Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Eine semantische Klassifikation von Systeminteraktionen

Johannes Reich[1]

Abstract: Es wird eine Klassifikation von Systeminteraktionen vorgestellt, die sich auf den Verarbeitungsmodus von Sender und Empfänger entlang dreier Dimensionen bezieht: Asynchron/synchron, deterministisch/nichtdeterministisch und zustandslos/zustandsbehaftet. Wegen der Vollständigkeit dieser Klassifikation lassen sich alle wohldefinierten Interaktionen in diese 8 Kategorien einordnen. Den Kategorien lassen sich unterschiedliche charakteristische syntaktische Mittel zuordnen.

Aus der Klassifikation folgt eine eindeutige Unterscheidung zwischen vertikaler und horizontaler Interaktion. Damit lässt sich zum einen im Bereich der Architektur- und Komponentenmodelle der formale Nachweis von Softwareschichten führen und zum anderen die Forderung motivieren, dass Komponentenmodelle für netzwerkartig operierende Applikationen Protokolldeklarationen unterstützen sollten.

Als Beispiel der Anwendung der Klassifikation werden die beiden sogenannten Architekturstile SOA und REST herangezogen. Es wird festgestellt, dass deren Grundbegriff des ”Service” nicht hinreichend klar definiert ist, dass er eine eindeutige Einordnung in die vorgestellte Klassifikation erlauben würde.

Abstract: A classification of system interactions is presented that is based on the processing mode of sender and receiver along the three dimensions: asynchronous/synchronous, deterministic/non-deterministic, and statless/statefull. Because of its completeness, every interaction can be attributed to one of the 8 categories. Different syntactical means can be attributed to each interaction class.

The classification offers a clear distinction between vertical and horizontal interaction. This allows on the one hand the formal proof of software layering in the area of architecture and component models and on the other hand motivates the requirement for component models for network-like operating applications to provide means for protocol declarations.

With respect to the so called architectural styles of SOA and REST, it is noticed that their basic notion of a ”service” is not sufficiently well defined to allow for an unambiguous attribution within the proposed classification.

Keywords: Interaktionen, synchron, asynchron, deterministisch, nicht-deterministisch, zustandsbehaftet

1 Einführung

Computersysteme interagieren auf vielfältige Weise. Dabei sind nicht alle Interaktionen von derselben Qualität, sondern es scheint verschiedene Interaktionsklassen zu geben.

Prominente Beispiele von Versuchen, Interaktionen im Bereich der Informatik zu klassifizieren sind etwa die ”Transaction Pattern” der UN/CEFACT Modelling Methodology

[1] SAP SE, Industry Standards & Open Source, Dietmar-Hopp-Allee 16, 69190 Walldorf, johannes.reich@sap.com

[UMM03] oder RosettaNet [RNI01], die durch die Theorie der Sprechakte motiviert wurden [Au62, Se69], oder die ”Transmission Primitives” [WSD01] bzw. ”Message Exchange Pattern” [WSD07a, WSD07b] aus der Welt der Web Services. Für beide Ansätze lässt sich mit Hilfe von Zustandsautomaten recht einfach zeigen, dass sie nicht gut geeignet sind, die Semantik von prozesshaften Interaktionen zu erfassen [Re08].

Insbesondere in Architekturmodellen können diese unklaren Klassifizierungen zu Unklarheiten bis hin zur Unbrauchbarkeit führen. Spätestens seit dem bekannten Open System Interconnection Modell [ITU94], wird zwischen vertikaler Interaktion zwischen Softwareschichten und der horizontaler Interaktion innerhalb einer Softwareschicht unterschieden - ohne dass klar gestellt wurde, wie sich diese beiden Interaktionsklassen formal - also genau und eindeutig - unterscheiden ließen. So redet das angesprochene OSI-Modell bezogen auf die vertikale Interaktion zutreffend von einer Verwendungsbeziehung, und bei der horizontalen Interaktion von Protokollen. Das Referenzarchitekturmodell Industrie 4.0 ([I4015], S. 43) hingegen spricht davon, dass innerhalb der Schichten eine ”hohe Kohäsion” und zwischen den Schichten eine ”lose Kopplung” herrschen soll.

Tatsächlich stellen Juha Savolainen and Varvana Myllärniemi in [SM09] fest, dass ”there is very little actual research done on layered architectures”. Diese Unklarheiten stellen aus meiner Sicht mittlerweile einen deutlichen Hemmschuh bei der technologischen Weiterentwicklung von netzwerkartig interagierenden Applikationen dar, dessen Abdruck sich auch in den Mühen, diese Weiterentwicklung systematisch durch Standardisierung zu fördern, niederschlägt, etwa im Bereich Industrie 4.0 oder dem Internet der Dinge [KLW11, BM12]. So wird im schon zitierten Ergebnisbericht der Plattform Industrie 4.0 ([I4015], S. 37) das Ziel formuliert, für Industrie 4.0-Szenarien eine formale, Computerverarbeitbare Form der Beschreibung zu entwickeln - Aber: Wie soll dies möglich sein, wenn die Unterschiede zwischen vertikaler und horizontaler Interaktion nicht formal - also genau und eindeutig - gefasst werden? Ziel dieses Artikels ist, eine einfache Klassifikation von Interaktionen aufzuzeigen, die insbesondere die beiden Klassen der vertikalen und der horizontalen Interaktion umfasst.

Ein weiterer sehr wichtiger Anwendungsfall einer Interaktionsklassifikation sind Komponentenmodelle. Mit einer vollständigen Klassifikation erhalten wir ein Kriterium, nach dem sich beurteilen lässt, ob Komponentenmodelle in diesem Maße tatsächlich vollständig sind.

Da die angesprochenen Unklarheiten meiner Ansicht nach auf einer unklaren Auffassung von der Bedeutung ausgetauschter Informationen beruhen, beschäftige ich mich im ersten Abschnitt mit der Frage, wie Informationen zu ihrer Bedeutung kommen. In den weiteren beiden Abschnitten wird die Klassifikation der Interaktionen vorgestellt. In Abschnitt 5 diskutiere ich die sogenannten Architekturstile der ”Service Orientierung” (SOA) und des ”Representational State Transfers” (REST). Der letzte Abschnitt fasst die wesentlichen Ergebnisse noch einmal zusammen.

