HERAKLIT-Fallstudie: Service-System

Peter Fettke[1,2][0000−0002−0624−4431] und Wolfgang Reisig[3][0000−0002−7026−2810]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken, Deutschland
peter.fettke@dfki.de
[2] Universität des Saarlandes, Saarbrücken, Deutschland
[3] Humboldt-Universität zu Berlin, Berlin, Deutschland
reisig@informatik.hu-berlin.de

Abstract. The modeling method HERAKLIT is presented on the basis of a business case study. The case study demonstrates how HERAKLIT can be used to systematically model, abstract, compose and analyze computer-integrated systems of business practice. Structural aspects, abstract descriptions of data and objects as well as behavioral models are integrated in a novel way.

Keywords: systems composition · data modelling · behaviour modelling · process modelling · composition calculus · algebraic specification · Petri nets · Systems Mining

Zusammenfassung. Anhand einer Fallstudie wird die Modellierungsmethode HERAKLIT vorgestellt. Die Fallstudie demonstriert, wie mit HERAKLIT rechner-integrierte Systeme der betrieblichen Praxis systematisch modelliert, abstrahiert, komponiert und analysiert werden können. Strukturelle Aspekte, abstrakte Beschreibungen von Daten und Gegenständen sowie Verhaltensmodelle werden auf neuartige Weise integriert.

Schlüsselwörter: Systemkomposition · Datenmodellierung · Verhaltensmodellierung · Prozessmodellierung · Composition Calculus · Algebraische Spezifikation · Petrinetz · Systems Mining

Vorbemerkung

Diese Fallstudie präsentiert die zentralen Konzepte von HERAKLIT anhand eines Beispiels. Dabei geht es um ein *Service-System*: ein Unternehmen, oder allgemeiner: eine Organisation, bietet eine oder mehrere Dienstleistungen für seine Kunden an. Ein solches Service-System findet sich häufig in Wirtschaft und Verwaltung [1][3]. Beispiele sind:

zu zitieren als: FETTKE, P.; REISIG, W.: HERAKLIT-*Fallstudie: Service-System*. 2020. – HERAKLIT–Arbeitspapier, v1, 8. November 2020, http://www.heraklit.org

- eine Bank mit Finanzdienstleistungen wie Anlageberatung, Baufinanzierung und Konsumentenkredit,
- eine Anwaltskanzlei mit Beratungsangeboten auf den Rechtsgebieten Straf-, Familien- und Mietrecht,
- ein Medizinzentrum mit verschiedenen (Fach-) Ärzten für Allgemeinmedizin, Augenheilkunde und Kardiologie,
- ein Bürgeramt für verschiedene Bürgerdienste wie Personalausweis, Anwohnerparken und Eheschließung,
- ein Friseursalon mit Angeboten wie Waschen, Schneiden und Föhnen.

Häufig betreibt ein Unternehmen nicht nur ein einziges Service-System, sondern gleichartige Service-Systeme an verschiedenen Orten, die dann je nach Art Filiale, Kanzlei, Salon oder ähnlich heißen. Jedes Service-System bedient seine Kunden nach einem einheitlichen Schema. Nach diesem Schema betritt ein Kunde das Service-System und nennt den Service, zu dem er bedient werden möchte. Wenn ein passender Experte verfügbar ist, geht der Kunde zu einem Beratungsraum, wo der Experte ihn bedient. Andernfalls verlässt der Kunde das Service-System wieder.

In dieser Fallstudie wird für solche Service-Systeme ein Schema entwickelt und als HERAKLIT-Modul dargestellt. Erste Ideen für diese Fallstudie finden sich in [2].

Zum Verständnis der Fallstudiemsind einige wenige elementare Kenntnisse zu den grundlegenden Ideen von Petrinetzen hilfreich: Plätze, auf denen Marken liegen, und Transitionen, die bei Eintritt von einigen Plätzen Marken entfernen und auf einige Plätze Marken ablegen. Generell sind die ersten Seiten von [4] empfehlenswert.

1 HERAKLIT-Module und ihre Komposition

Ein Service-System besteht im Allgemeinen aus Teilsystemen, die untereinander zusammenhängen. Auf dieser offensichtlichen Beobachtung basiert das zentrale Modellierungs-Konzept der HERAKLIT-*Module*: Ein HERAKLIT-Modul M ist ein Modell, graphisch dargestellt als ein Rechteck, mit zwei wesentlichen Aspekten:

- seinem *Inneren*: das kann ganz beliebig ausgestaltet sein; es besteht gegebenenfalls nur aus dem *Namen* des Moduls, es kann aber auch die Struktur oder das dynamische Verhalten des Moduls M beschreiben;
- seiner *Oberfläche*: sie besteht aus der *linken* und der *rechten Schnittstelle*, notiert als $*M$ und $M*$. Jede der beiden Schnittstellen von M enthält Elemente, die ein *Label* tragen. Die Elemente der Oberfläche werden als *Gates* bezeichnet.

Grafisch wird ein Modul als Rechteck dargestellt, mit dem Inneren des Moduls innerhalb des Rechtecks, und mit den Elementen der linken beziehungsweise rechten Schnittstelle auf dem linken beziehungsweise rechten Rand des Rechtecks. Dabei wird jedes Schnittstellen-Element mit seinem Label repräsentiert.

Abbildung 1a skizziert die vier Komponenten eines Service-Systems. Die Bedeutung der Schnittstellen-Elemente wird im Weiteren erläutert. Abbildung 1b zeigt eine alternative Darstellung der Oberfläche: Jedes Element wird als Kante dargestellt, mit seinem Label am Ende. Diese Darstellung ist vorteilhaft bei der Komposition von Modulen.

Module können mithilfe des Kompositionsoperators $\bullet$ komponiert werden. Zur Komposition $L \bullet M$ zweier Module L und M werden gleich gelabelte Elemente von L^* und von $^*M$ miteinander verschmolzen. Es entsteht wiederum ein Modul, $L \bullet M$. Wenn mehrere Module komponiert werden, ist es wichtig, dass die Komposition assoziativ ist, dass also für Module L, M und N gilt:

$$(L \bullet M) \bullet N = L \bullet (M \bullet N).$$

Für HERAKLIT-Module ist das nachweislich immer der Fall; das rechtfertigt die Notation $L \bullet M \bullet N$.

Abbildung 1c zeigt das komponierte Modul $clients \bullet admin \bullet rooms \bullet experts$. Dieses Modul hat drei Elemente in seiner linken Schnittstelle mit den Labeln a, b und c. Seine rechte Schnittstelle ist leer.

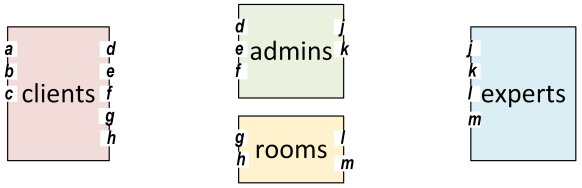
2 Objekte, Strukturen, Signaturen

Neben Modulen sind Objekte ein weiteres grundlegendes Konzept aller HERAKLIT-Modelle. Ein Objekt ist entweder ein realweltlicher Gegenstand oder ein Datenelement. Jedes Objekt hat einen Typ. Im systematischen Aufbau von HERAKLIT sind Objekte Elemente von *Mengen*; sie sind also nicht im speziellen Sinne der objektorientierten Modellierung oder Programmierung zu verstehen.

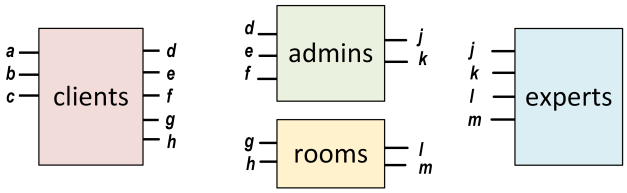
In unserem Beispiel gibt es in einem Service-System vier Sorten realweltlicher Gegenstände: Kunden, Personal, Experten und Räume. Im gegebenen Beispiel ist ein Datenelement oft ein digitaler Zwilling eines realweltlichen Gegenstandes. Beispielsweise hat jeder Kunde und jeder Experte einen digitalen Zwilling in der Service-System-Verwaltung. Services und Raumnummern sind Sorten mit Datenelementen ohne eine gegenständliche Entsprechung im Modell. Wir verwenden den Ausdruck „digitaler Zwilling“ also in einem ganz einfachen Sinne, der später noch präzisiert wird, aber nur entfernt mit dem allgemeinen Schlagwort „digitaler Zwilling“ zusammenhängt.

Neben Objekten gibt es im Allgemeinen noch Konstanten, Funktionen und Relationen über Mengen solcher Objekte. Alles das wird im Konzept einer Struktur zusammengestellt. Solche Strukturen sind zugleich der semantische Bereich, für den die Prädikatenlogik Aussagen formuliert. Das wiederum ist die Basis für den Nachweis der Korrektheit von HERAKLIT-Modellen. Abbildung 2 beschreibt die Objekte eines typischen Service-Systems.

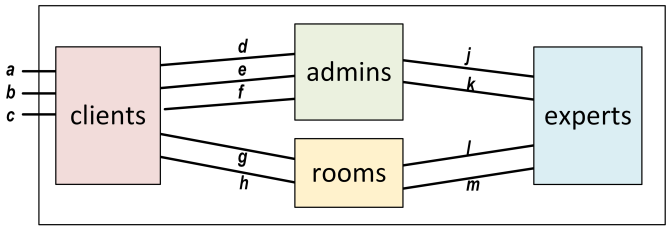
Die zentralen Konzepte eines Service-Systems können beschrieben werden mit einer Struktur S, die durchaus andere Objekte enthält, aber strukturell identisch ist, also sieben Grundsorten (Kunden, digitale Zwillinge von Kunden, einen Kundenbetreuer, Services, Experten, digitale Zwillinge von Experten und Raumnummern), und dazu eine Funktion, die zu jedem Service die verfügbaren



(a) die vier Module eines Service-Systems



(b) alternative Darstellung



(c) Komposition $clients \bullet admins \bullet rooms \bullet experts$

Jedes der vier Module wird hier lediglich als Rechteck mit seinem Namen skizziert. Jedes Modul M hat eine linke und eine rechte Schnittstelle, angedeutet mit Kanten und notiert als *M und M*. Die Schnittstelle *clients ragt aus dem Service-System hinaus.

Elemente von clients* werden mit Elementen von *admins und von *rooms verschmolzen. Elemente von admins* und von rooms* werden mit Elementen von *experts verschmolzen. Die Schnittstelle experts* enthält keine Elemente.

