SAPPHIRE: An Extensible Speech Analysis and Recognition Tool based on Tcl/Tk[1]

Lee Hetherington and Michael McCandless

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139 USA
{*ilh,mike*}*@sls.lcs.mit.edu — http://www.sls.lcs.mit.edu/*{*ilh,mike*}

ABSTRACT

The SAPPHIRE system is a powerful, extensible, object-oriented toolkit allowing researchers to rapidly build and configure customized speech analysis tools. Implemented in Tcl/Tk and C, the current version of SAPPHIRE provides a wide range of functionality, including the ability to configure and run the SUMMIT speech recognition system. We now use SAPPHIRE widely in almost all aspects of our speech analysis and recognition research.

1. MOTIVATION

Investing in the development of tools and other forms of infrastructure is a critical role that all research institutions must periodically undertake. Unfortunately, such an investment is time-consuming, and the benefits are not always apparent nor readily measurable. As a result, one can easily be so distracted by day-to-day affairs that such activities are indefinitely deferred.

Powerful tools are particularly necessary for research and development of technologies involving human language, since the processes of signal modelling, algorithm development, and system evaluation rely largely on empirical evidence. The situation is confounded by the fact that, more often than not, research activities involve not a single individual but a team of collaborators, each responsible for a specific aspect of the problem. A comprehensive set of research tools will greatly enhance researchers' ability to collectively explore, formulate, and test new ideas with minimum effort.

There are several existing tools geared towards the development of human language technologies. In the mid-eighties our group developed SPIRE [1], a set of Lisp-machine based signal analysis and display tools that enjoyed limited distribution. Another speech-related tool is the ISP package designed by Kopec [2]. Perhaps the most widely used is the signal processing and display package from Entropic, ESPS/Waves [3]. Recently, the CSLU at OGI has also developed and disseminated a set of tools, based on Tcl/Tk, that allow researchers to rapidly configure a dialogue-based speech interface [4].

Over the past few years, our group has been actively developing human language technologies, and embedding these technologies in conversational systems. In this process, we have become increasingly aware of the multiple roles played by tools, and the important properties that they must possess. First, the tools must be *intuitive.* A toolkit that requires significant training on the part of the user will not likely gain wide usage by people with varying computing skills, and will, in all likelihood, have minimal impact. Second, they must be *comprehensive.* Since the life cycle of research and development involves exploratory studies, signal modelling, algorithm development, system evaluation, and error analysis, the toolkit must support all these activities and more. Third, they must be *customizable.* For example, a user should be able to change the computing and display parameters easily to suit their individual needs. Finally, they must be *extensible.* It should be straightforward for an experienced user to add unanticipated functionalities with minimal effort, and to achieve interoperability and compatibility.

A more subtle, but no less important feature of tools for speech technology development is that they must be *integrated.* Because of the complexity of the problem, it is not uncommon for a research group to be splintered into separate subgroups, each responsible for a subset of the problem. Too often, R&D activities are supported by multiple, disjoint sets of tools—one for examining the signal and features, another for pattern classification, still another for system development and training, etc. If one researcher discovers a potentially useful feature, for example, he/she must ensure that the feature can be properly incorporated into other parts of the R&D food chain. The process can be extremely time-consuming if the tools are not interoperable and do not support group work. To deal with this problem, many research institutions typically keep multiple versions (e.g., development, evaluation, and demonstration) of their systems, effectively building firewalls around these activities. The net result is a decrease in productivity—the incorporation of good ideas into a working system takes much more time than should be necessary. Thus a desirable, or even essential, feature of tools is that they be able to support the seamless integration of research and system development activities.

2. SAPPHIRE

In order to address some of these issues we have developed SAPPHIRE, an extensible speech analysis toolkit based on the Tcl scripting language. SAPPHIRE allows users to quickly create new tools, or customize existing ones, by writing Tcl scripts. Within a script the user *declares* in a broad fashion the overall structure of the tool, and leaves implicit the many details of computation. It is the combination of a declarative computational paradigm with a simple scripting

[1] This research was supported by DARPA under contract N66001-94-C-6040, monitored though Naval Command, Control and Ocean Surveillance Center.

language that gives SAPPHIRE many of its unique enabling properties.

