Flexibilität in Future Networks durch Service Oriented Architectures (SOAs) und selbstbeschreibende Mikroprotokolle

Yannick Scherer
Betreuer: Heiko Niedermayer
Seminar Future Internet WS2010/2011
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
scherery@in.tum.de

KURZFASSUNG
Die Verwendung eines standardisierten, festen Protokollstacks in Netzwerken geht mit vielen Vorteilen einher, darunter die Interoperabilität zwischen Systemen, sowie ein hoher Grad an Robustheit und Vorhersehbarkeit. Doch neue Technologien wie *Voice over IP (VoIP)* oder *Video on Demand (VoD)* stellen auch neue Ansprüche an die zugrundeliegende Architektur, wodurch die Anpassung alter und die Einführung neuer Protokolle notwendig wird. Es soll hier gezeigt werden, wie sich durch Abkehr vom verbreiteten Schichtenmodell hin zu Service Oriented Architectures, sowie durch einen Mechanismus, der an individuelle Bedürfnisse angepasste Protokollstacks erstellt, die Akzeptanz und Verwendung neuer Protokolle erhöht und vereinfacht, die Evolvierbarkeit und Flexibilität in zukünftigen Netzen also gesteigert werden kann.

Schlüsselworte
SOA, SOCS, Mikroprotokolle, Future Networks, Dynamic Protocol Configuration

1. EINLEITUNG

Die Gründe, dass aus einem überschaubaren Forschungsnetz in Nordamerika das weltumspannende "Netz der Netze", das Internet, wurde, sind vielfältig, doch ist es der TCP/IP-Protokollstack, dem mit seiner Omnipräsenz der spürbarste Anteil am Erfolg zugestanden werden muss: Als gemeinsame performante Basis verschiedenster Anwendungen auf unterschiedlichsten Systemen ermöglicht er es, die weltweite Heterogenität auf einen gemeinsamen Nenner zu bringen, großen vernetzten Systemen also überhaupt erst einen Sinn zu geben.

Doch trotz des ungeheuren Erfolges kommt man v.a. in letzter Zeit nicht umhin, die Schwächen dieser starren Architektur, ja die Schwächen von Architektur im Allgemeinen zu erkennen: Neue Anforderungen, wie sie z.B. durch Streamingservices wie VoIP und VoD, aber auch durch die wachsende Nutzung des Internets mit Mobilgeräten entstehen, machen Anpassungen an bestehenden Kernkomponenten einzelner Netze unabdingbar. Diese sind jedoch viel zu komplex und starr ausgelegt, als dass die Adaption neuer Innovationen ohne großen Aufwand vonstatten gehen könnte. Das beste Beispiel hierfür ist IPv6, denn obwohl der Abschluss der Spezifikation mittlerweile Jahre zurückliegt, ist die Akzeptanz und Verbreitung im Vergleich mit dem Voränger IPv4 immer noch sehr gering.

Was kann man also tun, um neuen Gegebenheiten in Gegenwart eines festen (und sei es auch nur eines "moralischen") Rahmens gerecht zu werden? Die Antwort pragmatisch denkender Entwickler liegt nahe [1]: Man sprengt den Rahmen. Statt mit IPv6 begegnen wir der schwindenden Zahl freier IP-Adressen mit Network Address Translation (NAT), ganz egal, ob dadurch das *End-to-End*-Prinzip des Internets verloren geht oder nicht; dem Risiko, das durch die Möglichkeit von *Denial of Service (DoS)* Attacken entsteht, schieben wir einen Riegel vor, indem wir (ebenfalls End-to-End-unfreundliche) Firewalls verwenden; und neue Funktionalität lässt sich nicht auf Netzwerk- (IP) oder Transport-Ebene (TCP) einführen - zumindest nicht, wenn man noch irgendetwas mit dem Internet zu tun haben will - sondern nur auf höheren Schichten. Dies produziert möglicherweise unnötigen Overhead, doch v.a. wird es fast zwingend erforderlich, den einzelnen Schichten die Kommunikation untereinander zu erlauben, da nur so ein an die Art der Daten, die den unteren Ebenen ja prinzipiell nicht bekannt ist, angepasster, optimierter Fluss möglich wird. Und das wiederum ist in einer Schichtenarchitektur wie dem OSI-Modell schlichtweg nicht erlaubt.

Das Internet ist nicht das Netz, das es selbst sein wollte. Es besteht aus einem immer noch verlässlichen Rumpf, der jedoch von vielen, vielen Workarounds und "Hacks" immer mehr zersetzt wird [2]. Die Komplexität steigt und mit ihr die Fehleranfälligkeit; die Auflösung der strikten Trennung zwischen den einzelnen Schichten (so z.B. zu sehen bei HTTP-Firewalls oder der Verwendung von Portnummern zu einer Vielzahl an Zwecken [4]) führt zu einer starken Kopplung verschiedener Netzwerkebenen, was darin resultiert, dass Anpassungen erschwert, ja fast schon gefährlich und unvorhersehbar werden.

Es gibt zwei Möglichkeiten, mit dieser Situation umzugehen: Man gesteht sich entweder ein, dass das Internet in seiner aktuellen Form zwar Schwächen, jedoch auch weiterhin das Recht hat, zu existieren, man also eine Alternative *neben* dem bestehenden Netz aufbauen sollte. Oder man will sich nicht auf das Risiko einlassen, das durch etwaige Altlasten entsteht, und favorisiert einen kompletten Neuanfang.

doi: 10.2313/NET-2011-05-2_03

In dieser Arbeit wird letztere Sicht vertreten: Zunächst sollen in Kapitel 2 die Anforderungen, die an ein zukünftiges World Wide Web zu stellen sind (und warum die aktuelle Architektur nicht beibehalten werden kann), sowie deren Verbindung mit Service Oriented Architectures (Kapitel 3.1) und Mikroprotokollen (Kapitel 3.2) beschrieben werden. Anschließend, in Kapitel 4, wird ein Mechanismus dargestellt, der die automatische Verwendung eines *optimierten* Stacks selbstbeschreibender Mikroprotokolle ermöglicht, bevor letztlich auf die offenen Probleme, die damit einhergehen, Bezug genommen wird.