Abb. 1: vier Module und ihre Komposition

Experten nennt. Wir wollen nun alle Strukturen dieser Art charakterisieren. Dieses Problem ist in der formalen Logik wohlbekannt und wird mit Hilfe von *Signaturen* gelöst: eine Signatur für die Struktur S enthält für jede Grundsorte M ein Symbol (den Typ von M) und für jede Konstante, jede Funktion und jede Relation ein Symbol des entsprechenden Typs. Abbildung 3 zeigt eine solche Signatur Σ. Dabei sind C, $\underline{C}$, A, S, E, $\underline{E}$ und R Sortensymbole. Jedes andere Symbol der Signatur hat einen Typ, der aus den Grundsorten abgeleitet ist. Dieser Typ steht jeweils nach dem Doppelpunkt. Abbildung 3 nennt als Symbole mit abgeleiteten Sorten das Konstantensymbol adm, das Funktionssymbol f und die drei Relationssymbole EX, $\underline{EX}$und RO.

Eine *Instanziierung* der Signatur Σ ordnet zunächst jedem Sortensymbol eine Menge zu. Um ein Service-System zu charakterisieren, nimmt man dafür die

Grundsorten
Kunden:
C = *Menge aller Personen mit gültigem Ausweis*

digitale Zwillinge von Kunden:
C = {c | c ist digitaler Zwilling einer Person c aus C}

Kundenbetreuer:
A = {**Ron**}

Services:
S = {**loan, savings, credit card**}

Experten:
E = {**Max, Zoe, Kim**}

digitale Zwillingen von Experten:
E = {**Max, Zoe, Kim**}

Raumnummern:
R = {**001, 002**}

Konstanten
Ron**: A**

Funktion
f: **S** ⟶ P(***E)***
f(**loan**) = {**Kim**}
f(**savings**) = {**Kim, Max, Zoe**}
f(**credit card**) = {**Max, Zoe**}

Relationen (einstellig)
{Max, Zoe, Kim} Teilmenge von **E**
{Max, Zoe, Kim} Teilmenge von **E**
{001, 002} Teilmenge von **R**

Abb. 2: die Struktur S des Service-Systems

Grundsorten
C *Kunden*
C *digitale Zwillinge von Kunden*
A *Kundenbetreuer*
S *Services*
E *Experten*
E *digitale Zwillinge von Experten*
R *Raumnummern*

Konstantensymbole
adm: A *Kundenbetreuer*

Funktionssymbole
f: S ⟶ P(E) *Experten für Service*
_ : C ⟶ C
_ : E ⟶ E

Relationensymbole
EX Teilmenge von **E**
EX Teilmenge von **E**
RO Teilmenge von **R**

Forderungen
_ ist bijektiv

Abb. 3: Signatur Σ für das Service-System

Kunden und ihre digitalen Zwillinge, den Kundenbetreuer, die Services, die Experten und ihre digitalen Zwillinge, und die Raumnummern des Service-Systems. Das Funktionssymbol f wird als eine Funktion instanziiert, die jedem Service s eine Menge von Experten zuordnet; intuitiv: die Experten für den Service s. Dabei kann ein Experte selbstverständlich mehrere Services bedienen. Das Funktionssymbol „_“ wird intuitiv folgerichtig zur Bezeichnung digitaler Zwillinge

für Kunden und für Experten verwendet. Die drei einstelligen Relationssymbole werden mit Teilmengen des entsprechenden Typs instanziiert, die in der Darstellung des dynamischen Verhaltens von Service-Systemen den Anfangszustand beschreiben.

3 Lokale Zustände und Ereignisse

Generell beschreibt ein HERAKLIT-Modell das Verhalten eines Systems mit dem Konzept *lokaler Zustände*, die von *diskreten Ereignissen* aktualisiert werden. Das Resultat eines Ereignisses kann Anlass für weitere Ereignisse sein.

Die Beschreibung lokaler Zustände basiert auf *Prädikaten*: Für ein Prädikat p und ein Objekt o gilt: p trifft auf o zu, oder eben nicht. Damit ist ein lokaler Zustand ein Prädikat p zusammen mit einem Objekt o. Dynamik entsteht, indem das Objekt o den lokalen Zustand erreicht: damit trifft p auf o zu. Wenn o den lokalen Zustand verlässt, trifft p nicht mehr auf o zu. Das Erreichen und Verlassen lokaler Zustände wird durch Ereignisse beschrieben: indem ein Ereignis eintritt, erreichen oder verlassen einige Objekte einige lokale Zustände. Technisch ist ein Ereignis ein sehr einfaches Petrinetz: eine Transition, verbunden mit einigen Plätzen, und eingebettet in ein sehr einfaches Modul. Es ist bequem, mit dem Namen der Transition eines Ereignisses auch das ganze Modul zu bezeichnen.

Ein typisches Beispiel für ein Ereignis ist das Betreten eines Service-Systems: indem beispielsweise die Kundin Alice wegen eines Kredits das Service-System betritt, erreicht $(Alice, loan)$ das Prädikat *entered*. Abbildung 4a stellt das Ereignis *enter* grafisch dar. Die Schnittstelle $^*enter$ enthält also eine Transition mit dem Label *enter* und die Schnittstelle $enter^*$ enthält einen Platz mit dem Label $entered(Alice, loan)$. Der rote Hintergrund zeigt, dass das gesamte Ereignis im Verhaltensmodul Kunden liegt.

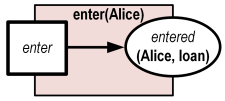
Ein weiteres Beispiel ist das Ereignis *serve clients* in Abbildung 4b: nachdem Alice mit ihrem Kreditwunsch das Service-System betreten hat, bittet der Kundenbetreuer Ron sie zu warten; Ron engagiert sich nun, um Alice zu bedienen. Alice bleibt im Kunden-Modul (roter Hintergrund); für die Verwaltung erzeugt das Ereignis *serve clients* einen digitalen Zwilling $\underline{Alice}$ von *Alice* und behält das Datenelement *loan* (grüner Hintergrund).

Zwei Ereignisse e und f können zu $e \bullet f$ komponiert werden. Wie schon in Abschnitt 1 erläutert, werden dafür gleich gelabelte Elemente von e^* und $^*f$ miteinander verschmolzen. In Abbildung 4 haben die Schnittstellen $enter^*$ und $^*serve\_clients$ gleich gelabelte Plätze; Abbildung 4c zeigt die Komposition $enter \bullet serve\_clients$.

Wie schon ein einzelnes Ereignis, hat auch die Komposition zweier Ereignisse die Struktur eines Petrinetzes; bestehend aus Plätzen, Transitionen und Pfeilen.

4 Verteilte Abläufe

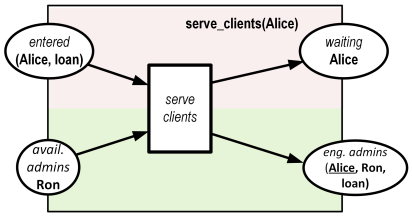
Die Komposition mehrerer Ereignisse beschreibt Verhalten; es entsteht ein *Ablauf*. Der Ablauf in Abbildung 4c kann im Service-System auf zwei unterschiedli-



Wie jedes Modul ist das Ereignis **enter(Alice)** dargestellt als ein Rechteck, mit den Elementen der Schnittstellen auf dem linken bzw. rechten Rand. Hier liegt die Transition mit dem Label *enter* auf dem linken Rand; das Prädikat *entered* zusammen mit dem Objekt **(Alice, loan)** ist das label für den Platz auf dem rechten Rand.

Der Pfeil von der Transition zum Platz zeigt, dass **(Alice, loan)** bei Eintritt von *enter* den lokalen Zustand *entered* erreicht. Der rötliche Hintergrund zeigt, dass das Ereignis innerhalb des Verhaltensmoduls Kunden liegt.

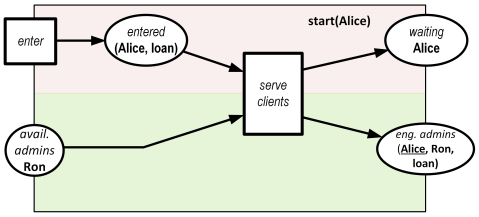
(a) Ereignis $enter(Alice)$



Die linke Schnittstelle dieses Ereignisses enthält Plätze, die mit den beiden lokalen Zuständen *entered* mit **(Alice, loan)** und *available admins* mit **Ron** gelabelt sind. Die rechte Schnittstelle enthält zwei Plätze mit den beiden lokalen Zuständen *waiting* mit **Alice** und *engaged admins* mnit **(Alice, Ron, loan)** als Label.

Der obere, rötliche Hintergrund zeigt, dass Alice im Verhaltensmodul *Clients* bleibt; Ron hingegen bleibt in der Verwaltung. Die Transition *serve cients* liegt in beiden Verhaltensmoduen und damit in einer Schnittstelle von clients und in einer Schnittstelle der Verwaltung.

(b) Ereignis $serve\_clients(Alice)$



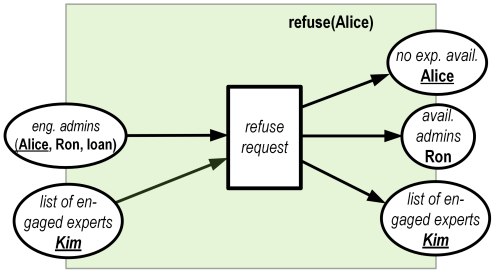
Der obere, rötliche Hintergrund zeigt, dass Alice im Verhaltensmodul *Clients* bleibt; Ron hingegen bleibt in der Verwaltung. Die Transition *serve cients* liegt in beiden Verhaltensmodulen und damit in Schnittstellen beider Verhaltensmoduleodule.

(c) Komposition $start(Alice) := enter(Alice) \bullet serve\_clients(Alice)$

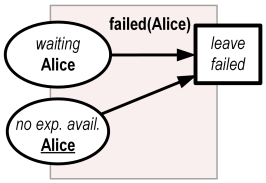
Abb. 4: zwei Ereignisse und ihre Komposition

che Weisen fortgesetzt werden: Wenn ein Experte für Kredite verfügbar ist, wird Alice bedient. Andernfalls wird ihre Bitte abgelehnt. Im zweiten Fall tritt das Ereignis $refuse\ request$ ein (Abbildung 5b), gefolgt von $leave\ failed$ (Abbildung 5b). Die Komposition beider Ereignisse zeigt der Ablauf $no_service$ (Abbildung 5c). Nicht nur einzelne Ereignisse, sondern auch Abläufe können (zu längeren) Abläufen komponiert werden. Abbildung 5d komponiert die beiden Abläufe $start(Alice)$ und $no_service$. Dieser Ablauf ist *vollständig*: Er beginnt im Anfangszustand und endet ohne eine aktivierte Transition.