2 Die Bedeutung von Informationen

Claude Shannon hat schon in seinem Artikel "A Mathematical Theory of Communication" ([Sh48]) darauf aufmerksam gemacht, dass die Betrachtung von Kommunikation als Informationsübertragung den Betrachter von der Bedeutung des Geschehens absehen lässt:

> "The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem."

Stattdessen wird die Betrachtung der Kommunikation zunächst auf die Frage der Reproduzierbarkeit von Zuständen reduziert, eben auf die Übertragung von "Information"[2].

Die Fragen der Semantik, also der Bedeutung der übertragenen Informationen ist damit natürlich nicht irrelevant geworden, nur weil sie durch die Theorie der Informationsübertragung ignoriert wird. Ganz im Gegenteil hat sich herausgestellt, dass sich das Problem der Informationsübertragung eher bescheiden ausnimmt gegenüber dem Problem der Verständigung.

Meiner Ansicht nach hat Claude Shannon mit seiner Trennung zwischen der Übertragung und der Verarbeitung von Informationen auch für ein adäquates Verständnis der Bedeutung von Informationen die richtige Grundlage gelegt. Eine sehr interessante und gleichwohl naheliegende Folgerung aus der scharfsinnigen Idee von Claude Shannon ist nämlich, dass sich Bedeutung materiell, also in einem physikalischen Sinn, gar nicht übertragen lässt. Stattdessen erhält Information Bedeutung schlicht durch ihre Verarbeitung, man könnte auch sagen, durch ihre Interpretation. Bedeutung wird nicht transportiert, sondern erteilt. Informationen, die nicht verarbeitet werden, sind bedeutungslos.

Die dieser Betrachtung zugrunde liegende Vorstellung von Semantik ist die einer Abbildung, der Interpretationsfunktion. Eine solche Vorstellung von Semantik kommt auch in vielen anderen wichtigen Bereichen der Informatik erfolgreich zur Anwendung. Die formale Semantik imperativer Programmiersprachen wird im Wesentlichen so verstanden. Dana Scott und Christopher Strachey ([SS71]) beschreiben etwa die Semantik eines Computerprogramms durch die Abbildung der Operationsterme auf Funktionen im mathematischen Sinne (die ihrerseits wieder entsprechende Abbildungseigenschaften von der Ein- auf die Ausgabe aufweisen), während Gordon Plotkin ([Pl81]) die Semantik eines Programms zusammen mit seinen Daten durch eine Abbildung auf ein Transitionssystem erklärt und damit ebenfalls auf die Operationen fokussiert, die ein System ausführen kann.

Als Beispiel sei die Zeichenfolge 'y=Sinus(x)' angegeben, die bei der Interpretation des Programmtextes auf ein System abgebildet wird. Dieses System bildet den Wert des Zu-

[2] Im weiteren verwende ich die Terme "Daten" und "Informationen" synonym, ebenso wie die Terme "Bedeutung" und "Semantik".

stands, der durch den Namen 'x' repräsentiert wird, in einem Zeitschritt auf einen Wert des Zustands, der durch den Namen 'y' repräsentiert wird, derart ab, dass zwischen beiden Zustandswerten eine mathematische Beziehung besteht, die der Funktionsname 'Sinus' suggeriert. Eine Bedeutung des Zustandswertes von 'x' (seine Belegung) wäre dann der anschließend errechnete Zustandswert von 'y'.

Auch in der formalen Logik wird die Semantik von Termen und Ausdrücken über eine Interpretationsfunktion erklärt, die diese Ausdrücke in der Regel rekursiv auf die Begriffe einer Metasprache abbildet. Auch in der formalen Logik ist zur Auswertung von Ausdrücken, die Variablen enthalten, eine Belegung dieser Variablen mit Werten notwendig.

Mit dieser Auffassung ist konsistent, dass allen Techniken, die die Verarbeitung von Informationen strukturieren sollen, ein "semantischer Gehalt" zugemessen wird. Das gilt sowohl für Typsysteme, in denen sich Datentypen als Paare von Daten- und Operationenmengen auffassen lassen, wie auch für Abstrakte Datentypen oder sogenannter "Objekte", die einer Menge von Daten eine feste Anzahl von Operationen zuordnen.

Und es ist ebenfalls konsistent zu dieser Auffassung, dass eines der ältesten, auf Gottfried W. Leibniz zurückgehende semantische Prinzip, dass zwei Ausdrücke die gleiche Bedeutung haben, wenn sie gegenseitig ersetzbar sind (salva veritate) mit Barbara H. Liskovs und Jeannette M. Wings "Substitutional Principle" ([LW94, LW01]) Eingang in die Informatik erhielt.

Aus dieser Auffassung leiten sich unmittelbar einige Folgerung ab. Zum einen macht diese Auffassung von Semantik keinen Unterschied zwischen einer menschlichen (oder allgemein biologischen) und einer technischen Verarbeitung. Ob mir mein Geld aufgrund meiner Eingabe am Geldautomat ausgezahlt wird oder von einem Bankangestellten ist bezüglich der Abbildung: Mein Wunsch $\rightarrow$ Geldinhalt meines Portmonaies, unerheblich. Gleichzeitig unterstreicht sie die Vielfalt unserer Bedeutungen. Sie inkludiert etwa die Fregesche Vorstellung von der Bedeutung einer Aussage als ihrem Wahrheitswert [Fr92]: Sobald sich eine Aussage auf die Wahrheitswerte "wahr" oder "falsch" abbilden lässt, ist dieser Wahrheitswert eine mögliche Bedeutung dieses Satzes. Und nicht zuletzt hat Bedeutung immer lokalen Charakter, im Gegensatz etwa zur These Hilary W. Putnams [Pu75][3].

