Verwirrende Informatik II – Interaktion und Koordination von Systemen

Johannes Reich, johannes.reich@sap.com

6. Juli 2022

Zusammenfassung

Dieser Artikel ist der zweite einer Viererreihe, die eine Diskussion über die Grundlagen der Informatik anregen soll. In ihm wird die Interaktion zwischen Systemen und die Koordination innerhalb von Systemen mit Hilfe des Kompositionskonzeptes thematisiert.

Dabei wird, basierend auf zwei unterschiedlichen Kompositionen, eine strukturelle von einer interaktionsorientierten Perspektive unterschieden. Einerseits die hierarchische, homogene Komposition, die Systeme als Ganzes erfasst und zur Bildung von Supersystemen und damit zu einer strukturellen Teil-Ganzes-Relation führt. Sie basiert auf den Kompositionsregeln berechenbarer Funktionen. Andererseits eine inhomogene Komposition, die Systeme nur partiell im Sinne einer Projektion erfasst, die "Rolle" genannt wird und zu vergleichsweise loser Kopplung in Form von Protokollen führt.

Diese beiden Kompositionsklassen sind die Basis für ein zweidimensionales Strukturmodell einer IT-Applikation entlang der grundsätzlichen Idee des OSI-Modells, in dem die Vertikale die Teil-Ganzes-Relation und die Horizontale die Interaktionsrelation repräsentiert.

Die Nichtdeterminiertheit der Rollen ist eine notwendige Voraussetzung zum einen für eine weitere Komposition, nämlich mehrerer Rollen eines Systems zu koordinieren – das konstruktive Gegenstück zur Projektion eines Systems auf seine Interaktion. Und zum anderen lässt die Nichtdeterminiertheit den Beteiligten einer Protokoll-basierten Interaktion Entscheidungsspielraum. Mithilfe von Entscheidungen lassen sich Rollen determinieren. Dies zeigt die enge Verwandtschaft des Protokollkonzepts mit dem Konzept des Spiels.

1 Einleitung

Dieser Artikel ist der zweite einer Viererreihe, die eine Diskussion über die Grundlagen der Informatik anregen soll. Im ersten hatte ich die Motivation geliefert und die, meiner Ansicht nach, Informatik begründenden Konzepte von System, Information, Berechenbarkeit und Daten vorgestellt. In diesem zweiten Teil thematisiere ich die Interaktion zwischen Systemen und die Koordination

innerhalb von Systemen mit Hilfe des Kompositionskonzeptes. Im dritten und vierten Teil gehe ich auf zwei Themen ein, bei denen meiner Wahrnehmung nach mehr Klarheit innerhalb der Informatik und auch zwischen Informatik und ihren Anwendungsdisziplinen notwendig ist. Zum einen auf das Verhältnis von Beschreibung und Beschriebenem in Teil drei und in Teil vier auf die Frage, welchen Beitrag die Informatik zum Verständnis des Begriffs der Semantik oder Bedeutung leisten kann.

Den ersten Artikel hatte ich mit der Bemerkung beendet, dass die Informatik eine Theorie der Interaktion ist. Wenn Systeme miteinander Informationen austauschen, bleibt das in der Tat nicht folgenlos – wenigstens zwei der im ersten Artikel erwähnten scheinbaren Widersprüche gehen darauf zurück: Einerseits scheint alles in der Informatik auf dem Transport von Informationen zu basieren – andererseits scheint dieser aber bei der Beschreibung von Operationen, wie sie die imperativen Programmiersprachen konzeptuell dominieren, gar keine Rolle zu spielen. Einerseits scheint die Informatik durch das Konzept der Berechenbarkeit geprägt zu sein, in dem Determinismus herrscht und in dem Zustand nur passager notwendig ist – andererseits scheinen die Interaktionen zwischen komplexen Systemen "auf gleicher Augenhöhe" in der Regel aber doch nichtdeterministisch und zustandsbehaftet zu sein. Diese sollen mit diesem Artikel aufgeklärt werden.

Im weiteren Artikel mache ich rege Gebrauch von der im ersten Teil eingeführten Verhaltensbeschreibung (klassisch) informatischer Systeme mittels I/O-Transitionssystemen (I/O-TSen). Deren Transitionen lassen sich als 4er-Tupel (i, o, p, q) aus Eingabezeichen i, Ausgabezeichen o, Startzustand p und Zielzustand q beschreiben. Beziehen sich dabei i und p auf den Zeitpunk t, dann beziehen sich o und q auf den nächsten Zeitpunkt $t' = t + 1$. Um auch das Verhalten von Multiinputsystemen darstellen zu können, die nur auf Teile ihrer Eingaben reagieren, hatte ich zusätzlich das leere Zeichen ϵ eingeführt und für ein Alphabet $A^\epsilon = A \cup \{\epsilon\}$ definiert.

2 Systemkomposition

Der Effekt von Interaktionen lässt sich mit dem Konzept der Komposition besser verstehen ([29, 22]), das ich im ersten Teil dieser Reihe eingeführt hatte. Tatsächlich lassen sich Systeme über ihr unterschiedliches Kompositionsverhalten verschiedenen Kompositionsklassen zuordnen, worauf meines Wissens erstmals David Harel und Amir Pnueli mit ihrer richtungsweisenden Unterscheidung zwischen "reaktiven" gegenüber "transformationalen" (diskreten) Systemen hingewiesen haben [9].

2.1 Homogene Systemkomposition: Systemhierarchien

Die einfachste Möglichkeit der Systemkomposition ist, dass Systeme mittels Interaktion ein Supersystem bilden. Dies ist die Welt der Berechenbarkeit und der deterministischen I/O-TSe: Aus den Systemfunktionen der Teilsysteme bildet

sich via Interaktion nach den Regeln der Berechenbarkeit eine Superfunktion und damit ein Supersystem.