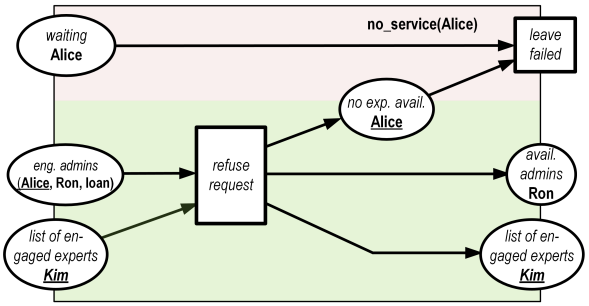
Alternativ zu $no_service$ ist der Ablauf $served(Bob)$ in Abbildung 6. Der Ablauf $start(Bob)$ in (a) ist strukturell identisch mit (c) in Abbildung 4. Anstelle von Alice, die einen Kredit wünscht, will hier Bob Geld anlegen. Dann aber geht es anders weiter: Wie Abbildung 6b darstellt, findet Ron diesmal in der Liste verfügbarer Experten dafür den Eintrag $\underline{Zoe}$, also einen digitalen Zwilling der Expertin Zoe. Das Ereignis *accept request* tritt nun ein mit Zoe in der Liste verfügbarer Experten und dem freien Raum 002. Nach dem Eintritt ist



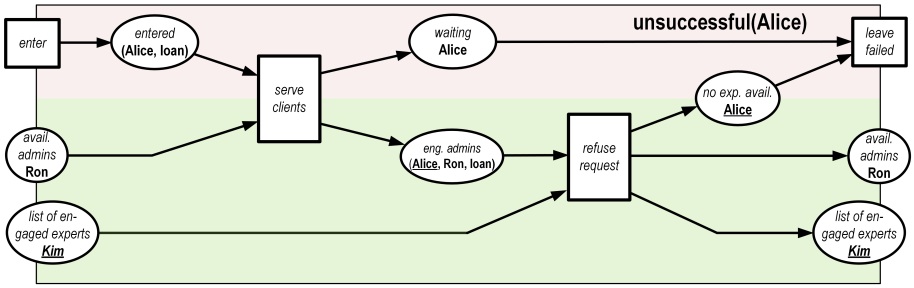
(a) Ereignis $refuse(Alice)$



(b) Ereignis $failed(Alice)$



(c) $no\_service := refuse(Alice) \bullet failed(Alice)$

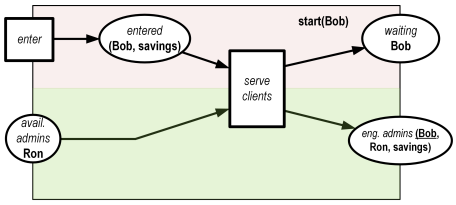


(d) ein vollständiges Ablauf: $unseccessful(Alice) := start(Alice) \bullet no\_service(Alice)$

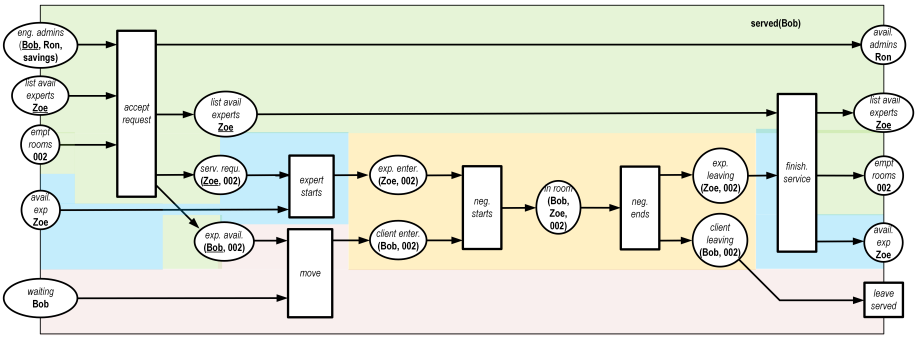
Abb. 5: Komposition eines erfolglosen Ablaufs

Zoe in der Liste *engaged experts*. Zudem werden Zoe und Bob in den Raum 002 gebeten. Zoe und Bob gehen in diesen Raum, verhandeln über den Kredit und verlassen den Raum wieder. Nach Ende der Verhandlungen wird mit $finish\ service$ Zoe wieder verfügbar und zugleich ihr digitaler Zwilling $\underline{Zoe}$ in die Liste der verfügbaren Experten aufgenommen. Raum 002 ist wieder frei. Unabhängig davon verlässt Bob das Service-System.

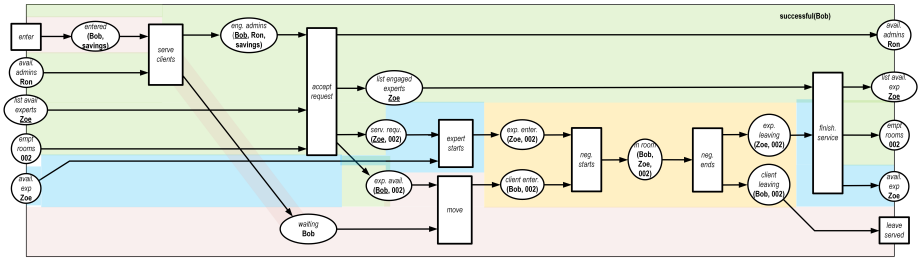
Betrachten wir nun den Fall, dass beide Kunden unabhängig voneinander das Service-System betreten. Der Kundenbetreuer *Ron* bedient erst *Bob* und dann *Alice*. Abbildung 7 zeigt diesen Ablauf: Nach Eintritt von *accept request* für *Bob*



(a) Ablauf $start(Bob)$



(b) Ablauf $served(Bob)$



(c) ein vollständiger Ablauf: $successful(Bob) := start(Bob) \bullet served(Bob)$

Abb. 6: Komposition eines erfolgreichen Ablaufs

ist *Ron* wieder verfügbar. Er bedient Alice, während Bob und Zoe miteinander verhandeln.

In diesem Ablauf werden Alice und Bob fast unabhängig voneinander bedient. Nur die Transition *serve clients* bringt sie in eine Reihenfolge. In einer Variante des Service-Systems mit mehreren Kundenbetreuern kann ein verteilter Ablauf aus ganz unverbundenen Komponenten bestehen. Das Konzept globaler Zustände eines Service-Systems schafft dabei wenig Nutzen.

5 Schematische Darstellung von Schritten

In Abschnitt 4 haben wir zwei konkrete Abläufe in einem Service-System dargestellt: Alice wünscht einen Kredit und Bob will Geld anlegen. Die Struktur

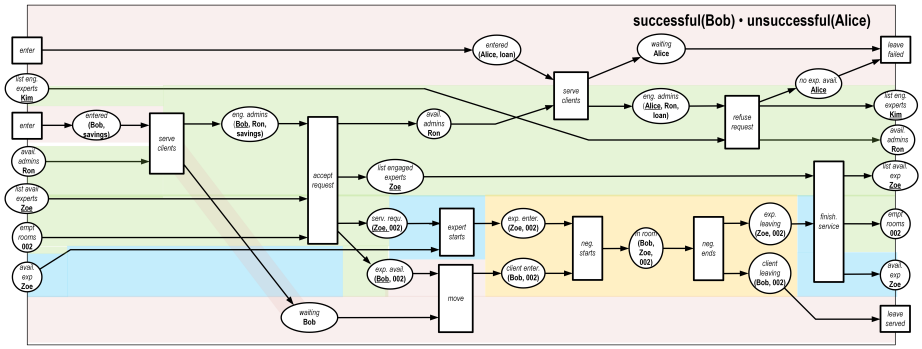


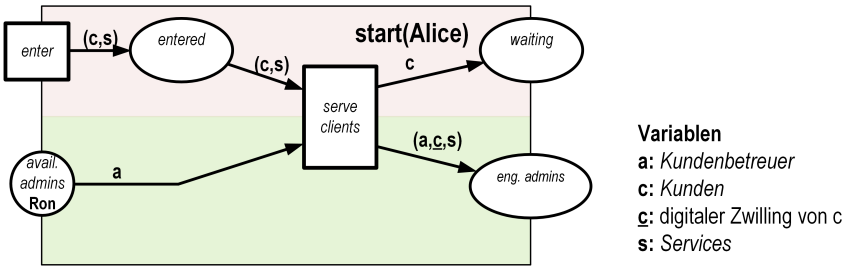
Abb. 7: Komposition $successful(Bob) \bullet unsuccessful(Alice)$

S in Abbildung 2 nennt aber weitere Kunden, Services, Experten und Räume. Insgesamt ergibt sich so eine Vielzahl von Abläufen. Deren zugrundeliegende Petrinetz-Struktur aus Plätzen, Transitionen und Pfeilen ist identisch mit der Struktur der beiden Abläufe aus Abbildung 5d und Abbildung 6b. Deshalb liegt es nahe, sie alle zusammen in einem einzigen Petrinetz-Schema darzustellen. Die Idee dabei: Ereignisse mit gleichen Prädikaten werden zusammengefasst. Ein Prädikat selbst wird statisch als Platz dargestellt; ein Objekt, dargestellt als Petrinetz-Marke, erreicht oder verlässt das Prädikat (und damit als Petrinetz-Marke den Platz), wenn eine Transition eintritt. Als Beispiel zeigt Abbildung 8a die Darstellung des Ereignisses des start-Ablaufs von Alice in der Darstellung als Schema. Zunächst hat kein Objekt das Prädikat *entered* erreicht. Der Pfeil von *enter* nach *entered* ist beschriftet mit zwei Parametern (Variablen), c und s. Die Belegung von c mit *Alice* und von s mit *loan* ist ein Modus für den Eintritt von *enter*. Der Eintritt von *enter* in diesem Modus führt dann zum Zustand in Abbildung 8b. In diesem Zustand kann nun die Transition *serve clients* eintreten, im Modus $c = Alice$, $s = loan$ und $a = Ron$. Den resultierenden Zustand zeigt Abbildung 8c.

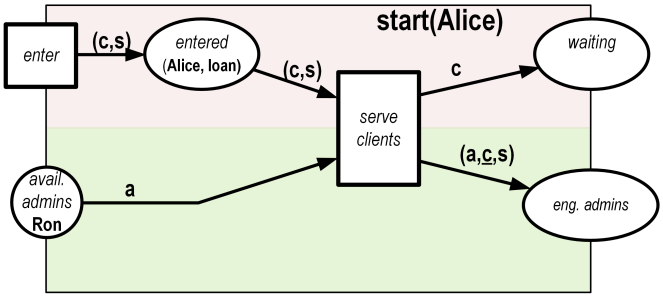
Im Anfangszustand kann die Transition *enter* in jedem Modus eintreten, der die Variable c mit einem Kunden und die Variable s mit einem Service belegt. Wer als Kunde und was als Service überhaupt in Frage kommt, beschreibt die Struktur S aus Abschnitt 2.