SAPPHIRE empowers every member of our group to be a tool builder. Researchers can quickly learn how to customize SAPPHIRE tools, and SAPPHIRE's extensible object-oriented design enables them to augment the standard objects with particular objects suitable for their projects. Since the same framework is shared by all group members, their work is then available for others. New members are more productive by spending less time learning tool details and constructing their own tools, and more time performing novel research. Rapid prototyping allows researchers to interactively explore new ideas.

SAPPHIRE began as a unique speech analysis tool with a sophisticated declarative computational model. However, using SAPPHIRE's extensible and flexible computational framework, we have now augmented it with the necessary objects to implement all of SUMMIT [5], our speech recognition system. This seamless marriage of speech analysis and recognition has fundamentally altered how we do our research. All aspects of SUMMIT recognition, including tools to perform batch-mode recognition in a distributed fashion, interactive exploration and analysis, and real-time demos, are unified under SAPPHIRE and share a single set of recognizer specification files. Note that other recognizers could be implemented within the SAPPHIRE framework (i.e., SAPPHIRE is not specialized for SUMMIT).

In addition, SAPPHIRE actively tracks our progress, preventing conventional redesigns and reimplementations. As new breakthroughs are made, we augment SAPPHIRE with new objects or modify existing objects. The new algorithms are then immediately available for everyone to test and improve upon.

SAPPHIRE is an excellent tool for educational purposes. Students can reconfigure the tool to explore the differences between signal representations and the tradeoffs of signal processing parameters such as the window type and window duration of a Fourier transform. SAPPHIRE now includes a healthy regiment of standard speech analysis objects such as LPC analysis, Mel-scale spectra and cepstral rotations, Fourier transforms and spectral slices, and transcriptions.

3. DESIGN

To meet these goals, we chose an object-oriented design based on the Tcl scripting language [6]. Tcl is an easy-to-learn language, and because it is interpreted, tools based on Tcl are rapidly prototyped and reconfigured. In addition, Tcl's powerful graphical user interface, Tk, makes building graphical tools quick and easy. Most computation in a SAPPHIRE tool is executed within C, not Tcl; Tcl serves only as the glue language to specify the configuration of the tool so that run-time efficiency is not sacrificed. We chose an object-oriented design in order to simplify the task of designing tools and recognition systems by breaking them into objects of manageable size and functionality. Such objects can be combined in various ways to create tools with unforeseen capabilities.

SAPPHIRE extends the standard Tcl language with additional commands and objects for speech analysis and recognition.[2] A user designs a tool by writing a suitable SAPPHIRE (i.e., Tcl) script, in which the user *declares* the connectivity of certain objects. SAPPHIRE's core computation module, described in the next section, then handles the computational details implicit in the user's declarations.

```
s_waveform w \
    -file timit.wav \
    -normalize yes

s_stft s \
    -waveform w \
    -window hanning \
    -windowdurms 5.0 \
    -db yes

s_waveform_view .wv \
    -waveform w

s_spectrogram_view .sv \
    -spectrum s \
    -width 15c \
    -height 6c

pack .wv .sv
```

Figure 1: Example SAPPHIRE script displaying a waveform and spectrogram.

The viewable objects in SAPPHIRE are designed as genuine Tk widgets. This means that a viewable object, once created, may be configured and packed along with other Tk widgets.[3]

The example script shown in Figure 1 computes a spectrogram from a waveform loaded from the file *timit.wav,* and draws the waveform and spectrogram above one another in a window. The user creates a waveform object *w,* attaches it to a short-time Fourier transform (STFT) object *s,* and then attaches *w* and *s* to waveform-view *.wv* and spectrogram-view *.sv,* respectively. In addition, any number of configuration options may be specified as the objects are created, to customize their computation or appearance. Finally, the script "packs" the graphical objects into the window. This script, when executed by SAPPHIRE, displays a waveform and spectrogram, in a pipelined fashion, as each frame of the STFT is computed. If instead the user wanted to see the spectrogram of a live waveform (recorded from a microphone), the waveform object could be replaced by a different waveform object, which collects samples from A/D hardware. Figure 2 shows a screen dump of a more sophisticated tool that incorporates both SAPPHIRE and SUMMIT objects. The tool takes waveform input from a file or microphone, performs recognition on it, and displays recognition output in the form of phonetic and orthographic transcriptions. It also shows some of the intermediate objects in SUMMIT, including segmentation and normalized classifier scores [7].

