie in Ihrem Projekt vorgenommen haben, können Sie über ein Commit in das Repository übertragen. Wie bei dem in Kapitel D.4 beschriebenen Update ist das Commit an unterschiedlichen Stellen möglich. Abb. 24 zeigt ein Commit im *Synchronize*-View.

In dem sich dann öffnenden Dialog werden noch einmal alle Dateien angezeigt, die mit diesem Commit in das Repository hochgeladen werden. Zudem sollten Sie hier einen aussagekräftigen Kommentar zu Ihren Änderungen angeben (siehe Abb. 25).

Ein Commit funktioniert natürlich nur, wenn Sie Schreibrechte in dem entsprechenden Repository besitzen. Hier beim Programmierpraktikum gilt dies nur in den Unterverzeichnissen `branches`, `tags` und `trunk` Ihres persönlichen Verzeichnisses `students/q1234567`.

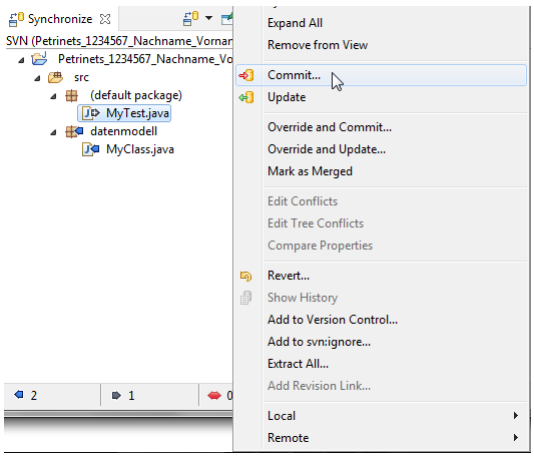


Abbildung 24: Commit einer Datei über das Kontextmenü im *Synchronize*-View

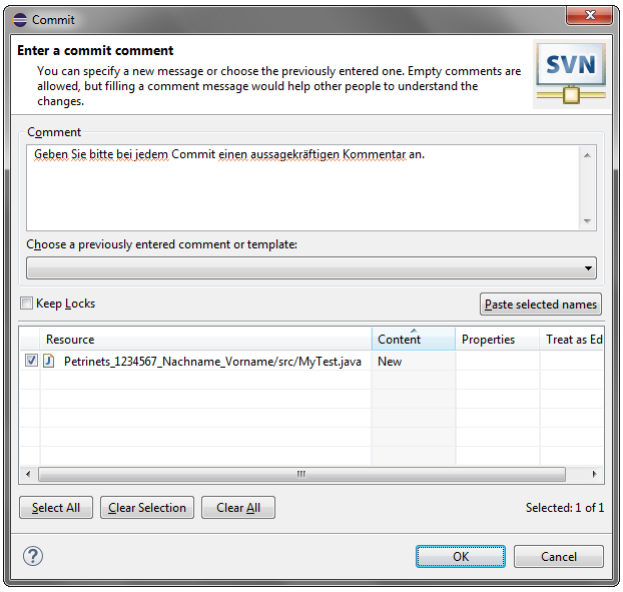


Abbildung 25: Angabe eines Kommentars zu dem Commit

D.6. Versions-Historie

Über den Befehl Team → Show History können Sie die Versions-Historie Ihres Projekts oder auch nur einzelner Unterverzeichnisse oder Dateien einsehen. Anhand des Screenshots (siehe Abb. 26) sieht man sehr schön, dass bei zwei Commits kein Kommentar angegeben wurde. Versuchen Sie, dies bei der Arbeit an Ihrem eigenen Repository zu vermeiden. Aussagekräftige Kommentare helfen auch Ihnen, falls Sie auf vorhergehende

Versionen Ihres Projekts zugreifen wollen.