Die Transition *serve clients* kann vor Eintritt von *enter* selbst nicht eintreten: Die Beschriftung (c, s) des Pfeiles von *entered* zu *serve clients* verlangt eine Marke aus einem Kunden und einem Service auf *entered*. Dort liegt aber gar keine Marke. Nach Eintritt von *enter* liegt $(Alice, loan)$ auf dem Platz *entered*. Deshalb kann nun *serve clients* eintreten, aber nur in dem oben genannten Modus $c = Alice$, $s = loan$ und $a = Ron$. Abbildung 9 zeigt den Ablauf *start* mit dem Modus $c = Bob$, $s = savings$ und $a = Ron$.

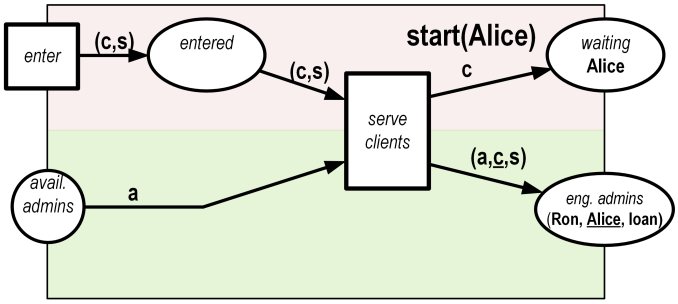
Im Anfangszustand des Ablaufs $start(Alice)$ (und $start(Bob)$) charakterisiert die Struktur S aus Abbildung 2 viele weitere Belegungen der Variablen c und



(a) Anfangszustand



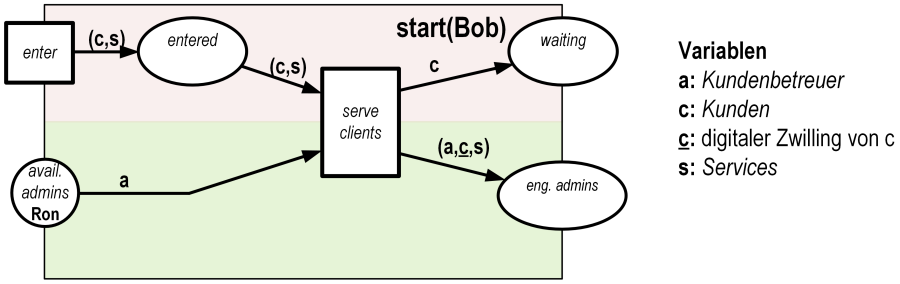
(b) nach Eintritt von *enter* mit $c = Alice, s = loan, a = Ron$



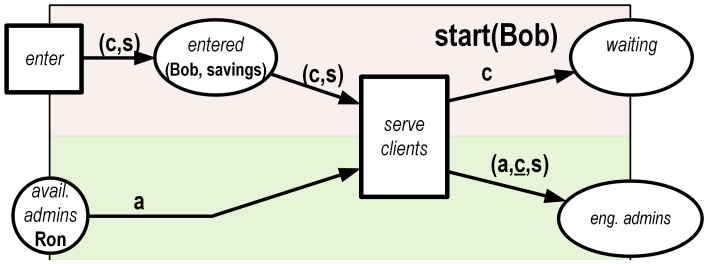
(c) nach Eintritt von *serve clients* mit $c = Alice, s = loan, a = Ron$

Abb. 8: der Ablauf $start(Alice)$ in schematischer Darstellung

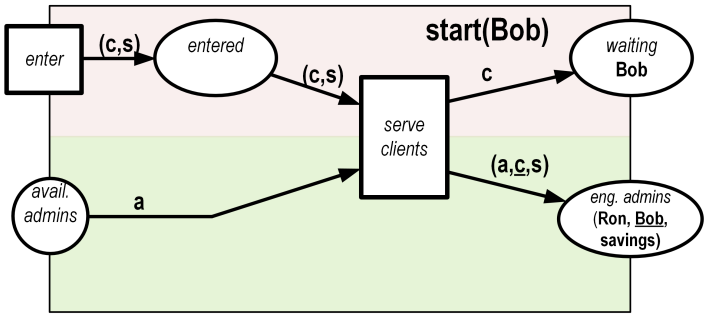
s, also viele Modi, mit denen *enter* eintreten kann: insbesondere mit anderen Kunden, und mit den Services *stock* und *credit card*. Jeder dieser Modi ist zugleich ein Modus für die Transition *serve clients*.



(a) Anfangszustand



(b) nach Eintritt von *enter* mit $c = Bob, s = savings, a = Ron$



(c) nach Eintritt von *serve clients* mit $c = Bob, s = savings, a = Ron$

Abb. 9: der Ablauf $start(Bob)$ in schematischer Darstellung

6 Alternativen, die *elm*-Notation, Zyklen: das Modell für die Struktur S

Die beiden Abbildungen 8c und 9c zeigen den Anfang der Abläufe der beiden Kunden Alice und Bob mit ihrem jeweiligen Beratungswunsch. Die zugrunde liegenden Netze sind gleich; nur die Marken unterscheiden sich. Der nächste Schritt

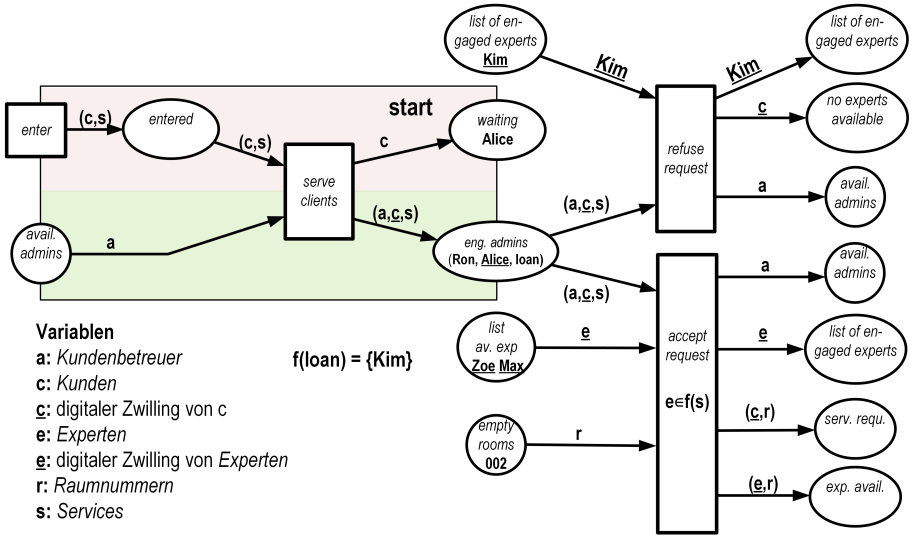
ist nun die Alternative zwischen Zustimmung und Ablehnung des Wunsches nach Beratung. Diese Alternative soll nun in das Netz integriert werden. Im Prinzip ist das einfach: Am Platz *engaged admins* beginnen zwei Pfeile: einer führt zur Transition *accept request* und der andere zu $refuse\ request$. Marken auf weiteren Plätzen im Vorbereich der beiden Transitionen regeln dann, welche der beiden Transitionen eintritt.

Abbildung 10a zeigt mit der Marke $(Ron, Alice, loan)$ auf dem Platz *engaged admins* diese Situation für den ersten Fall: Alice wünscht einen Kredit. Die Funktion f nennt Kim als einzigen Experten für Kredite. Der digitale Zwilling von Kim liegt nicht in der Liste verfügbarer Experten. Damit kann die Variable e nicht so belegt werden, dass die Beschriftung $e \in f(s)$ in der Transition *accept request* erfüllt ist. Somit kann *accept request* in der Situation von Abbildung 10a nicht eintreten.

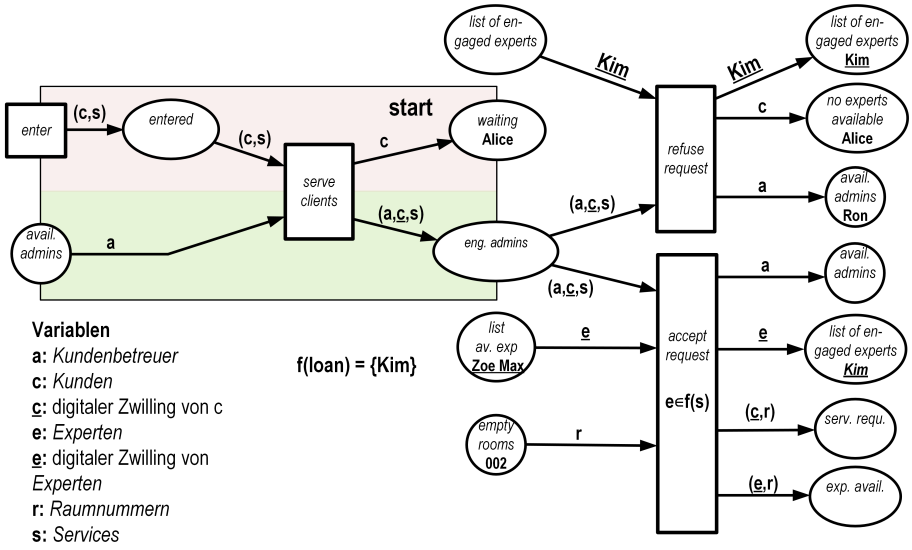
Indem Kim nicht verfügbar ist, liegt Kim in der Liste *engaged experts*. Damit ist die Transition $refuse\ request$ aktiviert. Ihr Eintritt führt zur Markierung von Abbildung 10b.

Den zweiten Fall zeigt Abbildung 11a zeigt mit der Marke $(Ron, Bob, savings)$ auf dem Platz *engaged admins*. Die Funktion f nennt Kim, Max und Zoe als Spezialisten für *savings*. Max und Zoe sind verfügbar; beide genügen der Bedingung der Beschriftung $e \in f(s)$ in der Transition *accept request*. Damit kann *accept request* mit zwei verschiedenen Belegungen eintreten. Abbildung 11b zeigt das Resultat für den Fall $e = Zoe$. Indem Max und Zoe verfügbar sind, liegen Max und Zoe in Abbildung 11a nicht in der Liste der *engaged experts*. Dort liegt nur Kim. Die Transition $refuse\ request$ ist aber nur aktiviert, wenn alle drei Experten *engaged* sind. Somit kann $refuse\ request$ in Abbildung 11a nicht eintreten.

