Client-directed Query

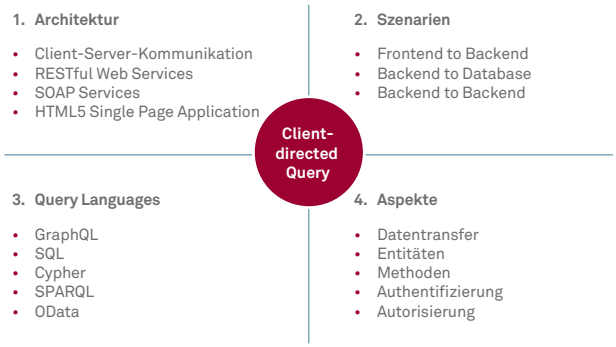
Eine universelle Schnittstelle für den Datenkonsumenten

Applikationen, die auf SOAP- oder REST-Architekturen setzen, haben sich im letzten Jahrzehnt bewährt. Es gibt jedoch Grenzen dieser Architekturen. Der Ansatz von Client-directed Query überwindet diese Grenzen und ermöglicht eine generische Anbindung von unterschiedlichen Clients an dasselbe Backend. Die Clients lassen sich so zu einem beliebigen Zeitpunkt definieren, mehrere Datenanfragen zu einer einzigen zusammenfassen und obendrein entscheidet der Client selbst über die Granularität der Daten.

Definition

Das Prinzip des Client-directed Query sieht vor, einem oder mehreren Clients so weit wie möglich die Kontrolle über den Inhalt und die Granularität ihrer vom Backend benötigten und gelieferten Daten zu überantworten. So lässt sich eine Vielzahl von einzelnen REST-, SOAP- oder RPC-Schnittstellen durch eine einzelne, zentrale Schnittstelle für alle Client-Anwendungen ersetzen.

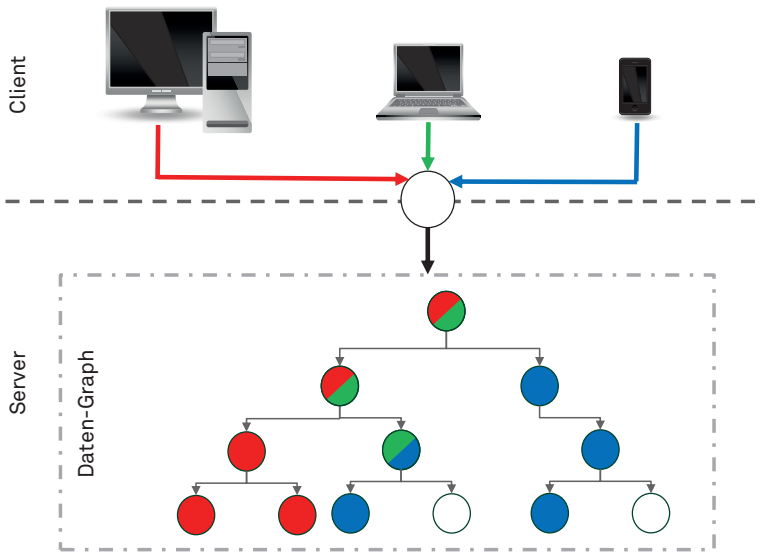
Der Mehrwert: Jede Client-Anwendung kann ihre individuellen Datenabfragen anhand einer DSL-basierten Abfragesprache definieren und stets die im aktuellen Anwendungskontext benötigten Daten abrufen. Starre, vordefinierte Schnittstellen weichen damit einer flexiblen und ausdrucksstarken Abfrageschnittstelle. Diese Datenabfragen fördern das Uniform Access Principle, sodass Lese- und Schreibzugriffe über dieselbe Notation der Attribute ermöglicht wird.



Beispiel: Anstatt „getSurname“ und „setSurname“ zu verwenden, kommt in beiden Fällen schlicht „surname“ zum Einsatz.

Referenzszenario

Verschiedene, sehr spezifische Situationen bieten sich für den Ansatz der Client-directed Querys an. Aber insbesondere in den nachfolgenden fünf sind sie möglich und sinnvoll, erst recht in Kombination.



Unbekannte Anzahl: Die Anzahl der möglichen Client-Anwendungen ist zu Beginn der Entwicklung weder bekannt noch absehbar. Gleichzeitig soll die Anzahl auch nicht durch Vorgaben bei der Backend-Entwicklung unnötig eingeschränkt werden.

Bündelung: Die Anzahl der Datenzugriffe auf das Backend soll so gering wie möglich ausfallen, indem mehrere zu einer einzelnen Abfrage gebündelt werden.

Wirtschaftlichkeit: Eine oder mehrere serverseitige Sonderschnittstellen je Client-Anwendung sollen allein schon aus wirtschaftlichen Gründen verhindert werden.

Komplexität: Die Datenbasis der Anwendung beruht auf einem komplexen Graphen.

Unbeständig: Es ist absehbar, dass die Datenbasis oft erweitert oder gar verändert wird. Die Client-Anwendungen sollen diese Änderungen möglichst schnell adaptieren können.