2. ANFORDERUNGEN AN FUTURE NETWORKS

Im Folgenden soll aufgezeigt werden, welche Eigenschaften ein zukünftiges Internet haben und welchen Anforderungen es genügen sollte.

2.1 Evolvierbarkeit

Das Internet der Zukunft muss sich weiterentwickeln können. Es muss möglich sein, Protokolle ohne großen Aufwand hinzuzufügen, zu entfernen oder anzupassen, d.h. die Abhängigkeiten zwischen Anwendungen und den zugrundeliegenden Protokollen müssen auf ein Minimum reduziert werden.

Aktuell besteht das World Wide Web aus Schichten, von denen jede eine große Anzahl an Aufgaben übernimmt. Die einzige Möglichkeit, auch nur winzigste Änderungen vorzunehmen, ist der Austausch der (als Black-Box gehaltenen) kompletten Schichten [4]. Da die Schnittstellen zu den angrenzenden Ebenen hier beibehalten werden müssen, lassen sich Modifikationen transparent verwenden. Dennoch können wir nicht einfach einen frischen, dem bisherigen Modell treu bleibenden Protokollstack entwickeln und einführen, da wir schlicht und einfach nicht wissen, welche Herausforderungen uns in Zukunft bevorstehen! [2]

Ein zukünftiges Netz sollte die Sache deshalb feiner angehen und ermöglichen, klein gehaltene Funktionalitätseinheiten (wie z.B. die in Kapitel 3.2 beschriebenen Mikroprotokolle) auf dynamische Art und Weise einzubinden und zu verwenden. Eine Schlüsseleigenschaft ist die lose Kopplung zwischen den einzelnen Einheiten selbst, wobei sich hier als erfolgsversprechender Ansatz *Service Oriented Architectures (SOAs)* in den Vordergrund drängen. Diese werden im nächsten Kapitel genau beschrieben.

2.2 Anwendungsorientierung

Nicht alle Anwendungen haben die gleichen Bedürfnisse: ein Instant Messenger hat andere Sicherheitsansprüche als ein Online-Banking-Client, ein Videostream legt nicht so sehr wert auf eine verlässliche Datenübertragung wie ein User, der eine Datei von einem Server bezieht. Und während diese Bedürfnisse früher einen eher diskreten Charakter hatten (z.B. “Verlässliche Verbindung oder nicht?”), muss heute in Abstufungen gedacht werden (z.B. “Wie effektiv soll die Fehlerkorrektur sein?”). [5]

Ein zukünftiges Internet sollte die Fähigkeit haben, alle oder zumindest einen Großteil der Ebenen und Teilaufgaben (z.B. Forwarding) an die Bedürfnisse der Anwendungen anzupassen, die über das Netz kommunizieren. Natürlich ist ein anwendungsorientierter Datenaustausch auch mit den bereits bestehenden Technologien möglich, allerdings hauptsächlich in höheren Schichten, was einen möglichen Daten-Overhead begünstigt und zu den bereits in der Einleitung angesprochenen Problemen bezüglich der effizienten, datenorientierten Kommunikation führt.

Als Ansatz, ein System möglichst anwendungsorientiert zu gestalten, lassen sich Mikroprotokolle nennen: Komplexe Aufgaben werden in ihre kleinsten Teile zerlegt, sodass aus einer geringen Menge an “Bauteilen” eine große Zahl komplexer Kompositionen mit unterschiedlichsten Eigenschaften erzeugt werden kann. (Kapitel 3.2)

2.3 Effizienz

Ein letzter Blick sei hier auf die Effizienz gelegt, denn was nützt das schönste Konzept, wenn am Ende das Netzwerk von einer Datenflut heimgesucht wird oder sich einzelne Knoten zu Tode rechnen? Vor allem ein minimaler Protokoll-Overhead ist nötig, um den Datenfluss in den Netzen der Zukunft zu optimieren.

Betrachtet man z.B. den IPv4-Header, sieht man Felder, die die Fragmentierung steuern und selbst dann präsent sind, wenn durch andere Maßnahmen wie PMTU (oder noch einfacher: dem DF-Flag im Header selbst) bereits sichergestellt ist, dass das Paket nicht zerlegt werden muss. Oder aber das *Header Checksum*-Feld, das von vielen Routern aus Performancegründen gar nicht erst ausgewertet wird.

Wir werden in Kapitel 3.2 sehen, dass Mikroprotokolle durch ihren modularen Aufbau und ihre namensgebende Kompaktheit, sowie durch die im vorhergehenden Abschnitt besprochene Anwendungsorientierung sehr gute Performance und geringen Overhead erzielen können.

3. GRUNDLAGEN

In diesem Kapitel geht es um die Bauteile, die, wenn zusammen verwendet, den Anforderungen in Kapitel 2 gerecht werden und damit die Basis für ein Future Internet legen können.

3.1 Service Oriented Architectures (SOAs)

Service Oriented Architectures ermöglichen es, Funktionalität dynamisch zu verwalten und lose gekoppelt zur Verfügung zu stellen. Die grundlegende Einheit sind hierbei sog. *Services* - Dienste, die sich v.a. durch die folgenden Eigenschaften auszeichnen [4] [6]:

Eigenständigkeit Ein Service ist in sich abgeschlossen, d.h. er stellt seine Funktionalität in einer Art und Weise zur Verfügung, die es ermöglicht, ihn unabhängig von anderen Diensten zu nutzen. Das bedeutet aber nicht, dass Services nicht miteinander kommunizieren dürfen, wie wir weiter unten sehen werden.

Wohldefinierte Schnittstelle Die Verwendung eines Dienstes muss nur durch Kenntnis seiner Schnittstelle, also ohne Wissen über interne Implementierungsdetails, möglich sein. So könnte ein Service, der die Integrität von Daten mithilfe einer Prüfsumme sicherstellen soll, auf verschiedene Arten realisiert werden, z.B. mithilfe von CRC oder einer Hashfunktion wie SHA-1.

doi: 10.2313/NET-2011-05-2_03

"Blindheit" Um eine möglichst lose Kopplung der einzelnen Services zu gewährleisten, müssen Annahmen über die internen Gegebenheiten anderer Dienste außen vor bleiben. So sollte ein Service A, der den Wert eines Prüfsummenservices B vorgesetzt bekommt, nicht davon ausgehen, dass die Berechnung dieser Summe z.B. mit CRC geschehen ist. Das ermöglicht den einfachen Austausch von B durch einen anderen Service.