Auf dem Weg zu einer rein schematischen Darstellung des Systems liegt noch ein Problem: Jedes Prädikat soll nur genau ein Mal dargestellt werden. In den beiden Abbildungen 10 und 11 kommt aber *available admins* zwei Mal und *engaged experts* sogar drei Mal vor. Wie Abbildung 12 zeigt, ist das für *available admins* kein Problem: Die beiden Vorkommen werden verschmolzen. Indem man nun auch die drei Vorkommen von *engaged experts* verschmilzt, entsteht eine Schlinge aus zwei Pfeilen mit der Transition $refuse\ request$. Auch das ist nicht weiter problematisch. Unklar ist aber die Wahl der Beschriftung der beiden Pfeile: Abbildung 10 verlangt die Beschriftung Kim, Abbildung 11 verlangt die Beschriftung Kim, Max und Zoe. Intuitiv formuliert beschreibt die Schlinge und die Stelle eine Nebenbedingung: $refuse\ request$ tritt für einen Kunden Service s_0 nur ein, wenn alle seine Experten $f(s_0)$ *engaged* sind. Man könnte nun die Pfeilbeschriftung „$f(s)$“ vorschlagen. Das würde aber für $s = loan$ die Marke Kim und für s = savings die Marke Kim, Max, Zoe auf dem Platz *engaged experts* verlangen. Wir erwarten aber stattdessen die Elemente dieser Mengen als Marken auf dem Platz *engaged experts*, wie in den Abbildungen 10a und 11a. Das erreichen wir mit der Notation elm. Für eine Menge A trifft ein Prädikat p (also ein Platz) auf $elm(A)$ genau dann zu, wenn p auf jedes Element von A zutrifft. Die Beschriftung $elm(f(s))$ bezeichnet also die einzelnen Elemen-



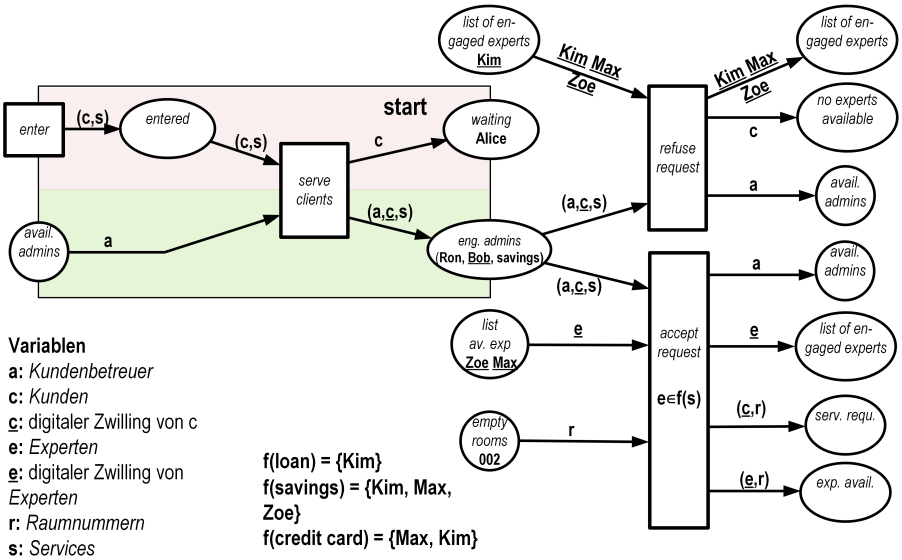
(a) *refuse request* ist aktiviert, *accept request* ist nicht aktiviert



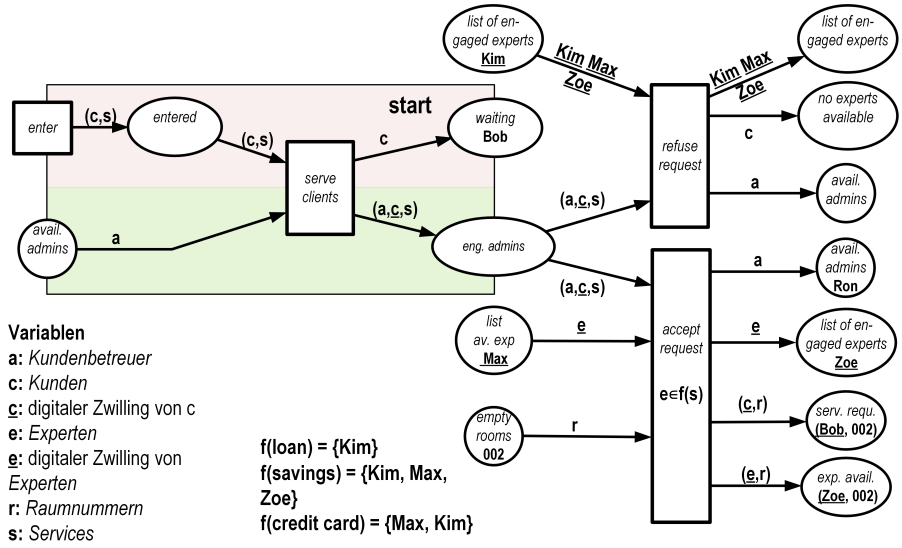
(b) nach Eintritt von *refuse request* mit $a = Ron, \underline{c} = Alice, s = loan$

Abb. 10: der Ablauf *refuse request*

te von $f(s)$ als Marken auf dem Platz *list of engaged experts*. Dabei ist $f(s)$ die Menge der digitalen Zwillinge der Experten für den Service s.



(a) *refuse request* ist nicht aktiviert, *accept request* ist aktiviert



(b) nach Eintritt von *accept request* mit $a = Ron, \underline{c} = Bob, s = savings$

Abb. 11: Eintritt von *accept request*

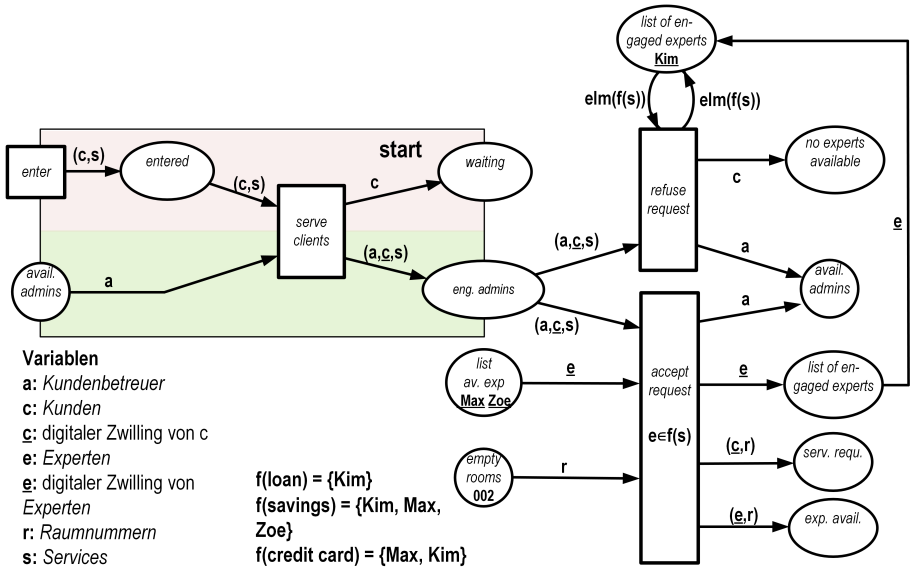


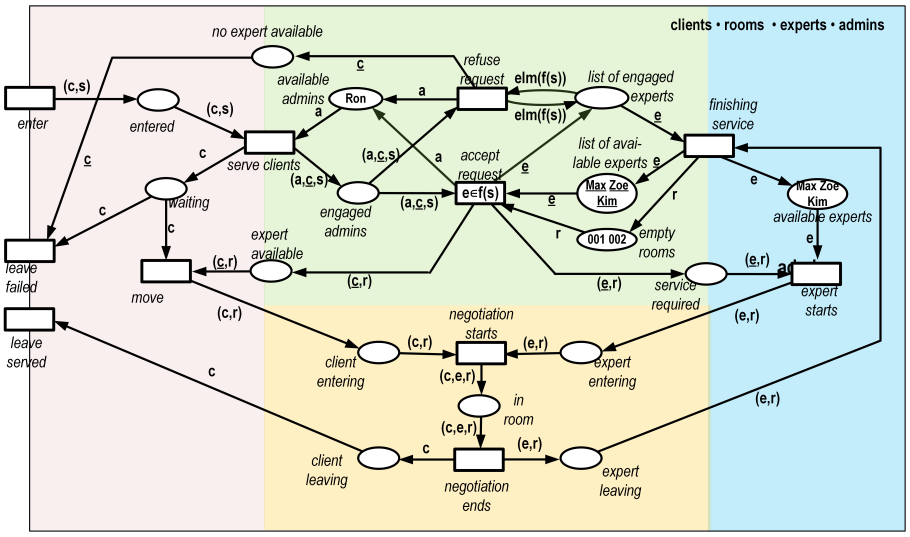
Abb. 12: schematische Darstellung der Alternative zwischen *accept request* und *refuse request*

Abbildung 12 zeigt einen Teil des gesamten Systems, das in Abbildung 13 dargestellt ist.

7 Das Modell der Service-Organisation

Abbildung 14 fixiert eine Reihe von Mengen und ihre Elemente: Kunde ist jede Person mit einem gültigen Ausweis, Kundenbetreuer ist Ron, Experten sind Max, Zoe und Kim mit fester Zuordnung zu vier Services, und für Beratungen sind die beiden Räume 001 und 002 verfügbar. Damit sind unendlich viele Abläufe beschrieben. Dennoch gibt es Service-Systeme mit ganz anderer Charakteristik. Ein Beispiel ist ein Medizinzentrum mit vorangemeldeten Patienten (Kunden), vielen Ärzten und anderem medizinischen Personal (Experten), vielen Diagnose- und Besprechungsräume et cetera.

Wie in Abschnitt 2 diskutiert, können mit einer Signatur beliebige solche Strukturen beschrieben werden. Abbildung 14 zeigt nun das Service-System auf der Basis der Signatur Σ. Diese Darstellung unterscheidet sich von Abbildung 13 nur in der Anfangsmarkierung: Die vier Plätze *available admin*, *list of available experts*, *available experts* und *empty rooms* sind mit Symbolen der Signatur beschriftet. Auch die Beschriftung der Transition *accept request* verwendet Symbole der Signatur. Jede Instanziierung der Signatur Σ erzeugt ein konkretes Service-System. Eine der Instanziierungen erzeugt das System aus Abbildung 13.



