

Design Strategies for a Distributed Web

Gareth Rushgrove

morethanseven.net

gareth@morethanseven.net

Web developers are increasingly turning to frameworks like [Django](#) and [Rails](#) in order to quickly develop high quality sites and application. But even with varied Open Source offerings we're still re-inventing the wheel when it comes to many common components. A vibrant ecosystem of commercial grade API providers, such as Amazon with their [Amazon Web Services](#) platform and now Google with [App Engine](#) could change all that. But what needs to happen to make this new web services dream a reality and where are the potential pitfalls for successful apps?

From language frameworks to APIs

Everyone is making use of increasingly mature and stable web application or javascript frameworks these days. That means in the main we've stopped reinventing things such as routing, event handling and object relational mapping, but we're all still building very similar components again and again. Frameworks might allow us to solve lots of fine grained problems easily but APIs could let us solve common course grained problems quicker.

Building blocks for your applications

There are already a few examples in the wild of APIs designed to be used as part of your application development process. Services where the API is the product, rather than an enhancement to an existing site or application. Amazon has been leading the way in providing remote services such as [S3](#), [EC2](#) and [SimpleDB](#). But other companies large and small are also making moves. Google first launched a perfect API product example in The [Social Graph API](#) and have since followed up with the Google App Engine platform. Microsoft are also rumoured to be looking to enter the game with a project code-named Red Dog and have [SQL Server Data Services](#) currently in beta. Smaller companies too, like the Scotland based [Flexiscale](#), are also entering the market – in many cases offering a more personalised service or providing features not yet available from the bigger players.

Quality engineering for free

The real value of outsourcing a discreet portion of your application to a third party API lies in quality. You could always use local storage and your programming language of choice to deal with a large volume of file read and write operations. But do you really think you'll beat Amazon or Google for reliability, scalability and speed of the resulting solution? The move from document based websites towards web based applications means more and more processing is being done online. The art of server configuration, of writing truly scalable and performant code are more important than ever – and not within most developers toolkits. Whereas learning the hard way may have previously been the only option, now we can rely on APIs developed by teams with real world experience of scaling. And although still an emerging area, some of these services are starting to offer Service Level Agreement that will be important for mainstream business adoption.

Functionality vs Data

It's not just processing that we can leverage from API providers – high quality data can be just as important to successful web applications. Historically some of this data has been available through often complex APIs and only after paying exorbitant fees. UK Postcode data, TV listings and sports fixtures are all good examples of this approach. Increasingly though we are seeing services using Semantic Web technologies, including [Microformats](#) and RDF, as well as simpler and free to use APIs which allow for easier access to data. [MusicBrainz](#) is a great example, providing a huge amount of data via a simple RESTful API about musicians and music releases. [GeoNames](#) does a similarly good job for all manner of geographical data and [DBpedia](#) is mining wikipedia and providing a simple semantic query engine to allow programatic access. With work ongoing on generic parsers capable of mining this data the whole web of data has the potential to be our API.

We're all fed up with entering and re-entering our personal data into each new service that comes along. With new technologies like [OAuth](#) and [OpenID](#), as well as a focus on data portability from the likes of the data portability working group, we might just be able to share this data too. The Google Social Graph API has already shown what can be done with publicly available [XFN](#) and [FOAF](#) data on the web. The issues surrounding personal information make this an area where particular care is required however.

Change the client as well as the server

Sometimes it's not enough to just change the server. The rise of specialised browsers such as [Joost](#) (for video) and [Songbird](#) (for music) allows for functionality that would be impossible otherwise within our everyday desktop browsers. Site specific browsers, along with advancements such as local storage in the latest builds of Firefox and Webkit, seem to indicate an increased interest in browser

innovation. Firefox has long had an impressive list of plugin and [Greasemonkey](#) scripts can make all sorts of interesting tweaks and enhancements to sites. But both Joost and Songbird have taken these small changes further, building integrated client/server environments around the concept. Google, with [Google Gears](#), and now Yahoo! with [BrowserPlus](#) have taken another approach to browser innovation; one which if we see wide spread adoption of the plugins could provide lots of scope for interesting applications in the future.

Problems

It's not all in place just yet. The reliability of your application is likely to be important, and making use of a distributed set of APIs could leave you in the unenviable position of being less stable than your least stable partner. If your monolithic web application can attain 99.9% uptime then you'll probably be down for around 8 hours and 45 minutes a year. If you're relying on four third-party services all promising 99.9% uptime as well you could suffer up to 43 hours, 45 minutes down time.

Although there are ways of mitigating this problem, stability could be a real issue for those making use of a number of different services. With some of the providers of APIs in this way being startups as well the risk is potentially even greater. What if the provider of a part of your application simply goes away one day? [Imeem](#), a social music site, recently purchased [Snocap](#), a company which provided a core part of the Imeem service under similar circumstances.

Relying on specific browsers or particular plugins also poses problems for adoption as we have seen before. As more and more services are provided via a browser, and with increasing awareness of security risks online, installation of new software or plugins becomes even more difficult.

The issue of lock-in could also raise its head. Without a vibrant ecosystem of different interoperable providers you could be subject to price hikes or serious service disruption. Individuals may be happy with our new Google or Microsoft overlords but companies are likely to see things slightly differently.

Possible Solutions and Design Challenges

Lots of things will need to come together to make a sustainable business case for widely distributed development. But that's not to say it's not possible now with a little care and attention, and there are examples already of possible solutions.

[Mapstraction](#) provides an abstraction library over popular mapping services including Google Maps, Yahoo! Maps and Microsoft Live Maps. Switching between multiple providers instantly and without re-developing your application makes life much easier, as long as comparable services exist. Another argument for competition benefiting everyone involved.

In an open source world we wouldn't need this sort of abstraction. Part of the solution to the lock-in problem might be a truly open source and open standards based platform, the original dream pursued by Zimki. This might still come to pass with work already looking at the Open Source licensed Google App Engine SDK and a proof of concept of that platform running on Amazon's EC2.

The uptime calculations above show the potential perils of relying on multiple third party services. However in many cases downtime in one of these APIs doesn't have to mean downtime of your application; caching, delayed processing and sensible error handling can all help. But this sort of distributed programming is not yet the norm amongst most developers. Herein lies maybe the main difference between the internet mashup culture and distributed business software development.

Service Level Agreements are also part of reassuring business of the utility of third party APIs. In theory centralisation of expertise and resources should allow specialised third parties to provide both cost savings and increased stability. But without an SLA in place most businesses would rather not take the risk. Amazon AWS now provides a rudimentary SLA for the S3 service but more robust and far reaching agreements are often demanded for business IT systems.

Flexiscale offers a competing service with a more comprehensive and personal SLA.

The Future

The use of third party commercial APIs has the potential to change the development landscape – bringing high quality middleware to the web. It could be the original *web services* dream realised. But without critical mass, new development practices and an active market it could also be a new achilles heel for successful startups.

The best place to be right now might be at the forefront of building this new middleware tier, or in convincing traditional business of the efficiency and cost savings possible under a new ecosystem of web based APIs.