

**Annual Report for Period:** 08/1998 - 08/1999**Submitted on:** 07/26/1999**Principal Investigator:** Picone, Joseph .**Award ID:** 9809300**Organization:** Mississippi State Univ

CARE: Internet-Accessible Speech Recognition Technology

**Participant Individuals****Senior Personnel****Name:** Picone, Joseph**Worked for more than 160 Hours:** Yes**Contribution to Project:****Post-doc****Graduate Student****Name:** Ganapathiraju, Aravind**Worked for more than 160 Hours:** Yes**Contribution to Project:**

senior graduate student responsible for the overall system development and design

**Name:** Zhang, Xinping**Worked for more than 160 Hours:** Yes**Contribution to Project:**

junior programmer - responsible for core HMM training technology and implementation of math foundation classes

**Name:** Wu, Yufeng**Worked for more than 160 Hours:** Yes**Contribution to Project:**

junior programmer responsible for foundation class implementation

**Name:** Mantha, Vishwanath**Worked for more than 160 Hours:** Yes**Contribution to Project:**

junior programmer responsible for development of signal processing code

**Undergraduate Student****Name:** Graff, Clayton**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Web site developer

**Name:** Rogers, Lorena**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Web site developer

**Name:** Brown, Robert**Worked for more than 160 Hours:** Yes**Contribution to Project:**

Java applet programmer

**Name:** Duncan, Richard

**Worked for more than 160 Hours:** Yes

**Contribution to Project:**

programmer responsible for low-level foundation classes

**Partner Organizations**

**Other Collaborators**

None

**Activities and Findings**

**Research Activities: (See PDF version submitted by PI at the end of the report)**

**RESEARCH AND EDUCATIONAL ACTIVITIES**

In the first year of this project, we focused our efforts in three major areas:

À Core Technology: extensions of the speech recognition system required to enhance its appeal to our customer base (driven by customer feedback);

À Foundation Classes: building blocks such as vectors, matrices, and data structures that simplify and standardize the development of higher-level classes;

À Web-Based Information: a comprehensive and informative web site that constitutes a central point of contact for everything related to the project.

We have seen interest in the project grow as evidenced by the fact our mailing list has grown to 150 participants, and we have received several serious inquiries about collaborations based on our system (one of which resulted in participation in a joint NSF/EU proposal [1]). Major milestones for the first year of the project included the release of a fully functional speech recognition system (including feature extraction and training), and the development of a remote job submission capability that lets users submit jobs to our system over the Internet.

**A. Core Technology**

State-of-the-art speech recognition technology is the foundation upon which this project is built. We must not lose sight of the importance of this core technology to this project. This technology is being developed in a parallel effort funded by the Department of Defense. The speech recognition system developed in this effort has improved dramatically since the start of our NSF project in August'98. We briefly review the enhancements made to the system in the first year of this project, and then discuss the impact this has made on our efforts within this project. Next, we describe some enhancements made to the system to broaden its appeal to our customer base, and review some initial attempts at cross-platform portability.

**A.1. System Status**

In the past year, three important capabilities have been added to the speech recognition system we have been developing: feature extraction, word graph generation, and Hidden Markov Model (HMM) training. Feature extraction is the process by which the speech signal is converted to a sequence of vectors that serve as input to the recognition system (a continuous density HMM system). This is often called the acoustic front-end. Our approach was to initially replicate an industry-standard front-end consisting of mel-spaced cepstral features [2] and their first and second-order derivatives. This is one step that allows users to duplicate results obtained with other commercial and proprietary systems. This capability was delivered as part of a general front-end capability summarized in Figure 1.

