

Studienarbeit

Strukturelle Reduktion von Verhaltensadaptern

Richard Müller

22. Dezember 2009



Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Zusammenfassung

Ein wesentlicher Bestandteil des Konzeptes des Service-oriented Computing ist das Komponieren bestehender Services zu einem neuen, komplexeren Service. Ein hierbei häufig auftretendes Problem ist, dass zwei Services nur aufgrund kleiner und behebbarer Unterschiede (z.B. in ihrem Verhalten) nicht sinnvoll komponiert werden können. Die Hinzunahme eines dritten Services (eines sogenannten Adapters), welcher diese Unterschiede ausgleicht, ist eine Möglichkeit dieses Problem zu lösen.

Es existiert eine große Zahl verschiedener Methoden und Modelle, um Services (und damit auch Adapter) zu modellieren, unter anderem eignen sich hierfür Petrinetze. Petrinetze sind ein Formalismus, welcher die Möglichkeit bietet, über die Reduktion der Struktur des Modells den Zustandsraum zu reduzieren, so dass aber gleichzeitig bestimmte Eigenschaften des Zustandsraumes bewahrt bleiben. Welche Eigenschaften genau bewahrt bleiben ist dabei abhängig von den angewandten Reduktionsregeln.

Die vorliegende Arbeit beschäftigt sich mit der strukturellen Reduktion von Petrinetzen im Rahmen der Synthese von Verhaltensadaptern.

Inhaltsverzeichnis

1	Einleitung	2
2	Grundlagen	4
2.1	Petrinetze	4
2.2	Offene Netze	8
2.3	Adaptersynthese	16
3	Strukturelle Reduktion	21
3.1	Isolierte Plätze	21
3.2	Siphons	25
3.3	Fallen	27
3.4	Anwendung auf Adaptersynthese	31
4	Fazit	35
4.1	Verwandte Arbeiten	35
	Abbildungsverzeichnis	37
	Tabellenverzeichnis	38
	Literaturverzeichnis	39

1 Einleitung

Entwickler von Software für den Unternehmenseinsatz sehen sich in der Regel mit hohen Anforderungen wie Flexibilität und Wiederverwendbarkeit bei gleichzeitiger Kosteneffizienz konfrontiert, welche für das Unternehmen im heutigen Geschäftsumfeld von essentieller Natur sind. Im Zuge dessen setzt sich allmählich eine Methode aus dem Bereich der verteilten Systeme durch, in welcher Dienste von Mitarbeitern und Organisationen als *Services* strukturiert und genutzt werden. Ein Service ist hierbei Software, welche eine bestimmte *innere Funktionalität* hinter einem *wohldefinierten Interface* kapselt, wobei die innere Funktionalität von einfachen Funktionen bis hin zu kompletten Geschäftsprozessen reicht. Typische und schon heute weit verbreitete Implementationen von Services sind die sogenannten *Web Services*, bekannte Vertreter gibt es z.B. von Google Inc. oder Amazon.com.

Service-oriented Computing ist ein neues Programmierparadigma, um aus Services große und komplexe Software zu entwerfen. Ein wesentlicher Bestandteil dieses Konzeptes ist das *Komponieren* bestehender Services zu einem neuen, komplexeren Service. Als Beispiel für zwei zu komponierende Services können wir uns einen Käufer und einen Verkäufer vorstellen. Der Käufer gibt eine Bestellung auf und wartet mit der Bezahlung bis zum zum Erhalt der zuvor bestellten Ware, wohingegen der Verkäufer eine Bestellung entgegen nimmt und anschließend mit dem Versand der bestellten Ware bis zum Erhalt der entsprechenden Bezahlung wartet. Wir merken, dass die Zusammenarbeit dieses Käufers mit diesem Verkäufer nicht ohne Weiteres erfolgreich möglich ist, da beide letzten Endes auf den jeweils Anderen warten werden. Ebenso können wir uns allerdings einen dritten Service in der Art einer Bank vorstellen, welcher die Bezahlung für den Käufer auslegt. Anschließend wird der Verkäufer die Ware abschicken und der dritte Service das ausgelegte Geld vom Käufer nach Ankunft der Ware zurückerhalten, was einer erfolgreichen Zusammenarbeit von Käufer und Verkäufer entspricht. Dies illustriert das häufig auftretende Problem, dass zwei Services nur aufgrund behebbarer Unterschiede (in diesem Fall in ihrem Verhalten) nicht komponiert werden können. Eine Möglichkeit, dieses Problem zu lösen, besteht aus der Hinzunahme eines dritten Service (eines sogenannten *Adapters*), welcher gerade diese kleinen Unterschiede ausgleicht.

Wie in der Informatik üblich, gibt es auch beim Konzept der Services Bestrebungen, diese mit standardisierten Methoden und/oder einem (möglichst formalen) Modell zu beschreiben. Als hilfreich haben sich hierbei *offene Netze*, eine spezielle Klasse von *Petrinetzen*, erwiesen. So existiert zum Beispiel für WS-BPEL, einer weit verbreiteten Sprache zur Beschreibung von Geschäftsprozessen, welche durch Web Services implementiert sind, eine vollständige, mit offenen Netzen modellierte Semantik (siehe [Loh08]). Des Weiteren wurden die Probleme der Synthese eines verhaltenskompatiblen Services zu einem gegebenen

Service, der endlichen Charakterisierung aller verhaltenskompatiblen Services ([Mas09]) sowie der Synthese eines Verhaltensadapters ([GMW08]) auf der Basis von offenen Netzen als Service-Modell gelöst. Die Wahl von Petrinetzen zur Modellierung hat mehrere Vorteile: Zum Einen können komplexe Systeme dank der explizit dargestellten Nebenläufigkeit gut modelliert werden, zum Anderen sind Petrinetze ein Formalismus, welcher die Möglichkeit bietet, über die Reduktion der Struktur des Modells den Zustandsraum zu reduzieren, so dass aber gleichzeitig bestimmte Eigenschaften des Zustandsraumes bewahrt bleiben. Welche Eigenschaften genau bewahrt bleiben ist dabei abhängig von den angewandten Reduktionsregeln. Inwiefern die strukturelle Reduktion von Petrinetzen bei der Synthese von Verhaltensadaptern hilfreich anwendbar ist, wollen wir innerhalb dieser Ausarbeitung erläutern.

In [GMW08] wird die Synthese von Verhaltensadaptern als zweistufiger Prozess beschrieben, dessen erster Schritt aus der Konstruktion eines speziellen Services (*Engine* genannt) besteht. Dieser Service modelliert genau alle Fertigkeiten der Nachrichtenverarbeitung des späteren Adapters. Im zweiten Schritt wird ein *Controller* genannter Service synthetisiert, welcher die Nachrichtenverarbeitung innerhalb der Engine genau so steuert, dass die Komposition von Engine und Controller den gesuchten Verhaltensadapter ergibt. Das Ziel dieser Arbeit ist es, diesen zweistufigen Prozess zu verbessern. Verbessern ist hierbei in dem Sinne zu verstehen, dass wir sowohl einen schnelleren Ablauf der Adapterynthese als auch kleinere Verhaltensadapter als Endergebnis wollen. Wir werden sehen, dass beide Ziele durch die gleiche Vorgehensweise, nämlich die strukturelle Reduktion der Engine, erreicht werden können.

Die vorliegende Arbeit gliedert sich in vier Kapitel, deren Schwerpunkte sich wie folgt darstellen lassen: In Kapitel 2 werden wir die grundlegenden Begriffe und Notationen von Petrinetzen und offenen Netzen definieren sowie den Ablauf der Adaptersynthese beschreiben. In Kapitel 3 schließlich beschäftigen wir uns mit Regeln zur strukturellen Reduktion offener Netze im Rahmen der Adaptersynthese. Abschließend werden wir in Kapitel 4 die Ergebnisse dieser Arbeit zusammenfassen.

2 Grundlagen

In diesem Kapitel beschäftigen wir uns mit den Grundlagen der Adaptersynthese. Zuerst werden Petrinetze und deren übliche Notationen eingeführt, anschließend gehen wir näher auf offene Netze als eine spezielle Klasse von Petrinetzen ein. Schlussendlich werden wir den genauen Ablauf der Adaptersynthese auf der Basis von offenen Netzen erläutern.

2.1 Petrinetze

Petrinetze sind ein Formalismus zur Modellierung und Analyse verteilter und reaktiver Systeme. In diesem Abschnitt werden wir die grundlegenden Begriffe und Notationen klassischer Petrinetze in einem Maß wiederholen, wie es für die folgende Arbeit notwendig ist. Umfangreichere Einführungen finden sich zum Beispiel in [Rei85] oder [Sta90].

Ein Petrinetz hat eine statische und eine dynamische Seite, wobei erstere durch ein *Netz* repräsentiert wird. Dieses wiederum ist ein endlicher, gerichteter, bipartiter Graph, dessen disjunkte Teilmengen der Knotenmenge als *Plätze* und *Transitionen*, und dessen Kanten als *Flussrelation* bezeichnet werden. Eine formale Definition geben wir im Folgenden an:

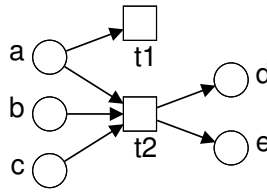
Definition 1 (Netz)

Ein Netz ist ein Tripel $N = (P, T, F)$, wobei

- P eine endliche Menge von Plätzen,
- T eine endliche Menge von Transitionen,
- $P \cap T = \emptyset$ und
- $F \subseteq (P \times T) \cup (T \times P)$ die Flussrelation ist. ┘

Ein Vorteil von Petrinetzen gegenüber anderen formalen Modellierungsmethoden ist deren einfache und intuitive grafische Repräsentation. Hierbei werden wir Plätze als Kreise, Transitionen als Quadrate und die Flussrelation als gerichtete Pfeile zwischen Plätzen und Transitionen bzw. zwischen Transitionen und Plätzen repräsentieren. Als Beispiel für die grafische Notation finden wir in Abbildung 2.1 das Netz $N = (\{a, b, c, d, e\}, \{t1, t2\}, \{(a, t1), (a, t2), (b, t2), (c, t2), (t2, d), (t2, e)\})$.

Um über die Struktur eines Netzes zu sprechen, werden wir häufig die Notation des *Vor-* oder *Nachbereiches* eines Knotens benutzen.

Abbildung 2.1: Die grafische Notation des Netzes N .**Definition 2 (Vor- und Nachbereich)**

Sei $N = (P, T, F)$ ein Netz und $x \in P \cup T$. Dann bezeichnet $\bullet x = \{y \mid (y, x) \in F\}$ den Vorbereich von x und $x^\bullet = \{y \mid (x, y) \in F\}$ den Nachbereich von x . Für $X \subseteq P \cup T$ sei $\bullet X = \bigcup_{x \in X} \bullet x$ und $X^\bullet = \bigcup_{x \in X} x^\bullet$. \lrcorner

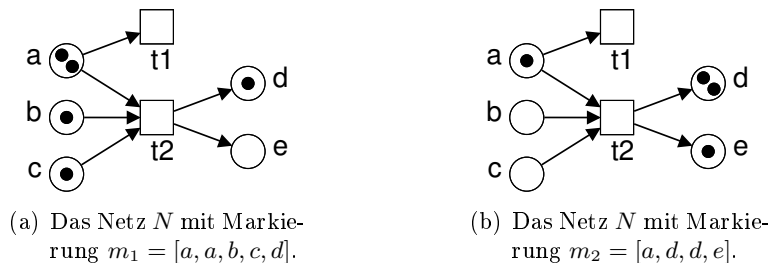
Beispielsweise hat die Transition $t2$ aus dem Netz N in Abbildung 2.1 den Vorbereich $\bullet t2 = \{a, b, c\}$ und den Nachbereich $t2^\bullet = \{d, e\}$. Der Nachbereich der Menge $\{a, b\}$ ist $\{a, b\}^\bullet = a^\bullet \cup b^\bullet = \{t1, t2\}$.

Um unterschiedliche Zustände eines Netzes beschreiben zu können, werden wir *Marken* einführen (grafisch dargestellt durch schwarze Punkte). Eine Verteilung von Marken auf die Plätze repräsentiert den Zustand eines Netzes und wird *Markierung* genannt.

Definition 3 (Markierung)

Sei $N = (P, T, F)$ ein Netz und $P' \subseteq P$. Dann bezeichnet eine Abbildung $m : P \rightarrow \mathbb{N}$ eine Markierung von N . Mit $m|_{P'} : P' \rightarrow \mathbb{N}$ wird die Projektion von m auf P' bezeichnet.

Eine Markierung von N kann als Multimenge über P angegeben werden. \lrcorner

Abbildung 2.2: Das Netz N mit zwei unterschiedlichen Markierungen.

In Abbildung 2.2 sehen wir das Netz N mit zwei unterschiedlichen Markierungen: m_1 und m_2 . Bei der Angabe einer Markierung als Multimenge über die Menge der Plätze werden wir die Notation in eckigen Klammern verwenden, um Multimengen besser von normalen Mengen unterscheiden zu können.

Ein Petrinetz ist nun genau ein Netz mit einer initialen Markierung.

Definition 4 (Petrietz)

Ein Petrietz ist ein 4-Tupel $N = (P, T, F, m_0)$, wobei

- (P, T, F) ein Netz und
- m_0 eine Markierung von (P, T, F) ist.