Wiederverwendbarkeit Eine sich aus den vorhergehenden Punkten ergebende Eigenschaft von Services ist die Wiederverwendbarkeit der Funktionalität in unterschiedlichsten Umgebungen. Noch wichtiger: Low-Level Services lassen sich zu neuen, komplexeren Diensten zusammensetzen, was z.B. in Service Oriented Communication Systems (SOCS, siehe Kapitel 3.3) genutzt wird.

Services sind also modulare Einheiten, die *on demand* instanziiert werden können.

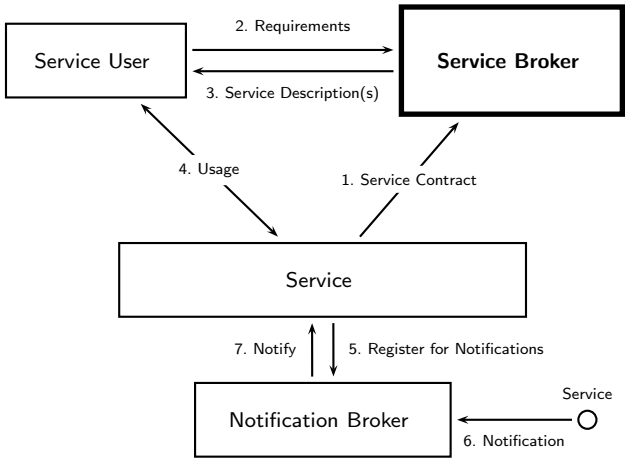

Abb. 1: Datenfluss in SOAs (nach [3], [4])

Der beste Service ist jedoch nutzlos, wenn er mangels Bekanntheit von niemandem in Anspruch genommen werden kann. An dieser Stelle kommt der *Service Broker* ins Spiel, der - wie ein Telefonbuch - eine Liste von ansprechbaren Diensten verwaltet, in die sich jeder Service erst einmal eintragen muss. Der dabei übertragene *Service Contract* beinhaltet u.a. die Spezifikation der Schnittstelle, sowie die Eigenschaften eines Dienstes, anhand derer der Broker in der Lage ist, eine passende Liste von Services zu einer gegebenen Menge an Anforderungen zu ermitteln. (siehe Abbildung 1, Schritte 1-3)

Als Beispiel seien Services gegeben, die Bücher in einer Bibliothek für eine bestimmte Zeit reservieren und dafür eine Bearbeitungsgebühr verlangen: Während manche User einen möglichst langen Reservierungszeitraum bevorzugen, ist für andere vielleicht wichtiger, dass die Gebühren minimal sind. Dies kann in den Anforderungen festgelegt werden, der Broker trifft dann die jeweils beste Entscheidung.

Es ist offensichtlich, dass für die Kommunikation zwischen Service und Broker bzw. zwischen Broker und User ein einheitliches Format verwendet werden muss, das von beiden Seiten verstanden wird und die notwendigen Informationen übermitteln kann. Im Bereich der WebServices (die einige SOA-Prinzipien widerspiegeln [6]) wird z.B. oftmals WSDL [8] für Schnittstellenspezifikationen genutzt; auch UDDI-Verzeichnisse [7] arbeiten hauptsächlich mit XML-Daten (SOAP-Schnittstelle). Auf die Frage, inwieweit eine formale Spezifikation eines Services überhaupt möglich ist bzw. an welchen Stellen es Probleme geben kann, wird aber im letzten Teil dieser Arbeit nochmals eingegangen.

Die Kommunikation zwischen Services geschieht (der losen Kopplung halber) indirekt, d.h. über eine Vermittlungsstation: Hat ein Service Daten, die er veröffentlichen/anderen zur Verfügung stellen will, so speichert er diese an einem von allen berechtigten Services zugänglichen Ort und benachrichtigt den sog. *Notification Broker*. Dieser informiert wiederum alle weiteren Services, die sich für die entsprechende Art von Benachrichtigungen (*Notifications*) registriert haben [4]. (siehe Abbildung 1, Schritte 5-7) So kann es z.B. einen Monitoring-Service geben, der alle Fehler-Notifications abfängt und in Log-Dateien niederschreibt.

SOAs ermöglichen es also, Funktionalität flexibel zur Verfügung zu stellen - eine Eigenschaft, die auch in den Netzen der Zukunft vorhanden sein soll.

3.2 Mikroprotokolle

Will man ein Haus bauen, so muss man sich über eine Reihe von Dingen Gedanken machen: Wo positioniert man beispielsweise die Fenster? Oder welche Art von Dach passt am besten? Natürlich kann man das alles umgehen und auf bewährte, schnell verfügbare Lösungen, ja vielleicht gleich auf ein ganzes Fertighaus zurückgreifen - doch selbst wenn bisher 80% der Nutzer zufrieden mit der Isolierung waren, heißt das nicht, dass sie auch für ein Domizil in Nordsibirien geeignet ist. Und ganz ehrlich: wenn alle Häuser gleich aussehen, sieht doch keines mehr gut aus.

Auf den heutigen Systemen ist TCP/IP so ein (wenn auch gut funktionierendes) Fertighaus - ja sogar eines, um das man nicht umhin kommt. Will man mit dem Rest der Welt (sprich: Anwendungen auf anderen Rechnern) in Kontakt bleiben, so *muss* man IP und evtl. TCP bzw. UDP "sprechen", egal ob man alle deren Features überhaupt braucht oder nicht.