A second key feature added to the system was the ability to generate word graphs. Speech recognition experiments are time-consuming primarily because of the large language models used in the decoding portion [2] of a system (these large language models are desirable because they maximize performance). Hence, to save time on subsequent experiments, a word graph is constructed that represents most plausible, or highly probable, hypotheses that can be generated by this network. This graph is then rescored with new acoustic or language models depending on the nature of the research. This network can be quickly rescored - a process that often runs at least ten times faster than the process required to generate the network. Because of these time savings, word graph rescoring is an extremely popular method of doing speech research for conversational speech recognition, and is therefore a modality that must be supported for a system to be widely accepted. This word graph generation feature was added to the system in early 1999, and required significant changes to the way the decoder manages the search process. Fortunately, the current implementation represents a vast improvement in the architecture and performance of the system [2]. Though it took longer than expected to add this feature to the system (we struggled with this for 6 months), the final implementation is extremely clean and efficient, and will have a positive impact on subsequent generations of the system. The system now supports more decoding modes than any commercially available system, and more than most proprietary systems as well.

A third essential feature that was added to the system this year was HMM training [3]. This essentially provided closure on the speech recognition system in that users are now able to build real systems from scratch (previously, one had to borrow some component of the system from another source). We first implemented an algorithm known as Viterbi training, in which only the best state sequence through a network is considered. This is an elegant algorithm in that it is efficient, fast, and consistent with a formal languages view of the speech recognition

problem. This allowed us to develop the proper control structures and code infrastructure quickly. Once this was complete, we added Baum-Welch training [4], which is more popular in state-of-the-art systems.

The current system is described in great detail in an upcoming publication [2]. This is one of the first such publications to provide a tutorial on the details of search algorithms, and is being published in a journal that emphasizes education of entry-level graduate students in signal processing. It is our hope that the time spent on this publication will pay off as more researchers are made aware of the availability of this system and project. Our system is also represented at an upcoming major speech conference [5] that will include a panel discussion on the merits of public domain speech technology.

#### A.2. System Enhancements

In any such project intended to develop public domain code, it is important to adapt to the changing needs of your user community. In the past few years, several major sites in speech research have shifted to the use of features based on an algorithm known as perceptual linear prediction [6]. This technique in some cases has been shown to deliver small improvements in performance. It is an important feature to include if such sites are attempting to replicate their best systems with our public domain system. We completed an initial implementation of this algorithm this year, and are in the process of integrating it into our front-end architecture. A summary of this approach is shown in Figure 2.

Another feature that was requested by several sites this year was the ability to switch language models in the middle of the recognition process. This feature is useful for two reasons: (1) it enables the development of command and control applications that switch between sub-grammars depending on what words are recognized (context-sensitive language models or menus); (2) it allows the recognition system to dynamically recurse through language models at runtime, rather than compile all language models into one big network. The first point impacts the development of voice-driven menu systems and real-time demonstrations. As a word or phrase is recognized, a new language model can be loaded, providing a small set of words or phrases as the next choice. This strategy is the same as that used by most systems to allow acceleration of desktop menus with voice commands.

A second, and equally compelling, reason to consider this feature is the development of systems that consist of a hierarchy of grammars (for example, sentences in terms of words, words in terms of syllables, syllables in terms of phones, etc.). Many existing systems will compile such a system into one large network. While this is sometimes attractive for efficiency reasons, for research flexibility it is often better to process this hierarchy of networks at runtime ('on the fly'). This allows users to change one small module of the system (for example, allowing a word to be represented multiple ways) without recompiling the entire system.

Implementation of this capability requires use of an approach called 'caching.' The system must only activate those portions of the network that represent active words, or words used in the recent past, and leave the remainder of the language model stored on disk. Fortunately, by implementing this strategy, we can handle much larger language models, such as those used in the broadcast news [7] and audio data mining research fields. Since there is currently a shift towards such applications in various research communities including NSF and DARPA, we feel it is important to provide a solution to this problem. Broadcast news language models tend to be large (because they deal with lots of words that occur infrequently) and cannot be managed in memory as a single unit. Our conversational speech recognition system could not handle such a large language model due to the number of bigram entries contained in this model.

In addition to these algorithmic enhancements, several supporting tools were developed to facilitate language modeling. These include a grammar compiler that accepts regular expressions as input and outputs a finite state machine description used by our system, and a grammar compaction algorithm that reduces the size of a word graph without compromising performance. We also developed several display utilities [8] that allow users to look at speech data and analyze its frequency content. We have interacted with several industrial sites on the development of these tools. In fact, one commercial site is making extensive use of this tool in a production data collection capacity, and hired one of our undergraduate programmers for the summer to customize this tool to better suit their unique needs.