Die Struktur S (Wiederh.):

Grundsorten
Kunden:
C = *Menge aller Personen mit gültigem Ausweis*

digitale Zwillinge von Kunden:

C = {c | c ist digitaler Zwilling einer Person c aus C}

Kundenbetreuer:
A = {**Ron**}

Services:
S = {**loan, savings, credit card**}

Experten:
E = {**Max, Zoe, Kim**}

digitale Zwillinge von Experten:
E = {**Max**, **Zoe**, **Kim**}

Raumnummern:
R = {**001, 002**}

Konstanten
Ron**: A**

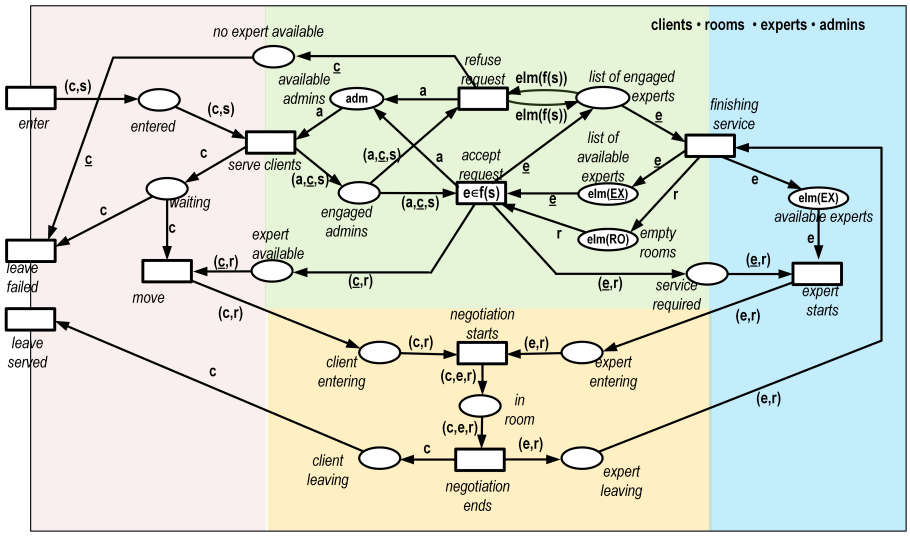
Funktion
f: **S** ⟶ P(***E***)
f(**loan**) = {**Kim**}
f(**savings**) = {**Kim, Max, Zoe**}
f(**credit card**) = {**Max, Zoe**}

Relationen (einstellig)
{Max, Zoe, Kim} Teilmenge von **E**
{*Max*, *Zoe*, *Kim*} Teilmenge von **E**
{001, 002} Teilmenge von **R**

Variablen
a: *Kunenbetreuer*
c: *Kunden*
c: digitaler Zwilling von c
e: *Experten*
e: digitaler Zwilling von *Experten*
r: *Raumnummern*
s: *Services*

Abb. 13: das Service-System auf Basis der Struktur S

Im Einzelnen: Der Platz *available admins* enthält nun nicht den konkreten Kundenbetreuer Ron, sondern das Konstantensymbol *adm*. Zu den vielen möglichen Instanziierungen dieses Symbols gehört natürlich auch Ron. Der Platz *empty rooms* enthält nun nicht die Raumnummern 001 und 002, sondern $elm(RO)$: Eine Instanziierung des Symbols RO ist eine beliebig gewählte Menge M, deren Elemente *Raumnummern* heißen. RO als Beschriftung von *empty rooms* würde eine einzige Marke generieren, nämlich die Menge M. Stattdessen wollen wir aber die einzelnen Elemente von M als Marken; die Notation



Die Signatur (Wiederh.):

Grundsorten
C *Kunden*
C *digitale Zwillinge von Kunden*
A *Disponenten*
S *Services*
E *Experten*
E *digitale Zwillinge von Experten*
R *Raumnummern*

Funktionssymbole
f: S ⟶ P(E) *Experten für Services*
(_): C ⟶ C
(_): E ⟶ E

Relationensymbole
EX Teilmenge von E
EX Teilmenge von E
RO Teilmenge von R

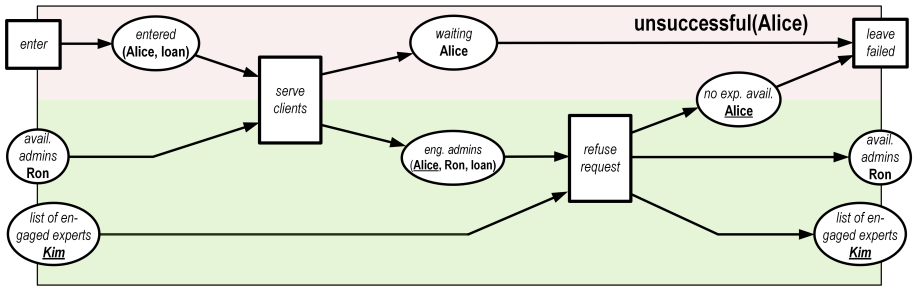
Variablen
a: *Kundenbetreuer*
c: *Kunden*
c: digitaler Zwilling von c
e: *Experten*
e: digitaler Zwilling von *Experten*
r: *Raumnummern*
s: *Services*

Abb. 14: Modell der Service-Organisation

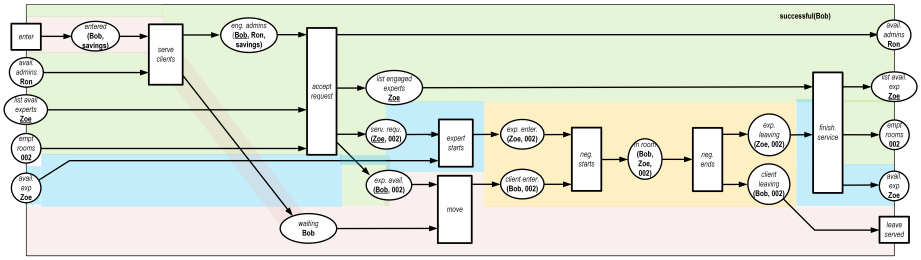
elm erzeugt diese einzelnen Marken (wie schon an den Pfeilen zwischen der Transition *refuse request* und dem Platz *list of engaged experts*). Analog dazu beschreibt eine Instanziierung für den Platz *available experts* eine Menge von Experten, und für den Platz *list of available experts* eine entsprechende Menge digitaler Zwillinge. In der Signatur garantiert die Forderung der Bijektivität für die „_“-Funktion diese Entsprechung.

Damit sind alle Aspekte modelliert, die zum Verständnis der Funktionalität der Service-Systeme eines Unternehmens beitragen: die Signatur Σ beschreibt den generellen Aufbau des Service-Systems; jede Instanziierung repräsentiert ein mögliches konkretes Service-System, also eine bestimmte Filiale, eine Kanzlei, ein Medizinzentrum, ein Bürgeramt, einen Friseursalon oder ähnliches.

Für jede Instanziierung gibt es zwei Sorten elementarer Abläufe: jeder Eintritt von *enter* führt entweder zu einer Beratung oder zu einer Ablehnung (siehe Abbildungen 5d und 6c, die in der Abbildung 15 aus Gründen besserer Übersichtlichkeit noch einmal gegenübergestellt werden). Wenn die beteiligten Mengen von Kunden, Experten und Räumen endlich sind, gibt es nur endlich



(a) der erfolglose Ablauf



(b) der erfolgreiche Ablauf

Abb. 15: die beiden elementaren Abläufe des Service-Systems

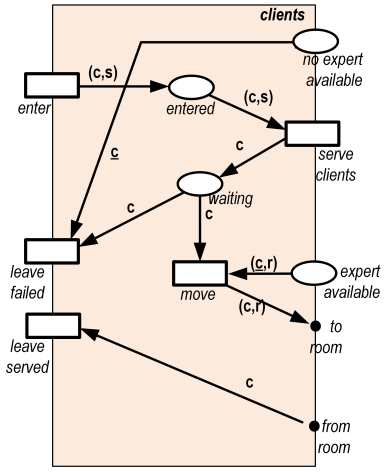
viele solcher elementarer Abläufe. Sie können beliebig hintereinander gesetzt werden und somit unbeschränkt lang werden.

8 Modelle der vier Module

In Abschnitt 7 haben wir das Modell des Service-Systems aus verteilen Abläufen heraus konstruiert. Alternativ dazu kann man es auch aus Modellen der vier Module *clients*, *rooms*, *experts* und *admin* herleiten, die wir nun als HERAKLIT-Module konstruieren und dann komponieren. Diese Herleitung betont die Flexibilität des Kompositionsoperators.

Im Weiteren wird erläutert, welches Verhalten jedes einzelne Modul darstellt, und was die Komposition der Module ergibt.

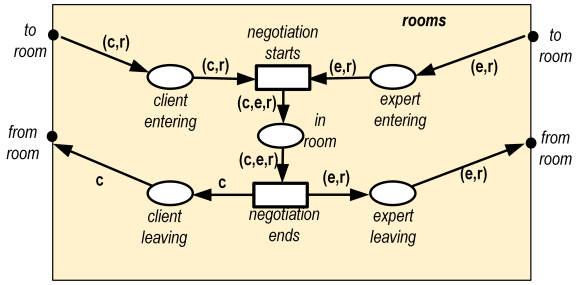
1. Das Modul *clients* (Abbildung 16): Ein Kunde betritt das Service-System mit dem Wunsch nach einem Service. Er wird gebeten zu warten. Dann erhält er entweder die Nachricht, dass kein passender Experte verfügbar ist; in diesem Fall verlässt der Kunde das Service-System wieder. Oder er wird in den Besprechungsraum gebeten, aus dem er später zurückkehrt und dann das Modul verlässt. Technische Einzelheiten dazu werden in der Abbildung beschrieben.



An der Transition *enter* endet kein Pfeil; damit ist *enter* immer aktiviert. *Enter* tritt ein mit einer Belegung der Variablen **c** durch einen frei wählbaren Kunden c0, und der Variablen **s** durch einen ebenfalls frei wählbaren Service s0, zu dem c0 eine Beratung wünscht.
Damit gelangt (c0,s0) auf den Platz *entered*. Die Transition *serve clients* liegt auf der Schnittstelle des Moduls; sie wird später mit einer Transition des Vewaltungsmoduls verschmolzen. Mit ihrem Eintreten erreicht c0 den Platz *waiting*. Wenn nun in der Schnittstelle von außen auf den Platz *no expert available* ein digitaler Zwilling c0 von c0 gelangt, verlässt c0 das Modul über die Transition *leave failed* (mit der Belegung c0 für **c**). Andernfalls kommt von außen eine Marke der Form (c0,r0) aus einem digitalen Zwilling c0 von c0 und einer Raumnummer r0 auf den Platz *expert available*. Über *move* verlässt dann c0 das Modul entlang des Pfeiles *to room*. Später kommt c0 nach einer Beratung zurück und verlässt das Modul über die Transition *leave served*.