Die Markierung m_0 wird Anfangsmarkierung von N genannt. ┘

Bisher wurde nur die statische Seite eines Petrietzes beschrieben. Die Dynamik ergibt sich aus der Möglichkeit, dass Transitionen unter bestimmten Bedingungen durch *Schalten* die Markierung des Petrietzes ändern können, d.h. Transitionen *konsumieren* bzw. *produzieren* Marken.

Definition 5 (Schalten)

Sei $N = (P, T, F, m_0)$ ein Petrietz, $t \in T$ eine Transition und m eine Markierung von N .

- (a) Die Transition t hat Konzession bzw. ist aktiviert in der Markierung m (kurz: $m \xrightarrow{t}$), wenn gilt: $m(p) \geq 1 \forall p \in \bullet t$. Andernfalls schreiben wir $m \not\xrightarrow{t}$.
- (b) Wenn t Konzession in m hat, dann kann t schalten. Durch das Schalten von t entsteht eine neue Markierung m' (notiert als $m \xrightarrow{t} m'$), wobei für alle $p \in P$ gilt:

$$m'(p) = \begin{cases} m(p) + 1 & , \text{ falls } p \in t^\bullet \setminus \bullet t, \\ m(p) - 1 & , \text{ falls } p \in \bullet t \setminus t^\bullet, \\ m(p) & , \text{ sonst.} \end{cases}$$
┘

Beispielsweise hat die Transition t_2 aus dem Netz N in Abbildung 2.2(a) Konzession in der Markierung m_1 . Nach dem Schalten von t_2 befindet sich N in der Markierung m_2 , zu sehen in Abbildung 2.2(b).

Durch das Schalten einer einzigen Transition wird eine Markierung in eine neue Markierung überführt. Dies kann auch durch das Schalten einer Sequenz von Transitionen erfolgen, was wir als *Schaltsequenz* induktiv wie folgt definieren:

Definition 6 (Schaltsequenz)

Sei $N = (P, T, F, m_0)$ ein Petrietz, m, m' Markierungen von N und $\sigma = t_1, \dots, t_n$ eine Sequenz von Transitionen. σ heißt Schaltsequenz von m nach m' in N (kurz: $m \xrightarrow{\sigma} m'$), wenn gilt:

Anfang: $\sigma = \epsilon$ und $m' = m$.

Schritt: $\sigma = t_1, \dots, t_n, t_{n+1}$

Dann existiert eine Markierung m'' von N , so dass:

$$m \xrightarrow{t_1, \dots, t_n} m'' \text{ und } m'' \xrightarrow{t_{n+1}} m'. \quad \text{┘}$$

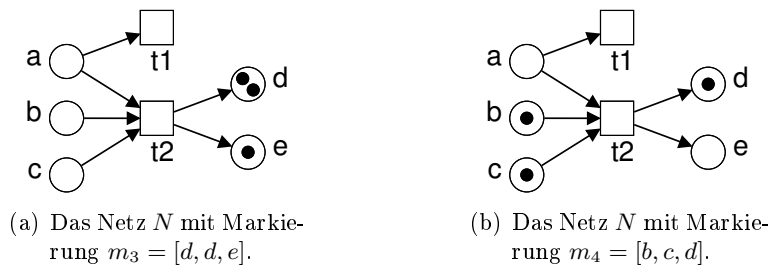


Abbildung 2.3: Das Netz N mit zwei weiteren Markierungen.

Die Transitionssequenz t_1, t_2 ist eine Schaltsequenz von m_1 nach m_3 im Netz N , wie wir in Abbildung 2.3(a) sehen. Da die Transitionen innerhalb einer Schaltsequenz nicht notwendigerweise verschieden sein müssen, ist z.B. auch die Transitionssequenz t_1, t_1 eine Schaltsequenz von m_1 nach m_4 in N . Die Markierung m_4 ist in Abbildung 2.3(b) dargestellt.

Wenn wir mittels einer Schaltsequenz eine Markierung in eine neue Markierung überführen können, so heißt die neue Markierung *erreichbar*.

Definition 7 (Erreichbarkeit und $R_N(m)$)

Sei $N = (P, T, F, m_0)$ ein Petrinetz und m, m' Markierungen von N . Die Markierung m' ist erreichbar von m in N , wenn eine Schaltsequenz von m nach m' in N existiert.

Die Menge $R_N(m)$ der von m in N erreichbaren Markierungen ist wie folgt definiert: $R_N(m) := \{m' \mid \text{es existiert eine Transitionssequenz } \sigma, \text{ so dass } m \xrightarrow{\sigma} m'\}$. ┘

Die von der Anfangsmarkierung eines Petrinetzes erreichbaren Markierungen bezeichnen schließlich den *Zustandsraum* des Petrinetzes.

Definition 8 (Zustandsraum)

Sei $N = (P, T, F, m_0)$ ein Petrinetz. Dann bezeichnet $R_N := R_N(m_0)$ den Zustandsraum von N . ┘

2.2 Offene Netze

Services werden konstruiert, um mit anderen Services zu kommunizieren. Diese Kommunikation kann asynchron oder synchron ablaufen, wobei wir im Folgenden nur die für Web Services praxisrelevante asynchrone Kommunikation betrachten werden. Da wir Petrinetze als Modell für Services verwenden, müssen diese natürlich auch eine Möglichkeit zur Kommunikation untereinander besitzen. Dies wird durch die Erweiterung von Petrinetzen zu sogenannten *offenen Netzen* realisiert, welche zusätzliche *Input-* und *Outputplätze* sowie *Endmarkierungen* besitzen. Input- und Outputplätze bilden das *Interface* eines offenen Netzes analog zum Interface eines Services und erlauben das Empfangen und Senden von Nachrichten. Endmarkierungen hingegen repräsentieren die erfolgreichen Terminierungen von Abläufen innerhalb eines Services.

Definition 9 (Offenes Netz)

Ein offenes Netz ist ein 7-Tupel $N = (P, P_i, P_o, T, F, m_0, \Omega)$, wobei

- (P, T, F, m_0) ist ein Petrinetz,
- $P_i \subseteq P$ mit $\bullet P_i = \emptyset$ ist eine Menge von Inputplätzen,
- $P_o \subseteq P$ mit $P_o^\bullet = \emptyset$ ist eine Menge von Outputplätzen und
- Ω ist eine Menge von Endmarkierungen in N mit $m(p) = 0 \forall m \in \Omega \forall p \in P_i \cup P_o$.

Des Weiteren gilt $P_i \cap P_o = \emptyset$ und $m_0(p) = 0 \forall p \in P_i \cup P_o$.

Die Menge $I := P_i \cup P_o$ heißt *Interface* von N . Ein Platz aus I wird *Interfaceplatz*, ein Platz aus $P \setminus I$ wird *interner Platz* genannt. Die Bestandteile eines offenen Netzes N werden wir häufig mit $P_N, P_{i_N}, P_{o_N}, T_N, F_N, m_{0_N}, \Omega_N$ oder I_N bezeichnen. \lrcorner

Graphisch werden Plätze, Transitionen und die Flussrelation auch in offenen Netzen wie gewohnt als Kreise, Quadrate und Pfeile repräsentiert. Plätze, welche mit mindestens einer Marke in mindestens einer Endmarkierung vorkommen, werden durch ein kleines ω gekennzeichnet. Das Interface wird durch eine gestrichelte Linie um das Netz, auf welcher genau alle Interfaceplätze liegen, dargestellt. Zwei Beispiele für die grafische Notation von offenen Netzen finden wir in der Abbildung 2.4.

Beispiel 1 In Abbildung 2.4 sehen wir das offene Netz *buyer* mit der Platzmenge $\{p1, p2, p3, p4, Order, Good, Pay\}$, der Transitionsmenge $\{!Order, ?Good, !Pay\}$, der Anfangsmarkierung $[p1]$ und der Endmarkierung $\{[p4]\}$ sowie das offene Netz *seller* mit der Platzmenge $\{p5, p6, p7, p8, Order, Good, Pay\}$, der Transitionsmenge $\{?Order, !Good, ?Pay\}$, der Anfangsmarkierung $[p5]$ und der Endmarkierung $\{[p8]\}$. Die entsprechenden Flussrelationen und Interfaceplätze sind in den Abbildungen 2.4(a) und 2.4(b) zu erkennen. Beide Netze werden wir innerhalb dieser Arbeit, analog zur Beschreibung in der Einleitung, als durchgängiges Beispiel betrachten. Das Netz *buyer* modelliert den Käufer,

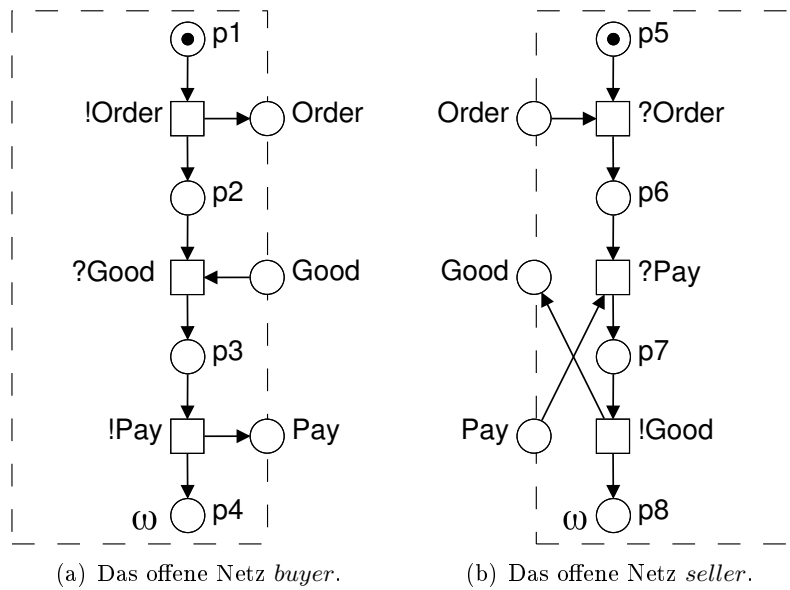


Abbildung 2.4: Die offenen Netze *buyer* und *seller*.

welcher eine Bestellung aufgibt (Outputplatz *Order*), auf die bestellte Ware wartet (Inputplatz *Good*) und anschließend bezahlt (Outputplatz *Pay*). Das Netz *seller* modelliert hingegen den Verkäufer, welcher eine Bestellung aufnimmt (Inputplatz *Order*), auf die Bezahlung der bestellten Ware wartet (Inputplatz *Pay*) und diese anschließend verschickt (Outputplatz *Good*). \lrcorner

Jedes offene Netz hat eine *innere Struktur*, bestehend aus dem zugrunde liegenden Petri-Netz ohne Interfaceplätze. Über diese innere Struktur können wir bestimmte Eigenschaften eines offenen Netzes definieren, wie zum Beispiel seine *Beschränktheit*.

Definition 10 (Innere Struktur)

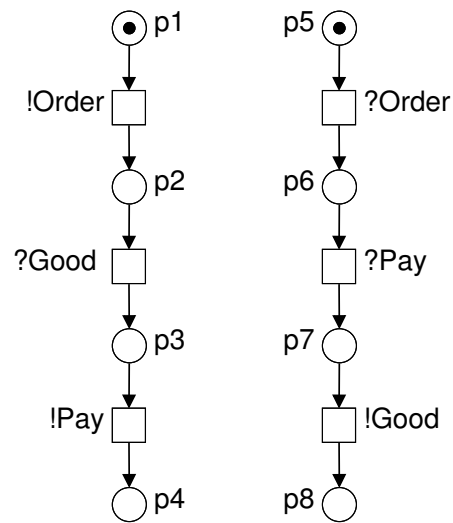
Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz. Seine innere Struktur $inner(N)$ ist definiert als das Petri-Netz $inner(N) = (P \setminus I, T, F \setminus ((I \times T) \cup (T \times I)), m_0|_{P \setminus I})$. \lrcorner

Definition 11 (Beschränktheit)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz. N heißt genau dann beschränkt, wenn die Menge $R_{inner(N)}$ endlich ist. \lrcorner

Beispiel 2 Auf jedem Platz der inneren Struktur der offenen Netze *buyer* und *seller* liegt maximal eine Marke, wie wir in den Abbildungen 2.5(a) und 2.5(b) erkennen können. Folglich sind die offenen Netze *buyer* und *seller* beschränkt. \lrcorner

Die Kommunikation zwischen zwei offenen Netzen wird als *Komposition* bezeichnet. Hierbei werden jeweils die Inputplätze des einen Netzes mit den passenden Outputplätzen



(a) Die innere Struktur von *buyer*. (b) Die innere Struktur von *seller*.

Abbildung 2.5: Die innere Struktur der offenen Netze *buyer* und *seller*.

des anderen Netzes verschmolzen. Ohne Beschränkung der Allgemeinheit nehmen wir dabei an, dass die Mengen der Transitionen und der internen Plätze aller beteiligten Netze paarweise disjunkt sind, was im Vorfeld durch simples Umbenennen erreicht werden kann.

