Reactive Extensions – Alles ist ein Event!

Einführung in die Programmierung mit Eventstreams

Marko Beelmann, Philips Medizinsysteme Böblingen

Die zunehmende Vernetzung von Geräten oder auch die Nutzung von Cloud-Services stellt auch an die Software-Entwicklung neue Herausforderungen. Messwerte von Sensoren oder Push-Benachrichtigungen aus der Cloud, immer mehr Events müssen verarbeitet werden. Mit Hilfe der Reactive Extensions können Events in Streams umgewandelt und leicht koordiniert werden. Durch Hilfe von Schedulern wird auch die asynchrone Verarbeitung deutlich erleichtert.

Jeder der ein Smartphone besitzt, schätzt die vielen Funktionen der Geräte. Sei es Navigation, Fitness-Tracking, Chat, Wasserwage oder Top-News direkt auf das Smartphone. All diese Funktionen beherrscht das Smartphone mehr oder weniger gut. Für die Softwareentwicklung heißt das auf der anderen Seite aber auch, auf viele Events zu reagieren. Seien es Koordinaten vom GPS, Daten von Sensoren oder Push-Nachrichten aus der Cloud. Diese Flut an Events muss eine Software intelligent und für den Nutzer transparent verarbeiten können. Bei Smartphones sollte eine Oberfläche aufgrund einer Eventflut nicht in einen blockierenden Zustand geraten.

Es sind aber nicht nur Smartphones die dieser „Eventflut“ gegenüber stehen. Auch die zunehmende Vernetzung der uns umgebenen Geräte wird zu einer großen Anzahl von Events führen, die wiederum von einer Software verarbeitet werden muss. Auch die immer weiter verbreiteten Cloud-Services sorgen für einen kontinuierlichen Fluss an Events. Der klassische Ansatz um auf diese asynchronen Events zu reagieren sind Callbacks, die beim Eintreffen eines Events gerufen werden. Allerdings kann dieser Ansatz bei einer großen Anzahl von Events und vor allem auch vieler verschiedener Eventquellen sehr unübersichtlich und ineffizient werden. Eventquellen mit einer hohen Frequenz an Events können eine Software schnell in Performanz-Problem bringen.

Um solch beschriebene Probleme zu verhindern bieten sich die Reactive Extensions an. Diese Software-Bibliothek(en) kommt ursprünglich aus dem Hause Microsoft und wurden 2012 als Open-Source freigegeben. Seit dieser Zeit wächst die Zahl der unterstützen Programmiersprachen kontinuierlich an. Für so ziemlich jede verbreitete Sprache existiert eine Erweiterung die das Konzept des „Reactive Programming“ unterstützt. Doch worum geht es dabei eigentlich genau? Im Kern geht es vor allem um die Abstraktion von Events oder anderer Datenquellen als ein Stream von Events.

Diese Eventstreams werden durch ein sogenanntes Obserservable abgebildet. Diese Observables ähneln vom Grundsatz dem Observer-Pattern der „Gang of Four“. Durch eine „Subscription“ kann der Entwickler sich auf das Observable registrieren. Für eine erfolgreiche Registrierung muss der Aufrufer einen sogenannten Observer übergeben. Dieser enthält bis zu drei Callbacks die auf den Eventstream reagieren sollen. Ein Callback ist für die eigentliche Eventverarbeitung gedacht. Taucht ein Event innerhalb des Observerable auf, so wird der Callback gerufen. In der Regel wird dann vom

Observable die Methode OnNext() des Observer gerufen. Neben der OnNext() Methode besitzt der Observer noch zwei weitere Callbacks. OnError() und OnCompleted() geben den Status des Observable wieder. OnError() wird bei einem Fehler in dem EventStream aufgerufen. Somit hat der Entwickler die Möglichkeit explizit auf Fehler in einem Eventstream zu reagieren. Wird OnCompleted() gerufen, so wird von diesem Observable kein Event mehr ausgehen. Der Eventstream ist geschlossen.

Vor allem die Benachrichtigung über Fehler im Eventstream kann sehr nützlich sein um sich z.B. wieder direkt auf einen neuen Stream zu registrieren oder den Nutzer über die Probleme einer Eventquelle zu informieren. Aber auch der Aufruf von OnCompleted kann nützlich sein um z.B. die Netzwerkverbindung eines Eventstreams zu schließen.