[3] Über dieses offensichtlich auch philosophisch sehr interessante Thema ließe sich sicherlich noch deutlich mehr sagen. An dieser Stelle seien mir zwei Bemerkungen zur Rolle unseres Verstandes und der Komplementarität von Verstand und Emotionen als Fußnote erlaubt.

Selber Bedeutung erteilen und verstehen, welche Bedeutung jemand drittes erteilt hat, sind zweierlei. Mit der vorgestellten Bedeutung von Bedeutung lässt sich unser Verstand, als das menschliche Organ zur Erfassung von Bedeutung auffassen, der damit das Individuum aus seiner existenziellen Isolation treten lässt.

Fasst man Emotionen als "Zustände der Interpretation" auf, also als innere Zustände, die unsere Verarbeitung von Informationen ganz wesentlich modulieren (was sie zweifelsfrei tun), dann sind Verstand und Emotionen untrennbar miteinander verbunden. Für eine solche Sicht spricht auch die enge Verbindung von Verstand und Emotionalität in den pathologischen Störungen unseres Denkapparates, den Psychosen, die sich stärker auf die Emotionen als affektive Psychosen oder auf den Verstand selbst, als kognitive Psychosen beziehen können.

3 Klassifikation von Interaktionen entlang dreier Dimensionen

Die Verarbeitung von Informationen wird durch die Theorie der Berechenbarkeit beschrieben, deren Grundlagen von Alan Turing ([Tu36]) mit der Turingmaschine, Alonso Church ([Ch36]) mit dem λ-Kalkül und von Kurt Gödel, Herbrand, Rózsa Péter und Stephen Kleene ([Kl36]) mit der Theorie berechenbarer Funktionen gelegt wurden.

3.1 Deterministische versus nichtdeterministische Verarbeitung des Empfängers

Bei der Betrachtung interagierender berechenbarer Systeme ist allerdings zusätzlich zu der Frage, wie berechenbare Funktionen grundsätzlich aufgebaut sind, die Frage relevant, welches System welche Informationen zu welcher Zeit zur Verfügung hat und ob die in einem System A ablaufende Berechnung auch aus Sicht des Interaktionspartners B eine solche ist. Mit anderen Worten, nur weil ein System nichts anderes als Berechnungen im Sinne deterministischer Zustandsänderungen ausführt, mithin vollständig durch seinen inneren Zustand und seine Systemfunktion beschrieben werden kann, heißt das nicht, dass es sich aus der Sicht seiner Interaktionspartner als solches darstellt. Wenn die Eingabe und Ausgabe-Partner ständig wechseln, wie es in Kooperationsnetzwerken die Regel ist, dann werden die Interaktionen aus Sicht der Interaktionsteilnehmer im Wesentlichen nichtdeterministischen Charakter haben ([Re12]). Ein nichtdeterministisches Verhalten des Empfängers von Informationen meint, dass der vom Empfänger vorgenommene Zustandsübergang zwar durch die empfangenen Informationen veranlasst, aber keineswegs vollständig durch sie bestimmt wurde. Dies lässt sich von den Informationen, die in einem Kommunikationsschritt ausgetauscht werden, auf alle Informationen, die dem Sender bekannt sind, verallgemeinern.

Auch können dann selbst deterministische Systeme in solchen Interaktionsnetzwerken scheinbar spontanes Verhalten zeigen, was allein mit dem Funktionsbegriff im Sinne einer Abbildung grundsätzlich nicht möglich ist, weil der Abbildung im Falle spontanen Handelns die ”Eingabe” fehlt.

Die erste Dimension, entlang derer die Interaktion zwischen Systemen klassifiziert werden kann, bezieht sich daher auf das Verhalten des Empfängers von Informationen aus Sicht des Senders: Ein Empfänger verhält sich entweder deterministisch oder nichtdeterministisch.

3.2 Synchrone versus asynchrone Verarbeitung des Senders

Bei vernetzten Interaktionen kommen die Eingabedaten in der Regel der Reihe nach von verschiedenen Sendern, mit der Folge, dass die interagierenden Systeme keine starren hierarchischen Systemverbünde eingehen.

Die Theorie der berechenbaren Funktionen beschreibt wie eine übergeordnete Funktion aus untergeordneten (wiederum berechenbaren) Funktionen geformt werden kann. Entsprechend lässt sich jeder Berechnung ein Zeitschritt zuordnen, der für die Berechnung

”abgewartet” werden muss. Die Berechnung der übergeordneten Funktion setzt voraus, dass die Berechnung der untergeordneten Funktionen abgeschlossen wurde. Diese Eigenheit ist die inhaltliche Grundlage der Bildung hierarchischer Systemverbünde. Geht Interaktion nun ohne die Bildung von übergeordneter Funktionalität einher, entfällt auch die Notwendigkeit auf Berechnungsschritte anderer Systeme warten zu müssen.

Die zweite Dimension, entlang derer die Interaktion zwischen Systemen klassifiziert werden kann, bezieht sich daher auf das Verhalten des Senders von Informationen aus Sicht des Empfänger: Ein Sender wartet entweder auf das Ergebnis der Berechung des Empfängers (synchrones Verhalten) oder eben nicht (asynchrones Verhalten).

3.3 Zustandslose versus zustandsbehaftete Verarbeitung des Empfängers

Ein Empfänger kann seine Verarbeitung der erhaltenen Informationen (abgesehen von eigenen Informationen) alleine auf diese stützen oder zusätzlich auf in früheren Interaktionen erhaltene Informationen. Den ersten Fall nennt man zustandslose Interaktion, den zweiten Fall zustandsbehaftet.

Dabei ist zwischen dem einzelnen Kommunikationsvorgang und der Interaktion zu unterscheiden. So ist etwa eine Telefon- oder eine HTTP-Verbindung zustandslos. Beide eignen sich aber für die Abwicklung zustandsbehafteter Interaktionen wie Bestellungen, etc., bei der sich jeder Interaktionsschritt auf die vorangegangenen beziehen kann.