Definition 12 (Komposition offener Netze)

Seien $N_1 = (P_1, P_{i_1}, P_{o_1}, T_1, F_1, m_{0_1}, \Omega_1)$ und $N_2 = (P_2, P_{i_2}, P_{o_2}, T_2, F_2, m_{0_2}, \Omega_2)$ offene Netze. N_1 und N_2 heißen genau dann komponierbar, wenn $P_{i_1} \cap P_{i_2} = \emptyset$ und $P_{o_1} \cap P_{o_2} = \emptyset$.

Wenn N_1 und N_2 komponierbar sind, dann ist ihre Komposition $N = N_1 \oplus N_2 = (P, P_i, P_o, T, F, m_o, \Omega)$ folgendermaßen definiert:

- $P = P_1 \cup P_2,$
- $P_i = (P_{i_1} \cup P_{i_2}) \setminus (P_{o_1} \cup P_{o_2}),$
- $P_o = (P_{o_1} \cup P_{o_2}) \setminus (P_{i_1} \cup P_{i_2}),$
- $T = T_1 \cup T_2,$
- $F = F_1 \cup F_2,$
- $m_0 = m_{0_1} \oplus m_{0_2}$ und
- $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}.$

Für Markierungen m_1 von N_1 und m_2 von N_2 ist die Markierung $m_1 \oplus m_2$ definiert als

$$(m_1 \oplus m_2)(p) = \begin{cases} m_1(p) & , \text{ falls } p \in P_1, \\ m_2(p) & , \text{ sonst.} \end{cases}$$

Für einen Platz $p \in P_1 \cap P_2$ ist diese Definition eindeutig, da es sich bei p um einen Interfaceplatz handelt und folglich $m_1(p) = m_2(p) = 0$ für Anfangs- und Endmarkierungen gilt. ┘

Das Ergebnis der Komposition ist wieder ein offenes Netz, wie wir beispielsweise in Abbildung 2.6 erkennen können.

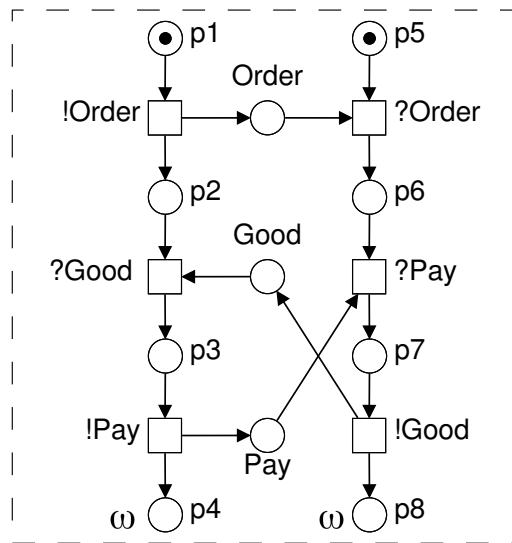


Abbildung 2.6: Das offene Netz $buyer \oplus seller$.

Beispiel 3 In Abbildung 2.6 sehen wir die Komposition der beiden offenen Netze *buyer* und *seller*: Das offene Netz $buyer \oplus seller$ mit der Anfangsmarkierung $[p1, p5]$, der Menge der Endmarkierungen $\{[p4, p8]\}$ sowie einem leeren Interface. ┘

Der Schwerpunkt unseres Interesses liegt auf dem Verhalten von Services, welche wir nun als offene Netze modellieren können. Folglich müssen wir auch das Verhalten von Services auf der Ebene der offenen Netze definieren. Hierbei kann zwischen unerwünschtem und erwünschtem Verhalten unterschieden werden: Unerwünschtes Verhalten bedeutet für uns beispielsweise, dass ein modellierter Service in einen „Fehlerzustand“ geraten kann, welcher kein Endzustand ist, und welcher auch nicht mehr verlassen werden kann. Solch ein Zustand wird als *Deadlock* bezeichnet.

Definition 13 (Deadlock)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz und m eine Markierung in N mit $m \notin \Omega$ und $m \not\stackrel{t}{\rightarrow}$ für alle $t \in T$, d.h. keine Transition hat Konzession in m . Dann heißt m Deadlock in N . ┘

Eine erwünschte Verhaltenseigenschaft hingegen ist, dass der modellierte Service immer einen Endzustand erreichen kann. Solch ein Service heißt *schwach terminierend*.

Definition 14 (Schwach terminierend)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz. N heißt genau dann schwach terminierend, wenn für jede Markierung $m \in R_N$ gilt: Es existiert eine Markierung $m' \in R_N(m)$ mit $m' \in \Omega$.

Im Folgenden werden wir schwach terminierend mit ST abkürzen. ┘

Wie wir anhand obiger Definitionen erkennen können, ist ein Deadlock eine Markierung, aus der keine Endmarkierung mehr erreicht werden kann. Folglich ist in einem schwach terminierenden offenen Netz kein Deadlock erreichbar, was wir im folgenden Korollar festhalten werden. Die Umkehrung dieses Korollares gilt im Allgemeinen nicht.

Korollar 1

Sei N ein offenes Netz. Falls N schwach terminierend ist, so ist keine Markierung m von N mit $m \in R_N$ ein Deadlock. ┘

Angenommen, wir haben zwei Services als offene Netze N_1 und N_2 modelliert. Dank Definition 14 können wir erwünschtes Verhalten der beiden modellierten Services untersuchen, indem wir Verhaltenseigenschaften der Komposition $N_1 \oplus N_2$ betrachten. Sollte diese Komposition nicht das erwünschte Verhalten aufweisen, so stellt sich uns intuitiv die Frage, ob dieses Fehlverhalten eindeutig an einem der beiden Services festgemacht werden kann. Dies wäre zum Beispiel bei N_1 der Fall, wenn überhaupt kein offenes Netz existiert, so dass seine Komposition mit N_1 das gewünschte Verhalten zeigt. Unsere Lösung zur Formalisierung hierfür ist die Idee der Bedienbarkeit: Ein offenes Netz ist genau dann bedienbar, wenn es mit einem anderen offenen Netz komponiert werden kann, so dass das Resultat ein offenes Netz mit der gewünschten Verhaltenseigenschaft ist.

In [MSSW08] wurde gezeigt, dass die Bedienbarkeit eines unbeschränkten Netzes unentscheidbar ist. Um unbeschränkte Netze trotzdem untersuchen zu können, werden wir sie nur in Verbindung mit einer Einschränkung der erreichbaren Markierungen der Komposition mit einem anderen offenen Netz betrachten. Diese Einschränkung werden wir in Form einer Funktion f realisieren, welche allen Plätzen (auch den Interfaceplätzen) eines Netzes eine maximale Anzahl an Marken zuweist, bis zu welcher eine Markierung der Komposition betrachtet wird. Mit einer gegebenen festen Funktion f ist damit der Zustandsraum eines Netzes de facto endlich und seine Bedienbarkeit entscheidbar.

Definition 15 (Bedienbarkeit)

Sei N_1 ein offenes Netz und $f : P_{N_1} \rightarrow \mathbb{N}$ beliebig. N_1 heißt genau dann ST,f-bediensbar, wenn ein komponierbares und beschränktes offenes Netz N_2 existiert, so dass $N_1 \oplus N_2$ ein schwach terminierendes offenes Netz ist und für alle Markierungen $m \in R_{N_1 \oplus N_2}$ gilt: $m(p) \leq f(p) \forall p \in P_{N_1}$.

Solch ein Netz N_2 wird ST,f-Controller von N_1 genannt. ┘

Da die Funktion f alle Plätze eines offenen Netzes in der Komposition mit einem ST,f-Controller beschränkt, muss eben diese Komposition auch beschränkt sein.

Korollar 2

Sei N_1 ein offenes Netz und N_2 ein ST,f-Controller von N_1 . Dann ist das offene Netz $N_1 \oplus N_2$ beschränkt. ┘

Beispiel 4 In Abbildung 2.7 sehen wir das offene Netz *buyer* sowie das beschränkte offene Netz *controllerB*. Da $buyer \oplus controllerB$ ein schwach terminierendes offenes Netz ist und in keiner erreichbaren Markierung mehr als eine Marke auf einem ehemaligen Platz von *buyer* liegt, ist *buyer* ST,f-bediensbar mit $f(p) \geq 1 \forall p \in P_{buyer}$. Somit stellt *controllerB* einen ST,f-Controller von *buyer* dar. Analog können wir in Abbildung 2.8 die offenen Netze *seller* und *controllerS* erkennen, wobei *controllerS* beschränkt und ein ST,f-Controller mit $f(p) \geq 1 \forall p \in P_{seller}$ von *seller* ist. ┘

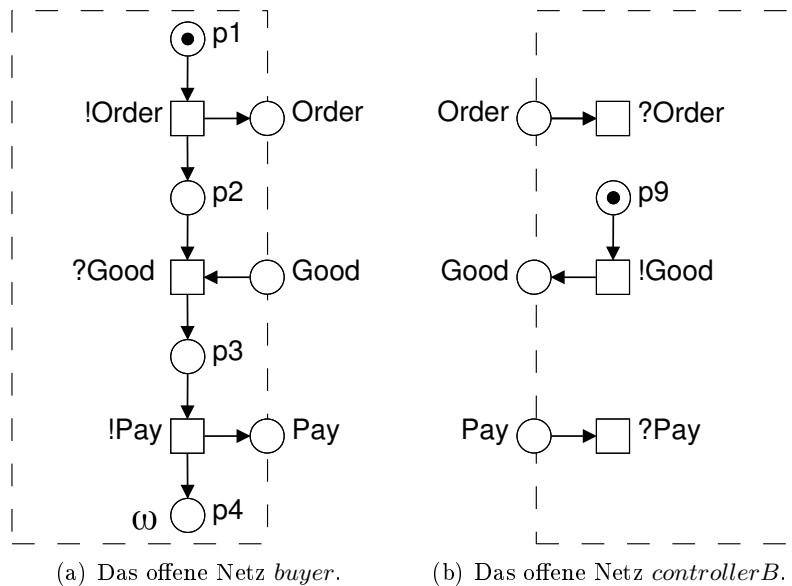


Abbildung 2.7: Die offenen Netze *buyer* und *controllerB*.

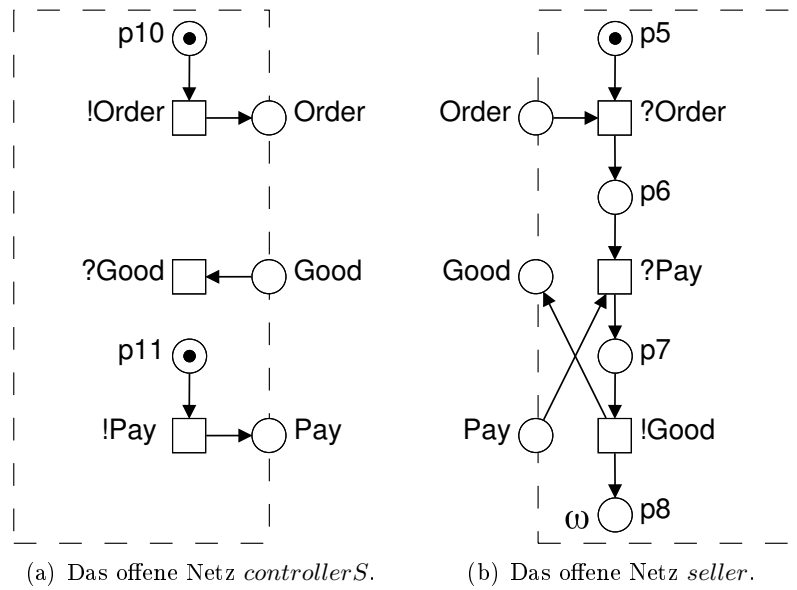


Abbildung 2.8: Die offenen Netze *seller* und *controllerS*.

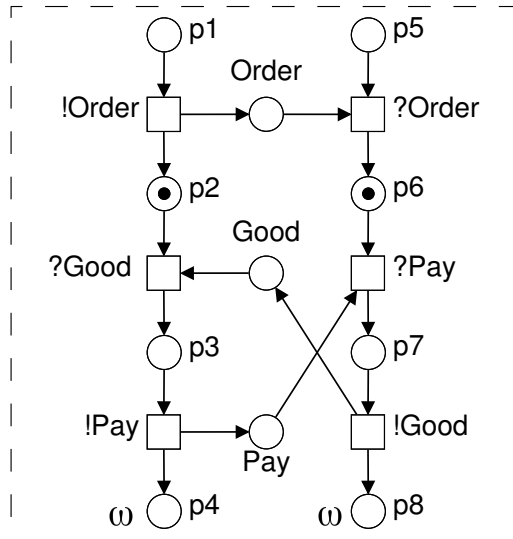


Abbildung 2.9: Das offene Netz *buyer* \oplus *seller* im Deadlock.

