Modellierung von Geschäftsprozessen in der Unified Modeling Language und ihre Transformation in Petrinetze

David Kreische
Lehrstuhl für Informatik 3, Universität Erlangen–Nürnberg
Martensstr. 3, D–91058 Erlangen
ddkreisc@informatik.uni-erlangen.de

Abstract: In diesem Paper wird gezeigt, wie ein Geschäftsprozess mit Hilfe der Unified Modeling Language (UML) so modelliert werden kann, dass daraus die Berechnung von dynamischen Kenngrößen wie Durchführungszeit oder Ressourcen-Auslastung möglich ist. Zuerst werden die dazu verwendeten UML-Konstrukte vorgestellt. Dann werden die wichtigsten Elemente eines Algorithmus zur Transformation in ein Generalized Stochastic Petri Net (GSPN) präsentiert. Zuletzt werden die Ergebnisse eines Beispiel-Prozesses erläutert.

1 Forderungen an die Geschäftsprozessmodellierung

Geschäftsprozessmodelle werden aus mehreren Gründen verwendet. Der momentan wichtigste Grund ist die Dokumentation. Ein übersichtliches, graphisches Modell kann das Verständnis und die Abstimmung aller am Prozess Beteiligten erheblich erleichtern und Missverständnissen vorbeugen. In vielen Bereichen hängt sogar die Auftragsvergabe von der sauberen Dokumentation der Geschäftsabläufe ab.

Wenn einiger Aufwand betrieben wurde, einen existierenden oder geplanten Geschäftsprozess zu modellieren, so ist es wünschenswert, wenn dieses Modell über den reinen Dokumentationszweck hinaus genutzt werden kann. In vorliegenden Paper geht es um die Berechnung von zu Einschätzung und Vergleich von Geschäftsprozessen tauglichen Kenngrößen.

Die Schwierigkeit dabei liegt darin, dass das verwendete Modell formalisiert und präzise genug sein muss, um die Berechnung von Kenngrößen überhaupt erst zu ermöglichen. Die direkte Verwendung von syntaktisch und semantisch wohldefinierten Modelliermethoden wie z.B. Petrinetzen oder Prozessalgebren scheitert gewöhnlich daran, dass sie vom Anwender nicht akzeptiert werden, weil die verwendeten Konzepte ihrer Erfahrung völlig fremd sind. Deshalb muss ein Modellierungstool für Geschäftsprozesse Konstrukte verwenden, die in der Begriffswelt des Anwenders vorkommen.

Solche Bausteine benutzt der *Prozessmodellierer*, um seinen Geschäftsprozess zu modellieren. Allerdings können nicht für alle denkbaren Fälle vorgefertigte Bausteine existieren. Deshalb muss es einem *Komponentenmodellierer* möglich sein, diese Bausteine zu modi-

fizieren oder neue Bausteine zu erstellen.

Im Projekt ERIKA[1] erfolgt die Eingabe und Darstellung des Modells in einer angepassten Version der Unified Modeling Language (UML). Dieses Modell wird dann in ein generalisiertes stochastisches Petrinetz (GSPN) transformiert, welches mit dafür verfügbaren Werkzeugen untersucht werden kann.

2 Modellierung der Geschäftsprozesse

Die Forderungen in Abschnitt 1 zieht nach sich, dass es zwei Modellsichten gibt: die Sicht auf den Ablauf des Geschäftsprozesses und die Sicht auf das Verhalten der Ressourcen. Der Geschäftsprozess wird vom *Prozessmodellierer* im *Hauptaktivitätsdiagramm* modelliert, während die Ressourcen vom *Ressourcenmodellierer* mittels *Treiberaktivitätsdiagrammen* sowie Zustands- und Klassendiagrammen erstellt werden. Diese Ressourcen können mittels definierter Schnittstellen im Hauptaktivitätsdiagramm verwendet werden.