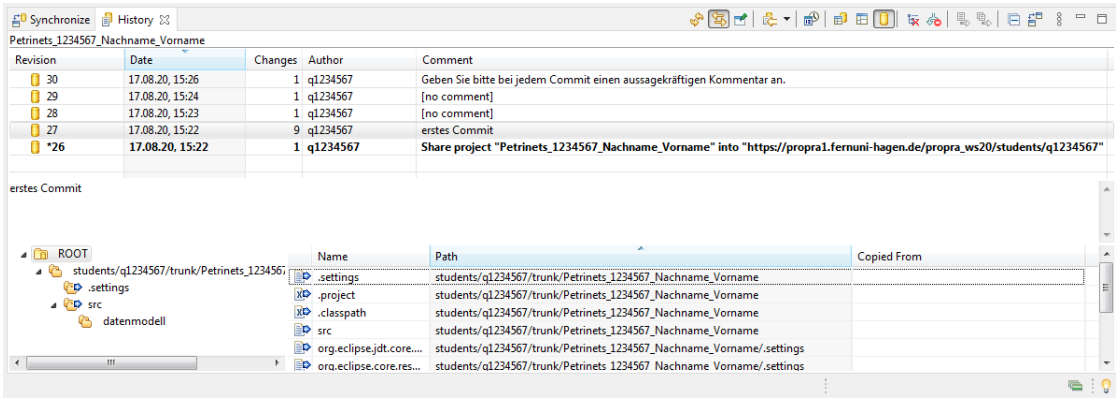


Abbildung 26: Versions-Historie eines Beispiel-Projekts

D.7. Kennzeichnen eines Projekts mit einem Tag

Für die Abgabe Ihrer Lösung müssen Sie Ihr Projekt mit dem Tag `Abgabe` kennzeichnen. Stellen Sie vorher sicher, dass Ihre lokale Version mit der Version auf dem SVN-Server synchron ist (Team → Synchronize with Repository, siehe Kapitel D.3). Dann wählen Sie über einen Rechtsklick auf Ihr Projekt den Befehl Team → Tag. . . . Füllen Sie den folgenden Dialog wie in Abb. 27 aus. Als Tag geben Sie `Abgabe` ein. Verwenden Sie in jedem Fall genau diesen Namen - auch Groß-/Kleinschreibung beachten - sonst funktioniert unser Script zum automatischen Erkennen aller eingegangenen Lösungen nicht. Achten Sie auch darauf, das Tag immer auf der Projekt-Wurzel anzuwenden, ansonsten ist Ihre Abgabe nicht vollständig.

In der *SVN Repository Exploring*-Perspektive können Sie über den Unterordner `tags` einsehen, welche Zwischenversionen Ihres Projekts Sie in das SVN Repository hochgeladen haben (siehe Abb. 28).

Ob die Abgabe Ihres Projekts funktioniert hat, können Sie auch mittels eines Internet-Browsers und den Aufruf Ihrer persönlichen URL überprüfen:
`https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567/tags`

In dem Verzeichnis `tags` sollte dann ein Unterordner mit der Bezeichnung `Abgabe` liegen. Dieser Ordner sollte wiederum einen Unterordner enthalten, der den Namen Ihres Projekts trägt und in dem Ihr gesamtes Eclipse-Projekt enthalten ist.

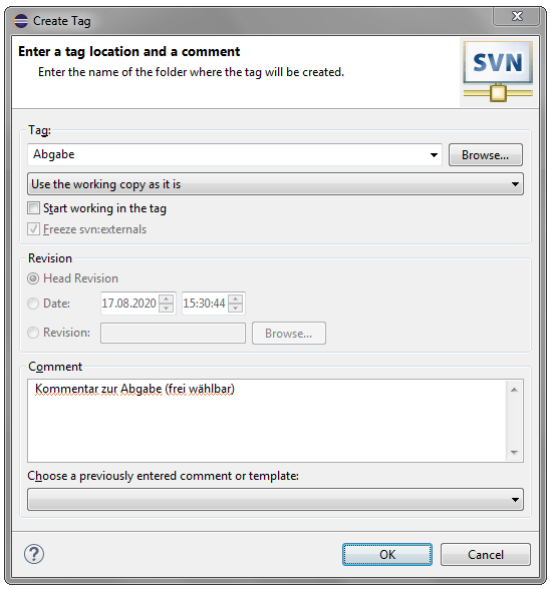


Abbildung 27: Kennzeichnen des Projekts mit dem Tag `Abgabe`

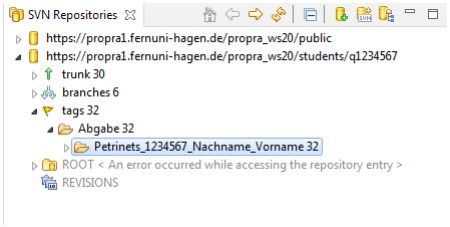


Abbildung 28: Kontrolle der Abgabe über die *SVN Repository Exploring-Perspektive*

