INTERVIEW MIT UDI DAHAN

Eine Bezeichnung wie SOA braucht es nicht mehr

Der Keynote-Sprecher der Developer Week 2018, Udi Dahan, spricht im Interview über den Sinn von serviceorientierter Architektur (SOA), Microservices und Nachrichtenbussen.

dotnetpro: *Udi, du nennst dich selbst den Software Simplist. Was meinst du damit?*
Udi Dahan: Das geht zurück auf meinen allerersten Blogpost im Jahr 2003 [1]. Ich habe damals einige Zeit nach einem passenden Titel für mich gesucht: Softwarearchitekt, Entwickler, Programmierer, Designer, Ingenieur, Tester, Denker beziehungsweise Evangelist … Ich fand aber, dass ich ein bisschen von all diesen Berufen war. Ich suchte daraufhin nach einem Weg, die pragmatischen Teile meiner Sichtweise auf die Softwareentwicklung zusammenzufassen. Und dann sprang mich dieses Wort an.

Heute bin ich zufrieden, nur Udi Dahan zu sein. Ich brauche heute keine anderen Titel mehr.

Du beschäftigst dich intensiv mit der Entwicklung verteilter Systeme. Aber verteilte Systeme sind alles andere als einfach. Wie passt das zusammen?
Dahan: Ich würde zustimmen, dass viele verteilte Systeme am Ende zu komplex sind. Ich habe die meiste Zeit meiner Karriere damit verbracht, diese Komplexität zu bekämpfen und den Leuten einen besseren, einfacheren Weg zu zeigen.

Udi Dahan
ist einer der weltweit führenden Experten für serviceorientierte Architektur und domänengesteuertes Design sowie Entwickler von NServiceBus, dem beliebtesten Servicebus für .NET. Auf der Developer Week 2018 erklärt er in seiner Keynote, wie Sie künftig bessere verteilte Lösungen schreiben, die dann nicht in einem großen Matschhaufen enden.

Es gab eine Zeit, als serviceorientierte Architektur (SOA) ein Synonym für verteilte Lösungen war. Ist das heute immer noch richtig?
Dahan: Ich würde sagen, dass das Wichtigste, das SOA erreichen sollte, die Ausrichtung von IT am Business war. Das ging definitiv über die Grenzen eines einzelnen Systems hinaus.

In diesem Sinne war die erwartete Softwareumgebung von SOA natürlich verteilt. Leider berücksichtigte die Art und Weise, wie SOA oft praktiziert wurde, die Gestaltung der einzelnen Systeme nicht intensiv genug.

Gibt es mittlerweile ein neues Akronym als Synonym für verteilte Systeme?
Dahan: Heutzutage sind alle Systeme verteilt. Sie bestehen aus ziemlich komplexem JavaScript im Browser auf den Anwendermaschinen und nativem Code auf Smartphones. Alle sprechen mit anderen Servermaschinen, die mit Datenbanken auf anderen Maschinen kommunizieren. Und all das arbeitet mit noch mehr Servern zusammen, die von verschiedenen Firmen betrieben werden.

Ich glaube nicht, dass wir ein spezielles Akronym oder einen Namen für etwas brauchen, was alle Entwickler tun.

Sind Microservices der richtige Weg für verteilte Systeme?
Dahan: Da es sogar einen richtigen Weg zum Aufbau komplexer Softwaresysteme geben könnte, würde ich sagen, dass dies eine entsprechende Modularisierung voraussetzt.

Auch die Auswahl der geeigneten Kommunikationsmuster für die Modulgrenzen – wie In-Memory, Queuing, Pub/Sub und Request/Response – ist wichtig, wobei die Wahl des Blocking- versus Non-Blocking-Ansatzes ebenso wichtig ist. Viele dieser Entscheidungen werden geschäftsspezifisch sein.

Ich glaube nicht, dass der allgemein beschriebene Microservices-Ansatz der synchronen, blockierenden Anfrage/Antwort mittels HTTP-Kommunikation über alle Grenzen hinweg sinnvoll ist.