Zunächst soll das Hauptaktivitätsdiagramm anhand des Beispieles in Abb. 1 dargestellt werden. Der zu modellierende Geschäftsprozess wurde dort in vier Teilschritte, die *Aktivitäten* aufgeteilt. Jede Aktivität ist klar von den anderen unterscheidbar und benötigt für ihre Abarbeitung eine gewisse Zeitdauer. Die Aktivitäten sind durch Transitionen miteinander so verbunden, dass *Aufträge*, die das System am Anfangsknoten *start* betreten, zum Endknoten *end* gelangen können. Die Aufträge können unterschiedliche Pfade durch das System nehmen. Bei *Entscheidungsknoten* wie nach der Aktivität *sort orders* können Wahrscheinlichkeiten angegeben werden. Parallele Abläufe sind ebenfalls möglich.

Zur Durchführung von Aktivitäten sind normalerweise einige Personen, Hilfsmittel, Werkzeuge u.ä. erforderlich. Diese werden unter dem Begriff *Ressourcen* zusammengefasst. Ressourcen werden jedoch nicht direkt verwendet. Stattdessen werden *Rollen* benutzt. Diese geben an, dass für die Durchführung einer Aktivität eine Ressource der angegebenen Klasse, welche die gewünschte Rolle übernehmen kann, vorhanden sein muss. Durch die Ausführung der Aktivität wird der Zustand der Ressource beeinflusst. Welche Zustände die Ressource vor, während und nach der Durchführung der Aktivität einnehmen muss, wird durch die an der Assoziation angegebene *Tätigkeit* bestimmt. Für *sort orders* in Abb. 1 muss ein *employee* zur Verfügung stehen, der als *sorter* arbeiten kann. Er soll dabei die Tätigkeit *work* ausführen.

Der Prozessmodellierer kann neue Rollen festlegen. Allerdings darf er nur bereits existierende Klassen und die dort definierten Tätigkeiten verwenden. Er kann sich somit ausschliesslich auf die Beschreibung des Geschäftsprozesses im Hauptaktivitätsdiagramm konzentrieren und benutzt die Ressourcen als „Black Boxes“ lediglich über ihre Schnittstellen.

[1]Das Projekt wird von der Bayerischen Staatsregierung als Teil des Programms: ”Informations- und Kommunikationstechnik” gefördert. Die anderen Projektpartner sind: Lehrstuhl Wirtschaftsinformatik II der Universität Erlangen–Nürnberg, MID GmbH und BIK mbH.

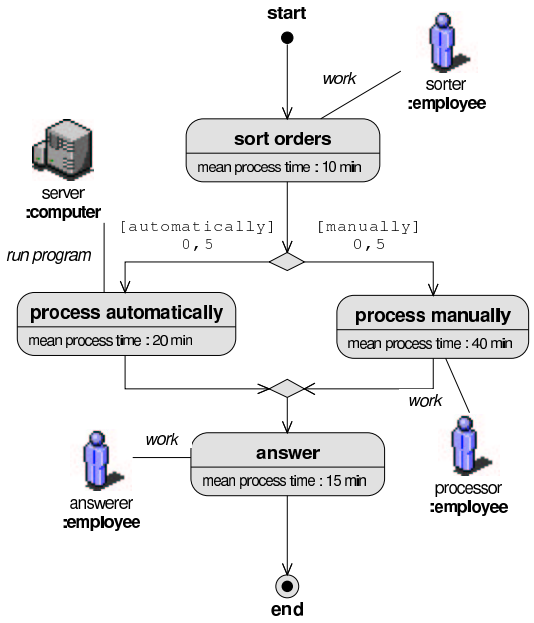
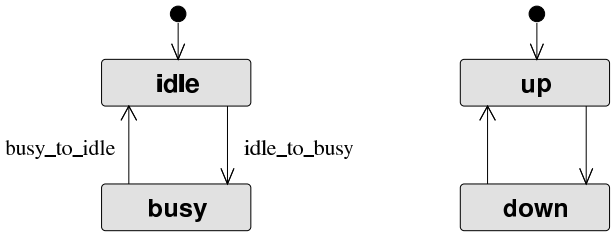


Abbildung 1: Hauptaktivitätsdiagramm

3 Modellierung der Ressourcen

Zur Beschreibung der Zustände einer Ressource und der möglichen Übergänge dazwischen bietet sich das Zustandsdiagramm an. Abb. 2 zeigt die Zustandsdiagramme der Klassen *employee* und *server*.