Man kann einfache Systeme parallel oder sequentiell verschalten oder in einer Rekursion mehrfach auswerten. Aus Sicht der Berechenbarkeit ist ein innerer Zustand nur im Rahmen von Rekursion und dort auch nur vorübergehend notwendig [21]. In jedem Fall entsteht eine Systemhierarchie im Sinne einer "Ist-Teil-von"-Relation: Die Subsysteme sind Teil des Supersystems.

Es gibt dabei einige wichtige Aspekte zu verstehen. Da die Supersysteme durch die Interaktion zwischen den Subsystemen entstehen, existiert zwischen den Supersystemen und ihren Subsystemen tatsächlich *keine* Interaktion.

Weiterhin ist wichtig, dass die Supersysteme bei sequentieller Kopplung ihrer Subsysteme sowie bei Rekursion eine andere Zeitskala aufweisen, was den engen Bezug zwischen dem System- und dem Zeitbegriff aufzeigt.

Und drittens lassen sich die "inneren Zustände", die die Interaktion vermitteln, aus der Darstellung der Supersysteme eliminieren.

Dies will ich an einer einfachen Komposition dreier Systeme, $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$, die zusammen ein Supersystem $\mathcal{S}_{ges}$ mit der Systemfunktion $f_{ges}(x) = 2x + 5$ bilden, darstellen (Beispiel aus [23]). Wie Abb. 1 zeigt, läuft die Berechnung (also ein Schritt für $\mathcal{S}_{ges}$) wie folgt ab: $\mathcal{S}_1$ nimmt den Wert in seinem Eingang 1 entgegen und gibt ihn in einem Schritt an $\mathcal{S}_2$ weiter. $\mathcal{S}_2$ nimmt den Wert entgegen und gibt den mit 2 multiplizierten Wert auf seinem Ausgang zurück, der wiederum ein weiterer Eingang von $\mathcal{S}_1$ ist. $\mathcal{S}_1$ bildet diesen Wert in einem weiteren Schritt auf den Eingang von $\mathcal{S}_3$ ab. $\mathcal{S}_3$ addiert 5 hinzu und gibt den Wert auf seinem Ausgang, dem dritten Eingang von $\mathcal{S}_1$, wieder zurück. Damit muss $\mathcal{S}_1$ diesen Wert nun nur noch in einem weiteren Schritt auf seinen ersten Ausgang abbilden um die Berechnung zu beenden.

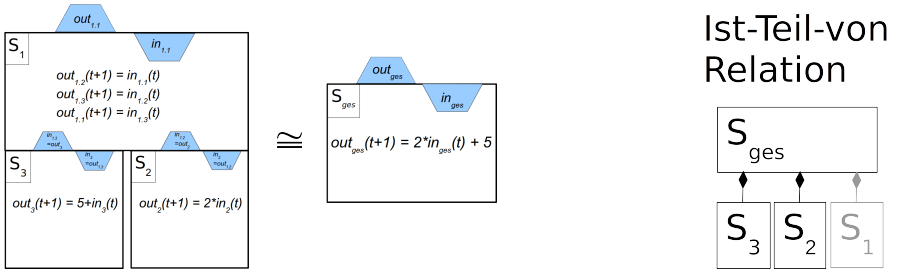

Abbildung 1: Links wird die Komposition dreier Systeme $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$ zu dem System $\mathcal{S}_{ges}$ mit der Funktion $f_{ges}(x) = 2x + 5$ dargestellt. Rechts wird die dabei entstehende "Ist-Teil-von"-Beziehung zwischen den Teilsystemen $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$ und dem Supersystem $\mathcal{S}_{ges}$ gezeigt, die durch die Interaktion entstehen. Das System $\mathcal{S}_1$ ist ausgegraut, weil es in dieser Art der Darstellung häufig weggelassen wird.

Interessant an diesem Beispiel ist, dass wir das System $\mathcal{S}_1$ und damit die gesamte "innere" Interaktion aus der Darstellung des Gesamtsystems eliminieren

können, wenn wir seine Systemfunktion mit $f_{ges}(x) = f_3(f_2(x)) = 2x + 5$ angeben. Hier tauchen nur die Teilfunktionen von $\mathcal{S}_2$ und $\mathcal{S}_3$ auf und die durch $\mathcal{S}_1$ realisierte Kompositionsvorschrift der Hintereinanderanwendung beider Funktionen.

Wir müssen also deutlich zwischen einer interaktionsbezogenen und einer strukturbezogenen Betrachtung bei deterministischen Interaktionen und ihrer korrespondierenden homogenen Systemkomposition unterscheiden. Tatsächlich ist für deterministische Interaktionen die strukturbezogene Darstellung viel ausdrucksstärker als die interaktionsbezogene, weswegen sie sich in imperativen Programmiersprachen mit dem rekursiven Operationskonzept weit verbreitet hat. Eine Operation ist der Kompositionsoperator für die gewünschte berechenbare Funktion. Das informatische Operationskonzept ist damit v.a. ein Konzept zur kompakten Darstellung der Komposition berechenbarer Funktionen unter Elimination der Beschreibung der zugrunde liegenden Interaktionen der beteiligten Systeme. Die "Intelligenz" einer Operation steckt somit in ihrer Komposition, die bekannterweise vom Softwareingenieur im Produktionsprozess einer IT-Applikation bestimmt wird.

2.2 Horizontale Systemkomposition: Die Kopplung interaktiver Systeme durch Protokolle

Es besteht weiterhin – glücklicherweise – die Möglichkeit, dass Systeme sinnvoll interagieren, aber dennoch kein Supersystem bilden, sich also vergleichsweise lose aneinander koppeln. Offensichtlich darf sich dann keine übergeordnete "Superfunktion" bilden. Das erfordert zwingend, dass die Interaktionen nichtdeterministisch sein müssen, dass also im Rahmen dieser Interaktionen die Systeme ihren Eingabewerten nicht eindeutig Ausgabewerte zuordnen.