D.8. Hilfe - das Verknüpfen meines Projekts mit SVN funktioniert nicht!

In Kapitel C.7.6 ist beschrieben, wie Sie Ihr Projekt mit SVN verknüpfen. Falls Sie bei dem sogenannten *Share* Ihres Projekts einen Fehler (Authentication error, wie in Abb. 29) erhalten, haben Sie vermutlich im Specify. . .-Dialog (siehe Abb. 18) den *Simple Mode* gewählt (richtig ist der *Advanced Mode*!).

Hintergrund: Für Ihr persönliches Verzeichnis wurden Ihnen nur Lese-Rechte zugewiesen. Nur in den Unterverzeichnissen `branches`, `tags` und `trunk` haben Sie jeweils auch Schreibrechte. Beim *Simple Mode* würde das Projekt jedoch direkt in dem Haupt-Verzeichnis abgelegt werden. Daher kommt es hier zu einem Fehler.

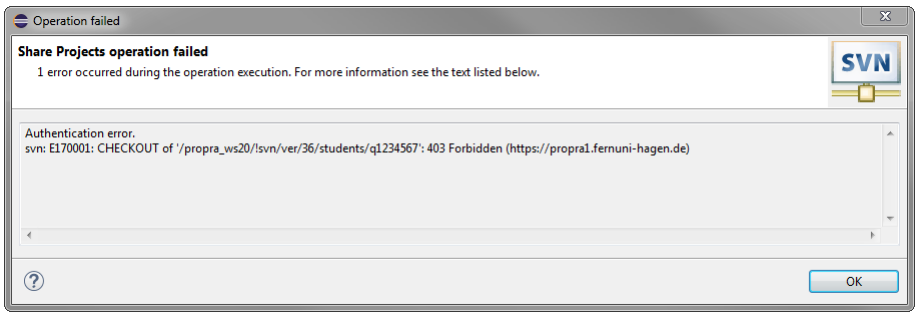


Abbildung 29: Authentication error beim Verknüpfen des Projekts mit SVN

D.9. Hilfe - mein Repository ist kaputt!

Es kommt leider immer mal wieder vor, dass es beim Arbeiten mit SVN zu Fehlern kommt, die sich nicht so einfach lösen lassen. In so einem Fall kann es manchmal helfen, auf dem Projekt den Befehl Team → Cleanup auszuführen.

Wenn es zu Konflikten bei einzelnen Dateien gekommen ist, für die es augenscheinlich keine Lösung gibt, kann man auch - nach vorheriger lokaler Sicherung der jeweiligen Datei - in dem *Synchronize*-View den Befehl Override and Update... aufrufen. Damit werden die lokalen Änderungen verworfen und die aktuelle Version aus dem Repository erneut in das lokale Projekt herunter geladen. Nachdem man wieder einen synchronen Zustand hergestellt hat, können die in der Sicherungskopie gesicherten Inhalte in das lokale Projekt kopiert werden, um diese dann mit dem üblichen Commit wieder in das entfernte Repository hochladen zu können.