(a) employee (b) server

Abbildung 2: Zustandsdiagramme

Eine Ressource der Klasse *employee* startet im Zustand *idle*. Sie wechselt genau dann in den Zustand *busy*, wenn das Ereignis *idle_to_busy* eintritt. Die Namen der Ereignisse werden immer nach dem Muster *<Startzustand>_nach_<Zielzustand>* gebildet, so dass

sie in den Zustandsdiagrammen üblicherweise ausgeblendet werden können.

In Abschnitt 2 wurde bereits angedeutet, dass die Zustandsübergänge der Ressourcen mittels der Tätigkeiten im Aktivitätsdiagramm gesteuert werden. Diese werden im Klassendiagramm definiert. Abb. 3 zeigt das Klassendiagramm für *employee*. Dort wird eine Assoziation *work* zwischen den Klassen *employee* und *ActionState* angelegt. Dies definiert die Tätigkeit *work*, die im Hauptaktivitätsdiagramm verwendet wurde (siehe Abb. 1). Das Anlegen solcher Assoziationen zu Objekten erweitert das Metamodell der UML-Aktivitätsdiagramme (siehe [Gro01], Seite 2-177) . Dies ist notwendig, um Ressourcen und Aktivitäten mittels einer Assoziation verbinden zu können.

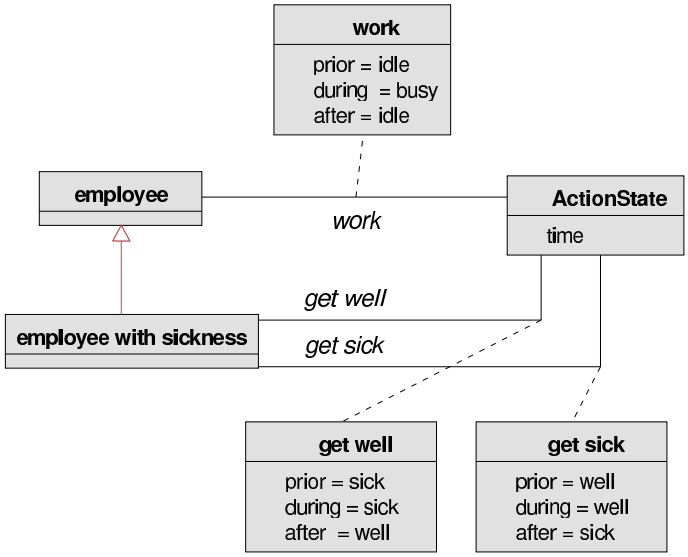


Abbildung 3: Klassendiagramm für die Klasse *employee*

Jede Tätigkeits-Assoziation besitzt eine gleichnamige Assoziationsklasse mit drei Attributen, die den Zustand der Ressource vor, während und nach der Ausführung der Aktivität angeben. Jede Ressourcen-Klasse besitzt genau ein Zustandsdiagramm. Nur dort vorkommende Zustände dürfen in der Assoziationsklasse verwendet werden. In Abb. 3 ist der Klasse *employee* das Zustandsdiagramm aus Abb. 2(a) zugeordnet. In der Assoziationsklasse von *work* wird festgelegt, dass *employee* vor der Ausführung einer Aktivität unter Verwendung dieser Tätigkeit im Zustand *idle*, während im Zustand *busy* und danach wieder im Zustand *idle* sein soll.

Wie aus Abb. 3 hervorgeht, ist es möglich, Unterklassen von Ressourcen anzulegen. Dies dient dazu, das Verhalten von Ressourcen abwandeln zu können, ohne das Hauptaktivitätsdiagramm anpassen zu müssen. Die abgeleiteten Klasse *employee with sickness* kann im Hauptaktivitätsdiagramm weiterhin mit der Tätigkeit *work* verwendet werden.

Um dies zu erreichen, muss ein Zustand im Zustandsdiagramm der Unterklasse das gesamte Zustandsdiagramm der Oberklasse als Verfeinerung enthalten. In Abb. 4 ist dies

bei dem Zustand *well* der Fall. Um zwischen den neuen Zuständen *well* und *sick* wechseln zu können, wurden die Tätigkeiten *get well* und *get sick* erstellt. Diese können nun in Aktivitätsdiagrammen verwendet werden.