4 Acht Kategorien der Interaktion

Die 8 Fälle, die aus der Kombination der drei Dimension entstehen, sind in Tabelle 1 zusammengefasst. Die wesentliche Behauptung ist nun, dass für jeden dieser 8 Fälle bestimmte syntaktische Mittel am besten geeignet sind, um die jeweils zugrunde liegende Interaktionssemantik zu repräsentieren.

Der wichtigste Fall einer **synchronen, deterministischen** Interaktion in der Informatik ist der Aufruf von Funktionen[4]. Die aufgerufene Funktion liefert ihren Wert im Verarbeitungskontext ab, der solange wartet, bis ihre Berechnung fertig ist. Die abgelieferten Informationen erhalten durch die deterministische Interaktion einen imperativen Charakter. D.h. sie beziehen sich auf das was gleich in der Zukunft ganz sicher passieren wird. Entsprechend werden in der Regel für die Beschreibung dieser Interaktionsform Verben in der Befehlsform verwendet. Besitzt der Empfänger der ursprünglich abgelieferten Information keinen eigenen Zustand, so ist es ein einfacher Funktionsaufruf, wie etwa die Berechung eines Sinus: y = sin(x). Für dem Umgang mit zustandsbehafteter Funktionalität hat sich die Objektorientierung herausgebildet, in der die sogenannten Objekte den Zustand kapseln. Etwa bei einem Iterator, der mit einem Startwert initialisiert wird und dessen getNext()-Methode jedesmal den nächsten Iterationswert liefert: iterator.getNext().

[4] Synonyme sind ”Methoden”, ”Operationen” oder ”Prozeduren”

Sender	Empfänger		Syntakt. Mittel zur Beschreibung	Beispiel
	Det.	Zust.		
s	d	zl	Funktionsaufruf	y = sin(x)
s	d	zb	Aufruf einer Objektmethode	y = iterator.getNext()
s	n	zl	Funktionsaufruf mit Ausnahmen die Zufälle abfangen	createConnection(String address) throw TimeoutException
s	n	zb	Aufruf einer Funktion/ Objektmethode mit Ausnahmen	writeTextToFile(File file, String text) throw NoSpaceOnDevice
a	d	zl	Funktionaufruf ohne Rückgabewert/Protokoll	Steuerung eines stabilen Systems, etwa Ausrichtung einer Videokamera oder einer Heizung.
a	d	zb	-	Problematisch, da Asynchronizität des Senders und Zustandsbehaftung des Verhaltens des Empfängers zu Inkonsistenzen führen.
a	n	zl	-	Problematisch, da ohne Zustandsbehaftung des Empfängers sich für die Interaktion kein prozessualer Fortschritt ergibt.
a	n	zb	Protokoll	Netzwerkartige Interaktionen, in denen die beteiligten Systeme wenigstens Eingaben aus zwei verschiedenen Interaktionen miteinander zu koordinieren haben.

Tab. 1: Klassifikation von Interaktionen anhand der Sender- und Empfängercharakteristiken. Ein Sender kann synchron (s) oder asynchron (a) arbeiten, ein Empfänger entweder deterministisch (d) oder nichtdeterministisch (n), sowie zustandslos (zl) oder zustandsbehaftet (zb).

Ein wichtiger Fall **synchroner, nichtdeterministischer** Interaktionen ist der Funktionsaufruf mit Ausnahmen. Dabei wird die im nichtdeterministischen Fall multiple Zuordnung der Eingabe- zu den Ausgabewerten in eine ”gewünschte” und die restlichen ”unerwünschten” Zuordnungen - die ”Ausnahmen” - partitioniert. Ein Beispiel ist etwa das Schreiben eines Textes in ein File. Dies geht in aller Regel gut, es sei denn, die Platte ist voll. D.h. de fakto ist das Verhalten des Filesystems nichtdeterministisch bezogen auf den Befehl, einen Text einem File hinzuzufügen: Es kann funktionieren oder auch nicht. Aber die Funktion ist die Regel und die Störung die Ausnahme. Ich nenne dieses Verhalten quasideterministisch. Das Schreiben eines Files ist zustandsbehaftet, weil es mit der Zeit dazu führt, dass die Platte tatsächlich voll sein wird. Der innere Zustand des Befehlsempfängers kann also dazu führen, dass sich sein Verhalten irgendwann ändert. Im nichtzustandsbehafteten Fall kann kein innerer Zustand den Nichtdeterminismus beeinflussen und entsprechend ist die Ausnahme aus Sicht des Aufrufers Zufall. Ein Beispiel ist etwa die Herstellung einer Kommunikationsverbindung, die an vielen externen Faktoren scheitern kann.

Ein typisches Beispiel von **asynchroner, deterministischer** Interaktion ist die Steuerung (”open loop control”) eines stabilen Systems, etwa einer Heizung. Eine Heizung, beste-

hend aus Brenner, Temperaturfühler, Heizkörper und Regler, lässt sich über ein Temperatursignal steuern. Dabei braucht es eine gewisse Zeit, bis die gewünschte Soll-Temperatur erreicht wird. Um eine neue Soll-Temperatur zu setzen, muss man nicht warten, bis die alte erreicht wurde. Wichtig dabei ist, dass jeder Befehl über kurz oder lang zu einem vom vorhergehenden Zustand unabhängigen Effekt im Befehlsempfänger führt, er also idempotent ist. Wäre dies nicht so, würde die Asynchronizität des Senders und die Zustandsbehaftung des Empfängers zu Inkonsistenzen führen. Zur Beschreibung dieser Form der Interaktion sind einerseits Funktionsaufrufe ohne Rückgabewert geeignet, als auch Dokumentenorientierte Protokolle.

Asynchrone, nichtdeterministische Interaktionen sind die bestimmenden Interaktionsformen in netzwerkartigen Interaktionen. Dort ist die Hauptaufgabe von (deterministischen) Sytemen die Koordination aller Interaktionen, in die sie eingebunden sind [Re12]. Dabei darf in der Regel die Interaktion nicht die Aktion der Systeme bestimmen. Stattdessen ist die lose Beziehung zwischen Interaktion und Aktion Voraussetzung für die Skalierung von Interaktionsnetzwerken, weswegen hier zu Recht von ”loser Kopplung” der Systeme gesprochen werden kann.