In order to support the high-level Tcl interface, SAPPHIRE provides significant core functionality to all objects. As mentioned earlier, SAPPHIRE tool specifications are largely declarative (rather than procedural) in nature: the tool builder declares and configures objects but does not have to worry about computational aspects such as order of evaluation or how to move data around from object to object in a pipelined manner. The specification for an object's configuration includes specifying its parents (i.e., the objects that provides its inputs). SAPPHIRE uses this information to construct and maintain a directed acyclic graph. In doing so it checks for dependency cycles which would interfere with the computational model. In addition, SAPPHIRE provides core functionality to support widgets (graphical objects), including a rich set of messages (e.g., for scrolling, cursors,

[2] Note that recognition capabilities are not part of the basic SAPPHIRE toolkit, but instead are part of SUMMIT.

[3] In fact, SAPPHIRE's object-oriented style is heavily modelled after the object-oriented nature of the Tk graphical user interface toolkit.

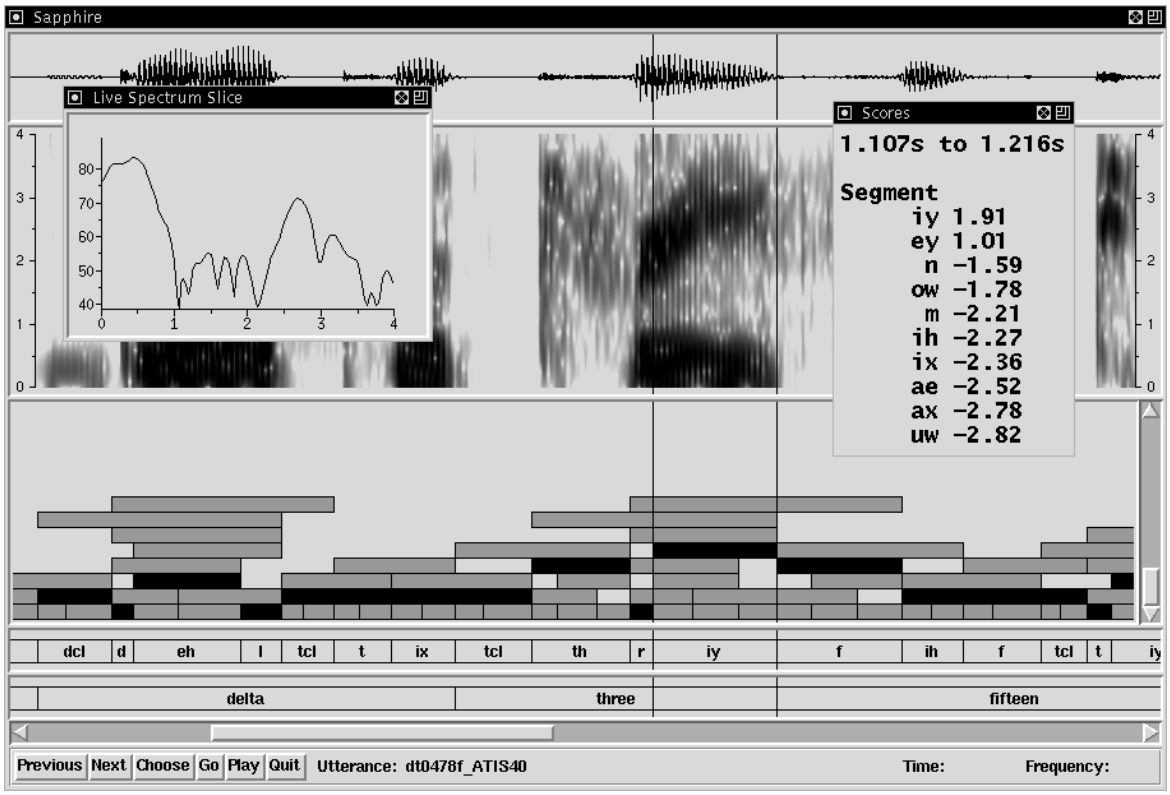

Figure 2: Example SAPPHIRE/SUMMIT tool displaying (top to bottom): a waveform, a wide-band spectrogram, a segmentation network with the computed recognition path (highlighted in black), the recognized phonetic and orthographic transcriptions, and, in separate pop-up windows, normalized classifier scores for the marked segment and a spectral slice from its midpoint. The user is able, among other things, to scroll, reconfigure signal processing parameters, set time marks, and play portions of the waveform. The full-fledged SUMMIT recognizer runs within this tool.

time marks, time scaling) and a mechanism to distribute these messages to relevant objects.