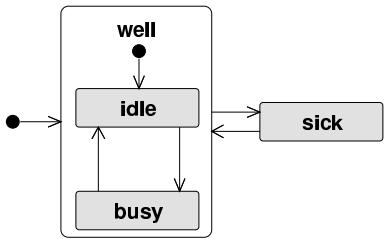


Abbildung 4: Zustandsdiagramm für die Klasse *employee with sickness*

Oftmals ist die Verwendung von abgeleiteten Klassen durch den Prozessmodellierer gar nicht gewünscht, weil die neuen Tätigkeiten und die zugehörigen Aktivitäten den eigentlichen Geschäftsprozess überfrachten. Das „Eigenleben“ der Ressource sollte von ihm nicht modelliert werden müssen, damit er sie weiter als „Black Box“ betrachten kann. Dem Ressourcenmodellierer hingegen stehen dafür die *Treiberaktivitätsdiagramme* zur Verfügung. Jeder Klasse kann genau ein solches Treiberaktivitätsdiagramm zugeordnet werden. Darin kann nur die zugehörige Ressource verwendet werden. Deshalb ist die Verwendung von Rollen hier sinnlos. Im Unterschied zum Hauptaktivitätsdiagramm ist ein Endzustand nicht notwendig, wenn die Ressource fortlaufend zyklisch zwischen ihren Zuständen wechseln soll.

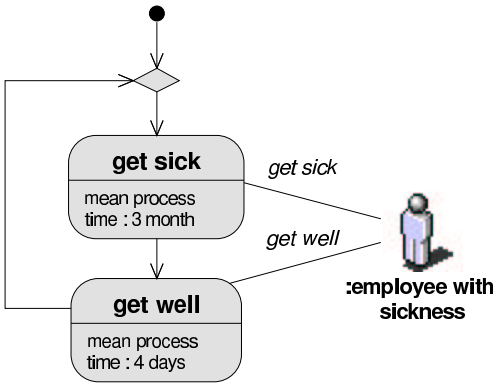


Abbildung 5: Treiberaktivitätsdiagramm für die Klasse *employee with sickness*

Im Hauptaktivitätsdiagram wird also modelliert, welche Ressourcen in welcher Rolle benötigt werden. In den Klassen-, Zustands- und Treiberaktivitätsdiagrammen wird das Verhalten dieser Ressourcen beschrieben. Nun muss noch angegeben werden, welche Ressourcen konkret zur Verfügung stehen und welche Rolle sie übernehmen können.

Bei dieser Zuordnung ist es möglich, die im Hauptaktivitätsdiagramm angegebene Klasse durch eine ihrer Unterklassen zu ersetzen. Dadurch muss die letztendlich verwendete Klas-

Ressource	Klasse	Rollen
ENIAC	computer	server
Müller	employee	sorter, answerer
Meier	employee with sickness	sorter
Schulze	employee	sorter, processor

Tabelle 1: Ressourcentabelle

se erst bei der Auswertung festgelegt werden. Eine Modifikation des Hauptaktivitätsdiagrammes ist dafür unnötig. Somit können die Auswirkungen unterschiedlicher Ressourcen ohne großen Aufwand ermittelt werden. In Tab. 1 wurde für *Meier* die Unterklasse *employee with sickness* verwendet.

Ebenfalls ersichtlich ist, dass alle drei Ressourcen der Klasse *employee* für die Rolle *sorter* verwendet werden können. Dies bedeutet, dass die Aktivität *sort orders* eine Auswahl treffen muss, welche und wie viele Ressourcen sie allokiert.

Normalerweise gilt, dass für die Durchführung einer Aktivität genau eine Ressource in der geforderten Rolle vorhanden sein muss. Ein davon abweichendes Verhalten kann mittels einer *Auswahltabelle* beschrieben werden, die einer Aktivität zugeordnet werden kann. Dort werden die gewünschten Kombinationen von Rollen und ihre Wirkung auf die Durchführungszeit der Aktivität aufgeführt.