Diese Interaktionsform ist grundsätzlich zustandsbehaftet. Die abgelieferten Informationen beziehen sich zunächst auf die Vergangenheit, als sie einen Zustandsübergang des sendenden Systems dokumentieren. In einer Interaktion, die gewisse Anforderungen der Konsistenz erfüllt, erfolgt diese Dokumentation in einer Weise die ”verständlich” für den Empfänger ist, d.h. dass der Empfänger sie vorbehaltlich seiner eigenen Entscheidungen passend verarbeiten, also einen passenden eigenen Zustandsübergang durchführen kann. Ohne Zustandsbehaftung des Empfängers ergäbe sich für die Interaktion kein prozessualer Fortschritt.

Die Beschreibung dieser Interaktionsform geschieht durch Protokolle[5] (z.B. [Ho91]). Man kann auch nichtdeterministische Zustandsübergänge mit Funktionen beschreiben. Da sich in dieser Interaktionsform der Nichtdeterminismus allerdings nicht absondern lässt, erhält man dann als Ergebnis keine einzelnen Zustandswerte, sondern Mengen von Zustandswerten. Damit wird die Beschreibung von zyklischen Interaktionsketten recht schnell sehr komplex. Dieses Problem wird mit dem Protokoll-Ansatz vermieden.

Die ausgetauschten Informationen haben, wie gesagt, Dokumenten-Charakter. Tatsächlich werden sie von Sender und Empfänger häufig völlig unterschiedlich verarbeitet, besitzen also für beide verschiedene Bedeutungen. So ist etwa ein Preis in einer Geschäftstransaktion für den Käufer die Menge Geld, die er bezahlen muss und für den Verkäufer ist es die Menge Geld, die er erhält. Die ausgetauschten Informationen werden in der Regel mit Hauptwörtern bezeichnet. Typische Beispiele finden sich etwa in Geschäftsinteraktionen: Kaufauftrag, Rechnung, Lieferavis, etc.

[5] Synonyme für ”Protokoll” sind ”Choreografie” oder auch ”Kollaboration” (z.B. [OM13a]).

4.1 Kombination mit der ”anderen” Richtung

Interaktion ist in der Regel nicht einseitig, stattdessen wechseln die Systeme oft die Rollen von Sender und Empfänger. Die Klassen lassen sich symmetrisch oder asymmetrisch kombinieren:

Die symmetrische Kombination (quasi-)deterministischer Interaktionsklassen führt zur rekursiven Supersystembildung [Re10]. Die entstehende Systemfunktion erstreckt sich über die gesamte deterministische Kette. Diese enge Beziehung nenne ich **”Symbiose”**. Sie sollte für die Interaktion von verschiedenen Komponenten möglichst vermieden werden.

Die asymmetrische Kombination aus einerseits synchroner und (quasi-)deterministischer und andererseits asynchroner, nichtdeterministischer, zustandsbehafteter Klasse nenne ich **vertikale Interaktion** oder **”Verwendung”**. Sie beinhaltet zum einen, dass ein per Funktionsaufruf aktivierter Kontext sein Berechnungsergebnis trivialerweise zurücksenden darf. Und es bedeutet weiterhin, dass er sich spontan melden kann, ohne allerdings den Kontext zu kennen, an den er die Indikation seiner Zustandsänderung abliefert, d.h. er kann einen generischen, nicht-Empfänger-spezifischen Event absetzen, etwa ”Tank ist voll”. Dies ist insbesondere bei von mehreren Verwendern benutzen Systemen wichtig, da ein Verwender den Zustand des verwendeten Systems derart verändert haben kann, dass dies für alle potentiellen Verwender interessant sein könnte.

Die symmetrische Kombination der asynchronen, nichtdeterministischen, zustandsbehafteten Interaktionsklasse nenne ich **horizontale Interaktion**. Hier senden sich die Systeme gegenseitig Dokumente im Sinne Empfänger-spezifischer Events. Der ”lose Charakter” der wechselseitigen semantischen Beziehung spiegelt sich dann in der Tatsache wider, dass für die wechselseitige Interaktionssemantik nur die Projektion der Verarbeitung auf die Interaktion relevant ist [Re12].

5 Bemerkungen zur SOA und dem REST

Die beiden von den Proponenten sogenannten Architekturstile der ”Service orientierten Architektur” (SOA) [OA06] und des ”Representational State Transfer” (REST) [Fi00, RR07] haben recht weite Verbreitung gefunden, weswegen ich sie hier in Bezug auf die eingeführte Klassifikation diskutiere.

5.1 Service orientierte Architektur

Die Idee der SOA geht auf R.W. Schulte und Y. V. Natis der Gartner Group zurück. [SN96]. OASIS definiert in ihrem Referenz Modell für die Service orientierte Architektur, Version 1.0 [OA06], eine SOA recht allgemein als ”paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.”. Ein ”Service” wird definiert als “The performance of work (a function) by one for another.” und als ”mechanism by which needs and capabilities are brought together”.

Tatsächlich wurde die SOA zwischenzeitlich von vielen großen Softwareherstellern im Umfeld der Unternehmenssoftware propagiert, allerdings ohne den erhofften Erfolg speziell bei der Vernetzung von Unternehmensapplikationen.

Die WSDL 1.1 Spezifikation [WSD01] definierte noch, gestützt auf das Muster der gesendeten Nachrichten, vier ”transmission primitives”, die ein sogenannter ”endpoint” unterstützen soll: ”One-way” (der ”endpoint” empfängt eine Nachricht), ”Request-response” (der ”endpoint” empfängt eine Nachricht und sendet eine ”korrelierte” Nachricht zurück), ”Solicit-response” (der ”endpoint” sendet eine Nachricht und erhält eine ”korrelierte” Nachricht zurück) und ”Notification” (der ”endpoint” sendet eine Nachricht). WSDL bezeichnet diese ”transmission primitives” als ”operations”. Damit wird die Semantik dieser ”operations” nicht über ihre Abbildungseigenschaften definiert.

