Client-Oriented API Design

CMPU4023 - Enterprise Application Development

Service-Oriented API Design

- Before considering client-oriented API design it is useful to reprise the more conventional service-oriented approach such as REST
- Recall that, in the RESTful API pattern, the service state and behaviour are modeled as resources
- The client interacts with one or more of these resources using a message passing API having a well-defined URI scheme over HTTP
- This resource-oriented approach is driven by service-side design considerations, namely how to abstract the service's functionality and capability
- In effect, this gives rise to a static interface specification

REST Consumption in Practice

- In practice, a statically-defined service-oriented API specification may not provide the best way for a given client to access or update the service resources
- When viewing, a client, often a user interface, typically has to present some aggregated view of two or more resources within some nested, hierarchical structure containing some, but not necessarily, all of the resource properties fetched from the service
- When updating, a client may have to change one or more properties across two or more logically-related resources
- A pure RESTful API doesn’t naturally fit either of these scenarios

Efficiency, Throughput and Performance

- As a consequence, a client may often have to make multiple API calls to the service to perform some read or update function
- This is a problem for several reasons:

  - Resource attributes may be fetched even if they are not needed by the client
  - The network overhead incurred negatively impacts service throughput and client performance
  - Updates across multiple resources are not necessarily transactionally safe

- A client with specific needs does not necessarily get to dictate these requirements back into the service API design

A Possible Solution

- In cases where particular client-side use cases can or need to be catered for, the service API can choose to export calls which more closely match its read and update requirements
- In other words, the REST interface can be extended in an ad hoc manner to cater for business case-driven *special cases*
- However, this approach doesn't really scale and it, by necessity, creates additional complexity on the server side implementation, makes it harder to develop, harder to understand and harder to test

The Real Problem

- The real issue here is the static nature a service-oriented API definition which is, in effect, a frozen-in-time view of the consumer requirements often with the objective of making the service easier to develop and maintain
- What's really needed here is a more dynamic API specification, one which didn't try to be prescriptive or predictive about what clients would need
- This kind of API would allow the client to discover the service capabilities and supported operations and then allow the client to ask for just what it needed (and nothing else) and in a format which would already be more logically closer to the client presentation and control abstractions

Towards a Dynamic Interface Specification

- Suppose, as a thought experiment, a relational database-backed service could expose a SQL interface to a client allowing it to query or update whatever entities it required
- Such a client could could craft exactly the right calls to the service to process data in the most efficient way possible
- This unrealistic scenario illustrates what we lose when we put REST APIs in front of relational systems
- For the benefit of security and higher order abstraction we lose a lot of flexibility
- What we need is a hybrid of RESTful principles with the power of SQL

Summary

- A REST interface represents a static API specification exposing a high level abstraction of the service state and behviour
- REST consumers may not necessarily find this kind of API convenient, efficient or even transactionally safe to use in many cases
- When viewing or updating, a client may have to wastefully make multiple API calls to the service which may return data it doesn't even need
- It is the static nature of the REST API specification which lends itself to such an inflexible API design
- In the next section, we will look at an alternative API specification system which incorporates RESTful API principles but which is more client-oriented

Client-Oriented API Design

CMPU4023 - Enterprise Application Development