Beispiel 5 Betrachten wir noch einmal die aus Abbildung 2.4 bekannten offenen Netze *buyer* und *seller*, welche beide jeweils ST,f-bediener sind (siehe vorheriges Beispiel). Bekannterweise modellieren sie einen Käufer und einen Verkäufer, deren Zusammenarbeit aufgrund ihres Verhaltens nicht sinnvoll möglich ist: Nach der Verarbeitung der abgegebenen Bestellung wird der Käufer, bevor er bezahlt, auf die Waren des Verkäufers warten, während hingegen der Verkäufer auf die Bezahlung wartet, bevor er die entsprechenden Waren verschickt. Dieses unerwünschte Verhalten können wir auch in der Komposition der beiden offenen Netze anhand des erreichbaren Deadlocks in Abbildung 2.9 erkennen. Somit ist *seller* kein ST,f-Controller für *buyer* (und umgekehrt) und für eine sinnvolle Zusammenarbeit der beiden Services ist ein Adapter erforderlich. ┘

2.3 Adaptersynthese

Wie bereits in der Einleitung beschrieben, bezeichnen wir einen Service, welcher Unterschiede zwischen anderen Services ausgleicht, als *Adapter*. Haben wir nun mehrere Services gegeben und möchten hieraus (möglichst automatisch) einen Adapter synthetisieren, so sprechen wir vom *Adaptersyntheseproblem*. Da wir Services als offene Netze modellieren und uns in dieser Arbeit nur um deren Verhalten kümmern, können wir das Adaptersyntheseproblem auch auf dieser Ebene formulieren: Seien n offene Netze S_1, \dots, S_n gegeben. Wie können wir ein offenes Netz A synthetisieren, so dass das offene Netz $(S_1 \oplus \dots \oplus S_n) \oplus A$ das gewünschte Verhalten (z.B. ST) hat? Ohne Beschränkung der Allgemeinheit gehen wir im Folgenden davon aus, dass die Interface der Netze S_1, \dots, S_n paarweise disjunkt sind, was im Vorfeld durch simples Umbenennen erreicht werden kann.

Definition 16 (Adapter)

Seien S_1, \dots, S_n offene Netze. Ein offenes Netz A heißt genau dann Adapter für S_1, \dots, S_n , wenn A ein ST,f-Controller von $S_1 \oplus \dots \oplus S_n$ ist. ┘

Ohne weitere Einschränkungen existiert für dieses Problem eine triviale Lösung: Für jedes der gegebenen offenen Netze S_i synthetisieren wir einen ST,f-Controller C_i . Anschließend komponieren wir die synthetisierten ST,f-Controller zu einem Netz $C_1 \oplus \dots \oplus C_n$, welches trivialerweise eine valide Lösung des Adaptersyntheseproblems und damit einen Adapter darstellt. Ein Beispiel für einen nach dieser Vorgehensweise synthetisierten Adapter finden wir in Abbildung 2.10(b).

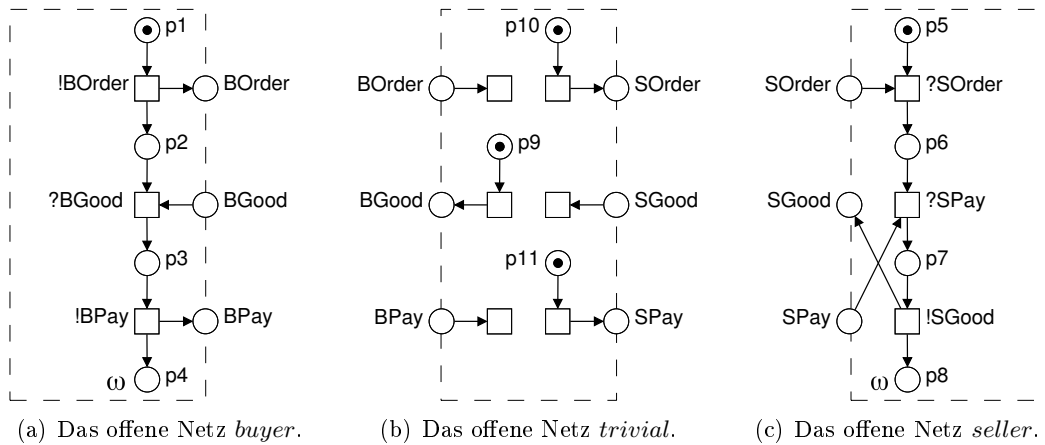


Abbildung 2.10: Die offenen Netze *buyer* und *seller* sowie ein Adapter *trivial*.

Beispiel 6 In Abbildung 2.10 sehen wir die offenen Netze *buyer* und *seller* mit disjunktem Interface. Für jedes dieser beiden Netze wurde jeweils ein ST,f-Controller mit $f(p) = 1 \forall p \in P_{buyer}$ bzw. $f(p) = 1 \forall p \in P_{seller}$ synthetisiert. Die Komposition dieser beiden ST,f-Controller bildet den Adapter *trivial*, da das Netz $(buyer \oplus seller) \oplus trivial$

schwach terminierend ist. Wie wir an den Outputplätzen *BGood* und *SPay* erkennen können, erzeugt der Adapter *trivial* Nachrichten, welche reellen Gütern entsprechen. Dies widerspricht unserer Intuition und stellt ein großes Hindernis bei einer möglichen Implementation von *trivial* dar. \lrcorner

Da die Netze C_i jeweils ST,f-Controller für S_i darstellen, erlaubt diese triviale Lösung unter anderem Adapter, welche Nachrichten mit reellen Gütern (z.B. Geld) oder semantisch relevantem Inhalt (z.B. Passwörter) erzeugen. Dies widerspricht gewöhnlich den Möglichkeiten, die wir einem Adapter in der realen Welt zugestehen, da ein Adapter, welcher z.B. Passwörter kennt, ein Sicherheitsrisiko darstellt. Folglich benötigen wir eine zusätzliche Anforderung an eine Lösung des Adaptersyntheseproblem: Eine Beschreibung aller Fertigkeiten, welche der Adapter zur Nachrichtenverarbeitung nutzen kann. Solche Fertigkeiten umfassen im Allgemeinen das Erzeugen, Löschen, Kopieren, Transformieren, Aufspalten, Zusammenfassen oder Umleiten von Nachrichten.

In [GMW08] wird eine *Spezifikation elementarer Aktivitäten* (kurz: SEA) als Beschreibung aller Fertigkeiten des Adapters vorgeschlagen. Diese besteht aus einzelnen *Transformationsregeln* der Form

$$X \mapsto Y,$$

wobei X, Y Multimengen über eine Menge M sind, welche wiederum die Menge aller Nachrichtentypen darstellt. Die Semantik einer Transformationsregel ist die Folgende: Durch die Transformation wird eine Nachricht jedes Typs in X verbraucht und eine Nachricht jedes Typs in Y produziert. Alle zuvor beschriebenen möglichen Fertigkeiten eines Adapters lassen sich jeweils als Transformationsregel beschreiben, wie wir in Tabelle 2.1 sehen.

mögliche Fertigkeit	Transformationsregel
Erzeuge a	$\mapsto a$
Kopiere a	$a \mapsto a, a$
Lösche a	$a \mapsto$
Transformiere a, b, c in d, e	$a, b, c \mapsto d, e$ oder $a, b, c, \mapsto a, b, c, d, e$
Spalte a in b, c, d auf	$a \mapsto b, c, d$ oder $a \mapsto a, b, c, d$
Fasse a, b, c zu d zusammen	$a, b, c \mapsto d$ oder $a, b, c \mapsto a, b, c, d$
Leite a zu Empfänger j anstatt zu Empfänger i	$a^i \mapsto a^j$

Tabelle 2.1: Fertigkeiten der Nachrichtenverarbeitung als Transformationsregeln.

Für die folgende Synthese eines Adapters nehmen wir an, das eine Spezifikation elementarer Aktivitäten gegeben ist. Diese Annahme ist gerechtfertigt, da es genug Technologien wie *Semantic Web*, *Ontologien* oder *Semantic Mapping* gibt, welche die Erzeugung einer SEA entweder automatisch übernehmen oder zumindest unterstützen können.

Die in [GMW08] präsentierte Lösung des Adaptersyntheseproblem für die offenen Netze S_1, \dots, S_n , einer Verhaltenseigenschaft wie ST und unter Nutzung einer gegebenen SEA besteht aus drei Schritten:

1. Aus der SEA und dem Interface von $S_1 \oplus \dots \oplus S_n$ wird ein offenes Netz E (*Engine* genannt) konstruiert.
2. Für das Netz $(S_1 \oplus \dots \oplus S_n) \oplus E$ wird ein ST,f-Controller C synthetisiert.
3. Die Komposition $E \oplus C$ liefert den gesuchten Adapter.

Die in Schritt 1 konstruierte Engine erlaubt per Konstruktion (wie wir später anhand von Definition 17 sehen werden) genau die in der SEA spezifizierten Fertigkeiten der Nachrichtenverarbeitung. Um einen Adapter zu erhalten, müssen also nur noch die erlaubten Nachrichtenverarbeitungen in den richtigen Reihenfolgen durchgeführt werden. Dies übernimmt der in Schritt 2 synthetisierte ST,f-Controller C , da das offene Netz $(S_1 \oplus \dots \oplus S_n) \oplus E \oplus C$ nach Definition 15 schwach terminierend ist. Da es sich bei C außerdem um ein beschränktes Netz handelt, ist nach Korollar 2 auch das Netz $E \oplus C$ beschränkt. Somit stellt das in Schritt 3 komponierten Netz $E \oplus C$ wirklich einen Adapter für S_1, \dots, S_n dar, was wir im folgenden Korollar festhalten werden.

Korollar 3

Seien S_1, \dots, S_n offene Netze mit der Platzmenge $P_S = \bigcup_{1 \leq i \leq n} P_{S_i}$, E die aus S_1, \dots, S_n und einer gegebenen SEA konstruierte Engine, $f : P_S \cup P_E \rightarrow \mathbb{N}$, $g := f|_{P_S}$ und C ein ST,f-Controller von $(S_1 \oplus \dots \oplus S_n) \oplus E$.

Dann ist das offene Netz $E \oplus C$ ein ST,g-Controller von $S_1 \oplus \dots \oplus S_n$ und somit ein Adapter für S_1, \dots, S_n . ┘

Eine schematische Repräsentation der Lösung finden wir in Abbildung 2.11 vor.

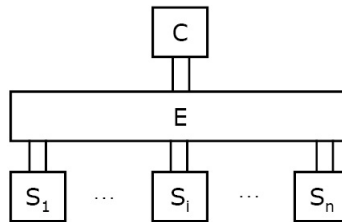


Abbildung 2.11: Die Struktur der Adaptersynthese.

Besonders wichtig bei dieser Vorgehensweise ist die Definition der Engine, da wir sicherstellen müssen, dass nur genau die in der SEA spezifizierten Nachrichtenverarbeitungen stattfinden können. Das Interface der Engine besteht aus Input- und Outputplätzen zur Kommunikation mit allen beteiligten Services sowie zusätzlich aus speziellen Input- und Outputplätzen zur Kommunikation mit einem Controller. Für jeden möglichen Nachrichtentyp $m \in M$ wird ein *interner Platz* (m, c) (c für „conceptual“) erzeugt, welcher eine Kopie der Nachricht enthalten kann. Nur auf diese Kopien können die spezifizierten Transformationsregeln angewandt werden, wobei jede Regel r durch genau eine *Regeltransition* (r, c) modelliert wird. Jeder Nachrichteneingang (Nachrichtenausgang) einer Nachricht x von (zu) einem der gegebenen Services wird über einen Interfaceplatz (x, n)

$((x, e))$ vom Controller erkannt (ermöglicht). Zusätzlich ist auch jede Regeltransition r über spezielle Interfaceplätze (r, e) (e für „execute“) und (r, n) (n für „notify“) vom Controller steuerbar. Formal wird die Engine E folgendermaßen konstruiert:

Definition 17 (Engine)

Seien $n \in \mathbb{N} \setminus \{0\}$, $S_x = (P_x, P_{i_x}, P_{o_x}, T_x, F_x, m_{0_x}, \Omega_x)$ offene Netze für $1 \leq x \leq n$ und seien $I := \bigcup_{1 \leq x \leq n} P_{i_x}$ und $O := \bigcup_{1 \leq x \leq n} P_{o_x}$. Sei R eine Menge, so dass eine SEA aus Transformationsregeln $X_r \mapsto Y_r \forall r \in R$ besteht. Sei M die Menge aller Nachrichtentypen aus I , O und der SEA. Weiterhin gelte $M \cap R = \emptyset$.

Dann ist die Engine $E = (P, P_i, P_o, T, F, m_o, \Omega)$ folgendermaßen definiert:

- $P = (M \times \{c\}) \cup P_i \cup P_o$
- $P_i = O \cup (R \times \{e\}) \cup (I \times \{e\})$
- $P_o = I \cup (R \times \{n\}) \cup (O \times \{n\})$
- $T = (O \times \{r\}) \cup (R \times \{c\}) \cup (I \times \{s\})$
- $F = F_r \cup F_c \cup F_s$
- $F_r = \bigcup_{o \in O} \{[o, (o, r)], [(o, r), (o, n)], [(o, r), (o, c)]\}$
- $F_c = \bigcup_{r \in R} (\{[(m, c), (r, c)] \mid m \in X_r\} \cup \{[(r, e), (r, c)]\} \cup \{[(r, c), (r, n)]\} \cup \{[(r, c), (m, c)] \mid m \in Y_r\})$
- $F_s = \bigcup_{i \in I} \{[(i, c), (i, s)], [(i, e), (i, s)], [(i, s), i]\}$
- $m_0(p) = 0 \forall p \in P$
- $\Omega = \{m_0\}$

Ein Beispiel für eine SEA und eine daraus konstruierte Engine finden wir in der Tabelle 2.2 und der Abbildung 2.12.