3.1. Core Computational Support

The computational model supported by SAPPHIRE is data-driven, flowing from input sources (e.g., microphones or files) through intermediate computational objects and on to outputs (e.g., displays or files). SAPPHIRE provides most of this functionality automatically by sorting objects topologically and maintaining a run list (i.e., a list of objects needing computation time). Conceptually, objects run as separate "threads," although SAPPHIRE's implementation is not really multi-threaded; SAPPHIRE provides the abstraction through simple time slicing. Pipelining is fully supported provided that the objects are capable of operating on partial input without blocking.

Initially, one or more objects receives inputs from previously saved files, sockets, or live devices (such as an A/D device). These objects are automatically placed on the run list by SAPPHIRE. When their output state changes (e.g., when new waveform samples become available) these objects notify SAPPHIRE, which in turn places the objects' immediate children on the run list. Based on input sources (previously saved files, socket, etc.), the user's demands, and object connectivity, SAPPHIRE will *activate* a subset of all objects; it is only these *active* objects that will compute. Order of evaluation in SAPPHIRE is simple: handle all pending graphical events and then traverse cyclically through the run list in topological order. This ensures that all objects on the run list receive a time slice to execute (although individual objects must not block), that execution happens in an order that supports the data-driven model, and that the response of the user interface is as immediate as possible.

3.2. Core Graphical Support[4]

One of SAPPHIRE's primary design goals was to aid in the design of graphical tools for speech recognition research. For this purpose we extend the set of Tk widgets (graphical objects) with additional widgets written for X windows. SAPPHIRE supports graphical tools through a rich set of graphical *messages.* These messages allow graphical state to be propagated from one widget to another throughout a set of interconnected objects. The graphical messages include movement of the cursor in time and frequency, changing the time scale, changing the time view (i.e., scrolling), and adding/removing time and frequency marks (e.g., for time-aligned transcriptions).

Because graphical objects are given higher priority for computation over non-graphical objects during core computation, graphical events and messages are responsive to the user, even when SAPPHIRE is busy with pipelined recognition.

[4] Note that SAPPHIRE scripts do not have to include graphical objects, and thus can be run as batch jobs.

4. CURRENT STATUS

The first version of SAPPHIRE is written in C and now includes a diverse set of objects types. We divide these objects into two groups: the core SAPPHIRE objects and the SUMMIT speech recognition objects. SAPPHIRE can run either locally on a single machine, or within our previously designed distributed computational framework. This allows a job to run in parallel across all currently available compute resources, sharing the resources with any other running jobs.[5] We have ported SAPPHIRE to a number of platforms, including: Sun SPARCstations running SunOS or Solaris; Digital AlphaStations running Digital Unix (OSF/1); Hewlett Packard workstations running HP-UX; and Intel x86, Pentium, and Pentium Pro–based PCs running Linux, Solaris, or NeXTStep. Architecture independent file formats are used to save intermediate computations to disk for future loading.

The core SAPPHIRE system includes 21 distinct object types: control objects to sequence through a set of data loaded from a database; an object to record and play "live" waveforms; waveform filtering objects to create various high-pass, low-pass, or band-pass filters given frequency cutoffs and filter orders;[6] container objects to load and store waveforms, spectra, and transcriptions; spectral computation objects to compute short-time Fourier transforms, Mel-scale spectral and cepstral coefficients, LPC coefficients and spectra, spectral energy in specified bands; and finally, graphical widgets to render waveforms, spectrograms, graphs, transcriptions, and spectral slices.

The SUMMIT component of SAPPHIRE includes 20 object types: numerous control objects to enable distributed computation via a client/server socket connection; objects to compute various aspects of segmentation; objects to extract segment-based features, (e.g., averages of spectral and cepstral coefficients), delta, duration, and derivative features; objects to train and compute principal components analysis; objects to train and evaluate various classifiers, including mixtures of diagonal and full-covariance Gaussians; search objects such as a Viterbi search and an A^* search; and objects which provide labels for segments during training, given transcriptions.