Mikroprotokolle sind demgegenüber mehr wie die fertigen Fenster, Dächer und Türen, die man sich aussuchen kann, um dem eigenen Haus eine gewisse Individualität zu verpassen. Jedes Mikroprotokoll übernimmt eine kompakte, klar definierte Aufgabe [2], was es ermöglicht, sie zu neuen, größeren Protokollen zusammenzusetzen. So könnte es z.B. ein Mikroprotokoll zur Fehlererkennung (beispielsweise über eine Prüfsumme) geben, das folgendermaßen aussieht:

Daten	Prüfsumme

Dazu noch eines, das mithilfe von Sequenznummern und Acknowledgements die richtige Reihenfolge und die verlässliche Ankunft der Daten sicherstellt:

Sequenznummer	Acknowledgement
Daten	

doi: 10.2313/NET-2011-05-2_03

Und zuletzt noch ein simpler Forwardingmechanismus anhand von Quell- und Zieladressen (nicht notwendigerweise IP-Format!):

Quelle	Ziel
Daten	

Kombiniert man diese Mikroprotokolle nun, erhält man ein neues, komplexeres Protokoll, das eine gewisse Ähnlichkeit zum bestehenden TCP/IP hat, allerdings z.B. auf Fluss- und Staukontrolle verzichtet und dementsprechend in Netzen, die diese Mechanismen nicht benötigen, den Overhead reduziert:

Quelle	Ziel
Sequenznummer	Acknowledgement
Daten	
Prüfsumme	

Mikroprotokolle ermöglichen es also, an die Bedürfnisse von Anwendungen und Netzen angepasste Protokolle durch Komposition kleinerer Teile zu erstellen. Wichtig ist jedoch, dass sich sowohl Sender als auch Empfänger darüber einig sind, in welcher Reihenfolge die einzelnen Teile zu bearbeiten sind. Am Beispiel unseres "Makroprotokolls" ist z.B. nicht direkt ersichtlich, ob sich die Prüfsumme auf das gesamte Paket oder möglicherweise nur die Daten bezieht.

An dieser Stelle liegt die Entscheidung darüber, welche "Bausteine" zu verwenden, aber immer noch bei der Anwendung selbst, d.h. die Akzeptanz neuer Protokolle ist nach wie vor gering, da ihre Verwendung nicht automatisch geschieht. Sehen wir Mikroprotokolle aber als Services im Sinne einer SOA, so können wir die letztliche Auswahl der einzelnen Funktionen in eine externe Einheit, nämlich den Broker, verlagern, und stehen mit einem Mal einer Fusion aus Flexibilität und Anwendungsorientierung gegenüber: *SOCS*.

3.3 Service Oriented Communication Systems (SOCS)

Die Funktionsweise von *Service Oriented Communication Systems* - also Systemen, die die Kommunikation zwischen zwei Endpunkten mithilfe von Services praktizieren - dürfte an dieser Stelle bereits erahnt werden: Eine Anwendung, die eine Verbindung aufbauen will, schickt eine Liste mit Anforderungen an den (lokalen) Service Broker, der sie mit den Servicespezifikationen in seiner Datenbank vergleicht. Anschließend ermittelt er den besten Treffer und stellt ihn der Anwendung zur Verfügung.

Hierbei gibt es zwei Möglichkeiten: entweder der Broker verwaltet eine Liste fertig verwendbarer Protokollstacks (*Communication Services*), oder er erstellt aus kleineren Services (z.B. Mikroprotokollen) einen neuen, individuell angepassten Communication Service. Die erste Möglichkeit ist mit weniger Rechenaufwand (dafür mehr Konfiguration) verbunden, schränkt aber auch die Anpassbarkeit der Kommunikation ein, während die zweite Möglichkeit einen effizienten Auswahlalgorithmus erfordert, demgegenüber aber sehr nahe an die Anforderungen herankommen kann. Natürlich lassen sich einmal erstellte Kombinationen auch zwischenspeichern und bei Bedarf anpassen, was einer Hybridlösung der beiden Ansätze entspricht.

Wichtig ist: Ein Service Broker ist bei seiner Auswahl nicht nur auf die formalen Spezifikationen beschränkt, die ihm übergeben werden! Er kann - und dies ist v.a. im Netzwerkumfeld entscheidend - auch auf externe Daten zurückgreifen [3], z.B. Rechnerauslastung, Firewallkonfigurationen, Netzwerkcharakteristika, etc... Am Ende steht dann ein optimierter Protokollstack, den die Anwendung über eine einheitliche Schnittstelle verwenden kann. (siehe Fig. 2)

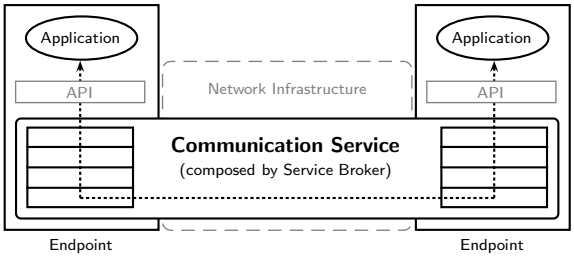

Abb. 2: Aufbau von SOCS (nach [3])

Wir betrachten in dieser Arbeit den Ansatz, Mikroprotokolle als Basis zur Komposition von SOCS zu verwenden. Die letzte offene Frage diesbezüglich ist nun: *Wie* kann ein Service Broker die beste Kombination, also den passendsten Protokollstack, bestimmen?

4. SELBSTBESCHREIBENDE PROTOKOLLE

Bisher haben wir uns mit der allgemeinen Funktionsweise einzelner Teile auseinandergesetzt: SOAs zur losen Kopplung, Mikroprotokolle zur Anwendungsorientierung und Effizienz, sowie SOCS als Fusion der beiden Ansätze. In diesem Kapitel werden wir uns nun jedoch detailliert mit einem Auswahlverfahren zur Komposition von Communication Services befassen, das anhand von Mechanismen und Effekten (siehe nächster Abschnitt) ein ideales Ergebnis erzielen soll.

4.1 Implementierung, Mechanismus & Effekt

Betrachten wir an dieser Stelle einmal das Standardwerk eines Programmieranfängers: Hello World. Es gibt viele verschiedene Arten, die bekannte Ausgabe zu erzeugen, z.B. in Java:

```
System.out.println("Hello World");
```

Genausogut wäre es möglich, den String zuerst in einer Variable zu speichern, bevor er ausgegeben wird:

```
String hello = "Hello World";
System.out.println(hello);
```

Beide Programme sind konkrete *Implementierungen* eines abstrakten *Mechanismus*, d.h. einer implementierungsunabhängigen Beschreibung von Funktionalität [2]. In unserem Fall lautet der Mechanismus: "Schreibe 'Hello World' auf die Konsole!".