$BOrder$	\mapsto	$SOrder, SPay, debt$
$BPay, debt$	\mapsto	
$SGood$	\mapsto	$BGood$

Tabelle 2.2: Eine mögliche SEA eines Adapters für *buyer* und *seller*.

Beispiel 7 Eine naheliegende Annahme für einen Adapter für *buyer* und *seller* ist, dass wir diesem die Fertigkeiten zur Nachrichtenverarbeitung ähnlich einer Bank geben, welche aufgeschrieben als SEA in Tabelle 2.2 zu sehen sind. Hierbei gehen wir davon aus, dass der Adapter Nachrichten mit Geld als Inhalt temporär erzeugen kann. Im Falle aller anderen Nachrichten, welche reelle Güter wie z.B. eine Bestellung oder eine

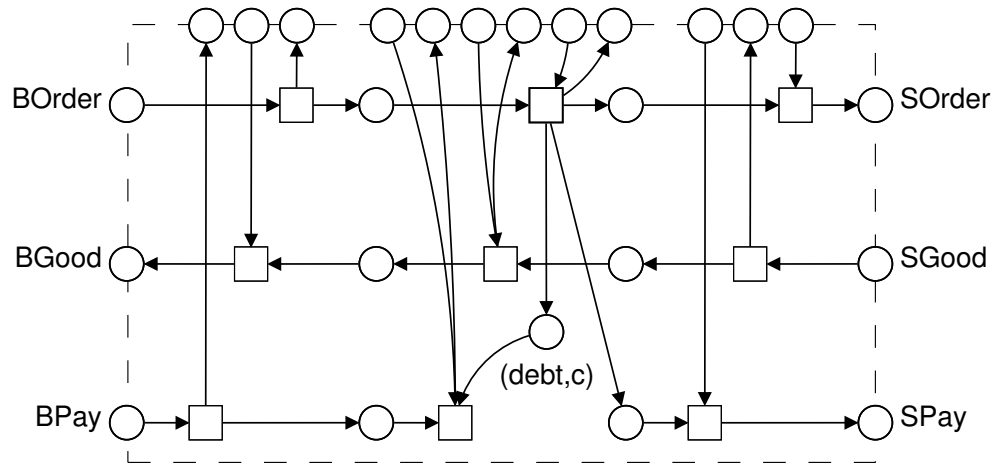


Abbildung 2.12: Die aus der SEA aus Tabelle 2.2 konstruierte Engine *engine*.

bestellte Ware modellieren, hat der Adapter maximal die Fertigkeit diese weiterzuleiten, aber keinesfalls sie zu erzeugen. Um das Verleihen von Geld zu modellieren, wurde ein Nachrichtentyp *debt* (in der Engine repräsentiert durch den Platz $(debt, c)$) eingeführt. Hiervon wird eine Nachricht beim Erzeugen einer Bezahlung für den Verkäufer *SPay* erzeugt und beim Vernichten einer Bezahlung vom Käufer *BPay* ebenfalls vernichtet. Aus der SEA in Tabelle 2.2 entsteht nach der Definition 17 eine Engine wie in Abbildung 2.12. Deutlich zu erkennen ist das Interface zu *buyer* (bestehend aus den Plätzen *BOrder*, *BGood* und *BPay*), das Interface zu *seller* (bestehend aus den Plätzen *SOrder*, *SPay* und *SGood*) sowie das Interface zu einem ST,f-Controller (bestehend aus den restlichen Interfaceplätzen von *engine*). Entsprechend der oben beschriebenen Vorgehensweise wird nun versucht, einen ST,f-Controller *controller* von dem Netz $(buyer \oplus seller) \oplus engine$ zu synthetisieren. Gelingt dies, d.h. ist das Netz $(buyer \oplus seller) \oplus engine$ ST,f-bediensbar, so stellt die Komposition $engine \oplus controller$ nach Korollar 3 einen Adapter für *buyer* und *seller* dar. ┘

Das Besondere an dieser Lösung ist, dass das Adaptersyntheseproblem für offene Netze auf das für offene Netze bereits gelöste Controllersyntheseproblem zurückgeführt wurde. Die Laufzeit der Synthese eines ST,f-Controllers *C* für das Netz $(S_1 \oplus \dots \oplus S_n) \oplus E$ hängt dabei von der Größe des Interface von $(S_1 \oplus \dots \oplus S_n) \oplus E$ ab. Folglich können wir beide für diese Arbeit gesteckten Ziele durch die strukturelle Reduktion der Engine erreichen: Einerseits führt eine Engine mit kleinerem Interface zu *C* zu einem kleineren Interface von $(S_1 \oplus \dots \oplus S_n) \oplus E$ zu *C* und damit zu einer Verbesserung der Laufzeit der Adaptersynthese. Andererseits führt eine kleinere Engine, d.h. eine Engine mit weniger Plätzen/Transitionen, zu einem kleineren Adapter $E \oplus C$.

3 Strukturelle Reduktion

In diesem Kapitel beschäftigen wir uns mit der Verbesserung der Adaptersynthese durch strukturelle Reduktion der Engine. Da die Engine ein offenes Netz ist, werden wir auch die Reduktionsregeln allgemein für offene Netze formulieren. Zuerst gehen wir näher auf das Entfernen von isolierten Plätzen ein, anschließend werden wir Reduktionen mittels des Entferns von speziellen Transitionen erläutern. Schlussendlich werden wir zeigen, wie jede vorgestellte Reduktionsregel im Rahmen der Adaptersynthese auf die Engine angewendet werden kann.

3.1 Isolierte Plätze

Isolierte Plätze sind Plätze eines offenen Netzes mit leerem Vor- und leerem Nachbereich. Sie sind ideale Kandidaten zur Reduktion, da ihr Vorhandensein weder das Schalten einer Transition noch das Schalten einer Schaltsequenz im Netz beeinflussen kann. Da die Anzahl der Marken auf einem isolierten Platz immer konstant bleibt, beeinflusst er allerdings sehr wohl die Erreichbarkeit bestimmter Markierung. Aus diesen Fakten formulieren wir folgende erste Reduktionsregel.

Reduktionsregel 1 (Isolierter Platz)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz und $p \in P$ mit $\bullet p = p\bullet = \emptyset$ und $m_0(p) = m(p)$ für alle Endmarkierungen $m \in \Omega$. Das nach Reduktionsregel 1 reduzierte offene Netz $N' = (P', P'_i, P'_o, T', F', m'_0, \Omega')$ von N ist definiert als:

- $P' = P \setminus \{p\}$
- $P'_i = P_i \setminus \{p\}$
- $P'_o = P_o \setminus \{p\}$
- $T' = T$
- $F' = F$
- $m'_0 = m_{0|P \setminus \{p\}}$
- $\Omega' = \{m|_{P \setminus \{p\}} \mid m \in \Omega\}$

Bei isolierten Plätzen eines offenen Netzes lohnt es sich, zwischen isolierten internen Plätzen und isolierten Interfaceplätzen zu unterscheiden, da Interfaceplätze im Gegensatz zu internen Plätzen die Komponierbarkeit mit einem anderen offenen Netz beeinflussen. Ein Beispiel für die Anwendung von Reduktionsregel 1 auf einen isolierten internen Platz

finden wir in Abbildung 3.1. Das Netz N'_1 ist das nach Reduktionsregel 1 reduzierte offene Netz von N_1 und das Netz CN_1 stellt sowohl für N_1 als auch für N'_1 einen ST,f-Controller dar.

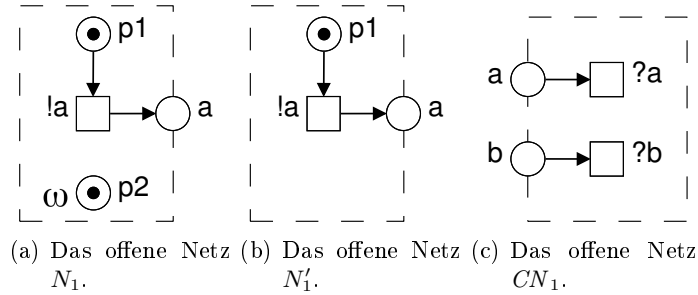


Abbildung 3.1: Die offenen Netze N_1 und N'_1 sowie ein Controller CN_1 .

Uns interessiert nur die ST,f-Bedienbarkeit offener Netze, deshalb beweisen wir für den einfacheren Fall, d.h. für isolierte interne Plätze, zuerst folgendes Lemma.

Lemma 4

Seien N ein offenes Netz, N' das nach Reduktionsregel 1 reduzierte offene Netz von N und $p \in P_N \setminus P_{N'}$ der reduzierte Platz mit $p \notin I_N$. Weiterhin sei $f : P_N \rightarrow \mathbb{N}$ beliebig mit $f(p) \geq m_{0_N}(p)$ und $g := f|_{P_{N'}}$.

Ein beschränktes offenes Netz C ist genau dann ein ST,f-Controller von N , wenn C ein ST,g-Controller von N' ist. ┘

Beweis. Dank $p \notin I_N$ haben N und N' ein identisches Interface und es gilt

$$N \text{ und } C \text{ sind komponierbar} \Leftrightarrow N' \text{ und } C \text{ sind komponierbar} \quad (1)$$

Aus $\bullet p = p \bullet = \emptyset$ folgt, dass in N und N' bzw. in $N \oplus C$ und $N' \oplus C$ die gleichen Schaltsequenzen möglich sind. Daraus folgt wiederum

$$m \in R_N \Leftrightarrow m|_{P_{N'}} \in R_{N'} \text{ bzw. } m \in R_{N \oplus C} \Leftrightarrow m|_{P_{N' \oplus C}} \in R_{N' \oplus C} \quad (2)$$

Da $f(p) \geq m_{0_N}(p)$ und (2) gilt

$$m(p) \leq f(p) \forall p \in P_N \forall m \in R_{N \oplus C} \Leftrightarrow m(p) \leq g(p) \forall p \in P_{N'} \forall m \in R_{N' \oplus C} \quad (3)$$

Aus $m_{0_N}(p) = m(p)$ für alle Endmarkierungen $m \in \Omega_N$ folgt, dass $m \in \Omega_N \Leftrightarrow m|_{P_{N'}} \in \Omega_{N'}$ bzw. $m \in \Omega_{N \oplus C} \Leftrightarrow m|_{P_{N' \oplus C}} \in \Omega_{N' \oplus C}$. Daraus folgt zusammen mit (2), dass

$$N \oplus C \text{ ist schwach terminierend} \Leftrightarrow N' \oplus C \text{ ist schwach terminierend} \quad (4)$$

Schließlich folgt aus (1), (3) und (4) nach Definition 15 die Behauptung. □

Leider lässt sich Lemma 4 nicht auf isolierte Interfaceplätze (und damit auf isolierte Plätze insgesamt) verallgemeinern, da ein ST,g -Controller von N' nicht zwangsläufig komponierbar mit N sein muss. Ein Beispiel hierfür finden wir in Abbildung 3.2. Das Netz N'_2 ist das nach Reduktionsregel 1 reduzierte offene Netz von N_2 , wobei der Platz b ein Inputplatz ist. Das offene Netz CN_1 aus Abbildung 3.1(c) ist ein ST,f -Controller von N'_2 , allerdings kein ST,f -Controller von N_2 , da N_2 und CN_1 beide einen Inputplatz b besitzen und deshalb nicht komponierbar sind.

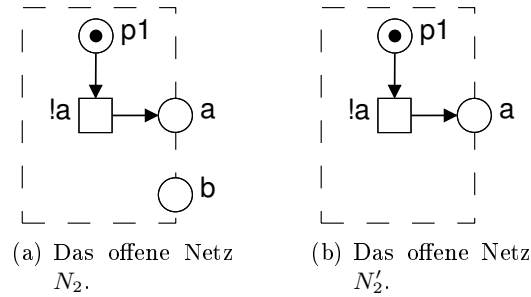


Abbildung 3.2: Die offenen Netze N_2 und N'_2 .

Allerdings bleibt nach der Anwendung der Reduktionsregel 1 auf einen isolierten Interfaceplatz immer noch die für uns interessante ST,f -Bedienbarkeit erhalten, wie wir mit dem folgenden Satz zeigen werden.

Satz 5 (Korrektheit von Reduktionsregel 1)

Seien N ein offenes Netz, N' das nach Reduktionsregel 1 reduzierte offene Netz von N und $p \in P_N \setminus P_{N'}$ der reduzierte Platz. Weiterhin sei $f : P_N \rightarrow \mathbb{N}$ beliebig mit $f(p) \geq m_{0_N}(p)$ und $g := f|_{P_{N'}}$.

N ist genau dann ST,f -bedienbar, wenn N' ST,g -bedienbar ist. ┘

Beweis. Im Folgenden werden wir davon ausgehen, dass $p \in I_N$ gilt, da andernfalls die Behauptung des Satzes direkt aus Lemma 4 folgt.