Ich bin mir nicht einmal sicher, ob das als Microservices bezeichnet werden sollte, da es sich nicht von dem seit Jahrzehnten bestehenden Remote-Procedure-Call-Muster (RPC) unterscheidet.

Was sind die Vorteile von Microservices, was sind die Nachteile?
Dahan: Die meisten Vorteile, die Microservices zugeschrieben werden, sind tatsächlich das Ergebnis einer passenden Mo-

dularität (klein, einfach, leicht verständlich), mit Vorteilen bei der Skalierung, die aus einem passenden Datenbesitz resultieren. Oder einfach das Ergebnis von Load-Balancing-RPC-Anfragen über Kopien desselben Prozesses. Nichts davon ist wirklich neu.

Das Laufzeitverhalten und das Tooling haben sich deutlich verbessert – sei es im Bereich der kontinuierlichen Integration und Bereitstellung, der Container oder der von Cloud-Anbietern zur Verfügung gestellten Infrastruktur.

All dies kann aber auch für sogenannte monolithische Einsätze genutzt werden.

Der Nachteil der synchronen, blockierenden Anfrage/Antwort über HTTP ist, dass das System genau so gekoppelt ist, als ob alle diese Request/Response-Interaktionen im Speicher zwischen regulären Objekten stattfinden würden. Allerdings ist die Latenz wesentlich höher, was zu einer viel schlechteren Performance führt.

In welchen Szenarien würdest du Microservices empfehlen?
Dahan: Ich würde empfehlen, gute Designentscheidungen zu treffen, unabhängig davon, welches das aktuelle Modewort des Tages ist.

Gibt es auch Projekte, bei denen du von der Nutzung von Microservices abraten würdest?
Dahan: Hier gilt das Gleiche wie bei den vorherigen Fragen.

Was hältst du von der Kombination aus Command Query Responsibility Segregation (CQRS) und Microservices?
Dahan: Wie viele andere Ideen, die an Popularität gewonnen haben, wird auch CQRS missverstanden. Der Stil von CQRS, bei dem separate physische Kopien von Daten für das Resultat von Lesezugriffen verwendet werden, wird viel zu oft verwendet.

Die gleiche Datenquelle zu haben, während wir logisch und physisch trennbare APIs für Abfragen und Befehle verwenden, ist ein guter Anfang und hat eine gewisse Ausrichtung auf Microservices.

Was nicht oft genug angewendet wird, ist die tiefere Geschäftsanalyse der CQRS-Prinzipien, sodass Befehle fast immer logisch erfolgreich verarbeitet werden, was zu möglicherweise sehr unterschiedlichen Benutzeroberflächen und Geschäftslogik führt. Das ist fast völlig orthogonal zu den Mikrodiensten.

Was ist deiner Meinung nach die beste Infrastruktur, um Microservices zu hosten?
Dahan: Es sollten Infrastrukturentscheidungen getroffen werden, die der Modularität, dem Datenbesitz und dem Kommunikationsstil des Systems angemessen sind. Es gibt hierbei kein „Bestes“.

Ist es eine gute Idee, ein verteiltes System ohne Nachrichtenbus einzurichten?
Dahan: Asynchrone, warteschlangenbasierte Ansätze haben in vielen verteilten Systemen ihren Platz.

Wie bei Caches, Datenbanken und anderen nichttrivialen Technologien kann es schwierig sein, zu entscheiden, welche ihrer Funktionen in welcher Weise für den konkreten Anwendungsfall verwendet werden sollen.

Produkte wie NServiceBus versuchen, dem entgegenzuwirken, indem sie einen Ansatz bieten, der viel von dieser Komplexität verbirgt. Während beispielsweise NServiceBus für die Teile eines Systems eine gute Wahl darstellt, die eine hohe Zuverlässigkeit und Konsistenz erfordern, ist ein Nachrichtenbus in Systemteilen, die einen extrem hohen Durchsatz (100 000 Nachrichten pro Sekunde und mehr) und/oder eine niedrige Latenz (einstellige Millisekunden und weniger) benötigen, überhaupt keine gute Wahl. ■

[1] Erster Blogpost von Udi Dahan, Starting Blogging, www.dotnetpro.de/SL1807Interview1