Für nichtdeterministische Interaktionen wird damit Zustandsbehaftung essentiell, weil Nichtdeterminismus ohne Zustand "Zufälligkeit" bedeuten würde. Tatsächlich ist intuitiv sofort klar, dass ein System, das durch viele parallel laufende Interaktionen Eingaben von anderen Systemen erhält und seine Ausgaben wiederum parallel an dieselben Systeme zurücksendet und dabei "auf gleicher Augenhöhe" unterwegs ist – selbst wenn es vollständig deterministisch funktioniert – von keinem der beteiligten weiteren Systeme vollständig bestimmt sein wird.

Solche Systeme nenne ich "interaktiv" [5, 21]. Andere Bezeichnungen für diese sehr wichtige Sytemklasse sind "Prozesse" (z.B. [17]), "reaktive Systeme" (wie in der Einleitung schon erwähnt, z.B. [9]) oder "Agenten" (z.B. [19]).

Die Konstruktion des Ergebnisses dieser Interaktionen kann nicht auf der homogenen Komposition berechenbarer Funktionen beruhen, da diese Konstruktion per definitionem *immer* zu berechenbaren Superfunktionen und damit Supersystemen führt. Stattdessen kommen wir zu einer interessanten Beschreibung der Interaktion wie auch der Komposition der Systeme, wenn wir die Systeme anders als in Abschnitt 2.1 nicht vollständig, sondern nur in ihrer Projektion auf die Interaktion, die ich "Rolle" nenne, betrachten. Dieser Projektionscharakter

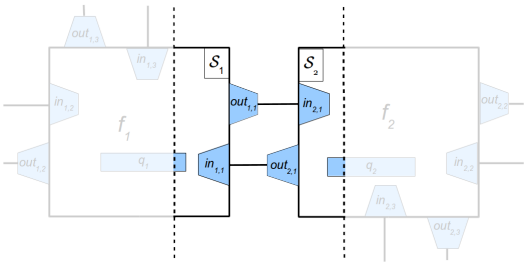

Abbildung 2: Zwei interaktive Systeme $\mathcal{S}_1$ und $\mathcal{S}_2$ interagieren miteinander und mit vielen weiteren Systemen über Protokolle. Der Kompositionsoperator des zugehörigen Protokolls operiert nur auf den Projektionen der Systemfunktionen auf der Interaktion, den sogenannten Rollen.

macht mit seiner erzwungenen Unvollständigkeit auch verständlich, weswegen Rollen in der Regel nichtdeterministisch sind.

Die interaktionsvermittelte Komposition von Rollen verschiedener Systeme ergibt nun kein System, sondern ein Protokoll $\mathcal{P} = C_{\mathcal{P}}(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ (s.a. [11]).

Diese Konstruktion illustriere ich anhand des Beispiels von zwei Zügen $\mathcal{Z}_1$ und $\mathcal{Z}_2$, die sich eine eingleisige Eisenbahnbrücke teilen [1] (s. Abb. 3), einer einfachen Version des allgemeinen Problems einer gemeinsam genutzten Ressource [11]. Zu diesem Zweck interagieren beide Züge mit einem gemeinsamen Controller $\mathcal{C}$ über ein Protokoll. Der Controller muss dafür sorgen, dass sich zu keinem Zeitpunkt mehr als ein Zug auf der Brücke befindet.

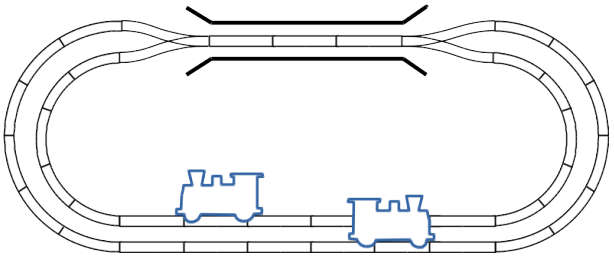

Abbildung 3: Eine eingleisige Eisenbahnbrücke, die von zwei Zügen benutzt wird. Um eine Kollision auf der Brücke zu vermeiden, interagieren beide Züge mit einem zentralen Controller.

Das Protokoll $\mathcal{P} = C_{\mathcal{P}}(\mathcal{Z}_k, \mathcal{C})$ zwischen einem Zug $\mathcal{Z}_k$ ($k = 1$ oder 2) und dem Controller $\mathcal{C}$ zeige ich in Abb. 4. Dazu beschreibe ich sowohl den Zug als auch den Controller nur in Bezug auf ihre Rolle, die sie in der Interaktion spielen mit einem Modell aus 3 Zustandswerten, $Q_{Z_k/C} = \{away, wait, bridge\}$. Es gilt $I_{\mathcal{Z}_k} = O_{\mathcal{C}} = \{go\}$ sowie $O_{\mathcal{Z}_k} = I_{\mathcal{C}} = \{arrived, left\}$.

Das Protokoll zwischen Zug und Controller ist vollständig, da keine weiteren externen Zeichen auftreten. Es ist wohlgeformt, da es für jedes gesendete Zeichen

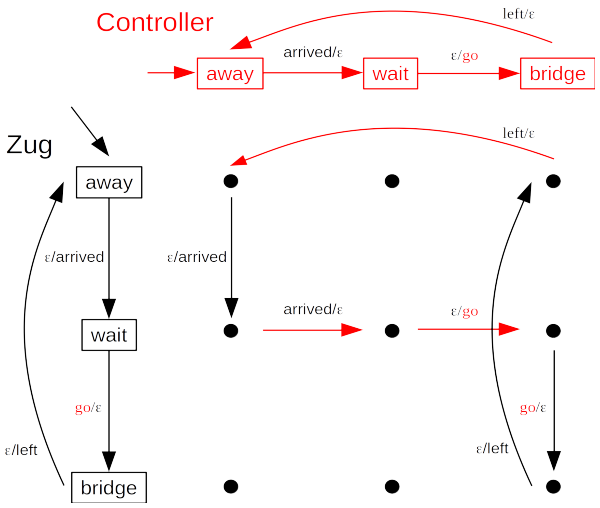

Abbildung 4: Darstellung des Protokolls zwischen Zug und Controller für das Problem der eingleisigen Eisenbahnbrücke. Zu Beginn befinden sich sowohl der Controller als auch der Zug im Zustand *away*. Wenn ein Zug eintrifft, signalisiert er dem Controller *arrived* und dieser wechselt in den Zustand *wait*. Der Controller gibt das Gleis mit *go* frei und der Zug signalisiert dem Controller mit *away*, dass er die Brücke wieder verlassen hat. Die Interaktion ist erfolgreich, wenn sowohl der Zug als auch der Controller ihre drei Zustände unendlich oft durchlaufen.

zum richtigen Zeitpunkt eine Transition gibt. Und schließlich ist es konsistent, da es nur endliche Interaktionsketten hat und seine Erfolgsbedingung erfüllt.