„ \implies “ : Sei C ein ST,f -Controller von N . Wir können $p \in I_C$ annehmen, da C sonst zu einem ST,f -Controller C' von N mit $p \in I_{C'}$ durch Einfügen von p in die entsprechenden Platzmengen und Anpassen der Anfangsmarkierung/Endmarkierungen erweitert werden kann. Sei also $p \in I_C$, womit p ein interner Platz in $N \oplus C$ und ein Interfaceplatz in $N' \oplus C$ ist. Aus $p \in I_N$ und $p \in I_C$ folgt, dass die Netze $N \oplus C$ und $N' \oplus C$ eine identische Anfangsmarkierung und identische Endmarkierungen besitzen. Somit unterscheiden sich $N \oplus C$ und $N' \oplus C$ nur durch den Platz p , und da $N \oplus C$ schwach terminierend ist, ist auch $N' \oplus C$ schwach terminierend. Folglich ist C ein ST,g -Controller von N' .

„ \impliedby “ : Sei C ein ST,g -Controller von N' . Wir können $p \in I_C$ annehmen, da C sonst zu einem ST,g -Controller C' von N mit $p \in I_{C'}$ (analog zur Konstruktion in der Hinrichtung) erweitert werden kann. Das Interface von N' und N unterscheidet sich nur durch

den Platz p . Wenn C nicht komponierbar mit N ist, so können wir mit dem Ersetzen von p durch einen Platz $\hat{p} \notin P_N$ einen ST,g-Controller von N' konstruieren, welcher komponierbar mit N ist. Es gelte also $p \in I_C$ und C ist komponierbar mit N . Dann ist p ein Interfaceplatz in $N' \oplus C$ und ein interner Platz in $N \oplus C$. Aus $p \in I_N$ und $p \in I_C$ folgt, dass die Netze $N' \oplus C$ und $N \oplus C$ eine identische Anfangsmarkierung und identische Endmarkierungen besitzen, sich ansonsten aber nur im Platz p unterscheiden. Somit ist C analog zur Hinrichtung ein ST,f-Controller von N . \square

Isolierte Plätze sollten in gegebenen offenen Netzen nicht vorhanden sein, da sie gewöhnlich nur durch Fehler in der Modellierung eines Services als offenes Netz entstehen. Nichtsdestoweniger können isolierte Plätze aber sehr wohl durch die Anwendung von Reduktionsregeln der folgenden Sektionen erzeugt werden, weshalb Reduktionsregel 1 trotz alledem ihre Berechtigung hat.

3.2 Siphons

Siphons und Fallen sind eine lange bekannte und besonders einfache Technik zum Beweis von Ungleichungen in Petrinetzen. Darüber hinaus eignen sie sich allerdings auch zur Reduktion von offenen Netzen, wie wir in dieser (für Siphons) und der folgenden Sektion (für Fallen) zeigen wollen.

Technisch gesehen ist ein *Siphon* eine Teilmenge der internen Plätze eines offenen Netzes, wobei jede Transition, welche mindestens einen Platz des Siphons im Nachbereich hat, auch mindestens einen Platz dieses Siphons im Vorbereich aufweist. Wir beschränken uns bei dieser Definition auf die internen Plätze, da sonst die gesamte Platzmenge eines offenen Netzes immer ein Siphon wäre.

Definition 18 (Siphon)

Sei N ein offenes Netz. Eine Menge $S \subseteq P_N \setminus I_N$ heißt genau dann Siphon in N , wenn $\bullet S \subseteq S^\bullet$ gilt. ┘

Intuitiv formuliert bedeutet dies: „Wenn eine Transition etwas in S hineinlegt, so nimmt sie vorher etwas aus S heraus.“ Hieraus können wir schlussfolgern, dass keine Transition mehr etwas in S hineinlegen kann, wenn sie nicht vorher etwas aus S herausnehmen konnte. Aus dieser Beobachtung resultiert unsere nächste Reduktionsregel.

Reduktionsregel 2 (Siphon)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz und S ein Siphon in N mit $m_0(p) = 0$ für alle Plätze $p \in S$. Dann ist das nach Reduktionsregel 2 reduzierte offene Netz $N' = (P', P'_i, P'_o, T', F', m'_0, \Omega')$ von N definiert als:

- $P' = P$
- $P'_i = P_i$
- $P'_o = P_o$
- $T' = T \setminus S^\bullet$
- $F' = F \setminus ((S^\bullet \times P) \cup (P \times S^\bullet))$
- $m'_0 = m_0$
- $\Omega' = \Omega$ ┘

Wir beschränken uns in Reduktionsregel 2 auf das Entfernen von nicht benötigten Transitionen aus einem offenen Netz. Hierbei entstehen in der Regel isolierte Plätze, welche wir allerdings mit der schon bekannten Reduktionsregel 1 entfernen können. In Abbildung 3.3 sehen wir das nach Reduktionsregel 2 reduzierte offene Netz N'_3 von N_3 , wobei die Menge $\{p_2\}$ einen Siphon nach der Voraussetzung der Reduktionsregel darstellt. Das offene Netz CN_1 aus Abbildung 3.1(c) ist sowohl für N_3 als auch für N'_3 ein ST,f-Controller.

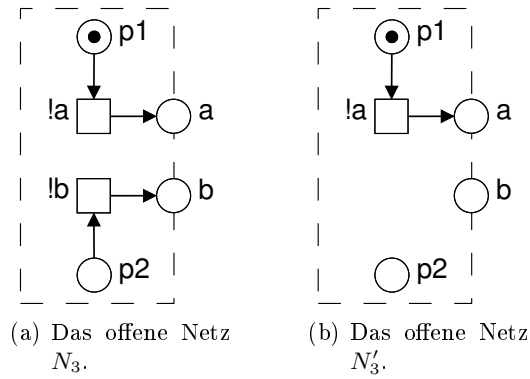


Abbildung 3.3: Die offenen Netze N_3 und N'_3 .

Auch Reduktionsregel 2 erhält die Verhaltenseigenschaft der ST, f -Bedienbarkeit eines offenen Netzes.

Satz 6 (Korrektheit von Reduktionsregel 2)

Seien N ein offenes Netz, N' das nach Reduktionsregel 2 reduzierte offene Netz von N und S ein Siphon in N mit $S^\bullet = T_N \setminus T_{N'}$. Weiterhin sei $f : P_N \rightarrow \mathbb{N}$ beliebig.

Ein beschränktes offenes Netz C ist genau dann ein ST, f -Controller von N , wenn C ein ST, f -Controller von N' ist. ┘

Beweis. Die offenen Netze N und N' haben ein identisches Interface und somit gilt

$$N \text{ und } C \text{ sind komponierbar} \Leftrightarrow N' \text{ und } C \text{ sind komponierbar} \quad (1)$$

Nach Voraussetzung der Reduktionsregel 2 gilt $m_{0_N}(p) = 0$ für jeden Platz $p \in S$. Daraus folgt, dass keine Transition $t \in S^\bullet$ aktiviert in m_{0_N} ist. Um eine Transition aus S^\bullet zu aktivieren, müsste vorher mindestens eine Transition aus $\bullet S$ schalten. Da S allerdings ein Siphon ist und damit $\bullet S \subseteq S^\bullet$ gilt, ist auch jede Transition aus $\bullet S$ nicht aktiviert in m_{0_N} . Folglich sind alle Transitionen aus S^\bullet in allen Markierungen $m \in R_N$ nicht aktiviert. Die Netze N und N' unterscheiden sich nur durch die Transitionen S^\bullet und die entsprechende Flussrelation, nicht aber in der Anfangs- und den Endmarkierungen. Somit haben sie identische Schaltsequenzen und es gilt $m \in R_N \Leftrightarrow m \in R_{N'}$ bzw. $m \in R_{N \oplus C} \Leftrightarrow m \in R_{N' \oplus C}$. Daraus folgt direkt

$$m(p) \leq f(p) \forall p \in P_N \forall m \in R_{N \oplus C} \Leftrightarrow m(p) \leq f(p) \forall p \in P_{N'} \forall m \in R_{N' \oplus C} \quad (2)$$

sowie

$$N \oplus C \text{ ist schwach terminierend} \Leftrightarrow N' \oplus C \text{ ist schwach terminierend} \quad (3)$$

Schlussendlich folgt aus (1), (2) und (3) nach Definition 15 die Behauptung. □

3.3 Fallen

Eine *Falle* ist das symmetrische Pendant zu einem Siphon. Technisch gesehen ist sie eine Teilmenge der internen Plätze eines offenen Netzes, wobei jede Transition, welche mindestens einen Platz der Falle im Vorbereich hat, auch mindestens einen Platz dieser Falle im Nachbereich aufweist. Wieder beschränken wir uns bei dieser Definition auf die internen Plätze, da sonst die gesamte Platzmenge eines offenen Netzes stets eine Falle wäre.

Definition 19 (Falle)

Sei N ein offenes Netz. Eine Menge $R \subseteq P_N \setminus I_N$ heißt genau dann *Falle in N* , wenn $R^\bullet \subseteq \bullet R$ gilt. ┘

Intuitiv formuliert bedeutet dies: „Wenn eine Transition etwas aus R herausnimmt, so legt sie nachher wieder etwas in R hinein.“ Hieraus können wir schlussfolgern, dass wenn in einer Markierung m mindestens ein Platz einer Falle mindestens eine Marke besitzt, so sind von m aus nur noch Markierungen erreichbar, in denen ebenso mindestens ein Platz dieser Falle mindestens eine Marke hat. Dies hat Auswirkungen auf die Erreichbarkeit einer Endmarkierung, wenn in ihr kein Platz der Falle eine Marke besitzen soll.

Lemma 7

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz und R eine Falle in N mit $\hat{m}(p) = 0$ für alle Plätze $p \in R$ und alle Endmarkierungen $\hat{m} \in \Omega$.

Von einer Markierung m von N mit $m(p) \geq 1$ für einen Platz $p \in R$ ist keine Endmarkierung $m' \in \Omega$ in N erreichbar. ┘

Beweis. R ist eine Falle in N und somit gilt $R^\bullet \subseteq \bullet R$. In jeder von m in N erreichbaren Markierung m' existiert somit ein Platz $p \in R$ mit $m'(p) \geq 1$. Somit ist m' keine Endmarkierung, da $\hat{m}(p) = 0$ für alle Plätze $p \in R$ und alle Endmarkierungen $\hat{m} \in \Omega$. □

Bekannterweise ist aus jeder erreichbaren Markierung eines schwach terminierenden offenen Netzes eine Endmarkierung erreichbar. Ist ein offenes Netz also schwach terminierend und hat jeder Platz jeder Falle in allen Endmarkierungen keine Marke, so sind sämtliche Transitionen aus dem Vorbereich jeder Falle des Netzes in keiner erreichbaren Markierung aktiviert. Diese Transitionen stellen hervorragende Kandidaten zur Reduktion dar. Allerdings wollen wir im Idealfall die Menge aller ST,f-Controller eines Netzes komplett erhalten, weswegen jede zu reduzierende Transition noch einen speziellen, immer leeren Vorplatz benötigt. Dies führt uns schließlich zu folgender Reduktionsregel.

Reduktionsregel 3 (Falle)

Sei $N = (P, P_i, P_o, T, F, m_0, \Omega)$ ein offenes Netz und R eine Falle in N mit $m(p) = 0$ für alle Plätze $p \in R$ und alle Endmarkierungen $m \in \Omega$. Des Weiteren existiert für jede Transition $t \in \bullet R$ ein Platz $p \in \bullet t$ mit $p^\bullet \subseteq \bullet R$ und $m_0(p) = m(p) = 0$ für alle

Endmarkierungen $m \in \Omega$. Dann ist das nach Reduktionsregel 3 reduzierte offene Netz $N' = (P', P'_i, P'_o, T', F', m'_0, \Omega')$ von N definiert als:

- $P' = P$
- $P'_i = P_i$
- $P'_o = P_o$
- $T' = T \setminus \bullet R$
- $F' = F \setminus ((\bullet R \times P) \cup (P \times \bullet R))$
- $m'_0 = m_0$
- $\Omega' = \Omega$

In Abbildung 3.4 finden wir ein Beispiel, warum wir für jede Transition aus dem Vorbereitung der Falle noch einen speziellen Platz fordern. Das nach Reduktionsregel 3 reduzierte offene Netz N'_4 von N_4 ist ST,f-bedenbar, wohingegen das offene Netz N_4 nicht ST,f-bedenbar ist. Dies liegt an der Transition t im Vorbereitung der Falle $\{p2\}$, welche unabhängig von einem mit N_4 komponierten Netz beliebig schalten und damit Marken in die Falle legen kann. Nach Lemma 7 wissen wir, dass dann kein Endzustand mehr erreichbar ist, womit das Netz nicht schwach terminierend sein kann.

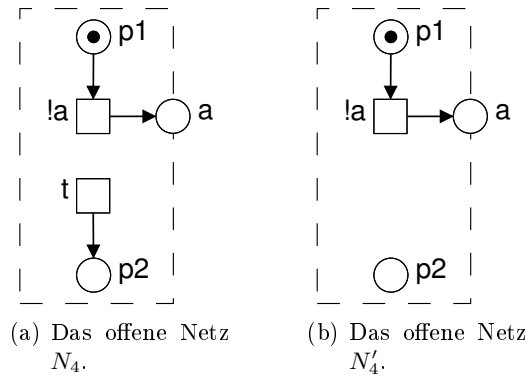


Abbildung 3.4: Die offenen Netze N_4 und N'_4 .

