Dissertation Proposal

Jim Higson

2013

Application of my dissertation

For a system engineered in a SOA style many of the resources a program accesses are resident on a remote machine. Although programmers often focus attention on the execution time of their algorithms, because transmission over a network can be as much as 10^6 times slower than local access, the interval in which a program waits for input usually contributes more to the degradation of performance than any local concern. As such the the sage use of io should rank above most other optimisation considerations when considering the efficiency of many modern programs.

For all but single-packet messages, data sent over a network is readable whilst transmission is still in progress. Hence, it is possible to view almost any transmission through the lens of a stream even if the sender of the data did not intend for it to be thought of it in this way and by doing so it is possible to start using the data from the resource earlier. In not utilising the progressive nature of resource availability I propose that today's common REST client libraries are failing to make best use of network bandwidth.

REST today is not solely the domain of server-to-server communication. It is also commonly employed under various AJAX patterns to make server-side resources available to client-side software executing locally inside a user's web browser. Data transmission often takes place over the mobile internet but AJAX clients commonly used in web browsers do not deal well with the fallible nature their networks. If a connection is lost whilst a message is in transit then the data downloaded to that point is discarded. Whilst it would of course be preferable to receive the entire resource requested, for many common use cases the incomplete data is nonetheless of considerable value. By discarding remote data at a time when the network is unreliable we are wasting a valuable resource at the time when it is the most scarce. As a practical example, for an application downloading an email inbox, if the connection is lost during transmission the program should be able to display *some* of the emails in preference to *none*.

Over the last decade or so a significant shift in web application architecture has been to push the presentation layer onto the client side. Rather than deliver

pages to a browser, data is sent instead so that it is the responsibility of an application running inside the browser to populate the page. Prior to this AJAX age progressive html rendering allowed a transmitted page to be viewed incrementally as it arrived over the network providing a fluid perception of performance. By not facilitating the use of a REST response before the entire message has been received, I observe that more recent web architecture has in this regard taken a regressive step. Given that the underlying http transport is the same, I propose that it should be equally possible to progressively consider content delivered as data as it is for content delivered as markup.

Prior art and new contributions

My thesis is centred on the creation of a novel style of REST client library which allows programmer specified items of interest from a resource to be used while the resource streams in and even if the download is only partially successful.

Sax provides progressive parsing to the XML world. SAX parsers however are little more than tokenisers, providing a low level interface which notifies the programmer of tokens as they are received. This presents poor developer ergonomics, often requiring that the programmer implement the recording of state with regard to the nodes that they have seen. For DOM-style parsing the programmer rarely directly concerns themselves with XML, taking advantage of the wealth of generic tools which automate the translation of markup into domain model objects as per a declarative configuration. Conversely, for SAX the equivalent logic is usually implemented imperatively; it to be difficult to read and programmed once per usage rather than programmed as the combination of reusable parts. For this reason, SAX is much less common and only generally used for fringe cases in which messages are extremely large or memory extremely limited.

I observe that this popularity of parsing models which require the whole message to be downloaded before any inspection can start hampers the performance of REST systems and propose the creation of a new, third way which combines the pleasant developer ergonomics of DOM with the progressive nature of SAX.

Delivery methodology

My thesis will be developed ‘in the open’ by committing all programming and analysis to a public Github repository. This will hopefully allow members of the developer community to comment on the work and contribute suggestions as the work progresses. With the creative process reflecting the thing that it creates, I plan to use Kanban to iteratively deliver my iterative REST client. To allow as wide an adoption as possible, it will be licenced for uncomplicated inclusion and modification under the 2-clause BSD licence.

TDD will be used to gain a reasonable assurance of correctness of the software. As well as demonstrating correctness against a set of known inputs, TDD will drive many of the design decisions in the programming. Designing so as to be easy testable encourages the separation of programming into smaller parts which each do one thing, such that each text becomes a trivial matter of asserting a single facet. As well as decomposition away from monoliths, TDD also encourages traits such as statelessness in programming because an assertion of correctness is more universally applicable once we can reasonably assume that the behaviour under test will not change as some state hidden inside the program varies.

The program design will be initially an exercise in creating the easiest expression that can possibly work and via constant work towards the emergence of elegance.

Timescales

I am registered on the Software Engineering Program until December 2013. I plan to complete and deliver the dissertation in late Summer or Autumn 2013.

Summary of deliverables

Having observed that it may be possible to improve over the current common use of REST, the question I am asking is *"What is the smallest possible work which is likely to bring a significant improvement in this area?"* I plan to focus on creating as small a library as is possible to meet my goals. The feature set will be minimal but contain no obvious omissions. To this end I will be focusing on receiving http responses rather than sending them. Firstly, asynchronous servers which encourage us to view REST through a streaming lens do already have some prominence. Secondly, In the creation of a receiver alone, an increased performance should be possible even given an entirely synchronous sender.

I propose to deliver a progressive rest client as a javascript library which builds on existing http libraries and may be deployed into either a server or a browser context. My ambitions for wider usage motivate a focus on programmer ergonomics, example-lead documentation and a packaging which allows drop-in replacement of existing tools. Because web programming is size-conscious I will deliver a micro-library; the size as sent to a web browser will not exceed 5KiB.

Finally, I will evaluate the effectiveness of my solution in comparison with existing tools in terms of the compactness of code, ease of programming, fault tolerance and speed. Speed will be judged both in terms of user perception and in terms of absolute total time required to complete realistic tasks such as the aggregation of data from several sources.