Dabei besteht die Komposition aus der Bildung des Produktautomaten sowie in der Einschränkung der Transitionsrelation dieses Produktautomatens durch den nun hergestellten Informationsaustausch – ihre Herstellung ist also denkbar einfach und von völlig anderer Natur als die Komposition von Funktionen zu Superfunktionen. Die "Intelligenz" steckt offenbar nicht in der Komposition sondern in den Rollen selbst.

Allerdings garantiert dieses Protokoll nicht unser ursprüngliches Ziel, dass höchstens ein Zug gleichzeitig die Brücke benutzt. Dies erfordert die "richtigen" Entscheidungen des Controllers. Darum geht es im nächsten Abschnitt.

2.3 Protokolle, Entscheidungen und Spiele

Offensichtlich bestimmt bei einer nichtdeterministischen Interaktion die Interaktion nicht (vollständig) die Aktionen der beteiligten Systeme. Die Verarbeitung der empfangenen Zeichen ist also nicht vollständig durch die Interaktion selbst bestimmt. Ein einfacher Ansatz zur Vervollständigung der Rollenbeschreibung zu einer Systemfunktion ist das Konzept der Entscheidungen: Entscheidungen

bestimmen das Verhalten, d.h. die Übergänge, wo es sonst unbestimmt wäre.

Entscheidungen in diesem Sinne sind ein weiteres, "inneres" Eingabealphabet D, das das Eingabealphabet I eines nichtdeterministischen I/O-TSs $\mathcal{A}$ zu $I' = I \times D^{\epsilon}$ so ergänzt, dass die ergänzte Übergangsrelation Δ' deterministisch wird [21]. Dazu sind die Entscheidungen eindeutig und verschieden von den bisherigen Zeichen zu bezeichnen. Das Entscheidungsalphabet eines schon deterministischen I/O-TSs ist offensichtlich leer.

Wie eine Entscheidung getroffen wird, haben wir bewusst offen gelassen. Damit können wir zu recht sage, dass es gerade die Eigenschaft eines Protokolls ist, die Frage, wie Entscheidungen getroffen werden, auszuklammern.

Diese Betrachtung zeigt den sehr engen Bezug der Informatik als Theorie der Interaktion mit ihrem Protokollbegriff zur Spieltheorie, wie sie Johann v. Neumann und Oskar Morgenstern begründet haben [30]. Tatsächlich lässt sich ein um Entscheidungen ergänztes Protokoll als "Spiel in Interaktionsform" (GIF) ansehen [21]. Die traditionelle Spielform entsteht dann durch Abstraktion von der Interaktion, in dem man alle Zustandswerte als äquivalent betrachtet, die unter derselben Entscheidung (also unter ϵ-Entscheidungen) erreichbar sind [20, 21]. In der Automatentheorie ist dieses Verfahren als ϵ-Elimination bekannt.

Das Nutzenkonzept der traditionellen Spieltheorie, das auf einer Präferenzrelation basiert, betrifft dann schon die Frage, wie Entscheidungen getroffen werden.

2.4 Koordination

Anschaulich gesprochen koordiniert der Controller unseres Beispiels seine Interaktionen mit den beiden Zügen. Tatsächlich entspricht dieser Anschauung eine weitere inhomogene, koordinationsbasierte, "innere" Komposition der Rollen *eines* Systems – im Gegensatz zur ebenfalls inhomogenen, aber interaktionsbasierten, "äußeren" Komposition der Rollen *verschiedener* Systeme zu einem Protokoll. D.h. der Controller $\mathcal{C}ontr = Comp_{\mathcal{C}}(\mathcal{C}_1, \mathcal{C}_2)$ lässt sich als Komposition seiner beiden Rollen $\mathcal{C}_1$ und $\mathcal{C}_2$ als System konstruieren.

Bei dieser Komposition geht es darum, dass Aktionen eines Systems im Sinne einer Ausführung seiner Systemfunktion in der Regel in mehreren Rollen "gleichzeitig" Relevanz entfalten können. Diese ist für das Beispiel des Controllers in Abb. 5 dargestellt. Dort etwa die als ein Schritt aufgefasste Transition $(away_1, wait_2) \xrightarrow{arrived_1/go_2} (wait_1, bridge_2)$. Damit ist diese Konstruktion das konstruktive Gegenstück zur Rolle als Projektion des Systems auf seine Interaktion.

Die Koordinierungsregel besagt, dass beide Züge nicht gleichzeitig auf der Brücke sein dürfen. Das bedeutet, dass der interne Zustand des Controllers nicht den Wert $(Bridge, Bridge)$ annehmen darf. Offensichtlich müssen die Zustandsübergänge des Produktautomaten von $\mathcal{C}_1$ und $\mathcal{C}_2$, die den Zustand $(Bridge, Bridge)$ als Zielzustand enthalten, eliminiert werden. Dies ist möglich, ohne die Rollen des Controllers als dessen Projektion auf die jeweilige Interaktion in den Protokollen zu verfälschen.