Auch in Reduktionsregel 3 beschränken wir uns auf das Entfernen von nicht benötigten Transitionen aus einem offenen Netz. Wie schon bei Reduktionsregel 2 entstehen hierbei in der Regel isolierte Plätze, welche wir wiederum mit Reduktionsregel 1 entfernen können. Ein Beispiel für die Anwendung von Reduktionsregel 3 finden wir in Abbildung 3.5, wobei die Menge $\{p2\}$ eine Falle nach den geforderten Voraussetzungen darstellt. Der spezielle Vorplatz aus $\bullet?c$ ist der Platz c , da Interfaceplätze die geforderte Bedingung trivialerweise erfüllen. Das Netz N'_5 ist das nach Reduktionsregel 3 reduzierte offene Netz von N_5 und das Netz CN_5 stellt sowohl für N_5 als auch für N'_5 einen ST,f-Controller dar.

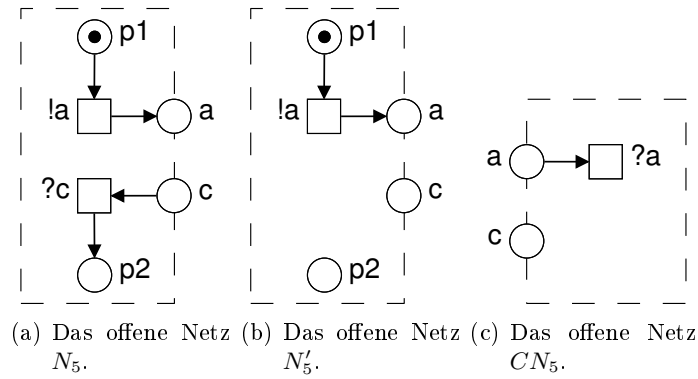


Abbildung 3.5: Die offenen Netze N_5 und N'_5 sowie ein Controller CN_5 .

Schlussendlich bleibt noch zu zeigen, dass auch Reduktionsregel 3 die ST,f-Bedienbarkeit eines offenen Netzes erhält.

Satz 8 (Korrektheit von Reduktionsregel 3)

Seien N ein offenes Netz, N' das nach Reduktionsregel 3 reduzierte offene Netz von N und R eine Falle in N mit $\bullet R = T_N \setminus T_{N'}$. Weiterhin sei $f : P_N \rightarrow \mathbb{N}$ beliebig.

Ein beschränktes offenes Netz C ist genau dann ein ST,f-Controller von N , wenn C ein ST,f-Controller von N' ist. ┘

Beweis.

Im Folgenden gehen wir daher davon aus, dass $m_{0_N}(p) = 0$ für alle Plätze $p \in R$ gilt, da andernfalls nach Lemma 7 sowohl N als auch N' nicht ST,f-bedienbar sind.

„ \implies “ : Sei C ein ST,f-Controller von N . Die offenen Netze N und N' haben ein identisches Interface und somit sind auch N' und C komponierbar. Aus Lemma 7 folgt, dass $m(p) = 0$ für alle $p \in R$ und alle $m \in R_{N \oplus C}$, da ansonsten $N \oplus C$ nicht schwach terminierend wäre. Damit sind alle Transitionen $t \in \bullet R$ in keiner Markierung $m \in R_{N \oplus C}$ aktiviert. Nach der Definition von N' gilt nun $m \in R_{N \oplus C} \Leftrightarrow m \in R_{N' \oplus C}$ sowie $m(p) \leq f(p) \forall p \in P_N \forall m \in R_{N \oplus C} \Leftrightarrow m(p) \leq f(p) \forall p \in P_{N'} \forall m \in R_{N' \oplus C}$. Folglich ist C auch ein ST,f-Controller von N' .

„ \impliedby “ : Sei C ein ST,f-Controller von N' . Die offenen Netze N und N' haben ein identisches Interface und somit sind auch N und C komponierbar. Nach der Definition von N' ist jede Schaltsequenz in $N' \oplus C$ eine Schaltsequenz in $N \oplus C$. Wenn wir zeigen können, dass jede Schaltsequenz in $N \oplus C$ eine Schaltsequenz in $N' \oplus C$ ist, dann folgt daraus direkt $m \in R_{N \oplus C} \Leftrightarrow m \in R_{N' \oplus C}$ sowie $m(p) \leq f(p) \forall p \in P_N \forall m \in R_{N \oplus C} \Leftrightarrow m(p) \leq f(p) \forall p \in P_{N'} \forall m \in R_{N' \oplus C}$. Folglich wäre C auch ein ST,f-Controller von N .

Angenommen es existiert eine Schaltsequenz $\sigma = t_1, \dots, t_n$ von $m_{0_{N \oplus C}}$ nach $m \in R_{N \oplus C}$ in $N \oplus C$, welche keine Schaltsequenz in $N' \oplus C$ ist. Dann kommt in σ mindestens eine Transition aus $\bullet R$ vor. Sei $t_i \in \bullet R$ das erste Vorkommen solch einer Transition. Dann

existiert eine Schaltsequenz $\sigma' = t_1, \dots, t_{i-1}$ von $m_{0_{N' \oplus C}}$ nach $m' \in R_{N' \oplus C}$ in $N' \oplus C$ mit $m'(p) \geq 1$ für alle $p \in \bullet t_i$. Da aber für jede Transition $t \in \bullet R$ ein Platz $p \in \bullet t$ mit $p^\bullet \subseteq \bullet R$ und $m_{0_N}(p) = m_N(p) = 0$ für alle Endmarkierungen $m_N \in \Omega_N$ existiert, gibt es in m' einen Platz $p \in \bullet t_i$ mit $p^\bullet = \emptyset$, $m'(p) \geq 1$ sowie $m(p) = 0$ für alle Endmarkierungen $m \in \Omega_{N' \oplus C}$. Damit wäre $N' \oplus C$ nicht schwach terminierend und C kein ST,f-Controller von N' , was einen Widerspruch zur Voraussetzung darstellt. \square

3.4 Anwendung auf Adaptersynthese

In den letzten Sektionen haben wir drei Reduktionsregeln für offene Netze formuliert und bewiesen, dass diese jeweils ST,f-Bedienbarkeit erhalten. Mit deren Hilfe verbessern wir nun die in Sektion 2.3 beschriebene Synthese eines Verhaltensadapters für gegebene offene Netze S_1, \dots, S_n , die Verhaltenseigenschaft ST und eine gegebene SEA, indem wir die Lösung aus [GMW08] um einen zusätzlichen Schritt erweitern:

1. Aus der SEA und dem Interface von $S_1 \oplus \dots \oplus S_n$ wird die Engine E konstruiert.
2. Das Netz $(S_1 \oplus \dots \oplus S_n) \oplus E$ wird mit Hilfe der Reduktionsregeln 1, 2 und 3 zu einem Netz $(S_1 \oplus \dots \oplus S_n) \oplus E'$ reduziert.
3. Für das Netz $(S_1 \oplus \dots \oplus S_n) \oplus E'$ wird ein ST,f-Controller C synthetisiert.
4. Die Komposition $E' \oplus C$ liefert den gesuchten Adapter.

Durch das Anwenden der Reduktionsregeln in Schritt 2 bleibt die ST,f-Bedienbarkeit des Netzes $(S_1 \oplus \dots \oplus S_n) \oplus E$ erhalten, folglich führen die Schritte 3 und 4 analog zur alten Vorgehensweise zu einem Adapter. Wir nehmen dabei an, dass die gegebenen Services, modelliert als offene Netze S_1, \dots, S_n , unveränderbar sind. Dies hat zur Folge, dass wir zwar die Reduktionsregeln auf das gesamte offene Netz $(S_1 \oplus \dots \oplus S_n) \oplus E$ anwenden, allerdings nur Plätze bzw. Transitionen aus der Engine E entfernen. Die Reduktionsregeln können wir solange anwenden, bis aus der Engine nichts mehr entfernt werden kann. Hierbei können wir die Anzahl der Regelanwendungen minimieren, indem wir bei der Anwendung von Reduktionsregel 2 einen maximalen Siphon und bei der Anwendung von Reduktionsregel 3 eine maximale Falle verwenden.

Durch das Entfernen von isolierten Interfaceplätzen von der Engine zu einem Controller gemäß Reduktionsregel 1 wird die Controllersynthese in Schritt 3 beschleunigt, da deren Laufzeit direkt von der Größe des Interfaces abhängt. Damit ist das erste Ziel dieser Arbeit erreicht. Zusätzlich sorgt jegliches Entfernen von Plätzen gemäß der Reduktionsregel 1 bzw. von Transitionen gemäß der Reduktionsregeln 2 und 3 für eine kleinere Engine E' und damit nach Schritt 4 auch für einen kleineren Adapter $E' \oplus C$. Dies entspricht der zweiten von uns erwünschten Verbesserung der Adaptersynthese.

$BOrder$	\mapsto	$SOrder, SPay, debt$
$BPay, debt$	\mapsto	
$SGood$	\mapsto	$BGood$
s	\mapsto	$BPay$
$SPay$	\mapsto	r

Tabelle 3.1: Eine andere SEA für einen Adapter für *buyer* und *seller*.

Beispiel 8 Um die Anwendung der erweiterten Adaptersynthese zu illustrieren werden wir wieder auf unsere aus den Abbildungen 2.10(a) und 2.10(c) bekannten Netze *buyer*

und *seller* zurückgreifen. Diesmal benutzen wir allerdings eine um in diesem Fall unbrauchbare Transformationsregeln erweiterte SEA, zu sehen in Tabelle 3.1. In Abbildung 3.6 finden wir die aus dem Interface von $buyer \oplus seller$ und der SEA konstruierte Engine $engineM$. Alle nicht beschrifteten Interfaceplätze stellen hierbei das Interface zu einem späteren Controller dar.

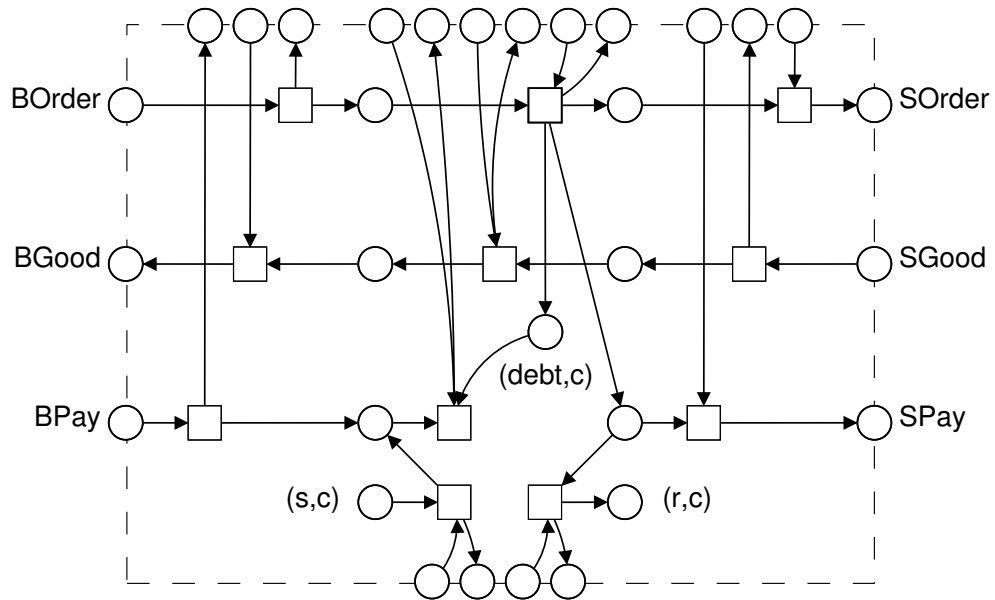


Abbildung 3.6: Die aus der SEA aus Tabelle 3.1 konstruierte Engine $engineM$.

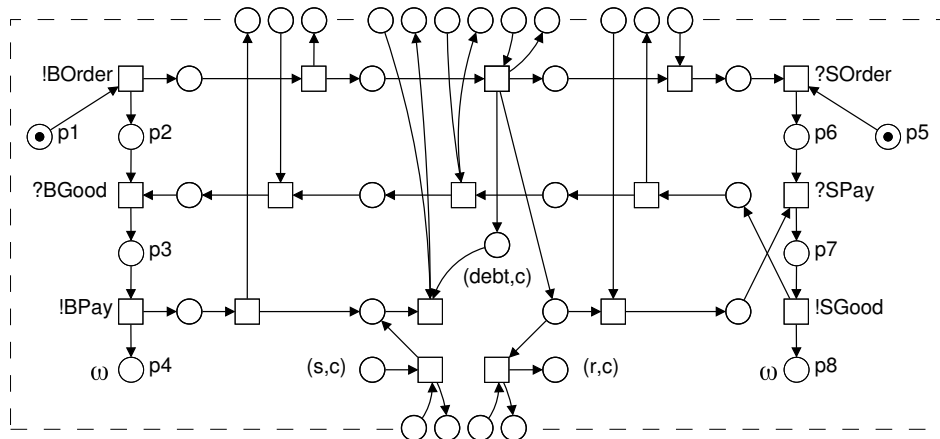


Abbildung 3.7: Das offene Netz $(buyer \oplus seller) \oplus engineM$.

In Abbildung 3.7 sehen wir das offene Netz $(buyer \oplus seller) \oplus engineM$, welches nun nach Schritt 2 reduziert werden kann. Dabei ist die Menge $\{(s, c)\}$ ein Siphon, welcher den Voraussetzungen der Reduktionsregel 2 genügt, und die Menge $\{(r, c)\}$ eine Falle, welche die Voraussetzungen der Reduktionsregel 3 erfüllt.