#### A.3. Cross-Platform Portability

We have begun to address issues related to portability across different computing platforms. Our plan remains to develop code under the Solaris operating system on two platforms - Sun Sparc and Intel PC (Solaris x86). Within these environments, we use a common compiler, gcc, provided by the Free Software Foundation (GNU). This compiler is supported across a wide range of platforms, including Microsoft Windows'9X and Windows NT. Since an ANSI standard for C++ has only recently been adopted, and most implementations of C++ are still not compatible (this likely will continue for a few years even after the adoption of the standard), using a common compiler across all platforms is the best way to guarantee portability. Currently, we periodically release a Windows version by porting the GNU environment to Windows, and compiling the code under gcc and the bash shell. This vastly simplifies the Windows porting effort.

Perhaps the fastest growing user population for our system is the Linux community. Fortunately, the backbone of the Linux system is GNU software and the gcc compiler in particular. In fact, GNU recently turned over development of its compiler, gcc, to an organization known as EGCS [9], which is a collection of free software advocates who have been informally developing compiler extensions to gcc for years. EGCS is responsible for all subsequent releases of gcc (the two have effectively merged) and is the compiler delivered on most production releases of the Linux operating system (RedHat for example). Hence, conformance to gcc standards covers a large portion of both the Windows and Unix markets, yet minimizes portability issues.

However, the Linux version of gcc distributed by EGCS currently lags the Sun Solaris version in one important dimension: wide character support. One of the founding principles of our system is that languages be handled in native format using Unicode character encoding. The current ANSI C/C++ standards support a wide character encoding with a collection of new functions. These must be supported by a runtime library, which Sun Solaris provides, but production releases of Linux do not. Hence, the current release of our code will not compile under Linux. We expect this problem to be resolved within the next three months. It doesn't make sense to provide an interim work around, because anything we do will be quickly obsoleted. It is expected that EGCS will do a much better job in the future of tracking standards in C++.

#### B. Foundation Classes

The most critical part of the first year of this project was to lay the proper foundation upon which all other software can be written. This is perhaps the most difficult part of any technology-specific project - balancing the efficiency and simplicity of the target application with extensibility and flexibility needed by future research. We believe this is the major strength of our project - the environment is object-oriented from the ground up and designed to be neutral to any particular algorithmic approach. Fortunately, we have been able to leverage pre-existing code developed for a much less ambitious project [10] involving speech recognition. However, most of this code needed revamping based on the changes in the C++ language definition and gcc compiler capabilities.

The hierarchy of classes is shown in Figure 10. Our goal in the first year of this project was to deliver everything through the DSP libraries, which we refer to as the ISIP foundation classes (IFCs). In the second year of the project, we begin the 'Great Convergence' as we rewrite the speech recognition system using these IFCs. We expect this task to be completed by the end of 1999. This latter version of the system will be the basis for our training workshops which will begin in the summer of 2000.

### B.1 Integral Types

At the bottom of our class pyramid is a header file that defines all low-level data types available to the user. These are known as the integral types, and in many ways mirror the syntax of the C programming language. Our current integral types file is shown in Figure 4. This deceptively simple file was the result of much hand-wringing about two competing design philosophies. The C programming language was designed to be somewhat architecture independent. For example, the datatype 'int' could be 16-bits long on one machine, and 32-bits long on another. A C program written properly would work on both machines regardless of the specific implementation. On the other hand, in signal processing, we often need access to specific data types. For example, speech signals are stored as 16-bit integers, and need to be represented as such in C. The problem in C is that a 16-bit integer doesn't really exist. Instead, one can use a 'short int' and must assume that an integer is 32 bits long.