Verschiedene Implementierungen können demnach derselben Spezifikation genügen. Dies gilt für einfache "Hello-World"-Programme genauso wie für Mikroprotokolle oder gar TCP: Es gibt viele Möglichkeiten, z.B. die Staukontrolle (einen

doi: 10.2313/NET-2011-05-2_03

Mechanismus!) zu implementieren, und alle sind zulässig, solange sie nur der formalen, schriftlich festgehaltenen Spezifikation von TCP entsprechen.

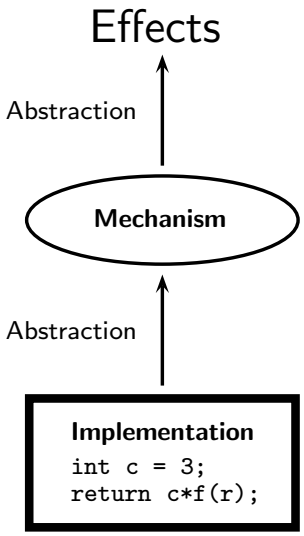

Abb. 3: Von der Implementierung zum Effekt [2]

Mechanismen lassen sich (siehe Abb, 3) anhand ihrer Ergebnisse beschreiben, den sog. *Effekten* [2]: So bewirkt z.B. die Verwendung des Mechanismus "CRC-Prüfsumme" den Effekt "Korrektheit der Daten", während etwa "Sequenznummern" die "Richtige Reihenfolge" begünstigen.

Neben den normalen Effekten existieren noch sog. *Meta-Effekte*, also solche, die sich durch Komposition mehrerer anderer ergeben. "Verlässlichkeit" setzt sich beispielsweise aus "Korrektheit der Daten", "Richtige Reihenfolge" und "Vollständigkeit" zusammen. (siehe Abb. 4). Natürlich kann ein Mechanismus auch mehrere Effekte mit sich bringen, genauso wie es möglich ist, dass ein Effekt erst durch Verwendung mehrerer Mechanismen entsteht. Doch dies ist in [2] detailliert beschrieben, weshalb hier nicht genauer darauf eingegangen werden soll.

4.2 Mechanismen, Services & Mikroprotokolle

Da Effekte eine solide Grundlage zur Beschreibung von Mechanismen darstellen, eignen sie sich sehr gut, um etwa in einem SOCS Service Broker zur Auswahl eines passenden Protokollstacks herangezogen zu werden. Hierbei werden Mechanismen (in unserem Fall also Mikroprotokolle) in Services gekapselt, die sich beim Broker unter Angabe mindestens der folgenden Daten registrieren müssen:

Liste von Effekten Jedes Mikroprotokoll hat gewisse Effekte - und der Broker muss wissen, um welche es sich dabei handelt, da ansonsten eine Auswahl passender "Bausteine" schwer wird. Hierbei sollte man unterscheiden zwischen *qualitativen* Effekten, die eine Bewertung des Protokolls zulassen [3] (z.B. im Bereich des *Quality of Service*), oder aber *inhärenten* Effekten, die einen absoluten Charakter (vorhanden oder nicht vorhanden) haben. In [2] werden nur letztere dargestellt, doch in meinen Augen darf man, v.a. in Anbetracht der Anforderungen, die durch VoIP et al erwachsen, nicht auf qualitative Betrachtungen verzichten.

Anforderungen Manche Mikroprotokolle funktionieren nur in Verbindung mit anderen richtig, z.B. macht es keinen wirklichen Sinn, Sequenznummern zur Sicherstellung der Reihenfolge zu verwenden, wenn nicht auch die Prüfsumme des Protokollheaders vorhanden ist, um zufälligen Änderungen in der Nummerierung von Paketen vorzubeugen. Dies muss der Service Broker wissen, um einen konsistenten Protokollstack erstellen zu können.

Kosten Jedes Mikroprotokoll ist mit einer gewissen Menge an Rechenaufwand verbunden, welcher natürlich (neben den o.g. qualitativen Aspekten) ebenfalls in die Berechnungen des Brokers einfließen sollte.

Hat ein Service Broker all diese Daten, kann er loslegen.

4.3 Aufbau

Die Komposition eines passenden Mikroprotokollstacks geschieht auf Basis einer bestehenden Netzwerkarchitektur, die sich aus den vorhandenen Mechanismen $\mathbb{M}$ (z.B. Prüfsumme, Sequenznummern, Acknowledgements, Staukontrollfenster, ...), den existierenden Effekten und Meta-Effekten $\mathbb{E}$ (z.B. Vertraulichkeit, Forwarding, Vollständigkeit, ...), sowie denjenigen Mechanismen $\mathbb{M}_f$ zusammensetzt, die in jedem Fall verwendet werden, da sie auf niedrigeren Netzwerkebenen implementiert sind (z.B. die Frame Check Sequence in Ethernet-Paketen). [2]

Der Service Broker hat, wie oben festgestellt, zu jedem Mechanismus eine Liste von Effekten gespeichert, was wir hier als Funktion $P : \mathbb{M} \rightarrow 2^{\mathbb{E}}$ darstellen wollen. Gleichzeitig existiert eine Liste von benötigten Mechanismen, die mithilfe von $R : \mathbb{M} \rightarrow 2^{\mathbb{M}}$ realisiert wird.

4.4 Konsistenz

Eine Auswahl von Mechanismen sei genau dann *konsistent*, wenn sie alle Mechanismen enthält, die benötigt werden:

$$consistent(M \subseteq \mathbb{M}) = \left[\bigcup_{m \in M} R(m) \right] \subseteq M$$

Dies stellt sicher, dass keine unvollständigen Protokollstacks als Communication Services verwendet werden.