Potenzial

Der Mehrwert von Client-directed Querys ergibt sich aus der hohen Flexibilität der Schnittstellenlösung. Müssen neue betriebliche Informationssysteme konzipiert werden, empfiehlt es sich, Client-directed Querys anderen etablierten Varianten vorzuziehen. Denn die Kosten für die Entwicklung und Pflege der Schnittstelle sind stets kalkulierbar, auch bei der späteren Anbindung von Fremdsystemen oder der Entwicklung weiterer Clients.

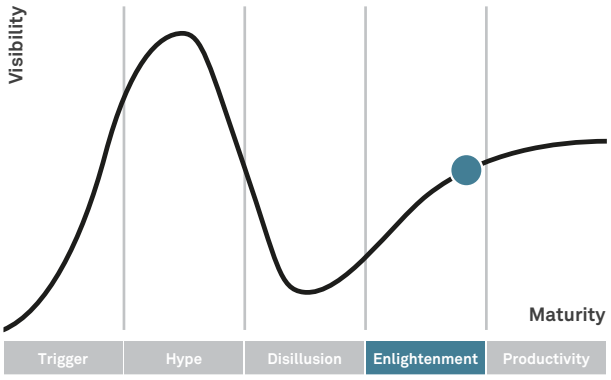
Sofern ein bestehendes betriebliches Informationssystem seine Daten mehreren anderen Systemen bereitstellt und über die Menge der Schnittstellenkontrakte feststellen kann, dass jedes System die Daten in anderer Form benötigt, sollte ein Wechsel zu Client-directed Querys in Betracht gezogen werden.

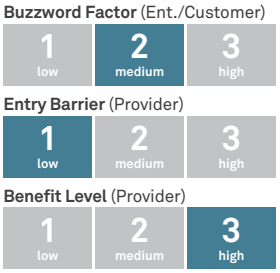
Reifegrad

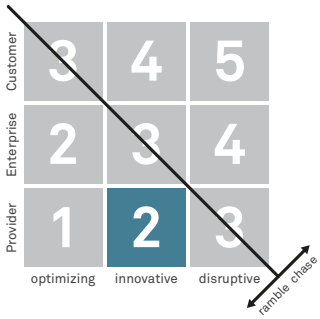
Client-directed Query ist kein neues Konzept und in Form von SQL als Abfragesprache für relationale Datenbanksysteme schon seit Jahrzehnten etabliert. Namhafte Hersteller wie Facebook, Github und Netflix nutzen den Ansatz der Client-directed Queries, in den Varianten GraphQL und Falcor, für ihre Produkte. Diese Firmen bieten verlässliche Dienste an, trotz der teils massiven Datenmenge, die sie bewegen müssen.

Marktübersicht

Vor allem Facebooks GraphQL und Netflix's Falcor sind durch Client-directed Querys inspiriert. GraphQL wurde in diesem Zuge bereits auf viele Programmiersprachen übertragen und steht für Java, Javascript, Go, Python, Perl, C# und weitere zur Verfügung.







Dementsprechend ist das Ökosystem enorm gewachsen. Es umfasst zahlreiche Tools und Frameworks, etwa für Input-Output in Form von GraphQL IO, User Interfaces in Form von GraphiQL und Data Access in Form des Moduls GraphQL-Tools-Sequelize, ein Object-Relational-Mapper. Ferner existieren Linter, IDE-Plugins und vieles mehr.

Alternativen

Eine sinnvolle Alternative zu Client-directed Query existiert nicht, weil dieses Konzept explizit solche Probleme adressiert, die in den aktuell häufig verwendeten Schnittstellenlösungen immanent sind. Möglich wäre es zwar, die individuellen Abfragen für jeden Client mit einzelnen REST- oder SOAP-Schnittstellen abzubilden. Das endet aber immer in einer starren Schnittstellendefinition. Kommen neue Clients mit neuen Datenanfragen hinzu, dann müssen dediziert weitere Schnittstellen für diese Clients gebaut werden. Ein Einsatz von REST mit HATEOAS (Hypermedia as the Engine of Application State) ermöglicht zwar eine erforschbare Schnittstelle, jedoch werden dabei immer noch zu viele Anfragen gestellt und eine ausdrucksstarke Abfragesprache fehlt.

Pro	Contra
Datenübertagung verschiedener Anfragen kann in einer Abfrage gebündelt werden	Performanz und deren Optimierung ist bei einer generischen SST ein größeres Problem als bei dedizierten Schnittstellen
reduziert die Schnittstellenanzahl und deren Abstimmung deutlich	spezielle Anfragen (Login, Logout, Uploads, Downloads, Streaming) sind in der DSL nicht vernünftig abbildbar
Entwicklung der Server-Schnittstelle kann zeitlich vor und unabhängig vom Client erfolgen	zusätzliche Komplexität durch Query-Auswertungs-Engine und somit komplizierte Fehlersuche
Flexibilität in der Abfrage	
explorative Schnittstelle	

https://msg.direct/techrefresh

Stand: September 2018

msg systems ag

Robert-Bürkle-Straße 1 | 85737 Ismaning/München | Telefon: +49 89 96101-0 | Fax: +49 89 96101-1113 | www.msg.group | info@msg.group