Dissertation Proposal

Jim Higson

2013

Application of my dissertation

For a system engineered in a SOA style, many of the resources required by a program reside on a remote machine. Although programmers often focus their concern on the execution speed 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 is waiting 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. Although some messages require the whole to be delivered before any part of it is useful, this is not the most typical case. So long as a partial message may be meaningfully interpreted, we can see that it should be possible to start using the resource earlier by examining it while transit is ongoing than if we had waited for the complete data. By not utilising the incremental, progressive nature of resource availability I propose that today's common REST client libraries are not making the most efficient use of remote resources.

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 via AJAX but the AJAX clients which are commonly used in web browsers do not deal well with the fallible nature of these networks. If a connection is lost whilst a message is in transit then the data downloaded to that point is discarded. Whilst of course it would be preferable to receive the entire resource, for many common use cases the incomplete data is nonetheless of non-zero 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 to the client side. Rather than deliver *pages* to a browser, *data* is sent so that an application running inside the browser is responsible for using this data 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, giving a fluid perception of performance. In contrast, by not facilitating the use of an AJAX 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 for content delivered as *data* it should be no less possible to employ an incremental consideration as were the case for content delivered as *markup*.

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

Prior art

In the XML world progressive parsing is provided by SAX. SAX parsers however exist at a level of abstraction only slightly above that of tokenisers, providing a low level interface which notifies the programmer of basic syntax as it is parsed. This presents poor developer ergonomics, requiring that the programmer implement the recording of state with regard to the nodes that they have seen. For non-incremental DOM-style XML parsers the programmer rarely directly concerns themselves with markup features. More likely is that they will take advantage of the wealth of modern, generic tools which automate a translation of markup into domain model objects as per a declarative configuration. For programmers using SAX, a conversion to their domain objects is usually implemented imperatively. This programming tends to be difficult to read and programmed once per usage rather than assembled as the combination of reusable parts. For this reason the use of SAX is usually reserved for fringe cases in which messages are extremely large or memory extremely scarce.

I hope that my REST client library will establish a third way, combining pleasant developer ergonomics of DOM parsing with the progressive nature of SAX parsing.

Delivery methodology

My thesis will be developed 'in the open' by committing all programming and analysis to a public Github repository. This will allow members of the developer community to contribute with comments and suggestions as the project unfolds. 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.

Test Driven Development, or *TDD*, will be used to gain a reasonable assurance of the 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 easily testable encourages the separation of programming into smaller parts which each do one thing, such that each test becomes a trivial matter of specifying the correct behaviour of 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 assume that the behaviour under test will not vary as some hidden state inside the program changes.

Timescales

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

Summary of deliverables

Having observed that it may be possible to improve over the presently 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?"*, as such I plan to focus tightly on as small a library as is possible to meet my goals. From the initial problem I am zooming in to concentrate on receiving http responses whilst de-scoping sending them. Firstly, asynchronous servers which encourage us to view the response as a stream do already have some prominence. Secondly, In the creation of a receiver alone, because all responses may be considered as streams, an increased performance should be possible even given an entirely synchronous sender architecture.

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. The feature set will be minimal but contain no obvious omissions. 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.