4.5 Auswahl von Mechanismen

Erhält der Service Broker eine Reihe von erwünschten Effekten $\mathbb{D}$, so berechnet er eine passende Menge von Mechanismen, also einen passenden Protokollstack, als konsistente Mechanismusliste mit minimalen Kosten, d.h. [2]:

$$S(\mathbb{D}) = \arg\min_{M \subseteq \mathbb{A}} C(M)$$

wobei C die Gesamtkosten einer Liste von Mechanismen berechnet und $\mathbb{A}$ alle konsistenten Mengen passender Mechanismen darstellt:

$$\mathbb{A} = \{M \subseteq \mathbb{M} | \mathbb{M}_f \subseteq M \wedge consistent(M) \wedge D \subseteq P^*(M)\}$$

(P^* ist die Menge aller Effekte, die durch eine Liste von Mechanismen erzielt werden.)

doi: 10.2313/NET-2011-05-2_03

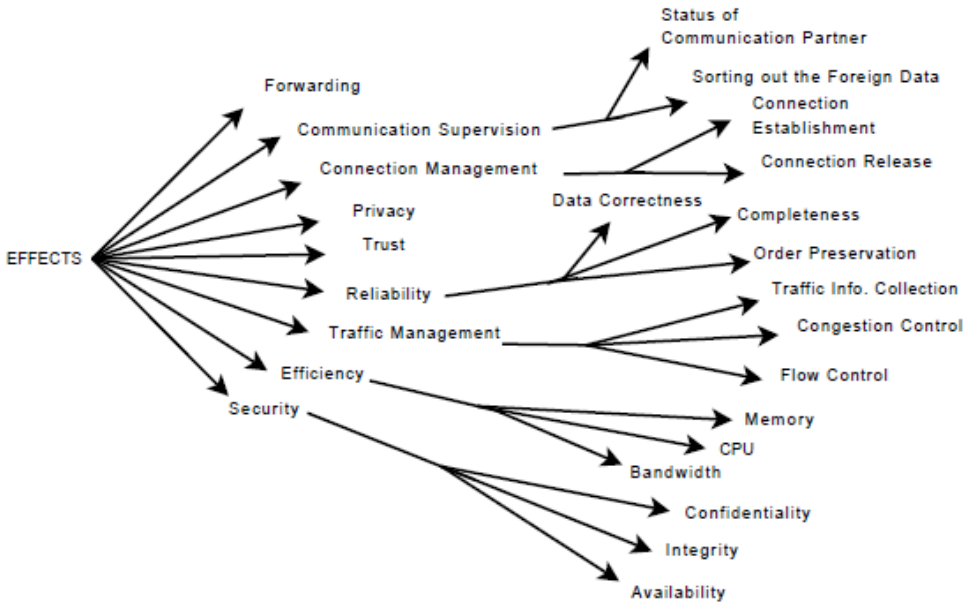

Abb. 4: Beispielhafte Effekthierarchie [2]

Ein sehr komplexer Punkt ist hierbei die Kostenfunktion C, da in verschiedenen Verbindungen von Mechanismen einzelne Gewichtungen ebenfalls unterschiedlich sein können (z.B. wenn Werte im Protokollheader als Zwischenergebnisse wiederverwendet werden). Zusätzlich sollte die Qualität von Mechanismen - relativ zu den entsprechenden Anforderungen - mit in die Berechnung einfließen, was keine simple Aufgabe ist.

Alles in allem ist der Algorithmus, der $S(\mathbb{D})$ berechnet, die wohl kritischste Komponente des hier beschriebenen Ansatzes, denn arbeitet dieser nicht effizient, kommt es zu Verzögerungen beim Verbindungsaufbau, die nicht akzeptabel sind. In [2] werden mehrere Implementierungsmöglichkeiten angeschnitten, darunter eine naive Herangehensweise (alle konsistenten Lösungen - und das sind sehr viele - finden und bewerten); eine, die mit zuvor schon berechneten Elementen arbeitet und sie nur noch an die Gegebenheiten anpasst (wie so eine Anpassung aussehen kann, ist in [3] beschrieben); und letztlich eine, die auf parallel ausgeführte Heuristiken baut.

Wie die Auswahl am Ende geschieht, ist offen und kann von Netz zu Netz unterschiedlich sein. Wir sehen aber, dass die Beschreibung von Mikroprotokollen anhand ihrer Effekte ein vernünftiges Mittel zur Optimierung von SOCS darstellt.

4.6 Ein Beispiel

Das war nun alles sehr theoretisch und formal, weshalb der Sachverhalt an einem Beispiel verdeutlicht werden soll. Gehen wir davon aus, dass der Service Broker die folgenden, beispielhaft definierten Mechanismen in seiner Datenbank gespeichert hat (wobei die Funktionalität niedrigerer Schichten an dieser Stelle ausnahmsweise ignoriert wird):

Header-Prüfsumme	
benötigt	-
erzielt	Integrität
kostet	2

Sequenznummern	
benötigt	Header-Prüfsumme
erzielt	Reihenfolge
kostet	1

Acknowledgements	
benötigt	Header-Prüfsumme
erzielt	Vollständigkeit
kostet	1

Prüfsumme	
benötigt	-
erzielt	Korrektheit
kostet	2

Sliding Window	
benötigt	Sequenznummern, Acknowledgements
erzielt	Flusskontrolle, Effizienz (Bandbreite)
kostet	3

Stop-and-Wait	
benötigt	Acknowledgements
erzielt	Flusskontrolle
kostet	1

doi: 10.2313/NET-2011-05-2_03

(Hier haben wir übrigens ein Beispiel, in dem Services miteinander kommunizieren müssen, um eine vernünftige Funktionalität herzustellen, denn wie können die Flusskontrollmechanismen arbeiten, wenn sie keine Daten vom Acknowledgement-Service bekommen?)