WSDL 2.0 [WSD07a] spricht in Abschnitt 2.2.1 von einer ”Interface component” als einer ”sequences of messages that a service sends and/or receives”. Nach WSDL 2.0 ist eine ”operation” eine ”interaction with the service consisting of a set of (ordinary and fault) messages exchanged between the service and the other parties involved in the interaction”. Auch hier wird die Semantik einer ”operation” nicht über ihre Abbildungseigenschaften definiert.

Die BPMN-Version 2.0.2 [OM13b] bezieht sich formal auf die WSDL 2.0 Spezifikation (wenn auch nur als nichtnormative Referenz) und so ist es keine Überraschung, dass in Abschnitt 8.5.3 nur festgestellt wird: ”An Operation defines Messages that are consumed and, optionally, produced when the Operation is called.” - Wieder keine Definition der Semantik einer ”operation” über ihre Abbildungseigenschaft.

Es lässt sich feststellen, dass die in diesem Artikel vertretene Anschauung von Semantik in der SOA nicht zur Anwendung gekommen ist, ohne dass ein klares alternatives Modell erkennbar wäre.

Die Semantik eines SOA-”Service” könnte ein Funktionsaufruf im Sinne einer synchronen (quasi-) deterministischen Interaktion sein, dessen Semantik dann nach Dana Scott und Christopher Strachey denotational oder entlang der Theorie von Gordon Plotkin operational festgelegt wäre. Wegen des funktionalen Charakters lassen sich dann die Sende- und Empfangsfunktionalität und damit die Erstellung und die Verarbeitung der serialisierten Daten hinter dem Interface verstecken (mit Ausnahme der durch den Transport zusätzlich möglichen Ausnahmen).

Die Semantik eines SOA-”Service” könnte aber auch der Versand eines Dokuments im Sinne einer asynchronen, nichtdeterministischen Interaktion sein, bei der die einzige vom Sender kontrollierte Funktion des Empfängers die des Empfangens ist, die für alle Dokumente aber gleich sein kann. Damit wäre die Semantik aller vom Sender kontrollierten ”SendDataXYZ()” Operationen nach der in diesem Artikel vertetenen Auffassung identisch. Die Semantik der Dokumente ergibt sich hingegen aus den unter ihrer Beteiligung erfolgenden Zustandsübergänge des Empfängers. Da diese nicht Teil der Empfangsoperationen sind, wird die Semantik der Dokumente auch nicht von den Empfangsoperationen berührt.

Die semantische Vagheit trug wohl mit zu Aussagen bei wie ”Everything is a service” (etwa [GI00]) bei und könnte einer der wichtigsten Gründe sein, warum die Konzepte im Umfeld der SOA insgesamt recht komplex geworden sind, häufigen Änderungen unterworfen waren und trotz der Marktmacht der beteiligten Unternehmen nicht den erhofften Durchbruch in der Kommunikation zwischen Unternehmensanwendungen brachten.

5.2 Representational State Transfer

Der sogenannte ”Representational State Transfer” (REST) wird mittlerweile als Alternative zu SOAP und WSDL positioniert, um die Interaktion verteilter Anwendungen im Internet zu beschreiben. Er geht auf Arbeiten von Roy Fielding ([Fi00]) zurück.

Roy Fielding hatte richtig erkannt ([Fi00], S.80), dass das Design Prinzip des Internets das der zustandslosen Dokumentenübertragung war, und versucht, dieses Prinzip auf die Interaktion von vernetzten Applikationen zu übertragen. Entsprechend bezieht sich der REST nicht unmittelbar auf die Operationen eventuell relevanter Anwendungsobjekte (die als Ressourcen bezeichnet werden), sondern allein auf die Manipulation der zu übertragenden Daten mittels der dem HTTP-Protokoll [Fi99] entlehnten Operationen.

Ein Rest-Dienst soll den folgenden 3 Prinzipien genügen (z.B. [RR07]):

- Adressierbarkeit: Jede Ressource wird durch einen eindeutigen URI benamt.
- Zustandslose Kommunikation: Eine REST-Nachricht soll alle Informationen enthalten, die für den jeweilig anschließenden Verarbeitungsschritt des Empfängers notwendig ist.
- Idempotenz: die aufgerufenen Methoden sollen bei jedem Aufruf denselben Effekt haben, so wie es die HTTP-Spzifikation ([Fi99], Abschnitt 9.1.2) für die Operationen GET, HEAD, PUT und DELETE fordert.

Auch bezüglich des REST-Ansatzes lässt sich sagen, dass die Semantik eines REST-Services bezüglich der Anwendungsschicht nicht definiert ist, in diesem Fall allerdings absichtlich. Genau genommen definiert REST nur eine Briefkastensystematik. Im Gegensatz zum traditionellen Briefkasten, der, bis auf das Abliefern der Post, in der Hoheit des Empfängers steht, wird nun diese Hoheit teilweise an den Sender übergeben.

Nimmt man die REST-Prinzipien ernst, dann hat dies insbesondere für netzwerkartig interagierende Applikationen recht merkwürdige Forderungen zur Folge: Die Ablieferung eines Kaufauftrags muss jederzeit wiederholt (idempotentes PUT) und jederzeit widerrufen werden können (idempotentes DELETE). Und anstelle des bewährten einen Briefkasten für alle abgesendeten Dokumente erhält man eine Vielzahl von verschiedenen Briefkästen, für jede sogenannte ”Ressource” einen eigenen. Auch dürfte etwa eine Überweisung keine Referenz auf den Kaufauftrag oder eine (Teil-) Lieferung enthalten, da die Interaktionen ja zustandslos sein sollen.

Damit ist REST in gewissem Sinn ein Zwitter, der nur Teile einer Objekt-Zugriffsschicht in die Administrationsschicht übernommen hat. Entsprechend gehen wichtige Eigenschaften für die Beschreibung netzwerkartiger Interaktionen verloren und für die Beschreibung von Funktionsaufrufen sind weiterhin wesentliche Ergänzungen notwendig, wie das OData-Protokoll zeigt [ODa13].