Und sollte es dann doch nochmal zu einem Konflikt kommen, der sich gar nicht lösen lassen will, können Sie Folgendes probieren:

- Machen Sie sich eine Sicherungskopie des aktuellen Arbeitsstandes.
- Entfernen Sie das lokale Projekt in Eclipse (Häkchen setzen bei *Delete project contents on disk*).
- Danach checken Sie das Projekt aus dem Repository erneut aus. Der lokale Stand ist dann mit dem im Repository synchron.
- Kopieren Sie den Arbeitsstand Ihrer Dateien aus der Sicherungskopie in das neu ausgecheckte Projekt. Achten Sie dabei unbedingt darauf, dass das versteckte Verzeichnis `.svn` *nicht* mitkopiert wird.
- Zum Schluss können Sie die Neuerungen mit *Commit* in das Repository übernehmen. Achten Sie aber darauf, dass Sie nur die wirklich gewollten Änderungen hochladen; ggf. überprüfen Sie dies im *Compare Editor*.

D.10. Hilfe - meine Abgabe ist nicht komplett!

Leider kommt es auch immer mal wieder vor, dass nach dem Einreichen der Lösung auffällt, dass in dem als Abgabe gekennzeichneten Projekt noch ein Fehler enthalten ist oder dass eine Abgabe nicht vollständig war. In so einem Fall kann man sich wie folgt helfen:

Beheben Sie alle Fehler in Ihrem lokalen Projekt und übertragen Sie diese mit einem einfachen Commit in das SVN. Sie können Ihr Projekt allerdings nicht (unmittelbar) erneut mit dem Tag `Abgabe` kennzeichnen, da dieses Tag bereits existiert. Wenn Sie Team → Tag mit dem Tag `Abgabe` ausführen würden, dann würde eine entsprechende Fehlermeldung erscheinen (ungefähr wie in Abb. 30 dargestellt).

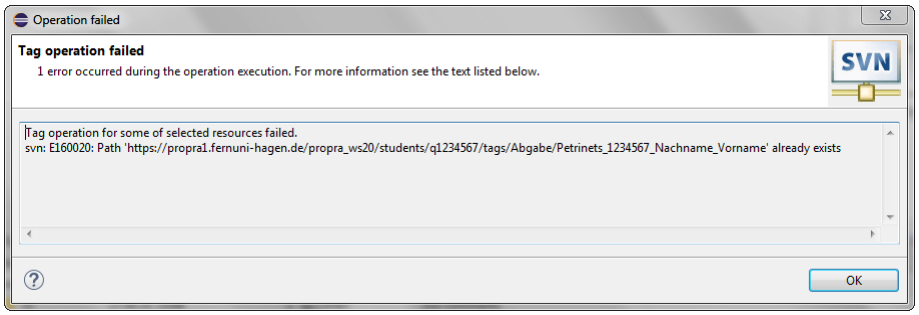


Abbildung 30: Fehler beim Tag: der gewählte Pfad existiert bereits

Um die Abgabe dennoch vornehmen zu können, wechseln Sie in die *SVN Repository Exploring*-Perspektive. Hier können Sie sich durch Ihr SVN-Repository bewegen und das Verzeichnis

`https://propra1.fernuni-hagen.de/propra_ws20/students/q1234567/tags`

aufklappen. Durch einen Rechtsklick auf den Ordner `Abgabe` öffnet sich ein Kontextmenü und Sie können das Verzeichnis mit dem Befehl Delete... entfernen.

Danach ist ein erneutes Kennzeichnen Ihres Projekts mit dem Tag `Abgabe` wie in Kapitel D.7 beschrieben wieder möglich.

D.11. Abschließender Hinweis

Am besten spielen Sie mit dem SVN ein bisschen herum und probieren die hier vorgestellten Befehle einfach mal aus. Da die in den Verzeichnissen `trunk` und `tags` abgelegten Inhalte auch wieder gelöscht werden können, ist fast alles wieder rückgängig zu machen. Und an der vorbereiteten Struktur `trunk`, `tags`, `branches` können Sie keine Veränderung vornehmen. Die Historie Ihrer Übungen wird im SVN zwar verzeichnet, aber das stört bei der weiteren Arbeit nicht.

Literatur

[DJ01] Jörg Desel and Gabriel Juhás. "What Is a Petri Net?". In Hartmut Ehrig, Gabriel Juhás, Julia Padberg, and Grzegorz Rozenberg, editors, *Unifying Petri Nets*, volume 2128 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2001.

[Rei13] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies.* Springer, 2013.