Wenn der Service Broker einen Communication Service zusammenstellen soll, der Flusskontrolle und Korrektheit garantiert, wird er (bei naiver Herangehensweise) zuersteinmal die folgenden Kombinationen ermitteln:

- {Sliding Window, Prüfsumme}
- {Stop-and-Wait, Prüfsumme}

Anschließend werden die Mengen so lange mit benötigten Mechanismen angereichert, bis sie konsistent sind:

- {Sliding Window, Prüfsumme, Sequenznummern, Acknowledgements, Header-Prüfsumme}
- {Stop-and-Wait, Prüfsumme, Acknowledgements, Header-Prüfsumme}

Die Kostenfunktion wird (ohne Rücksicht auf irgendwelche Abhängigkeiten) die beiden Werte 9 und 6 berechnen, weshalb nur die Stop-and-Wait-Variante in den Protokollstack einfließt, den der Nutzer verwenden soll. Wäre in den Anforderungen noch die effiziente Ausnutzung der Bandbreite aufgetaucht, hätte das Sliding Window den Vorzug erhalten.

Hier sehen wir auch einen Grund dafür, qualitative Effekte zuzulassen, v.a. im Bereich der Effizienz und Übertragungsqualität: Was passiert etwa, wenn ein Nutzer angeben will, dass die Effizienz der Bandbreite nicht zu 100%, sondern nur zu 40% in die Entscheidungen des Brokers einfließen sollen (weil z.B. in Firmennetzwerken ab einer gewissen Uhrzeit sowieso weniger los ist)? Wenn dann nicht bekannt ist, *wie* effizient ein gegebener Mechanismus ist - und wie sich das auf die Kosten auswirkt - hat man ein Problem. (Es sei hier auf [3] verwiesen, wo sich ein Beispielverfahren zur Auswahl anhand qualitativer Eigenschaften findet.)

5. BEWERTUNG UND OFFENE PROBLEME

5.1 Heterogenität

Durch den ständigen Wandel der Funktionalität einzelner Knoten, wie sie durch SOAs ermöglicht wird, entsteht zwangsläufig Heterogenität, der entsprechend begegnet werden muss. Nicht alle Netzelemente können mit allen Mechanismen umgehen [4] und das simple Aushandeln der verwendeten Protokolle durch die Endpunkte garantiert nicht, dass ein Paket auch wirklich den gesamten Weg übersteht.

Es gibt verschiedene Ansätze, dieses Problem zur Laufzeit zu lösen [4]:

Funktionalität ignorieren So ist z.B. Flusskontrolle wünschenswert, aber nicht notwendig, um Daten zu übertragen, und kann dementsprechend von einzelnen Knoten ignoriert werden.

Funktionalität auflisten Existieren nur wenige Mechanismen, die betrachtet werden sollen, so können diese zwischen den Knoten ausgetauscht werden, um entsprechend interoperable Lösungen zu wählen. Bei einer größeren Menge an Protokollen wird das aber nicht mehr effizient möglich sein.

Funktionalität delegieren Ist ein Knoten bekannt, der mit einem Mechanismus umgehen kann, so kann man die Daten an diesen zur Verarbeitung weiterleiten. Dies ist jedoch nicht immer praktikabel und erhöht außerdem die Netzauslastung.

Generell sind Lösungen, die zur Laufzeit angewandt werden, nicht unbedingt ideal, da meist zusätzlicher Traffic erzeugt wird, der den Vorteil des geringen Overheads von Mikroprotokollstacks zunichte macht. Machbarer und auch logisch sinnvoller ist die Klärung der Kompetenzen einzelner Knoten bereits im Vorfeld, wobei zwei Prinzipien zu nennen sind:

Capability Domains Ein Knoten kann sich in einer Capability Domain registrieren, wenn er eine bestimmte Funktionalität (bzw. einige wenige Funktionalitäten) besitzt. So wären mögliche Domains z.B. definiert durch die Fähigkeit, Stau- oder Flusskontrolle zu verwenden, IP-Adressen zu verstehen, etc... Die entsprechenden Informationen könnten ähnlich zu Routing-Protokollen verteilt werden. [4]

Overlays Ein anderer Ansatz ist es, das Gesamtnetz in kleinere (virtuelle) Teile, sog. Overlays (vgl. SpovNet [9]), zu zerlegen, in denen alle enthaltenen Knoten eine bestimmte Menge an Funktionalität besitzen. Dementsprechend darf sich auch niemand in einem Overlay registrieren, der nicht alle Vorraussetzungen bzgl. der Funktionalität erfüllt. [4]

Der Unterschied ist subtil: Während bei Capability Domains alle Rechner im selben Netz liegen und somit beim Aufbau der Kommunikation zwischen zwei *bestimmten* Knoten immer noch geklärt werden muss, welche Funktionen - sprich: Services - auf beiden Seiten vorhanden sind (anhand des Vergleichs der Capability Domains), stellen Overlays Netze dar, in denen *alle* Knoten eine bestimmte Reihe von Fähigkeiten besitzen, aus denen ohne weitere Abstimmung frei gewählt werden kann. Demgegenüber erschwert das abgrenzende Prinzip der Overlays aber die Kommunikation im Gesamtnetz, während Capability Domains eine Anpassung an unterschiedliche Gegebenheiten ermöglichen. Welche der beiden Varianten die bessere ist, lässt sich zu diesem Zeitpunkt allerdings noch nicht sagen.

Eine letzte Möglichkeit wäre die Festlegung eines Standard-(Transport-)Protokollstacks, der als *Fallback*-Mechanismus agiert, wenn keine vernünftige Lösung gefunden wird. Dies birgt aber die Gefahr, dass der Einfachheit halber die Kommunikation zwischen zwei Endpunkten über Gateways geleitet wird, die den Fallback-Mechanismus verwenden. Protokollspezifische Eigenschaften hätten dann nämlich keinen Wert mehr.

5.2 Spezifikation von Services & Optimierung

Es ist schwierig, ein einheitliches Format festzulegen, das von Services verwendet wird, um ihre Beschreibungen an den Service Broker zu übermitteln: Wir können nicht wissen, welche Anforderungen zukünftige Dienste an dieses Format haben, genausowenig wie wir sicher sein können, welche Da-

doi: 10.2313/NET-2011-05-2_03

ten ein “neuartiger” Broker bräuchte, um ideale Ergebnisse zu liefern.

