Web Beeps, and a Strategy for Optimizing Web Systems

Danny Ayers (independent R&D)

danny.ayers@gmail.com

2012-02-09

The proposed presentation will have the following sections, with approximately the same time devoted to each:

- how it works : the anatomy of the Web Beeps service
- how it happened : development process outline, in particular:
  - how a declarative description was derived and used for for configuration
  - how the system was optimized using a genetic algorithm
- generalization of the overall technique to other problems (e.g. optimization of Web Services)
  - utilities to aid that generalization

These sections are briefly described below. (The presentation will have diagrams :)

Introduction

On the 31st December 2011, while everyone else was out celebrating, I switched on a service at http://webbeep.it. From the site:

Web Beeps are short, quasi-musical sounds which encode text. They can be thought of as sonic QR Codes. You can use them to create audio representation of Web links and identify things with tones. They are easy to create, decode and use and backed by an open specification together with an open source implementation.

The project started as an experiment to see if it could be done, and from that perspective the project is effectively finished, goal achieved. But the development process was interesting and suggests possible approaches to a broad class of problems likely to be encountered when developing Web systems. In other words, although conceived as a fun project, it turned out to be (IMHO) a useful piece of research.

How It Works

Encoding

- Unicode text input is converted into ASCII (Punycode with a checksum)
- each character is mapped to a high note, low note and duration (notes taken from a pentatonic scale)
- pure tones are generated and shaped according to the mapping
- the tones are mixed and rendered as mp3 files

Decoding

- incoming tone is trimmed
- tone is split into chunks, each corresponding to one character
- each chunk is analysed to find tones present and durations (using Goertzel algorithm)
- tones mapped to ASCII characters
- text reassembled and converted back to Unicode

More detailed description is available online (I'll make this more concise and hopefully more coherent for presentation :)

Development Process

"Spike" Solutions

Initially, for **encoding** Web Beeps, an algorithm was chosen that fitted the "quasi-musical" requirement but was otherwise fairly arbitrary. Because the audio encoding within messages was relatively simple (a sequence of pairs of tones) it was assumed that **decoding** *would* be possible, but the choice of algorithms was not determined in advance. A general idea of which digital signal processing algorithms might be useful was was sketched out, and the algorithms implemented/obtainined. Various hard-coded experiments were made to explore the problem, roughly corresponding to what the Extreme Programmers call spike solutions.

A small set of algorithms were found that could decode the signal *some of the time*. The difficult part of the problem was, given a signal composed of a mix of one or two sine waves, identify the frequency of those sine waves. The process used centered around Fast Fourier Transform-based spectral analysis.

It was quite dependent on the specific characters in the text message being transmitted. However, each algorithm relied on two or three numeric parameters, the values of which impacted the overall accuracy. Although the processing was built as a pipeline of discrete, independently controlled modules, there was some interdependence between the parameters of the modules. The target system viewed as a whole not only comprised a signal generator and decoder but also transfer across external systems - lossy (mp3) codecs and potentially real-world acoustic transfer between a loudspeaker and a microphone. Degradation of the signal through this transfer would occur due to noise and distortion. As result, it seemed reasonable to view the system as a whole as nondeterministic, which would have additional impact on the interdependence of the control parameters.

At this point is seemed possible that selection of the right parameters would solve the problem : 100% fidelity of message transmission when the encoder was connected directly to the decoder, and maintenance of 100% fidelity for "most" real-world scenarios. But there were a lot of parameters. Some kind of automatic optimization seemed like a good idea.

Module Parameterization

To address the problem of manipulating many values at once it was decided to refactor the code, allowing configuration through a set of parameters expressed in a consistent manner. The code already had fairly systematic print statements expressing the configuration for debugging purposes, and the output of these were

used as a basis for an ad hoc configuration format. Although this step occurred as a simple consequence of how the development had proceeded so far, in retrospect it is believed to be powerful technique (if approached systematically).

System Modelling

In the process of refactoring, the individual modules of the system were modified to expose consistent interfaces (essentially signal in, signal out and a set of named numeric parameters). The real-world part of the system as a whole was simulated by building modules of the same form, placed in the audio part of the pipeline between the encoder and decoder (adding noise, harmonic distortion and reverberation). Text messages were simulated in a random fashion (weighted to favour messages that seemed likely, e.g. those beginning "http://").

As a result the system configuration as a whole could be defined declaratively (there were some compromises to the convenience of hard-coding, but the pieces under investigation were all exposed).

Optimization

In addition to the parameterization of the controls of the modules in the system, there was another factor of critical significance: the accuracy of the results. This was very easy to measure, simply taking the proportion of the output characters that matched the input characters.

Given a system like this, a large variety of optimization strategies and algorithms are possible. Because of the way the system was componentized, it was felt that a Genetic Algorithm might be a good choice.

An estimate was then made of a system that would contain a subsystem that would be optimal. For example, it seemed likely that both low- and high-pass filtering of the signal going into the decoder would be desirable, although the exact shape of the filters was unknown. So several filters were put in series in the pipeline. In addition to their control parameters (FIR filters were used with parameters for the number of stages and cutoff frequency) an "on/off" parameter was added. When this parameter had the value "off", the filter wasn't used, the signal passed through that stage unmodified.

(Sorry, a diagram or two would help a lot here but I don't yet have one prepared and the submission deadline is approaching...)