There is a growing movement within the C++ community to support data types such as int32 for a 32-bit integer. In fact, Microsoft seems to be leaning this way with its Visual C++. The integral types file shown in Figure 4 represents a synthesis of the two approaches. Programmers used to constructs such as 'long' have these available. However, the scalar classes, to be discussed later, are defined to be of specific sizes, and hence use the size-specific data types. In the future, as architectures and compilers expand to 64-bits and 128-bits, our code will be backwardly compatible with no modifications, because the types are tied to a specific number of bytes. By providing new types, such as 'double double' or 'long long,' we can also accommodate the new wider formats.

### B.2. System and I/O Classes

Our goal in the design of our system is to abstract the user from details of the operating system. The system library serves this function by encapsulating all operating system specific activities, such as file I/O and character string processing. Both libraries represent a significant improvement over the previous implementation of this environment [10]. The System library supports low-level operations such as file management (opening, closing, reading, writing), character processing (Unicode support), error handling and error notification. All of these require interacting with operating system-specific C functions, and hence must be centralized and abstracted from user-level code.

The I/O library contains a novel approach to file formats. All files in our environment are represented as Signal Object Files (Sof) [10]. Sof alleviates the need for users to read and write data manually (formatting of data tends to be one of the most time-consuming aspects of speech research). All objects know how to read/write from/to a Sof object. In fact, higher-level objects just recurse through their hierarchy of objects for I/O. Sof is simply a smart indexing scheme that keeps track of what objects are stored in a file. It allows multiple instances of an object (a String object can be written several times to the file), and handles ASCII or binary files transparently. It is also machine-independent in that users need not worry about byte-ordering or floating point representations across platforms - this is handled automatically within Sof.

A centralized data storage strategy is essential in speech research, and other data-intensive research areas as well. There is nothing speech-specific about Sof. Sophisticated database management strategies have yet to provide a clean solution for researchers, so the tendency has been to use proprietary formats or unstructured formats (ASCII). Most database packages do not want to deal with byte-formatted data, such as an MPEG audio or video stream, and do not allow partial I/O of these types of data (e.g. retrieve only the middle three seconds of an audio file). There are some public domain file formats being supported within the community [11], but these do not interface well to C++ and have certain limitations in terms of the flexibility (only one instance of an object can be written to a file). Sof is an important part of our overall strategy to ease the programming burden of our users.

### B.3 Math Classes

As shown in Figure 3, the next step up from our low-level IFCs are the math classes. This is actually the first level we expect application developers to interact with - users should not use the system classes directly, and should only use the I/O classes for programming I/O into new class definitions. The math classes consist of scalars, vectors, and matrices. Their implementation follows that of the C++ Standard Template Libraries (STL), with two important exceptions. Our data types are allowed to be size-specific. For example, a Long is always a 32-bit integer, a VectorLong is always a vector of 32-bit integers, etc. This gives the programmer complete control of data sizes. Also, our matrix class is a vector of vectors, where each vector can have a different dimension. This costs very little in overhead, and makes the matrix class much more useful as a container class.

We began implementing the math classes with a simple strategy that did not rely on templates. This was mainly due to a legacy of gcc not supporting a useful model of templates. Gcc was based on an inclusion model in which all code related to a template had to be included in the header file. Obviously, for the system of the scale we are talking about, this is impractical. Fortunately, with the release of version 2.8.X of the gcc compiler, appropriate hooks have been provided to allow source and header files to be separated for template definitions. A header file for a template version of a scalar class is shown in Figure 5. Our benchmarks on code implemented with and without headers seems to show that the template approach is every bit as efficient as the non-template version, and requires much less time to compile. It is a win-win situation thus far. This is summarized in Table 1 below.

Description	Non-Template	Template
Scalar Long:		
executable (kB)	564	1484
memory usage (kB)	1484	10.9
runtime (ms)	31.8	566
compilation time (secs)	1484	11.4
VectorLong:		
executable (kB)	708	1592
memory usage (kB)	1592	12.9
runtime (ms)	76.8	76.8
compilation time (secs)	713	1596
MatrixLong:		
executable (kB)	13.1	66.1
memory usage (kB)	66.1	13.1
runtime (ms)	1596	66.1
compilation time (secs)	13.1	66.1
	4590	1640
	1640	17.6
	17.6	99.4
	99.4	4605
	4605	1648
	1648	17.9
	17.9	76.9

Table 1. A comparison of template and non-template implementations of the math classes. Templates appear to finally be competitive with traditional code.