In [3] wird deshalb vorgeschlagen, eine Liste von *Properties*, also Name-Wert-Paaren zu übermitteln, die beliebig erweiterbar ist. Dies erhöht die Flexibilität bei der Definition von Services, doch löst noch lange nicht alle Probleme, denen wir hier begegnen.

Zwischen Services existieren unterschiedlich geartete Abhängigkeiten: Die *Header-Prüfsumme* macht z.B. keinen Sinn, wenn sie sich nicht auf den gesamten Header bezieht; und ein Acknowledgements-Mechanismus erzielt alleine vllt. die Effekte “Vollständigkeit” und “Richtige Reihenfolge” (wenn nämlich als Stop-and-Wait-Mechanismus implementiert), kann die Reihenfolge der Daten aber nicht mehr sicherstellen, wenn zusammen mit dem Sliding-Window-Ansatz verwendet. Auch die Präsentation von Rechenkosten einzelner Mechanismen ist sehr komplex (siehe Kostenfunktion in Kapitel 4.5), da manche Mikroprotokolle z.B. “billiger” werden, wenn sie auf Zwischenergebnisse anderer Services zurückgreifen können.

Die Beibehaltung der Flexibilität der Service-Spezifikation bei gleichzeitiger Darstellbarkeit komplexer Beziehungen ist somit ein Punkt, der kritisch für die Optimierung zukünftiger Netze ist.

5.3 Verlässlichkeit

Man kann in keiner Architektur garantieren, dass alle Dienste zu jedem Zeitpunkt verfügbar sind. Während dies allerdings bei “traditionellen” Systemen zum Totalausfall führen könnte, bietet die Dynamik, die mit SOAs einhergeht, ein sehr starkes Mittel zur Anpassung an solche Vorfälle. [6]

5.4 Performance

Die Performance des hier beschriebenen Ansatzes hängt stark davon ab, wie gut und schnell der Service Broker Entscheidungen trifft, sowie von den verfügbaren Mikroprotokollen. Dies sind jedoch Fragen geeigneter Optimierung und Auswahl, welche von Netz zu Netz unterschiedlich sein wird.

Es lässt sich hier jedoch getrost festhalten, dass sich nicht alle Anwendungsgebiete für den Einsatz von Mikroprotokollstacks eigenen: So scheinen z.B. im Bereich der Hochleistungsrechner stark spezialisierte Protokolle besser geeignet als “ungefähr optimale” Kompositionen.

5.5 Implementierungs- & Dokumentationsaufwand

Die Verwendung von SOAs erfordert v.a. eine Anpassung der Denkweise einzelner Programmierer, da die Beziehungen zwischen Services nicht mehr statisch, sondern extrem wandelbar sind. So wird es in erster Linie unbedingt notwendig, die Dokumentation solcher Beziehungen so ausführlich und detailliert wie möglich zu gestalten, um (z.B. in großen Firmen) nicht aus Angst vor Änderungen letztlich wieder an ein starres System gebunden zu sein.

Was einzelne Services angeht, wird sowohl Implementierung und Dokumentation einfacher: Jeder Dienst hat eine bestimmte Aufgabe, ist also nicht übermäßig komplex, und stellt durch seinen selbstbeschreibenden Charakter schon alle Informationen zu Schnittstelle & Co. zur Verfügung. (Dokumentation muss “sowieso” betrieben werden, kann also nicht mehr als lästige Zusatzaufgabe gesehen werden.)

Der Broker und der Auswahlalgorithmus für einzelne Services sind die wohl kritischsten Teile der Implementierung und erfordern sehr hohen Aufwand. Zukünftige Verbesserungen werden vermutlich v.a. hier ansetzen, was ebenfalls einen hohen Dokumentationsgrad mit sich bringt.

Alles in allem wird der Aufwand bei Implementierung und Dokumentation durch SOAs nicht geringer werden, doch die sich daraus ergebenden Vorteile, besonders im Bereich der Netzwerkorganisation, werden sich irgendwann bezahlt machen.

6. ZUSAMMENFASSUNG

Die Verwendung von Service Oriented Communication Systems auf Basis selbstbeschreibender Mikroprotokolle ermöglicht es, ein Netz aufzubauen, das für zukünftige Anforderungen bestens gewappnet ist. Dennoch existieren noch einige offene Fragen - allen voran das Problem der Heterogenität - deren Klärung entscheidend für den Erfolg und die Akzeptanz solcher Systeme ist. Der nächste logische Schritt ist deshalb der Aufbau einfacher Prototypen (so z.B. in [2] und [3] angekündigt), die dann die Praktikabilität und Konkurrenzfähigkeit dieses in meinen Augen erfolgsversprechenden Ansatzes untermauern.

7. LITERATUR

[1] T. Roscoe: *The End of Internet Architecture*, in *Proceedings of the fifth workshop on hot topics in networks (HotNets-V)*, S.55-60, Irvine, USA, November 2006

[2] D. Schwerdel, Z.Dimitrova, A. Siddiqui, B. Reuther, P. Müller: *Composition of Self Descriptive Protocols for Future Network Architectures*, Karlsruhe, Germany, August 2009

[3] B. Reuther, D. Henrici: *A model for service-oriented communication systems (Journal Version)*, in *Journal of Systems Architecture*, June 2008

[4] B. Reuther, J, Götze: *An Approach for an Evolvable Future Internet Architecture*, in *1st Workshop, New Trends in Service & Networking Architectures*, November 2008

[5] N. Van Wembeke, E. Expósito, M. Diaz: *Transport Layer QoS Protocols: The Micro-Protocol approach*, OpenNet Workshop 1, March 2007

[6] M.H. Valipour, B. Amirzafari, Kh. N. Maleki, M. Valadbeigi, N. Daneshpour: *Concepts of Service Orientation in Software Engineering: A Brief Survey*, in *MASAUM Journal of Reviews and Surveys, Vol. 1, No. 3*, S.244-250, November 2009

[7] Universal Description Discovery and Integration (UDDI), online: `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`

[8] WebService Description Language (WSDL), online: `http://www.w3.org/TR/wsdl20/`

[9] Spontaneous Virtual Networks, online: `http://www.spovnet.de`

doi: 10.2313/NET-2011-05-2_03