Variablen
c: *Kunden*
c: digitaler Zwilling von c
r: *Raumnummern*
s: *Services*

Abb. 16: das Modul *clients*



In der linken und der rechten Schnitt-stelle liegen nur unterbrochene Pfeile. Die Transition *negotion starts* tritt ein, wenn es einen Raum r0 gibt, der in einer Marke (c0,r0) auf *client entering* und auch in einer Marke (e0,r0) in *expert entering* vorkommt. Nach dem Eintritt dieser Transition beschreibt die Marke (c0,e0,r0) in *in room*, dass der Experte e0 den Kunden c0 in Raum r0 berät. Über *negotion ends* verlassen c0 und e0 den Raum wieder.

Variablen:
c: Kunden **e:** Experten **r:** Räume

Abb. 17: das Modul *rooms*

In der linken Schnittstelle des Moduls liegen nur Transitionen; hingegen ist die rechte Schnittstelle des Moduls überaus heterogen: sie besteht aus zwei Plätzen, einer Transition und zwei Pfeilen.

2. Das Modul *rooms* (Abbildung 17): ein Kunde und ein Experte betreten einen Besprechungsraum; sie beginnen und beenden ihre Besprechung, und verlassen den Raum wieder. Die Komposition der beiden Module Kunden und Räume ist wiederum ein Modul (Abbildung 18).

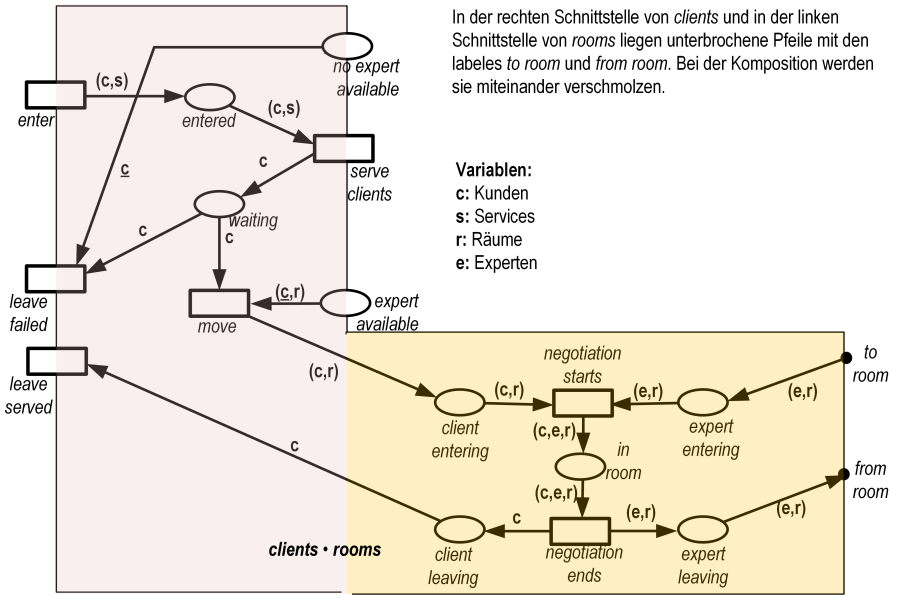
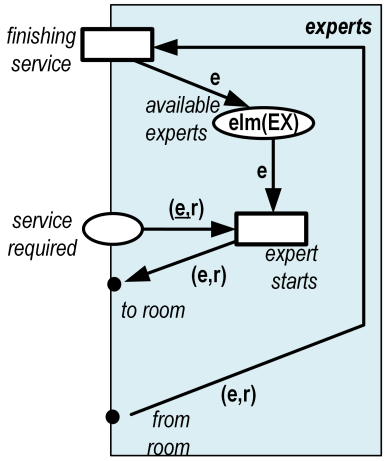


Abb. 18: das Modul *clients* • *rooms*

3. Das Modul *experts* (Abbildung 19): Dieses Modul zeigt ein wichtiges Konzept von HERAKLIT: das Konstantensymbol EX. Es wird verwendet, um den Anfangszustand des Moduls zu beschreiben. Das Modul kann ausgeführt werden, nachdem zunächst EX durch eine Menge E (von Experten) instanziiert wurde. Wie in Abschnitt 6 dargelegt, bezeichnet die Beschriftung $elm(EX)$) also die Elemente von E als einzelne Marken auf dem Platz *available experts*. Wenn nun ein Experte zu einem Raum gehen soll, erreicht ihn über *service required* eine entsprechende Nachricht, und die Transition *expert starts* tritt ein.

   Abbildung 20 zeigt das Zusammenspiel der Module *clients*, *rooms* und *experts*. Es fehlt nun noch eine Verwaltung, die den Kunden Experten und Räume zuordnet.

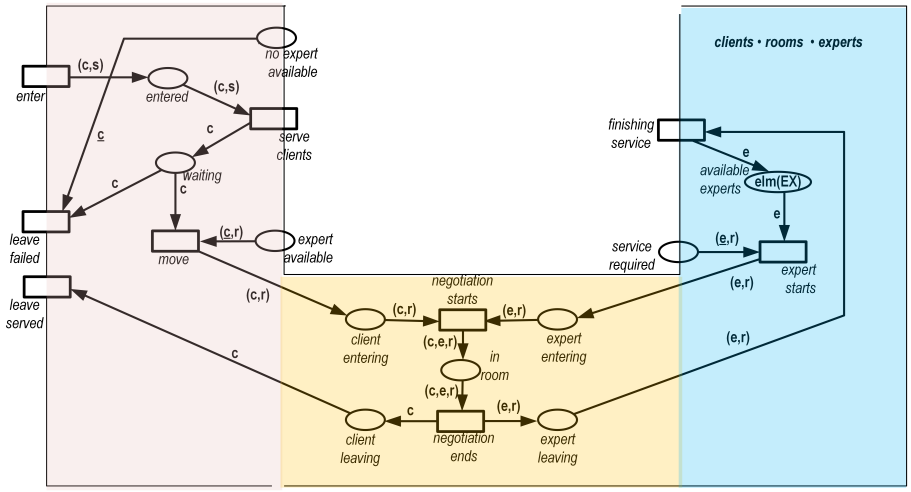
4. Das Modul *admin* (Abbildung 21): Dieses Modul verwendet drei Konstantensymbole. Im Anfangszustand wird das Konstantensymbol *adm* durch einen Kundenbetreuer (im Modell kurz: *admin*) belegt. Der Kundenbetreuer a_0 kann einen Kunden c_0 bedienen, der einen Service s_0 wünscht. Bei der Komposition der Module verschmelzen die Transition *service clients* in der rechten Schnittstelle von Kunden und in der linken Schnittstelle von *admins* zu einer Transition. Ein *engaged admin* weist den Kundenwunsch entweder über die Transition $refuse\ request$ oder nennt dem Kunden c_0 sowie einem geeigneten Experten e_0 einen Besprechungsraum r_0 über die Transition *accept request*.



Indem **EX** beispielsweise durch die Menge M = {e0, e1, e2} instanziiert wird, entstehen auf *available experts* die drei Marken e0, e1 und e2 (und nicht die Marke M). Dann tritt *expert starts* ein, wenn die Umgebung auf *service required* eine Marke (a,b) ablegt, mit a aus {e0, e1, e2} und b aus {001, 002}.

Variablen:
c: Kunden
s: Services
r: Räume
e: Experten

Abb. 19: das Modul *experts*

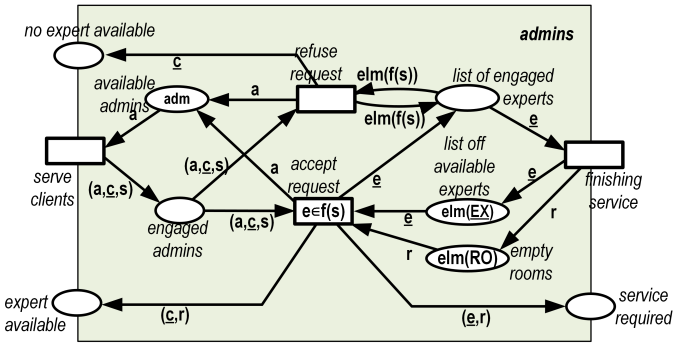


Mit einer Marke (e0,r0) auf dem Platz *service required* tritt *expert starts* ein (imit der Belegung e = e0 und r = r0), und e0 geht zum Raum r0. Dort trifft er einen Kunden c0, den er berät.

Variablen:
c: Kunden
s: Services
r: Räume
e: Experten

Abb. 20: das Modul $clients \bullet rooms \bullet experts$

Der Rest des Moduls organisiert die Kriterien für die Auswahl zwischen den beiden Transitionen $refuse\ request$ und $accept\ request$: Das Konstantensymbol RO auf dem Platz $empty\ rooms$ wird durch eine Menge von Raumnummern belegt. Jede Raumnummer ist anfangs eine Marke auf dem Platz $empty\ rooms$.



Der *available admin* a0 erfährt über den Eintritt von *serve clients*, welcher Kunde c0 welchen Service s0 nachfragt. Für einen *engaged admin* (a0, c0, s0) tritt die Transition *acceept request* ein, wenn ein Raum frei ist und die Liste verfügbarer Experten einen Experten e0

Wenn die Liste keinen solchen Experten enthält, sind alle diese Experten in der *List of engaged experts*. Dann ist die Transition refuse request aktiviert (mit a = a0, c = c0 und s = s0). Wenn ein Experte verfübar, aber kein Raum frei ist, wird gewartet, bis (über *finishing service*)

Abb. 21: das Modul *admin*

Analog dazu wird das Konstantensymbol EX im Platz *list of available experts* durch digitale Zwillinge der Experten belegt, die im Experten-Modul anfangs auf *available experts* liegen. Das garantiert beispielsweise, dass zu jedem Zwilling im Platz *list of available experts* der entsprechende Experte auch im Platz *available experts* vorkommt.

Abbildung 22 zeigt die Komposition der drei Module *clients*, *rooms* und *admin*. Das gesamte Modell eines Service-Systems ist nun die Komposition der vier Module in Abbildung 14.