Richtig nützlich wird die Library aber durch die Komposition und Filterung solcher Eventstreams. Es existiert eine große Anzahl an Operatoren bzw. Methoden um den Stream auf die eigenen Bedürfnisse anzupassen. Der eigentliche Eventstream, die echte Quelle der Events, bleibt dabei allerdings unverändert. So kann ein Eventstream mit einer hohen Eventfrequenz durch entsprechende Operatoren entschärft werden. Liefert beispielsweise ein Sensor alle 10ms sein Wert, so kann der Operator Sample() einen neuen Eventstream erstellen, der nur alle 100ms den aktuellen Wert meldet. Nützlich könnte hier aber auch eine Eventstream sein, der der mit jedem neuen Wert auch den bisherigen Durchschnitt berechnet. Sinnvoll ist es in diesem Fall den bereits entschärften Eventstream als Basis zu nutzen. Somit wird ein zweiter Eventstream erstellt, der als Event den Durschnitt aller bisher aufgetretenen Event-Werte berechnet. Abb. 1 verdeutlicht den Verlauf der Events aller drei Eventstreams. Diese Form von Diagrammen wird als Marble-Diagramm bezeichnet.

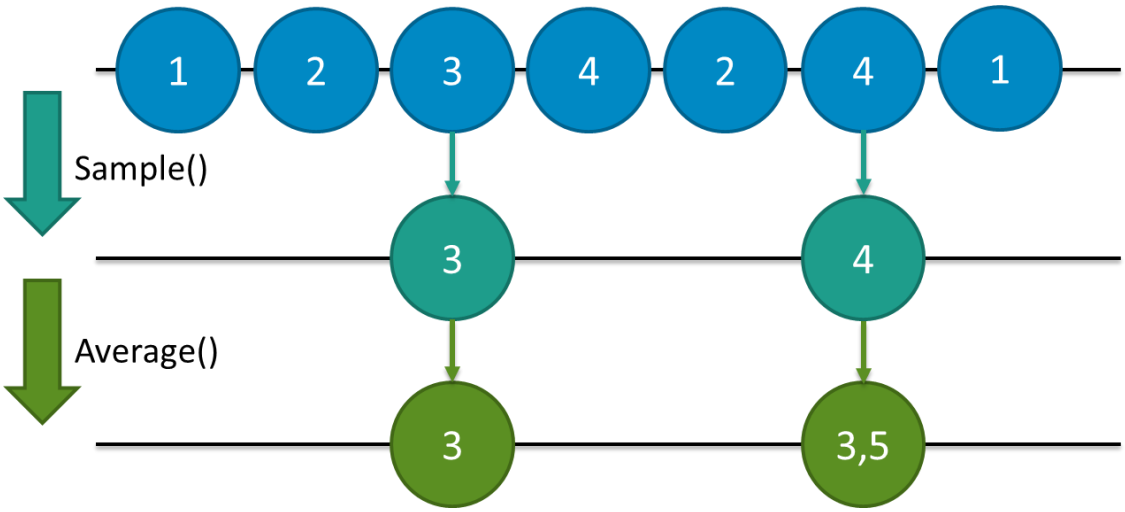

Abb. 1: Aus einem Eventstream werden drei. Mit Hilfe der Methoden Sample() und Average().

Die Reactive Extensions beinhalten eine große Zahl an Methoden, die aus einem Stream weitere Streams ableiten können.

Eine weitere Stärke der Bibliothek liegt in der Kombinationsmöglichkeit von Eventstreams. Neben dem Erstellen von neuen Streams ist es genauso möglich auch Streams miteinander zu kombinieren. Dies kann vor allem sinnvoll sein, wenn es darum geht mehrere Eventstreams ein einem zusammenzufassen. Mit der Methode Merge() wird in einem vorhandenen Eventstream ein zweiter Stream eingefügt. Ein typisches Beispiel sind Eventstreams von Sensoren, die durch einen dedizierten Thread verarbeitet werden. Dabei werden die Streams der einzelnen Sensoren in einem neuen zentralen Stream zusammengeführt, welcher dann in einem exklusiven Thread verarbeitet wird.