Rampe	Gabelstapler	Fahrer	Zeitfaktor
1	1	1	1
1	2	2	0.5

Tabelle 2: Auswahltabelle für *Fahrzeug beladen*

Für eine Aktivität *Fahrzeug beladen* werden eine Rampe sowie Gabelstapler mit Fahrer benötigt. Mit der Tabelle aus Tab. 2 wird folgendes Verhalten beschrieben: es wird nie mehr als eine Rampe belegt und ein Gabelstapler muss auch immer einen Fahrer haben. Die Verwendung von zwei Gabelstaplern halbiert die Ladezeit. Mehr als zwei können jedoch nicht verwendet werden, weil die Rampe nicht groß genug ist.

4 Transformation in ein GSPN

Nachdem alle nötigen Informationen in den UML-Diagrammen und Tabellen vorhanden sind, kann mit der Berechnung von Resultaten begonnen werden. Diese wird nicht auf direktem Wege vorgenommen, sondern das Modell wird zunächst in ein Generalized Stochastic Petri Net (GSPN) transformiert. GSPN sind weit verbreitet und es existiert eine Vielzahl von Tools zu ihrer Untersuchung. Außerdem lagen bereits Erfahrungen bei der Transformation von Zustandsdiagrammen zur Beschreibung technischer Systeme vor [KH00]. Die Zustandsdiagramme der Ressourcen und die Aktivitätsdiagramme werden zunächst getrennt transformiert und dann zu einem einzigen Petrinetz verschmolzen.

Für jede in der Ressourcentabelle aufgeführte Ressource wird aus dem seiner Klasse zugeordneten Zustandsdiagramm ein Petrinetz erzeugt. Dazu wird das in Algorithmus 1 beschriebene Verfahren verwendet. Dort wird für jeden Zustand eine Stelle im Petrinetz angelegt. Für jede Kante wird eine Transition erzeugt, die über jeweils eine Eingabe- und Ausgabekante mit den Stellen verbunden ist, die ihrem Start- und Zielzustand entsprechen. Bei hierarchischen Zuständen kommen noch Kanten hinzu, die sicherstellen, dass beim Verlassen des Oberzustandes alle Token in den Stellen des Unterzustandes entfernt werden, bzw. die beim Betreten des Oberzustandes in die Anfangsstelle des Unterdiagramms ein Token ablegen.

```
1  for ∀ Zustände z ∈ ZD* do erzeuge für Zustand z_i eine Stelle s_i im PN† od
2  for ∀ Kanten k ∈ ZD do
3      erzeuge für Kante k_i mit Startzustand z_start und Zielzustand z_ziel eine
4      Transition t_i im PN
5      build_input_arc(s_start, t_i)
6      setzte Multiplizität der neu erzeugten Kante gleich 1
7      build_output_arc(t_i, s_end)
8  od
9  Z_top = {z ∈ ZD | z liegt in der obersten Hierarchieebene}
10 put_start_tokens(Z_top)
11
13 funct build_input_arc(s, t) ≡
14   ⌈erzeuge neue Kante a_j mit der Multiplizität m = mark‡(s) im PN
15    if z hat Unterzustände Z_sub
16      then for ∀ z' ∈ Z_sub do
17                build_input_arc(s', t) od fi ⌋.
18
20 funct build_output_arc(t, s) ≡
21   ⌈erzeuge neue Kante a_k im PN
22    if z hat Unterzustände Z_sub
23      then for ∀ z' ∈ Z_sub do
24                if z' ist Startzustand
25                  then build_output_arc(s', t) fi od fi ⌋.
26
28 funct put_start_tokens(Z) ≡
29   ⌈for z ∈ Z do
30        if z_i ist Startzustand
31          then lege ein Token in die zu z_i gehörende Stelle s_i
32                if z_i hat Unterzustände Z_sub
33                  then put_start_tokens(Z_sub) fi fi od ⌋.
```

Algorithmus 1: Transformation eines Zustandsdiagrammes in ein Petrinetz

---

* Zustandsdiagramm

† Petrinetz

‡ $mark(s)$ liefert die momentane Anzahl von Marken in der Stelle s zurück

Das untere Petrinetz in Abb. 6 ist auf die erläuterte Weise aus dem in Abb. 4 gezeigten Zustandsdiagramm konstruiert worden.

