Dissertation Proposal

Jim Higson

2013

Application of my dissertation

In the profession of software engineering, Much attention is focused on the speed of execution of a program, often neglecting to acknowledge that for a system engineered in a SOA style many of the resources a program accesses are resident on a remote machine. 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 io can be much more significant to overall performance than execution time on the CPU. As such the efficiency of many modern programs is much more dependant on sage use of io than it is on other optimisation considerations. For any non-trivial message sent as text over a network, the transmission is readable before it is complete but by ignoring the progressive nature of the resource availability I propose that today's common REST client libraries are failing to make best use of network bandwidth, this most precious of resources.

REST today is not solely used for communication between servers, it is also widely employed under the AJAX pattern to expose server-side resources to clients executing as scripts in web browsers. often sending data to mobile devices such as phones and tablets. Once we are operating over mobile connections, network bandwidth becomes an even more precious resource. However, existing REST clients do not deal well with the fallible nature of the mobile internet; if the connection is lost while a message is in transit then the potentially valuable data downloaded so far is discarded. For many common use cases the incomplete data would be complete enough to be useful to the user and by discarding it at a time when the network is unreliable we are wasting valuable resources at the time when they are most scarce. As a practical example, for a web 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*.

Prior to the web's AJAX age, browser implementations of progressive html rendering allowed server-side generated pages to be viewed in parts as they arrive over the network. By not allowing the use of a downloaded resource before the entire message has been received today's AJAX clients have taken a backwards step in terms of the fluid perception of performance. I propose that it should be

equally possible to program a progressive presentation of the data regardless of which side of the network the html is generated on.

To improve all of these areas, my thesis is centred on the creation of a novel style of REST client library which allows use of interesting parts of the resource before the entire resource has been downloaded and even if the download is never entirely completed.

Advances over prior art

Progressive parsers exist already as a SAX-style but are much less than the DOM-style alternative. SAX parsers are little more than tokenizers, providing a very low level interface which notifies of tokens as they are received. This presents a difficult API, requiring the programmer to record state regarding nodes that they have seen and implement their own pattern-matching. Whereas for DOM-style parsing there exists a wealth of tools to transform structured text into domain model objects based on declarative configuration, for SAX-style parsing this is usually done in the programming language itself. This logic tends to be difficult to read and programmed once per usage rather than assembled from easily reusable parts. For this reason, SAX parsing is much less common than DOM parding when connecting to REST services. I observe that this trend towards DOM style parsing which requires the whole resource to be downloaded before inspection can commence hampers the performance of the REST paradigm and propose the creation of a third way combining the developer ergonomics of DOM with the responsiveness of SAX.

In which a callback call is received not just when the whole resource is downloaded but for every interesting part which is seen while the transfer is ongoing. The definition of 'interesting' will be generic and accommodating enough so as to apply to any data domain and allow any granularity of interest, from large object to individual datums. With just a few lines of programming

Http libraries feeding into the parser. In browser, generally single callback when whole message received.

Client-side web scripting via Javascript is a field which at inception contributed no more than small, frequently gimmicky, dynamic features added to otherwise static webpages. Today the scope and power of client side scripting has increased to the extent that the entire interface for large, complex applications is often programmed in this way. These applications are not limited to running under traditional web browsers but also include mobile apps and desktop software.

Focus tightly on creating a small, high-quality piece of code with a narrow feature set but no obvious omissions. Hence, only client, not server, tools already exist to send asynchronously. Easier to improve REST with a client than a server because async clients still bring benefits with existing servers but

Delivery methodology

My thesis will be developed 'in the open' by committing all code and writing to a public Github repository. This should allow members of the developer community to contribute comments and suggestions as the work progresses. I plan to use Kanban to deliver my dissertation, including the written parts. Because Kanban focusses on always having a potentially releasable product, it mitigates problems which could otherwise lead to non-delivery and allows the direction to be changed as necessary. For each unit of work (under Kanban, a card), an entire vertical slice of planning, design, implementation and reflection must be complete before going onto the next card. Alongside each software feature, every written chapter will be expanded and refactored in much the same way as the code. Just as for well designed software, the order of implementation should not be apparent to a user, my plan is that the written work should not feel disjointed for having been written non-sequentially. I plan to manage the Kanban process using paper only, with cards on a physical board.

Will code using TDD and design code to be easily testable via TDD. This includes stateless and separation of programming into many collaborating parts. Constant refactoring, emergent design.

Timescales

I am registered on the Software Engineering Program until December. I plan to complete and deliver the dissertation towards the end of Summer 2013.

Summary of deliverables

I propose to deliver a progressive rest client as a javascript micro-library which runs on either the server or client side and sits on top of existing http libraries. My ambitions for wider usage motivate a focus programmer ergonomics, packaging so as to allow drop-in replacement of today's commonly used tools but also liberal BSD-style licencing so as to encourage inclusion in free and non-free software projects. Because web programming is size-conscious I will deliver a micro library meaning that the size on the wire when sent to a web browser will not exceed 5kib.

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