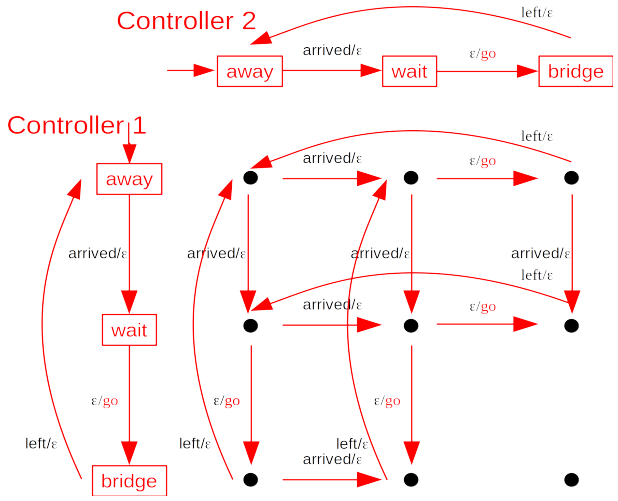

Abbildung 5: Zustandsdiagramm des zentralen Controllers $Contr$, der zwei Züge $\mathcal{Z}_1$ und $\mathcal{Z}_2$ koordinieren soll. Es zeigt die innere Kopplung seiner beiden Rollen $\mathcal{C}_1$ und $\mathcal{C}_1$ durch Eliminierung aller Übergänge zum Zustand $(Bridge, Bridge)$.

Es ist zu beachten, dass die Korrektheit, man könnte auch sagen die Wahrheit der Darstellung des Zugzustandes im Controller von der Korrektheit des Protokolls abhängt. D.h. der Zustand des Controllers $(Bridge, Bridge)$ repräsentiert die Tatsache, dass sich beide Züge auf der Brücke befinden, nur dann, wenn die Interaktion zwischen Controller und Zug dies tatsächlich sicherstellt.

Der in Abb. 5 dargestellte Controller muss im Zustand $(wait, wait)$ eine Auswahlentscheidung treffen, welchem der beiden Züge er die Erlaubnis zur Weiterfahrt gibt. Belässt man die Symmetrie, stellt sich für den Konstrukteur des Controllers die Frage, wie die Auswahlentscheidung berechnet werden sollte. Hier sind viele Möglichkeiten denkbar. Löst man diese Frage hingegen durch Entfernung einer der beiden Transitionen $(wait, wait) \rightarrow (wait, bridge)$ oder $(wait, wait) \rightarrow (bridge, wait)$, eröffnet diese Symmetriebrechung die Möglichkeit, dass einer der Züge nie über die Brücke gelassen wird - ein Fairness-/Starvationproblem.

Dieses Beispiel veranschaulicht sehr gut, dass Nichtdeterminismus in einer Interaktion notwendig ist, um sie mit anderen Interaktionen intern zu koordinieren oder um Entscheidungsspielräume in einer Interaktion zu beschreiben. Es illustriert weiterhin die Tatsache, dass zusätzliche Interaktionen den bestehenden Nichtdeterminismus in der Bestimmung eines Systems einschränken können, indem sie zusätzliche Koordinierungszwänge auferlegen - aber dass sie ebenso die Möglichkeiten, Entscheidungen zu treffen, vermehren können.

3 Hierarchische Systemarchitektur

Die Struktur eines IT-Systems im Sinne der Kompositionsbeziehungen seiner Komponenten wird als seine Architektur verstanden [14, 25]. Was ist der Unterschied zwischen einer Komponente und einem System? Clemens Szyperski schreibt prägnant: "Components are for composition." [26] (S. 3). Wir können das nun präzisieren, indem wir Komponenten als Systeme betrachten, die wir für eine bestimmte Komposition vorsehen.

Dieser Idee folgend definiere ich, wie schon in [22] vorgeschlagen, den Teil eines Systems, der in eine Komposition eingeht, als ein Interface dieses Systems. Unsere Intention, ein System für eine bestimmte Komposition vorzusehen, es also als Komponente anzusehen, äußert sich dann darin, dass wir es so konstruieren, dass dieses Interface sich in dieser Konstruktion explizit wiederfindet.

Wir haben zwei ganz unterschiedliche Kompositionsklassen kennengelernt, die auf der Systeminteraktion basieren. Zum einen die homogene, hierarchische Komposition von Systemen zu Supersystemen und zum anderen die inhomogene, Rollen-bezogene, nicht-hierarchische Komposition zu Protokollen.

Damit werden durch die Interfaces der verschiedenen Klassen ganz verschiedene Aspekte eines Systems "nach außen" offenbart und andere bleiben verborgen. Das Interface "Operation' verbirgt die Struktur der Berechnung der Systemfunktion, also quasi den Algorithmus, sowie die Interaktion selbst. Das Interface "Rolle" hingegen offenbart in der Protokollkomposition nach außen die Interaktion und verbirgt die Berechnung der eigenen Entscheidungen, während sie nach innen in der Koordinationskomposition die Abhängigkeiten von den den Transitionen der anderen Systemen ignoriert.

Wir können nun die hierarchische, strukturelle Beziehung der Komponenten eines Systems und die interaktionsbezogene, semantisch ungerichtete Beziehung zwischen Komponenten verschiedener Systeme in einem zweidimensionalen Komponentenmodell ausdrücken, wie es Abb. 6 zeigt. Die Vertikale repräsentiert die hierarchische Ordnung der "ist-Teil-von" Relation, wie sie durch die funktionale Komposition entsteht. Die Horizontale repräsentiert die Interaktionsbeziehung der Komponenten gleicher Schicht über Protokolle.

Bei dieser Darstellung ist zu beachten, dass zum einen Informationen nur in horizontaler Richtung, aber *nicht* in vertikaler Richtung ausgetauscht werden. Und zum anderen, dass die Systemgrenzen sowohl in horizontaler als auch in vertikaler Richtung durch die Interaktionen, also durch die Mathematik bestimmt werden und nicht durch die Zeichnung. Man kann also ohne Weiteres mit dieser Art der Darstellung die Unwahrheit sagen, noch dazu sehr suggestiv.