5. DISCUSSION

SAPPHIRE has had a profound effect on the way we do speech recognition research. It is currently used very widely in our group by both staff members and students. Breakthroughs made by one member are immediately available to others and to our group's real-time demonstration systems. For examples of some of the recent research aided by SAPPHIRE, see [7–10]. SAPPHIRE is also being used and extended at three other sites around the world.

We have developed a large library of scripts to address our research needs. Using interactive exploratory tools, we can examine SUMMIT as it recognizes an utterance. We explore details of the segmentation and classification, understand why the Viterbi or A^* search chose a certain path, and witness the interaction of the acoustic and language models. Rapid turnaround allows us to vary thresholds and parameters and see the resulting impact. Porting to a new recognition domain is now a simpler process. Exploration, batch-mode evaluation, and real-time demonstration now take place under the same framework.

The extensive usage SAPPHIRE has seen in our group has prompted us to redesign its internals. The primary design emphasis in the first version was to make it easy to develop and customize *tools* by writing SAPPHIRE Tcl scripts, and this was very successful. However, not enough emphasis was placed on making it easy to develop new SAPPHIRE *objects*. Improving this situation involves better software engineering and adding functionality to SAPPHIRE's core.

We have started to design and implement the second version of SAPPHIRE. This time we chose C++ because it is an excellent match for SAPPHIRE's object-oriented structure. By using C++'s class *inheritance*, we will enable object designers to derive a new SAPPHIRE object from an existing object by coding only the extra (or different) functionality. Existing SAPPHIRE scripts will require little, if any, changes to work with the new version. Initial indications are very encouraging that this SAPPHIRE redesign will meet its design goals.

[5] However, each SAPPHIRE tool must run as a single process. SAPPHIRE does not currently support computation with multiple processes or on different machines; the parallelism is not that fine-grained.

[6] Use of this object requires MATLAB.

6. ACKNOWLEDGMENTS

We wish to acknowledge the valuable contributions many in the SLS group have made to SAPPHIRE. Because SAPPHIRE is in such widespread use within our group, many have made valuable contributions that have become an integral part of the system, including Jane Chang, Jim Glass, Manish Muzumdar, Ray Lau, Tim Hazen, and Ray Chun. Victor Zue was very supportive of our building SAPPHIRE, and was helpful in writing this paper.

7. REFERENCES

1. V. W. Zue, D. S. Cyphers, R. H. Kassel, D. H. Kaufman, H. C. Leung, M. Randolph, S. Seneff, J. E. Unverferth III, and T. Wilson, "The development of the MIT lisp-machine based speech research workstation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing* (Tokyo), pp. 329–332, Apr. 1986.
2. G. E. Kopec, "The integrated signal processing system ISP," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, no. 4, pp. 842–851, Aug. 1984.
3. "ESPS/Waves," http://www.entropic.com/products.html.
4. S. Sutton, J. de Veilliers, J. Schalkwyk, M. Fanty, D. Novick, and R. Cole, "Technical specification of the CSLU toolkit," Tech. Report No. CSLU-013-96, Center for Spoken Language Understanding, Dept. of Computer Science and Engineering, Oregon Graduate Institue of Science and Technology, Portland, OR, 1996. Also http://www.cse.ogi.edu/CSLU/toolkit/toolkit.html.
5. V. Zue, J. Glass, M. Phillips, and S. Seneff, "Acoustic segmentation and phonetic classificaion in the SUMMIT system," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing* (Glasgow), pp. 389–392, May 1989.
6. J. K. Ousterhout, *Tcl and the Tk Toolkit*. Reading, MA: Addison-Wesley, 1994.
7. J. Glass, J. Chang, and M. McCandless, "A probabilistic framework for feature-based speech recognition," in *these proceedings*.
8. M. S. Spina and V. W. Zue, "Automatic transcription of general audio data: Preliminary analyses," in *these proceedings*.
9. S. Seneff, R. Lau, and H. Meng, "ANGIE: A new framework for speech analysis based on morpho-phonological modelling," in *these proceedings*.
10. M. D. Muzumdar, *Automatic Acoustic Measurement Optimization for Segmental Speech Recognition*. M.Eng. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, June 1996.