An organism for the algorithm was genetically encoded as an ordered list of the parameters of each of the modules in the encoding and decoding pipelines (including "on/off" switches). Each organism was a whole **(encoder) -> (real-world degradation simulation) -> (decoder)** system.

At this stage in the development process the prototype system was taking significant time to encode and decode the messages (of the order of a second) so the total time taken for this processing was factored into the overall fitness measure.

Population behaviour for crossover and mutation of each generation was initially set up as a rough guess of what might work (based on the literature) which was subsequently modified based on observation of how the optimization progressed.

The algorithm was run with a total population of around 200 organisms. The code included a serializer/parser for

the ad hoc configuration format so it was possible to persist a set of organisms across runs. Even when the fitness measure and other aspects of the setup were modified, this was found to provide a seed of "fairly good" candidates.

However the system when using the FFT-based tone extraction algorithm **fell short of the requirements**. Although it appeared to get to a 100% encoding/decoding success rate with a direct encoder-decoder signal path, even a small amount of "real-world" signal degradation made the accuracy drop sharply.

Plan B (and some validation of the optimization technique)

The failure of the core algorithm at this point led to a return to the literature. Fortunately, although it had somehow been overlooked initially, there was an algorithm available that was a perfect match for the kind of processing required: the Goertzel Algorithm. Where the FFT technique will reveal any component sine waves in a signal (and for this application the peaks need to be detected), the Goertzel algorithm will check the amplitude of a single, known-frequency component. Because the encoder only uses tones with certain frequencies, testing for these with Goertzel was not only much more accurate, it was also significantly faster (at least an order of magnitude). What's more it was trivial to implement.

So the tone detection module of the decoder was replaced with a Goertzel-based one, with everything else left as it was for the FFT version.

Running the genetic algorithm over this new kind of organism was very revealing. Many of the modules in the "guesstimate" pipeline (notably the filters) were optimised to the "off" position. This can be explained by the fact that a focused measure of a single frequency isn't dependent on the levels of other frequencies, in a sense it is already optimally filtered.

Generalization of the approach to other problems

While the system optimized here was for digital signal processing, it shares many characteristics with other systems encountered around the Web:

- composed of interconnected processing components (pipelines being a common case)
- the components have adjustable parameters (even if it's just "on/off")
- a fitness measure is available (performance and scale are common measures)

The primary thing needed to apply the optimization technique as described here, whether using a genetic algorithm or one of the many others available, is a declarative codification of the system configuration.

The way this codification was derived for the Web Beeps system was through directly exploiting the prototype system itself. The components, actually objects in the running code (it was written in Java) were in effect asked to describe themselves and their subcomponents recursively.

"dork" Utilities

For Web Beeps the declarative language used was ad hoc and application-specific. However work is ongoing to generalise this. Structurally, in general, the system in question can be described as a graph, and the Resource

Description Framework (RDF) is an eminently suitable language for expressing this, especially when dealing with systems that are to be deployed on the Web.

To this end a small utility library, dork ("Description of Runtime Klasses") is under development in which a description of objects may be obtained from the objects themselves (by implementing a `describe()` method or via reflection). The output from these utilities is in the human-friendly Turtle format. (Where the `describe()` method is used it can be seen as a machine-friendly output version of the standard `toString()` method of Java objects).

Essentially the assumption is that, to get a system description "the code knows best".

An aim for dork is for it to be an easy, lightweight supplement to existing systems (it doesn't depend on any other RDF libraries, it treat Turtle as simple strings). To ensure it will work in practice, the current Web Beep parameterization/configuration serialization setup is being refactored to use it. *(Work in progress, about half done at the time of writing).*

The application of the genetic algorithm to the system parameters of Web Beeps was straightforward. All it really demanded codewise was the ability to express them as an ordered list (which was implemented as an `ArrayList` of `Parameter` objects), together with a variable to retain the value of each organism's derived fitness. No significant problems are anticipated translating this to RDF data structures so the optimization can be fully decoupled from the implementation. However more work will be required to generalise beyond pipeline-based processing and to support other optimization algorithms, but that too should be reasonably straightforward (probably via declarative transformations in both cases).

The proposed strategy for system optimization currently looks like this:

1. build a prototype
2. have the prototype describe itself
3. refactor the prototype to use the description for system configuration
4. encode the description in a suitable form for an algorithm (such as the genetic algorithm)
5. wrap the system in an optimization testbed and run it
6. take the "winning" configuration and plug it into the system
7. *deploy*

With foresight, steps 1. 2. and 3. can be rolled into one: *build a prototype which supports declarative configuration*. (Systems are often built this way already - configured using text files with extensions like .xml, .conf and .ini, though the potential for reuse of the data contained in these files is usually extremely limited).

With a little more work it is believed it should be possible to drop step 4. and have at least one suitable optimization algorithm implemented such that it can operate directly on a broad range of systems.

The "optimization testbed" part of 5. could take a variety of forms. In Web Beeps it has been implemented as a thin hardcoded wrapper which creates objects that are instances of the whole encode-degrade-decode system. For particularly heavyweight systems the instance under test could be run in its own virtual machine and observation of the VMs resource utilization could be used as factors in the fitness measure.

An interesting, incidental observation about the RDF produced so far in modeling the Web Beep digital signal

processing system is that it is largely isomorphic to the core of the OWL-S ontology designed for the description of Web Services (so much so I'll probably use that vocab). While hardly strong evidence, it does favour the possibility that the kind of optimization described here can be applied to Web systems.