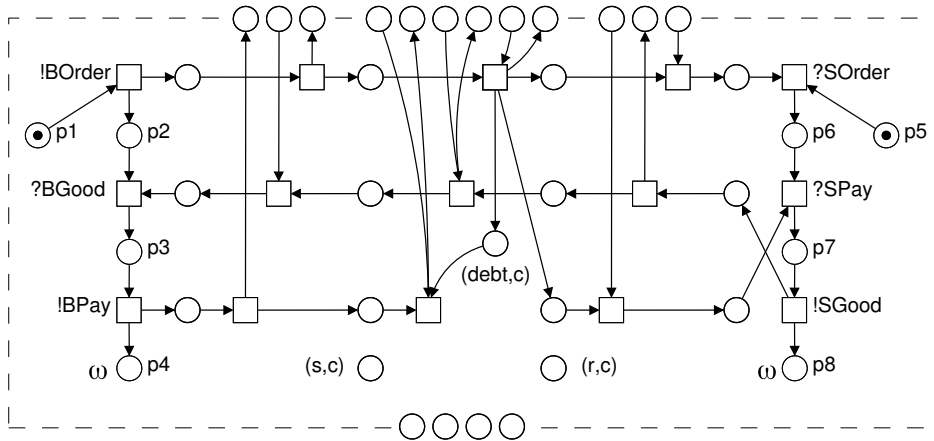


Abbildung 3.8: Das offene Netz $(buyer \oplus seller) \oplus engineM'$.

Das Netz $(buyer \oplus seller) \oplus engineM'$ in Abbildung 3.8 ist das Resultat der Anwendung beider Reduktionsregeln. Wir können weiterhin die Reduktionsregel 1 anwenden, da die Plätze (s, c) , (r, c) sowie vier Interfaceplätze den Voraussetzungen genügen. Es entsteht das Netz $(buyer \oplus seller) \oplus engineM''$ in Abbildung 3.9, welches durch keine weitere unserer Reduktionsregeln reduziert werden kann.

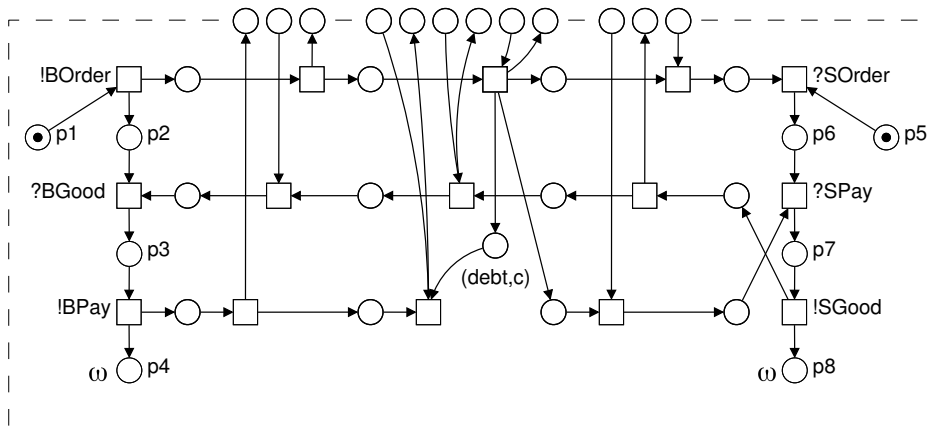


Abbildung 3.9: Das offene Netz $(buyer \oplus seller) \oplus engineM''$.

Für dieses offene Netz können wir wie in Schritt 3 beschriebenen einen ST,f-Controller synthetisieren, welcher laut Schritt 4 in Komposition mit dem offenen Netz $engineM''$ einen Adapter für *buyer* und *seller* ergibt. Einen fertigen Adapter *adapter* finden wir schlussendlich in Abbildung 3.10(b). ┘

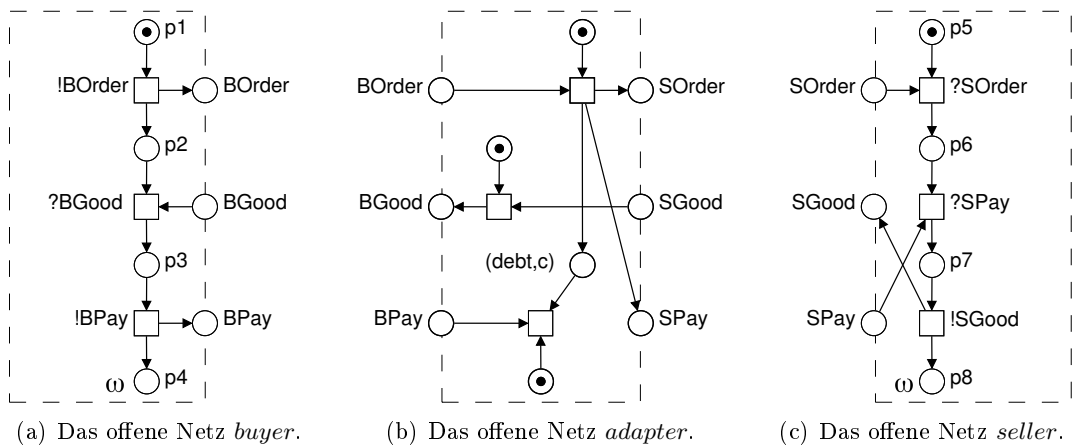


Abbildung 3.10: Die offenen Netze *buyer* und *seller* sowie ein Adapter *adapter*.

4 Fazit

In dieser Arbeit haben wir uns den Formalismus der offenen Netze angeschaut und uns auf dessen Basis mit der Synthese von Verhaltensadaptern beschäftigt. Wir haben den Begriff der schwachen Terminierung offener Netze und darauf aufbauend den der ST,f-Bedienbarkeit definiert. Des Weiteren haben wir drei neue Reduktionsregeln für offene Netze formuliert. Mit Hilfe der Reduktionsregel 1 können wir isolierte Plätze entfernen, wohingegen sich die Reduktionsregeln 2 und 3 auf die Nach- bzw. Vortransitionen bestimmter Siphons und Fallen beziehen. Für jede der Reduktionsregeln konnten wir zeigen, dass sie mindestens die ST,f-Bedienbarkeit des reduzierten offenen Netzes erhält. In einigen Fällen bleibt sogar die Menge aller möglichen ST,f-Controller gleich.

Darauf aufbauend haben wir den bestehenden Vorgang der Synthese von Verhaltensadaptern modifiziert. Durch die Nutzung der formulierten Reduktionsregeln ist es sowohl möglich, kleinere Verhaltensadapter zu synthetisieren, als auch den Vorgang an sich zu beschleunigen. Beides haben wir an einem durchgängigen Beispiel demonstriert. Da die Reduktionsregeln allgemein für offenen Netzen formuliert und bewiesen wurden, ist deren Anwendung nicht auf die Verhaltensadaptersynthese beschränkt.

Die Ergebnisse sollen auch noch praktisch umgesetzt werden. Hier bietet es sich an, sie in die bestehende Werkzeuglandschaft des Lehrstuhls zu integrieren, so dass die zukünftige Synthese schneller abläuft und zu kleineren Adaptern führt.

4.1 Verwandte Arbeiten

In der verfügbaren Literatur finden sich neben [GMW08] auch andere Ansätze zur Synthese von Adaptern, wie zum Beispiel in [DS06], [NBM⁺07], [BCG⁺05] oder [BBC05]. Der Ansatz in [DS06] unterscheidet sich vom hier vorgestellten Ansatz dadurch, dass der in [DS06] synthetisierte Adapter die gegebenen Services modifiziert. Solch eine Adaptersynthese funktioniert demnach nicht, wenn die gegebenen Services nicht unserer Kontrolle unterliegen. Des Weiteren läuft die Adaptersynthese aus [NBM⁺07] nur semi-automatisch ab, im Gegensatz zur automatischen Synthese in [GMW08]. Der Ansatz aus [BBC05] schliesst Rekursion aus dem Modell für die gegebenen Services aus, wohingegen der hier vorgestellte Ansatz mit offenen Netzen zwar beliebig beschränkte Services voraussetzt, aber keine Einschränkung bezüglich vorhandener Schleifen macht. In [BCG⁺05] und [BBC05] werden Spezifikationssprachen ähnlich der SEA verwendet, allerdings sind die Transformationsregeln dort symmetrisch, im Gegensatz zu den asymmetrischen Transformationsregeln aus [GMW08]. Asymmetrie jedoch ist häufig wichtig, zum Beispiel wenn die Transformation eine Verschlüsselung mit einem öffentlichem Schlüssel beinhaltet.

Allen Ansätzen gemein ist, dass sie sich nur mit der grundsätzlichen Umsetzung der Adaptersynthese befassen. Wir hingegen beschäftigen uns in dieser Arbeit mit der Verbesserung einer bestehenden und funktionierenden Vorgehensweise zur Adaptersynthese, was in keinem der genannten Papiere vorkommt.

Abbildungsverzeichnis

2.1	Die grafische Notation des Netzes N .	5
2.2	Das Netz N mit zwei unterschiedlichen Markierungen.	5
2.3	Das Netz N mit zwei weiteren Markierungen.	7
2.4	Die offenen Netze $buyer$ und $seller$.	9
2.5	Die innere Struktur der offenen Netze $buyer$ und $seller$.	10
2.6	Das offene Netz $buyer \oplus seller$.	11
2.7	Die offenen Netze $buyer$ und $controllerB$.	13
2.8	Die offenen Netze $seller$ und $controllerS$.	14
2.9	Das offene Netz $buyer \oplus seller$ im Deadlock.	14
2.10	Die offenen Netze $buyer$ und $seller$ sowie ein Adapter $trivial$.	16
2.11	Die Struktur der Adaptersynthese.	18
2.12	Die aus der SEA aus Tabelle 2.2 konstruierte Engine $engine$.	20
3.1	Die offenen Netze N_1 und N'_1 sowie ein Controller CN_1 .	22
3.2	Die offenen Netze N_2 und N'_2 .	23
3.3	Die offenen Netze N_3 und N'_3 .	26
3.4	Die offenen Netze N_4 und N'_4 .	28
3.5	Die offenen Netze N_5 und N'_5 sowie ein Controller CN_5 .	29
3.6	Die aus der SEA aus Tabelle 3.1 konstruierte Engine $engineM$.	32
3.7	Das offene Netz $(buyer \oplus seller) \oplus engineM$.	32
3.8	Das offene Netz $(buyer \oplus seller) \oplus engineM'$.	33
3.9	Das offene Netz $(buyer \oplus seller) \oplus engineM''$.	33
3.10	Die offenen Netze $buyer$ und $seller$ sowie ein Adapter $adapter$.	34

Tabellenverzeichnis

2.1	Fertigkeiten der Nachrichtenverarbeitung als Transformationsregeln. . . .	17
2.2	Eine mögliche SEA eines Adapters für <i>buyer</i> und <i>seller</i>	19
3.1	Eine andere SEA für einen Adapter für <i>buyer</i> und <i>seller</i>	31

Literaturverzeichnis

- [BBC05] BRACCIALI, Andrea ; BROGI, Antonio ; CANAL, Carlos: A formal approach to component adaptation. In: *Journal of Systems and Software* 74 (2005), Nr. 1, S. 45–54
- [BCG⁺05] BENATALLAH, Boualem ; CASATI, Fabio ; GRIGORI, Daniela ; NEZHAD, Hamid R. M. ; TOUMANI, Farouk: Developing Adapters for Web Services Integration. In: *CAiSE*, 2005, S. 415–429
- [DS06] DUMAS, Marlon ; SPORK, Murray ; 0002, Kenneth W.: Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In: *Business Process Management*, 2006, S. 65–80
- [GMW08] GIERDS, Christian ; MOOIJ, Arjan J. ; WOLF, Karsten: Specifying and generating behavioral service adapter based on transformation rules / Universität Rostock. Rostock, Germany, August 2008 (CS-02-08). – Preprint
- [Loh08] LOHMANN, Niels: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: *Lecture Notes in Computer Science* 4937 (2008), April, S. 77–91
- [Mas09] MASSUTHE, Peter: *Operating Guidelines for Services*. University Press Facilities, Eindhoven, 2009. – ISBN 978–90–386–1702–2
- [MSSW08] MASSUTHE, Peter ; SEREBRENIK, Alexander ; SIDOROVA, Natalia ; WOLF, Karsten: Can I find a partner? Undecidability of partner existence for open nets. In: *Inf. Process. Lett.* 108 (2008), Nr. 6, S. 374–378
- [NBM⁺07] NEZHAD, Hamid R. M. ; BENATALLAH, Boualem ; MARTENS, Axel ; CURBERA, Francisco ; CASATI, Fabio: Semi-automated adaptation of service interactions. In: *WWW*, 2007, S. 993–1002
- [Rei85] REISIG, Wolfgang: *Monographs in Theoretical Computer Science. An EATCS Series*. Bd. 4: *Petri Nets: An Introduction*. Springer, 1985. – ISBN 3–540–13723–8
- [Sta90] STARKE, Peter H.: *Analyse von Petri-Netz-Modellen*. B. G. Teubner, Stuttgart, 1990