9 Weitere Aspekte: Annahme von Fairness; kalte und heiße Transitionen

Die Gesamtschau auf das Schema in Abbildung 14 zeigt, dass der Kundenbetreuer einen Kunden e_0 in *entered* oder in *waiting* willkürlich oft übergehen kann, indem er unbeschränkt viele anderer Kunden (oder andere Kunden unbeschränkt oft) vorzieht, aber e_0 beständig ignoriert. Man kann nun versuchen, das mit zusätzlichen Maßnahmen zu verhindern. Im realen Leben, beispielsweise in einem Bürgeramt, installiert man zu diesem Zweck am Eingang einen Ticket-Automaten, der auf Knopfdruck nummerierte Tickets abgibt, und die vom Kundenbetreuer in der entsprechenden Reihenfolge bedient werden (Abbildung 23). Das ist ein pragmatisch vernünftiges Verfahren, löst aber das Problem nicht grundsätzlich: das Fairness-Problem wird nun auf den Zugang zum Ticketautomaten verlagert. Wenn plötzlich Tausende Bürger in das Bürgeramt wollen, kann ein einzelner Bürger ebenfalls dauerhaft am Ticket-Automaten benachteiligt werden. Ganz entsprechend verhält es sich mit digitalen Zugangsverfahren über das

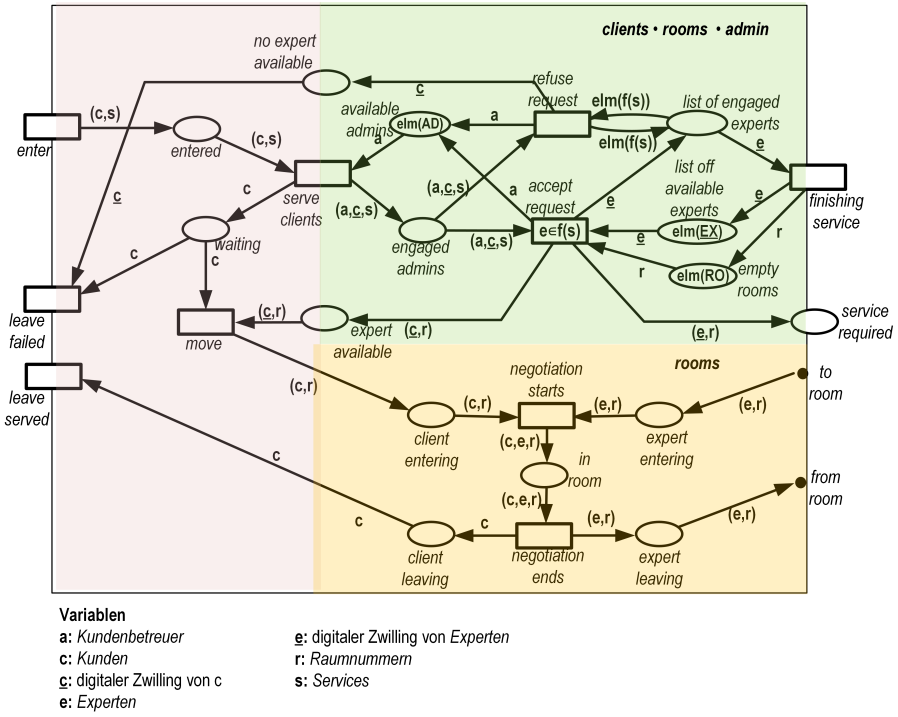


Abb. 22: das Modul *clients • rooms • admin*

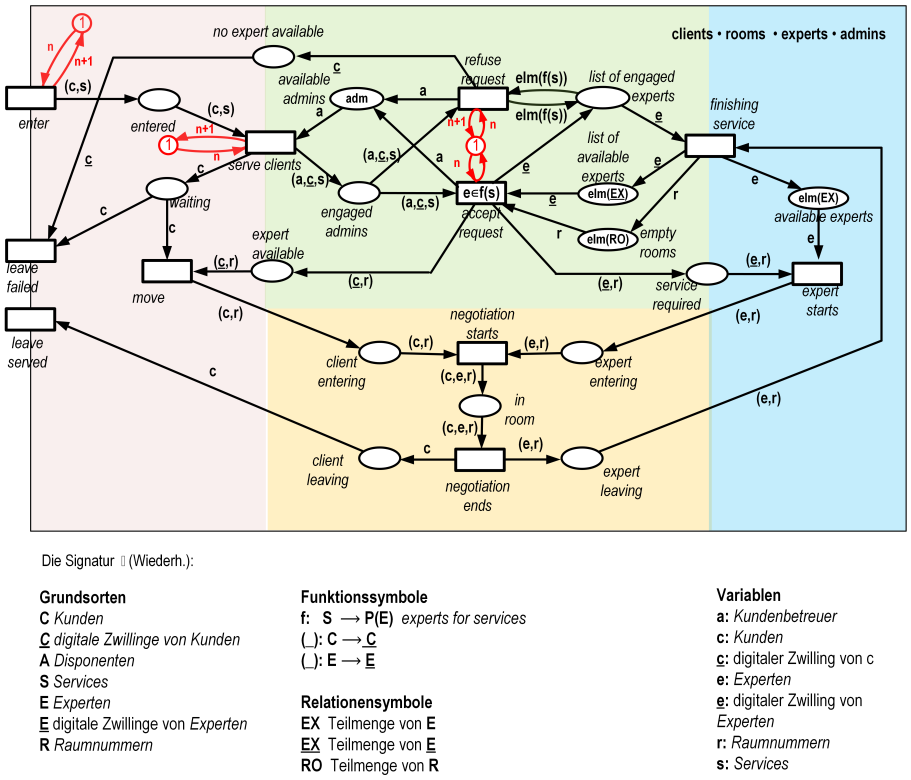
Internet. Bei zu vielen Anmeldungen brechen sie zwangsläufig zusammen. Mehr zum Problem der Annahme von Fairness findet sich im HERAKLIT-Handbuch.

Die beiden Abläufe in Abbildung 15 sind vollständig in dem Sinn, dass sie nicht weiter voranschreiten können, denn am Ende hat der Kunde das System verlassen. Vorher erwarten wir allerdings, dass das System nicht einfach in irgend einem erreichten Zustand stehen bleibt, obwohl es weitergehen könnte. Die Transition *enter* ist davon ausgenommen; sie erzeugt ja jeweils einen Ablauf. Wir charakterisieren die Transition *enter* als *kalt* und alle anderen Transitionen als *heiß*, und definieren einen Ablauf als *vollständig*, wenn er keine heiße Transition aktiviert. Die Bedeutung eines Schemas wie das in Abbildung 14 ist damit definiert als die Menge seiner vollständigen Abläufe.

Schlussbemerkung

Diese Fallstudie verwendet eine Reihe spezifischer Konzepte von HERAKLIT, insbesondere:

– heterogene Schnittstellen (Plätze, Transitionen und Pfeile in einer Schnittstelle),



Die Signatur (Wiederh.):

Grundsorten
C *Kunden*
C *digitale Zwillinge von Kunden*
A *Disponenten*
S *Services*
E *Experten*
E digitale Zwillinge von *Experten*
R *Raumnummern*

Funktionssymbole
f: **S** → **P(E)** *experts for services*
(_): C → C
(_): E → E

Relationensymbole
EX Teilmenge von **E**
EX Teilmenge von **E**
RO Teilmenge von **R**

Variablen
a: *Kundenbetreuer*
c: *Kunden*
c: digitaler Zwilling von c
e: *Experten*
e: digitaler Zwilling von *Experten*
r: *Raumnummern*
s: *Services*

Abb. 23: Modell der Service-Organisation mit Ticket-Automaten

– digitale Zwillinge,
– unterschiedliche Instanziierungen einer Signatur,
– den Umgang mit Fairness,
– kalte und heiße Transitionen.

Zweck der vorliegenden Fallstudie ist es, in die zentralen Konzepte von HERAKLIT einzuführen. Gleichwohl kann das modellierte Service-System auch Ausgangspunkt sein, um eine Reihe weiterer Aspekte eines realen Service-Systems zu erfassen:

– Im präsentierten Modell fragt ein Kunde immer nur genau einen Service nach, wenn er das Service-System betritt. Allerdings könnten Kunden auch mit mehreren Service-Wünschen das Service-System betreten.
– Das Erbringen eines Services wird als elementar verstanden. Ein realer Service wird nicht selten in verschiedenen Teilschritten erbracht, an dem auch unterschiedliche Experten beteiligt und verfügbar sein müssen.
– Das Erbringen eines Services könnte sogar nicht nur aus mehreren Teilschritten bestehen, sondern eine größere Menge von Aufgaben umfassen, deren sachlogische Abhängigkeiten über ein Petrinetz zu beschreiben wären.

- Der Kunde, der das Service-System betritt, weiß, welche Services vom System angeboten werden und welches Services-Bedürfnis er selbst hat. Dagegen muss in realen Service-Systemen der Kundenberater häufig zunächst herausfinden, welches Service-Bedürfnis ein Kunde hat. In manchen Service-Systemen wird sogar erst dann ein bestimmtes Service-Bedürfnis beim Kunden geweckt, wenn er bereits im Service-System verweilt.
- Das Service-System bietet ausschließlich standardisierte Services an. Die im Modell definierte Grundsorte kann folglich als ein standardisierter *Service-Katalog* verstanden werden. Dabei werden alle Services in einer bestimmten Qualität angeboten und es werden zwischen dem Kundenbetreuer und Kunden im Service-System keine Services ausgehandelt, die nicht in diesem Katalog verzeichnet sind.

Vor dem Hintergrund der zuvor genannten Aspekte wird deutlich, dass das hier präsentierte Modell eines Service-Systems bestimmte Annahmen über ein reales Service-System trifft, die nicht bei allen realen Service-Systemen zutreffen. Folglich bestehen vielfältige Möglichkeiten, das präsentierte Modell zu erweitern.

Literatur

1. Chesbroug, H. ; Spohrer, J. : A Research Manifesto for Service Science. In: *Communications of the ACM* 49 (2006), Nr. 7, S. 35–39
2. Fettke, P. ; Resig, W. : Modelling service-oriented systems and cloud services with Heraklit. In: *CoRR* abs/2009.14040 (2020). `http://arxiv.org/abs/2009.14040`. – presented at the 16th International Workshop on Engineering Service-Oriented Applications and Cloud Services, Heraklion, Greece, September 28-30, 2020
3. Leimeister, J. M.: *Dienstleistungsengineering und -management: Data-driven Service Innovation.* Springer, 2020
4. Reisig, W. : *Understanding Petri Nets.* Springer, 2013