Die Transformation der Struktur von Aktivitätsdiagrammen ist unkompliziert, weil sie der Struktur von Petrinetzen sehr ähnlich ist. Deshab sei dazu auf [GGW99] und [Kun00] verwiesen. Für die Verschmelzung mit den Petrinetzen essentiell ist jedoch der Aufbau des Petrinetzes einer einzelnen Aktiviät. Dieser muss die drei Phasen: Allokation, Verwendung und Freigabe der Ressourcen ermöglichen. Im oberen Petrinetz in Abb. 6 ist der Aufbau einer Aktivität beispielhaft dargestellt. Das Feuern einer der Transitionen zwischen *prior* und *during* dient dem Allokieren von Ressourcen. Jede dieser Transitionen allokiert alle für die Ausführung der Aktivität nötigen Ressourcen. Die Anzahl der Allokierungs-Transitionen hängt sowohl von der in der Ressourcentabelle angegebenen Anzahl der verfügbaren Ressourcen als auch von dem in der Auswahltabelle definierten Allokationsmuster ab. Für jede sich daraus ergebende Alternative existiert eine Transition.

Nach dem Feuern einer Auswahl-Transition wird auf das Feuern der zeitbehafteten Transition zwischen *during* und *after* gewartet. Diese Transition hat eine Feuerrate λ, die dem Kehrwert der an der Aktivität angegebenen Zeit entspricht. Das Feuern dieser Transition gibt alle allokierten Ressourcen frei. Um dies korrekt durchführen zu können, ist noch eine Hilfs-Stelle pro Ressource erforderlich, die in Abb. 6 nicht dargestellt ist.

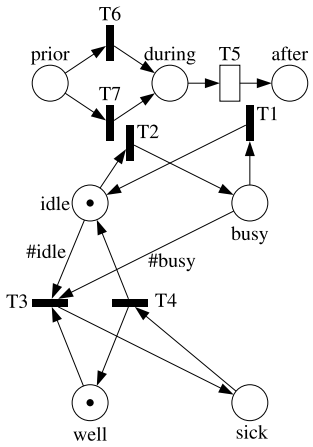


Abbildung 6: Petrinetz für *employee with sickness*

Wie die Verschmelzung durchgeführt wird, ist in Abb. 6 angedeutet. Durch die Tätigkeit *work* in Abb. 3 ist festgelegt, dass *T2* mit *T7* und *T1* mit *T5* verschmolzen werden muss. Alle Kanten von *T2* und *T1* werden kopiert und auf *T7* bzw. *T5* umgelenkt.

Wenn die Transitionen der Zustandsdiagramme mit allen dafür in Frage kommenden Transitionen der Aktivitätsdiagramme verschmolzen worden sind, werden sie entfernt. Nun liegt ein einziges, nicht-hierarchisches Petrinetz vor, welches mit verschiedenen Methoden untersucht werden kann.

5 Ergebnisse

Das momentan verwendeten Werkzeug PANDA (siehe [AK00]) ermöglicht es, die Warteschlangenlänge von Aufträgen vor den Aktivitäten und die davon abhängende Wartezeit sowie die Auslastung von Ressourcen zu berechnen. Dazu wird der komplette Zustandsraum aufgestellt, in eine Markovkette umgewandelt und für den stationären Fall gelöst (siehe [MBC+95]).

Um dies um dies durchführen zu können, sind noch zusätzliche Elemente im Petrinetz nötig, die Aufträge mit einer bestimmten Rate in das System senden und sie nach dem Durchlauf entfernen. Ausserdem muss die Anzahl der gleichzeitig im System befindlichen Aufträge beschränkt werden, weil der Zustandsraum sonst unendlich groß werden würde. Die Wahl der richtigen Anzahl ist einerseits für die Genauigkeit der Ergebnisse sowie andererseits die Größe des Zustandsraumes und damit die praktische Berechenbarkeit von Bedeutung. Dies im folgenden verdeutlicht werden.

Für das in diesem Paper vorgestellte Beispiel wurde zunachst eine Ankunftsrate der Aufträge von 1/h gewählt und die maximale Anzahl von Aufträgen auf 25 gesetzt. Es wurden 5 verschiedene, in Tab. 3 angegebene Varianten für die Rollen/Ressourcenzuordnung der Klasse *employee* gewählt. Die Rolle Server wurde dabei ignoriert.