Dieses Architekturmodell ist eine Präzisierung der Schichtungsidee des wegweisenden Open Systems Interconnection (OSI) Modells [15]. Allerdings baut es kurioser Weise auf einer, dem OSI-Modell entgegenstehenden Annahme auf. Während das OSI-Modell noch davon ausging, sich nur mit dem Informationsaustausch und keinesfalls mit der internen Verarbeitung der Informationen zu beschäftigten, "OSI is concerned with the exchange of information between open systems (and not the internal functioning of each individual real open system).", geht das vorgestellte Modell gerade davon aus, dass die Struktur der Systeme

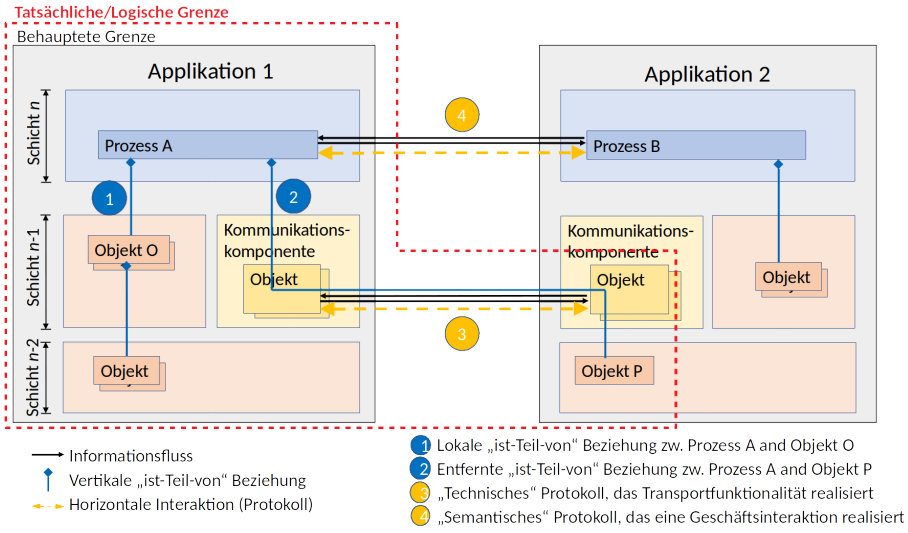

Abbildung 6: Eine geschichtete Applikationsarchitektur. Die Vertikale repräsentiert die Ordnung durch die ”ist-Teil-von” Relation, wie sie durch die funktionale Komposition entsteht. Die miteinander über Protokolle interagierenden Komponenten finden horizontal in derselben Schicht.

und ihrer Interaktion eben durch die Art ihrer Informationsverarbeitung bestimmt wird. Bezogen auf unser konkretes Systemmodell ist es schlicht inkonsistent, einerseits gänzlich von der Struktur der Verarbeitung der Informationen abzusehen und andererseits weitreichende Behauptungen über die Struktur (der Informationsverarbeitung[!]) eben dieser Systeme aufstellen zu wollen.

Folgerichtigerweise versagte das OSI Modell in der Praxis dort, wo diese Inkonsistenzen wirksam wurden. Nämlich ab der 5. Schicht bei der es um die Verwaltung eines ”Session”-Zustandes geht. Nur im Falle der hierarchischen Komposition kann ein interaktionsbezogener Zustand in einer zwischengeschalteten ”Sitzungsschicht” gekapselt und damit ”nach oben” eliminiert werden. Im Falle der horizontalen Interaktion gehört der interaktionsbezogene Zustand tatsächlich zu den Komponenten derselben semantischen Schicht, die miteinander über ein Protokoll interagieren.

Weil es sich auf die Verarbeitung der Informationen bezieht, ist dieses Schichtenmodell ein semantisches. Zusätzlich geht mit dieser Hierarchie ebenfalls eine Hierarchie der Abstraktion einher, weil die hierarchische Komposition den inneren Aufbau der Berechungen an ihren Interfaces verbirgt.

Der Erfolg des Interfacekonzepts der Operation zeigt sich nicht zuletzt in den inzwischen allgegenwärtigen Paketmanagern, von denen in vielen Programmiersprachen jeder mehr als 100.000 hierarchisch komponierende Pakete verwaltet [6]. Problematisch wird es hingegen, wenn die Auffassung vertreten wird, dass

alleine den Operationen ein Interfacecharakter zukommt (z.B. aktuell [4]). In ihrem Überblick über Komponentenmodelle unterscheiden Ivica Crnkovic et al. [7] zwischen "operationsbasierter" und "portbasierter" Schnittstellenunterstützung und zeigen damit auf, dass viele der derzeit wichtigen Komponentenmodelle tatsächlich die Deklaration von Protokollschnittstellen nicht unterstützen - ein gravierendes Defizit für alle, die interaktive Systeme beschreiben wollen.

4 Diskussion

Die Informatik, verstanden als Theorie der Interaktion, sagt uns nicht nur unter welchen Umständen wir Interaktionen als Austausch von Informationen verstehen können, sondern sie lässt uns auch besser die Konsequenzen verstehen, die sich aus den verschiedenen Möglichkeiten ergeben, die ausgetauschten Informationen jeweils zu verarbeitet.

Es gab einige Ansätze in der Informatik, Interaktionen ohne Bezug auf die Informationsverarbeitung zu beschreiben. Dazu neben dem schon erwähnten OSI-Modell zwei weitere Beispiele: Anfang der 1990er Jahre wurde das Client-Server-Modell als Request/Reply-Schema verstanden [2, 27]: Der Client sendet die Anfrage und der Server die Antwort. Das passt zwar grundsätzlich zum Operationskonzept, aber es wäre völlig verfehlt, den Erfolg verteilter IT-Anwendungen auf dieses Schema zurückzuführen. Im Sinne unseres Modells der Systeme und ihrer Interaktionen beschränkt sich der wesentliche Unterschied zwischen Client und Server auf den zwischen Anrufendem und Angerufenem (s.a. Spezifikation des TCP/IP-Protokolls (RFC 793, 7323)). Mit diesem Kriterium lässt sich in der Tat eine Ordnung erklären. Die Frage ist nur, wie aussagekräftig sie ist.