6 Abschließende Bemerkungen

Die vorgestellte Klassifikation entlang der Sender-bezogenen Synchronizität und der Empfänger-bezogenen Determiniertheit und Zustandsbehaftung ist vollständig. D.h. jede wohldefinierte Interaktion lässt sich entsprechend zuordnen.

Die in der Informatik sehr verbreitete Vorstellung, dass die Interaktion von Systemen allein durch benamte Operationen zu beschreiben ist, ist damit unzureichend. Sie liegt bemerkenswerter Weise etwa auch den Prozesskalkülen von Charles H. Hoare [Ho04] als auch von Robin Milner [Mi89, MPW92] zugrunde. Tatsächlich impliziert die in diesem Artikel vertretene Auffassung von Semantik unmittelbar, dass die Angabe von Operationen bzw. sogar ihre feste Verküpfung mit Daten im Sinne der Objektorientierung eine sehr starke semantische Festlegung darstellt. Diese Festlegung ist in netzwerkartigen Interaktionen ungünstig, da ihre Flexibilität ganz wesentlich auf der Interpretationshoheit der Informationsempfänger beruhen.

Ein wesentliches Ergebnis dieses Artikels ist ein klares und einfaches Kriterium, um horizontale und vertikale Interaktionen eindeutig zu unterscheiden. Insbesondere ist das Absetzen generischer Events kompatibel zur Steuerung zustandsbehaftete Objekte mit traditionellem funktionalen APIs. Hier wäre denkbar, dass Komponentenmodelle das State-Pattern [Ga95] deklarativ unterstützen und so deutlich machen, welche Verhaltensänderung gegenüber den Objekt-Verwendern per Event generisch signalisiert werden kann.

Es folgt unmittelbar, dass Komponentenmodelle, die keine syntaktischen Mittel zur Verfügung stellen, um Protokolle zu beschreiben, für Komponenten, die in Interaktionsnetzwerken Verwendung finden sollen, ebenfalls unzureichend sind. Ivica Crnkovic, Severine Sentilles, Aneta Vulgarakis und Michel R. V. Chaudron [Cr11] unterscheiden in ihrem Überblick über Komponentenmodelle zwischen ”Operation based” und ”port based” Interface-Unterstützung und zeigen damit auf, dass viele wichtige Komponentenmodelle bisher tatsächlich keine Protokolldeklarationen unterstützen.

Komponentenmodelle, die nur prozedurale oder objektorientierte Mittel zur Verfügung stellen, eignen sich daher nur zur Beschreibung von zusammengesetzten Systemen, die in einer hierarchischen Beziehung stehen. Und ohne die Möglichkeit der Beschreibung generischer Eventkonsumption der Verwender auch dazu nur eingeschränkt.

Bei zyklischen Abhängigkeiten von Komponenten, die durch funktionale Schnittstellen beschrieben werden, entstehen rekursive Funktionen über Komponenten hinweg, die das Design der Komponenten sehr stark wechselseitig - symbiotisch - abhängig machen. Das heißt nicht, dass sich mit prozeduralen oder objektorientierten Mitteln keine nichtdetermi-

nistischen Beziehungen zwischen Komponenten aufbauen ließen, sondern nur, dass man diese Beziehungen mit den syntaktischen Mitteln des Komponentenmodells dann nicht adäquat beschreiben kann. Ein gutes Beispiel ist etwa die sogenannte AJAX-Architektur für Web-Applikationen, bei der über die Beziehung zwischen dem Web-Client und dem Server nur soviel deklarativ bekannt ist, als sie über HTTP kommunizieren. Diese Unterspezifikation des Verhältnisses der beiden Systeme stellt in einer Umgebung, in der etwa die Laufzeitumgebung des Web-Client als nicht-vertrauenswürdig eingestuft werden muss, ein enormes Sicherheitsrisiko dar.

Ohne eine deklarative Unterscheidung horizontaler und vertikaler Interaktion ist auch jede formale Analyse zyklischer Abhängigkeiten auf Komponentenebene nur begrenzt aussagefähig.

Dieses Defizit verbreiteter Komponentenmodelle ist angesichts der Tatsache, dass ab einer bestimmten Komplexität der Komponenten diese in der Regel in nichtdeterministische Interaktionen eingebunden sind, erstaunlich. Protokolle ermöglichen ganz wesentlich "Programming in the Large", um die Terminologie von Frank DeRemer and Hans Kron aufzugreifen [DK75, Si04].

Danksagung: Mein besonderer Dank geht an Christoph Rechsteiner. Die vorgestellte Klassifizierung entstand in einer ersten Form in einer gemeinsamen Diskussion zwischen uns beiden im Herbst 2014 im Zusammenhang unserer Arbeiten im Umfeld der Standardisierung zu Industrie 4.0.

Literatur

[Au62] Austin, John Langshaw: How to Do Things with Words. Cambridge (Mass.), 1962.

[BM12] BMBF, Referat Grundsatzfragen der Innovationspolitik: . Bericht der Bundesregierung, Zukunftsprojekte der Hightech-Strategie (HTS-Aktionsplan), 2012.

[Ch36] Church, Alonzo: An unsolvable problem of elementary number theory. American Journal of Mathematics, 58:345 – 363, 1936.

[Cr11] Crnkovic, Ivica; Sentilles, Séverine; Vulgarakis, Aneta; Chaudron, Michel R. V.: A classification framework for software component models. Software Engineering, IEEE Transactions on, 37(5):593–615, 2011.

[DK75] DeRemer, Frank; Kron, Hans: Programming-in-the Large Versus Programming-in-the-small. SIGPLAN Not., 10(6):114–121, April 1975.

[Fi99] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T.: , RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. `http://tools.ietf.org/html/rfc2616`, 1999. Aufruf 2015-01-17, überholt seit 2014 durch RFCs 7230, 7231, 7232, 7233, 7234, and 7235.

[Fi00] Fielding, Roy: Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, 2000.

[Fr92] Frege, Gotlob: Über Sinn und Bedeutung. Zeitschrift für Philosophie und philosophische Kritik, 100:25–50, 1892.