Ressource	Klasse	Rollen
e1	employee	sorter, processor, answerer
e1	employee	sorter, processor, answerer
e2	employee	sorter, processor, answerer
e1	employee	sorter, answerer
e2	employee	processor
e1	employee	sorter, processor, answerer
e2	employee	sorter, processor, answerer
e3	employee	sorter, processor, answerer
e1	employee	sorter
e2	employee	processor
e3	employee	answerer

Tabelle 3: Ressourcentabelle für das Beispiel

In Abb. 7 sind die Auslastung der jeweils eingesetzten Mitarbeiter sowie die Aktivität, bei der sie jeweils beschäftigt sind, zu erkennen. Der Fall, in dem nur ein *employee* alle drei Rollen wahrnehmen muss, ist der kritischste und wurde deshalb weiter untersucht. Dabei wurden die Ankunftsraten auf 1/50 min, 1/45 min, 1/40 min, 1/30 min und 1/20 min verwendet.

Abb. 8(a) zeigt die Zeiten, die ein Auftrag im Mittel vor einer Aktivität warten muss. Abb. 8(b) zeigt das Verhältnis von gewünschter Ankunftsrate und der sich durch die Begrenzung der Auftragszahl ergebenden tatsächlichen Ankunftsrate. Für die Raten 1/20 min, 1/30 min und 1/40 min schwingt sich die errechnete Rate auf 1/45 min ein. Ab 1/50 min besteht Übereinstimmung zwischen gewünschter und errechneter Rate.

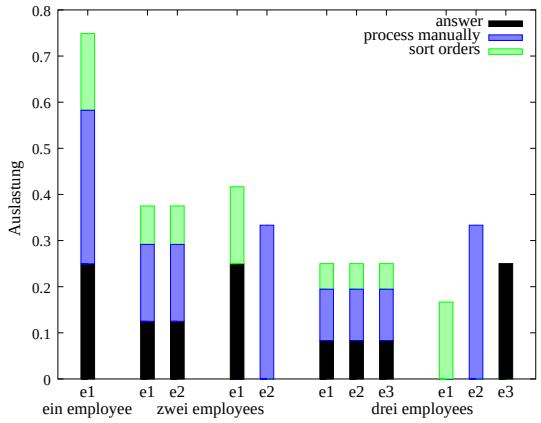
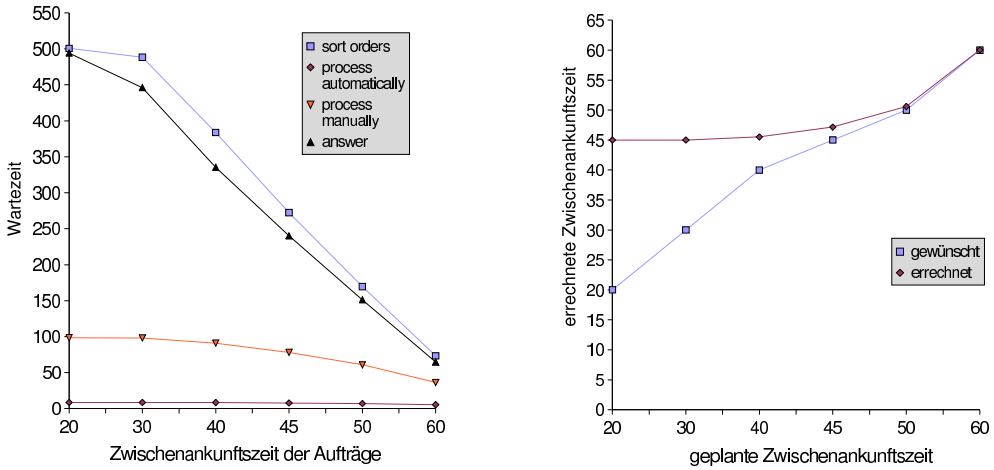


Abbildung 7: Auslastung der *employees*



(a) Wartezeiten vor den Aktivitäten

(b) Auswirkung der Begrenzung

Abbildung 8: Verhalten bei Überlastung des Systems

Die Erklärung für dieses Verhalten besteht darin, dass die maximal mögliche Durchsatzrate des Systems bei 1/45 min liegt. Höhere Ankunftsraten können damit nicht zu einem stabilen Systemzustand führen und das Tool kann somit auch keine sinnvollen Ergebnisse berechen.