Das zweite Beispiel für die Beschreibung von Interaktionen ohne den Bezug zur Informationsverarbeitung stammt aus dem Kontext der Enterprise Application Integration. Dort wird häufig von "message-based integration" geredet (z.B. [8]). Im Alltagsleben käme tatsächlich niemand auf die Idee, die Interaktion etwa zwischen Kunde und Bank oder Bürger und Finanzamt als "Nachrichten-basierte Integration" zu erklären. Auch das ist nicht falsch, aber dass zum Zwecke der Interaktion "Nachrichten" auzutauschen sind, ist schlicht zu inhaltsleer um sich in der Alltagssprache zu verankern. Im Sinne unseres Modells sind es sowohl im Alltag als auch in der Informatik neben den ausgetauschten Dokumenten v.a. die Regeln nach denen sie ausgetauscht werden und ihre Bedeutung, die diese Interaktion prägen.

Mit dem Konzept der Komposition haben wir einen Ansatz, der uns in gewisser Weise von selbst auf tatsächlich wesentliche Aspekte von Interaktion lenkt: Was müssen wir alles wissen, um sinnvoll über eine gewisse Klasse von Interaktionen zu reden? Und warum im Kontext der nur Werte repräsentierenden Zustandsfunktionen der Unterschied zwischen Determinismus und Nichtdeterminismus so wichtig ist.

Der deterministische Fall eröffnete die Möglichkeit einer strukturellen Darstellung hierarchischer Funktionalität, die auf die Darstellung eines Informationsaustausches innerhalb der Komposition gänzlich verzichten konnte – ganz

entlang der Vorstellung, dass Informationsaustausch eigentlich nur zwischen Systemen und nicht innerhalb von Systemen stattfindet. Das Kompositionskonzept erlaubte die Definition der Operation als Interface, die die Art ihrer Berechnung "nach oben" verbirgt, aber ihre Abbildungseigenschaft offenbart.

Um Interaktionsnetzwerke besser zu verstehen, ist das Konzept der Operation wenig hilfreich – was die Unzulänglichkeit aller Ansätze von verteilten Objekten wie der "Common Object Request Broker Architecture (CORBA)", des "Distributed Component Object Model (DCOM)" oder auch der "Open Platform Communications Unified Architecture (OPC-UA)" in diesem Umfeld erklärt.

Stattdessen steht unser unvollständiges Wissen im Vordergrund. Systeme, die ihre Eingabe intermittierend aus vielerlei Interaktionen erhalten, werden sich an ihren Schnittstellen nichtdeterministisch und zustandsbehaftet verhalten. Hier tritt das Rollen- bzw. Protokollkonzept in den Vordergrund, verkörpert es doch unser unvollständiges Wissen um diese Systeme in ihren Interaktionen. Das Protokollkonzept setzt mit seiner Abgeschlossenheit unserem erforderlichen Wissen um die Interaktionsteilnehmer klare Grenzen.

Wie aufgezeigt ist die Bedeutung von Zustand je nach Kontext sehr unterschiedlich: Im Rahmen der Berechnung von Funktionen ist Zustand etwas, das nur im Rahmen von Rekursion temporär notwendig ist, weswegen die funktionale Programmierung hier auch (wenig überraschend, aber nichtsdestotrotz unzutreffend) von "Seiteneffekten" spricht [12]. Im Rahmen einer nichtdeterministischen Interaktion stiftet der Zustand hingegen den Zusammenhang zwischen den einzelnen Interaktionsschritten und bildet damit auch die Basis, um im Rahmen der Koordination mehrere Interaktionen eines Systems sinnvoll miteinander in Einklang zu bringen.

Beide Modelle, das strukturorientierte, funktionale und das interaktionsorientierte, relationale ergänzen sich – richtig verstanden – sehr gut zu einem "geschichteten IT-Applikationsmodell". Falsch verstanden geben beide Modelle Anlass zu endlosen Diskussionen. Man stelle sich einen Informatiker vor, der der (irrigen) Auffassung ist, dass Client-Server-Interaktionen auf dem Prinzip der Remote-Operation aufzubauen sind (z.B. [31]). Dieser trifft auf eine Betriebswirtin und versucht ihr zu erklären, wie Interoperabilität zwischen komplexen Systemen wie etwa einem Enterprise-Ressource-Planning (ERP) und einem Manufacturing-Execution-System (MES) effizient herstellbar ist. Nach langem Hin-und-Her einigen beide sich auf das Wort "Dienste" und dass Informatik eben nur von Informatikern zu verstehen ist. Sie versteht darunter aber weiterhin Dienstleistung im Sinne eines Protokolls, bestehend aus Fertigungsauftrag, etc., er hingegen einen deterministischen, möglichst zustandslosen, weil dann einfacheren SOA-Webservice, mit dem man sogar alle bisherigen Dokumente eliminieren kann. Was wird passieren? Nach einigen Jahren, die Betriebswirtin hat dem Informatiker angekündigt, den Geldhahn für sein Projekt demnächst zuzudrehen, weil es halt nicht funktioniert, kommt ein anderer oder vielleicht auch derselbe Informatiker, behauptet, er habe nun die Prinzipien des Internets auf die Schnittstellentechnik übertragen, nennt es "REST-Dienst" und das Spiel geht von vorne los.

Es liegt nahe, die Überlegung dieses Artikels, dass ein hierarchisches Mo-

dell ein einheitliches Ordnungskriterium aufweisen muss, als generelle Anforderung aufzustellen, wie es der Bitkom Arbeitskreis Interoperabilität I40 getan hat [3]. Eine wichtige Klasse solcher Modelle sind die von mir in der Einleitung des ersten Artikels angesprochenen sogenannten "Referenzmodelle", die die US-amerikanische Standardisierungsorganisation OASIS definiert als "an abstract framework for understanding significant relationships among the entities of some environment" und das v.a. der Entwicklung konsistenter Standards und der Verständigung mit Nichtspezialisten dienen soll [18].