[Ga95] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1. Auflage, 1995.

[GI00] Gottschalk, Karl; IBM Web Services Architecture Team: , Web Services architecture overview. http://www.ibm.com/developerworks/webservices/library/w-ovr/, 2000. Augerufen am 2015-01-17.

[Ho91] Holzmann, Gerard J.: Design and validation of computer protocols. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[Ho04] Hoare, Charles.A.R.: Communicating Sequential Processes. Prentice Hall, 1985/2004.

[I4015] Umsetzungsstrategie Industrie 4.0, Ergebnisbericht der Plattform Industrie 4.0. http://www.plattform-i40.de/sites/default/files/150410_Umsetzungsstrategie.pdf, 2015.

[ITU94] Information Technology - Open Systems Interconnection – Basic Reference Model. http://handle.itu.int/11.1002/1000/2820, 1994. Früher "CCITT Recommendation", Identischer Standard: ISO/IEC 7498-1:1994 (Common).

[Kl36] Kleene, Stephen: General recursive functions of natural numbers. Mathematische Annalen, 112(5):727–742, 1936.

[KLW11] Kagermann, Henning; Lukas, Wolf-Dieter; Wahlster, Wolfgang: Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. VDI Nachrichten, 13, April 2011.

[LW94] Liskov, Barbara H.; Wing, Jeannette M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst., 16(6):1811–1841, 1994.

[LW01] Liskov, Barbara H.; Wing, Jeannette M.: Behavioural subtyping using invariants and constraints. In (Bowman, Howard; Derrick, John, Hrsg.): Formal methods for distributed processing: a survey of object-oriented approaches, S. 254–280. Cambridge University Press, New York, NY, USA, 2001.

[Mi89] Milner, Robin: Communication and Concurrency. Prentice Hall, 1989.

[MPW92] Milner, R.; Parrow, J.; Walker, D.: A calculus of mobile processes (parts I and II). Information and Computation, 100(1):1–77, 1992.

[OA06] OASIS: , Reference Model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/, 2006. Aufgerufen am 2015-01-17.

[ODa13] Open Data Protokoll, Version 4.0. http://www.odata.org/documentation/odata-version-4-0/, 2013. Aufgerufen am 2015-01-17.

[OM13a] OMG: , Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0.2, 2013. Accessed 2015-01-17, also published as ISO/IEC 19510:2013.

[OM13b] OMG: , Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0.2, 2013. Aufgerufen am 2015-01-17, auch veröffentlicht unter ISO/IEC 19510:2013.

[Pl81] Plotkin, Gerald D.: A structural approach to operational semantics. Bericht, Aarhus University Computer Science Department, 1981. DAIMI FN-19.

[Pu75] Putnam, Hilary: The meaning of meaning. In (Gunderson, Keith, Hrsg.): Language, Mind, and Knowledge. University of Minesota Press, Mineapolis, USA, 1975. German translation: "Die Bedeutung von Bedeutung", 2. Ed. printed by Klostermann Verlag 1990, Frankfurt a. Main, Deutschland.

[Re08] Reich, Johannes: Modelling the Interaction of Distributed Systems as Protocols. In: MCETECH. IEEE Computer Society, Los Alamitos, CA, USA, S. 16–24, 2008.

[Re10] Reich, Johannes: Finite System Composition and Interaction. In (Fähnrich, Klaus-Peter; Franczyk, Bogdan, Hrsg.): 40. GI Jahrestagung (2). Jgg. 176 in LNI. GI, S. 603, 2010.

[Re12] Reich, Johannes: Processes, roles and their interactions. In (Reich, Johannes; Finkbeiner, Bernd, Hrsg.): Proceedings Second International Workshop on Interactions, Games and Protocols, IWIGP 2012, Tallinn, Estonia, 25th March 2012. Jgg. 78 in EPTCS, S. 24–38, 2012.

[RNI01] RosettaNet Implementation Framework: Core Specification, V02.00.00, 2001. http://xml.coverpages.org/RNIF-Spec020000.pdf, aufgerufen am 2007-3-11.

[RR07] Richardson, Leonard; Ruby, Sam: RESTful Web Services, Web services for the real world. O'Reilly Media, 2007.

[Se69] Searle, John: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, 1969.

[Sh48] Shannon, Claude E.: A Mathematical Theory of Information. Bell System Technical Journal, 27:379–423, 623–656, 1948.

[Si04] Singh, Munindar P; Chopra, Amit K; Desai, Nirmit; Mallya, Ashok U: Protocols for processes: programming in the large for open systems. ACM Sigplan Notices, 39(12):73–83, 2004.

[SM09] Savolainen, Juha; Myllärniemi, Varvana: Layered architecture revisited - Comparison of research and practice. In: Joint Working IEEE/IFIP Conference on Software Architecture 2009 and European Conference on Software Architecture 2009, WICSA/ECSA 2009, Cambridge, UK, 14-17 September 2009. IEEE, S. 317–320, 2009.

[SN96] Schulte, Roy W.; Natis, Yefim V.: "Service-Oriented" Architectures, Part 1 and 2. SSA Research Notes SPA-401-068, -069, Gartner Group, 1996.

[SS71] Scott, Dana; Strachey, Christopher: Toward a mathematical semantics for computer languages. In (Fox, J., Hrsg.): Proc. Symp. Computers and Automata. Poytechnic Inst. of Brooklyn Press, 1971. Also Technical Monograph PRG-6, Programming Research Group, Oxford University.

[Tu36] Turing, Alan M.: On Computable Numbers, With an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, Series 2, 42:230–265, 1936.

[UMM03] UN/CEFACT Modelling Methodology (UMM) User Guide, V20030922, 2003.

[WSD01] Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001. Aufgerufen am 2015-01-17.

[WSD07a] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/wsdl20/`, 2007. Aufgerufen am 2015-01-17.

[WSD07b] Web Services Description Language (WSDL) Version 2.0: Additional MEPs. `http://www.w3.org/TR/wsdl20-additional-meps`, 2007. Aufgerufen am 2015-04-15.