Im Bereich zwischen 1/45 min und 1/50 min kommt es zu Verfälschungen des Ergebnisses durch die Beschränkung der Auftragsanzahl im System. Ab 1/50 min ist kaum noch eine Abweichung vorhanden.

6 Schlussfolgerungen und Ausblick

Die vorgestellte Modellierungsmethode ermöglicht es dem Prozessmodellierer, den Geschäftsprozess aus der Sicht des Ablaufes zu modellieren. Er muss sich nicht um Details des Verhaltens von Ressourcen kümmern, sondern kann diese aus einer vorgefertigten Bibliothek auswählen. Die Ressourcenbibliothek kann durch den Ressourcenmodellierer bei Bedarf leicht erweitert werden. Durch die Verwendung von Rollen wird die Untersuchung des Prozessverhaltens bei verschiedenen Ressourcenkonstellationen erheblich vereinfacht.

Das verwendete GSPN-Lösungstool weist prinzipbedingt zwei bekannte Schwächen auf: die Zustandsraumexplosion und die Beschränkung auf exponentiell verteilte Zeiten. Die in Kapitel 5 vorgestellte Begrenzung der Aufträge im System schränkt den Zustandsraum ein, verfälscht aber gleichzeitig die Ergebnisse bei kritischer Belastung. Der Modellierer hat dadurch aber die Möglichkeit, sich systematisch einem akzeptabelen Kompromiss zwischen Genauigkeit und Rechenaufwand anzunähern.

Um leichter verschiedene Auswertetools, welche die erwähnten Schwächen abgemildern bzw. umgehen (z.B. durch Simulation, Phasenverteilungen, effizientere Speichernutzung etc.), leichter anbinden zu können, wird das generierte Petrinetz in der Petri Netz Markup Language (PNML) (siehe [JKW00]) gespeichert. Dies soll auch die Verwendung von Tools zur Untersuchung qualitativer Aspekte der Geschäftsprozesse ermöglichen.

Die Transformationsalgorithmen für Aktivitäts- und Zustandsdiagramme wurden bereits implementiert. Als nächste Aufgabe steht die Verschmelzung hierarchischer Zustandsdiagramme an. Diese können dazu verwendet werden, einer Aktivität bereits allokierte Ressourcen während der Ausführung zu entziehen, was für die Modellierung von Störungen im Geschäftsablauf notwendig ist.

Literaturverzeichnis

[AK00] S. Allmaier and D. Kreische. PANDA — Petri Net ANalysis and Design Assistant Users Guide. Technical report, Institut für Informatik 3, FAU Erlangen-Nürnberg, 2000.

[GGW99] Thomas Gehrke, Ursula Goltz, and Heike Wehrheim. Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen. In E. Schnieder, editor, *Proceedings of The 6th Symposium on Development and Operation of Complex Automation Systems*, 1999.

[Gro01] Object Management Group. *OMG Unified Modeling Language Specification*, 2001. Version 1.4.

[JKW00] M. Jungel, E. Kindler, and M. Weber. The Petri Net Markup Language. In S. Philippi, editor, *Proceedings of AWPN 2000 - 7thWorkshop Algorithmen und Werkzeuge für Petrinetze*, pages 47–52. Research Report 7/2000, Institute for Computer Science, University of Koblenz, Germany, 2000.

[KH00] Konstantinos Kosmidis and Gabor Huszerl. UML Extensions for Quantitative Analysis. In *Proceedings UML 2000 Workshop "Dynamic Behaviour in UML Models: Semantic Questions"*. LMU-München, Insitut für Informatik, 2000.

[Kre01] D. Kreische. Performance and Dependability in Business Process Modeling. In *Proc. 5th Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS 5)*, Erlangen, Germany, 2001. Arbeitsberichte des Instituts für Informatik, FAU Erlangen-Nürnberg.

[Kun00] M. Kunert. Design und Implementierung einer Komponente zur Transformation von UML-Aktivitätsdiagrammen in stochastische Petrinetze. Studienarbeit, Institut für Informatik 3, FAU Erlangen-Nürnberg, 2000.

[MBC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. Wiley, Series in Parallel Computing, 1995.