Entsprechend viele solcher Referenzmodelle wurden entwickelt. Tatsächlich genügt nach meinem Verständnis kaum eines der mir bekannten Modelle diesem Referenzanspruch und damit auch nicht der Anforderung das gegenseitige Verständnis wesentlich zu fördern. Teilweise wird sogar das Gegenteil erreicht. Eine kritische Diskussion des Referenzmodells Industrie 4.0 (RAMI4.0) findet sich in [24]. Die sogenannte Wissenspyramide [10] hatte ich im ersten Teil der Serie angeführt. Die Inkonsistenz des ebenfalls recht weitverbreiteten "Level of Conceptual Interoperability Model (LCIM)" [28], das etwa vom US-amerikanischen Industrial Internet Consortium (IIC) in 2018 aufgegriffen wurde [13], diskutiere ich in [22]. Auch die Hierarchisierung der sogenannten "Automatisierungspyramide" ist nicht wirklich konsistent, was meiner Ansicht nach einer der Gründe dafür ist, weshalb sie in so vielen Ausprägungen verwendet wird [16].

Und, last but not least, ermöglicht der vorgestellte Ansatz es uns, über die Interaktion zwischen Maschine und Mensch einheitlicher zu denken – zum Vorteil des Menschen. Das lässt sich mit dem Schachspielen gut erläutern. Schachspieler müssen sich – unabhängig von ihrer Natur – schlicht an die Schachregeln halten, die sich als Rollen in Form von I/O-TSen beschreiben lassen. Die wesentlichen Unterschiede zwischen Mensch und Maschine liegen nicht in der konkreten Interaktion. Nein, ihr Verhalten lässt sich durch dieselben Rollen modellieren. Die wesentlichen Unterschiede liegen unter anderem in der herausragenden Eigenschaft des Menschen, den Kontext seiner Interaktion ad hoc zu adaptieren und zum in der Art und Weise, wie sie Entscheidungen treffen. Menschen, die nachweislich höchsten eine Handvoll Stellungen pro Sekunde bewerten können, treffen ähnlich gute Schachentscheidungen wie Maschinen, die nachweislich Millionen, ggfs. sogar Milliarden Stellungen pro Sekunde bewerten – ist das nicht faszinierend?

Literatur

[1] R. Alur. *Principles of Cyber-Physical Systems.* MIT Press, 2015.

[2] G. R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys (CSUR)*, 23(1):49–90, 1991.

[3] Bitkom. *Impulse zur Weiterentwicklung der Normungsroadmap Industrie 4.0.* White paper, 2021.

[4] *Details of the Asset Administration Shell. Part 2 – Interoperability at Run-*

time – Exchanging Information via Application Programming Interfaces (Version 1.0RC01). Bundesministerium für Wirtschaft und Energie (BMWI), 2020.

[5] M. Broy. A logical basis for component-oriented software and systems engineering. *Comput. J.*, 53(10):1758–1782, 2010.

[6] R. Cox. Surviving software dependencies. *Communications of the ACM*, 62(9):36–43, 2019.

[7] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron. A classification framework for software component models. *Software Engineering, IEEE Transactions on*, 37(5):593–615, 2011.

[8] D. L. Freire, R. Z. Frantz, F. Roos-Frantz, and S. Sawicki. Survey on the run-time systems of enterprise application integration platforms focusing on performance. *Software: Practice and Experience*, 49(3):341–360, 2019.

[9] D. Harel and A. Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498. Springer-Verlag, New York, 1985.

[10] J. Hey. The data, information, knowledge, wisdom chain: the metaphorical link. *Intergovernmental Oceanographic Commission*, 26:1–18, 2004.

[11] G. J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[12] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, 1989.

[13] IIC: The Industrial Internet of Things, Volume G5: Connectivity Framework, 2018.

[14] Systems and software engineering - Architecture description, 2011.

[15] ITU-T. X.200 Information Technology - Open Systems Interconnection – Basic Reference Model, 1994.

[16] T. Meudt, M. Pohl, and J. Metternich. Die Automatisierungspyramide – Ein Literaturüberblick. Technical report, 2017.

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

[18] OASIS. Reference model for service oriented architecture 1.0. `http://docs.oasis-open.org/soa-rm/v1.0/`, 2006. Aufgerufen am 2015-01-17.

[19] S. Poslad. Specifying protocols for multi-agent systems interaction. *ACM Trans. Auton. Adapt. Syst.*, 2(4), Nov. 2007.

[20] J. Reich. The relation between protocols and games. In S. Fischer, E. Maehle, and R. Reischuk, editors, *39. Jahrestagung der GI, Lübeck*, volume 154 of *LNI*, pages 3453–3464. GI, 2009.

[21] J. Reich. Composition, cooperation, and coordination of computational systems. *preprint arXiv:1602.07065*, 2016/2020/2021.

[22] J. Reich. Komposition und Interoperabilität von IT-Systeme. *Informatik Spektrum*, 40:339–346, 2021.

[23] J. Reich and T. Schröder. A simple classification of discrete system interactions and some consequences for the solution of the interoperability puzzle. *ifac 2020*, 2020.

[24] J. Reich, L. Zentarra, and J. Langer. Industrie 4.0 und das Konzept der Verwaltungsschale – eine kritische Auseinandersetzung. *HMD Praxis der Wirtschaftsinformatik*, pages 1–15, 2020.

[25] I. Sommerville. *Software engineering, Deutsche Ausgabe*. Pearson, 9 edition, 2012.

[26] C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1 edition, 2002.

[27] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall International, Inc, 1992.

[28] A. Tolk, C. D. Turnitsa, S. Y. Diallo, and L. S. Winters. Composable M&S web services for net-centric applications. *The Journal of Defense Modeling and Simulation*, 3(1):27–44, 2006.

[29] S. Tripakis. Compositionality in the Science of System Design. *Proceeding of the IEEE*, 104:960 – 972, 2016.

[30] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[31] P. Wegner. Interoperability. *ACM Computing Surveys (CSUR)*, 28(1):285–287, 1996.