Daneben gibt es noch weitere nützliche Methoden welche Eventstreams miteinander vereinen. Die Methode Zip() generiert aus zwei Streams einen neuen Stream, der erst ein Event auslöst, wenn die beiden zugrunde liegenden Streams jeweils ein Event auslösen.

Neben der Möglichkeit „echte“ Events zu modellieren bietet das Modell des Observable/Observer einen anderen sehr interessanten Aspekt. Mit Hilfe von Observables ist es möglich, Arrays/Collections über die Zeit abzubilden. Über die Zeit heißt in diesem Sinne das die Daten in aller Regel asynchron dem Konsumenten zur Verfügung gestellt werden.

Ein Vergleich zwischen dem klassischen Iterator-Pattern und einem Observable sollte den Unterschied klar machen. Der Iterator ist von einem Anbieter (Aggregat) abfragbar und ermöglicht das sequentielle Abfragen der Daten. Mit Hilfe von Methoden wie z.B. MoveNext() wird zum nächsten sequentiellen Element gesprungen. An dieser Stelle kann es ein Problem geben, denn MoveNext() würde so lange blockieren bis das nächste Element bereit ist. Dies kann aber z.B. bei I/O basierten Quellen, wie z.B. einer Internetverbindung, eine lange Wartezeit mit sich bringen. Bei großen Anwendungen oder Services kann dies zu einem Problem werden, da viele Threads einfach mit blockierenden Methoden ungenutzt Ressourcen verbrauchen. Diese könnten an andere Stelle sinnvoller eingesetzt werden.

Aus Sicht des Konsumenten müssen die Daten aktiv aus der Quelle geholt werden. Daher wird dieses Model auch als Pull-Model bezeichnet.

Wäre es nicht einfacher der Konsument bekommt die angefragten Daten einfach zugestellt, sobald diese vorhanden sind? Genau an dieser Stelle kommt das Push-Model ins Spiel. Neben der Zustellung der Daten wird der Konsument auch informiert, wenn keine Daten mehr vorhanden sind. Sollte die Quelle einen Fehler bzw. Exception melden, wird diese ebenso gemeldet. Genau dieses Push-Model wird durch die Observables repräsentiert. Durch dieses Konzept sollte der Konsument nicht mehr in die Lage kommen blockierend auf das nächste Element einer Sequenz zu warten. Durch die Verwendung von Schedulern ist auch das Koordinieren von nebenläufigen Operationen etwas entschärft. Viele Operatoren der Reactive Extensions ermöglichen eine einfache und komfortable Verarbeitung.

Da solche Datenquellen aber letztendlich auch nur Eventstreams repräsentieren, können diese beliebig mit anderen Streams kombiniert und verbunden werden.
Dem Entwickler wird damit ein Werkzeug in die Hand gegeben, mit dem er Eventstreams einfach und effizient verarbeiten, kombinieren und (asynchron) koordinieren kann.

Somit bietet sich dieses Konzept vor allem bei Software an, die z.B. mit sehr vielen Events arbeitet oder so designt wurde, Stichwort „Event Based Components“. Aber auch bei der (asynchronen) Verarbeitung großer Datenmengen kann diese Library überzeugen.

Quellen

Introduction to RX – auch als kostenloses eBook (http://www.introtorx.com/)
Übersicht für eine Vielzahl von Programmiersprachen (http://reactivex.io/)
Das reaktive Manifesto (http://www.reactivemanifesto.org/)
The Cloud Programmability Group at Microsoft (https://github.com/Reactive-Extensions/)

Autor

Marko Beelmann ist bei Philips Medizinsysteme Böblingen tätig.
Funktion: Aktive Produktentwicklung im Bereich Patientenüberwachung mit dem Schwerpunkt PC basierte Erweiterungen und Systemlösungen für Echtzeit-Patientenmonitore.
Fachliche Schwerpunkte/Interessen: Entwicklungen im .NET Umfeld, Multithreading, Reactive Extensions, Application Lifecycle Management, Open Source und Softwarearchitektur.

Kontakt

Email: Marko.Beelmann@gmail.com