In the first year of this project, we have completed implementation of the math classes based on our new template approach. This was somewhat of a paradigm shift for us, and hence required some additional training of our programmers. This approach will pay great dividends when we move to higher-level libraries such as data structures, where one expects a template capability. With templates, users can build generic lists, hash tables, etc. of whatever object they desire. This is an extremely important capability for C++ programming. The only drawback of our approach is that the template implementation is specific to gcc, since there exists no clear standard for implementing templates in the ANSI C++ specification. We continually monitor standards activities to see how we can improve the portability of our software.

#### B.4 Concurrent Versions System (CVS) and Anonymous CVS Servers

One of the interesting aspects of our project is that truly concurrent development is being done by a large number of programmers (between six and eight people work on the system continuously). This places great stress on most non-commercial software management systems.

Commercial packages, on the other hand, are expensive (typically \$1K per seat) and hamper our ability to do concurrent development in a distributed fashion as we describe below. Hence, we spent some time this year converting our software management environment from the popular revision control system (RCS) [12] to a state-of-the-art package called Concurrent Versions System (CVS) [12].

CVS allows multiple users to check out the same code, make revisions, and check it back in. The tool automatically merges the code to produce what it thinks is the correct version. CVS also allows one to manage utilities, classes, libraries, etc., all within the same framework. It allows users to check out an entire software tree as a single unit, rather than manage this as individual files as is done in RCS. Unfortunately, this is not as simple as it sounds, and there is a steep learning curve for CVS. Hence, we spent some time developing wrappers for common CVS

functions so that users were protected from the details of CVS [14]. This is important because CVS is unwieldy at times, and can corrupt files if not used carefully. However, we are now routinely using it in production, with satisfactory results.

One of the strongest arguments for using CVS is the capability it has for doing distributed software development. We have implemented an anonymous CVS server that functions much like an ftp server. Users can log into this server, and grab a snapshot of the code currently under development, and do concurrent development if necessary. This is a fairly new capability introduced into CVS in the last year, and is being used by several public domain software projects. In our case, it is an extremely useful capability because it allows users to update a portion of the environment as we make incremental changes, rather than download the entire environment each time we make a small change. Since our final environment will be large, the anonymous CVS distribution strategy allows users to maintain a current copy of the environment without repeatedly doing massive downloads. It also offers the potential for others to contribute to the environment. An on-line tutorial [15] on how to use our anonymous CVS server is available on the web.

### B.5 Software Quality Control

Maintaining quality software releases in a rapidly developing environment is always a challenge, especially when dealing with student programmers. In fact, this is one reason we will add a staff member to the project next year. We have been continually refining our software quality control process. Three important facilities were added to our environment to enhance this process. First, we adopted a public domain memory checking system known as dmalloc as a routine part of our software development cycle. Dmalloc checks for memory leaks and memory related programming errors. Though not an industrial strength product (such as Pure Software's Purify), dmalloc runs on all our supported platforms and does a good job of catching memory problems. It has made a big difference in the quality of our code. Typical released code that used to have two or three memory problems per class now is released with virtually no defects.

A second important step we have taken to improve the quality of our software is to introduce a diagnostic method in each class. As a programmer implements a class, a diagnose method is provided that exercises all methods in the class, and produces predetermined output. We have integrated this into our make facility as well: the 'make diagnose' option automatically generates a test program that uses this method. This facility, coupled with dmalloc, allows the programmer to do a fairly complete test and verification of the class before it is released.

Finally, we have instituted a web-based checklist facility that programmers use to make sure they complete all steps required of a release. As a programmer works through a class, the checklist is updated with each major step. The checklist includes items such as initial design, design review, implementation, diagnostics, debugging, dmalloc, documentation, cross-platform check, and release. Pertinent information, such as the programmer's name and date of completion, are automatically generated and stored in database. Everything is done via the web and an SQL database interface, which makes it extremely easy for the project manager to track progress.

### C. Web-Based Information

Dissemination of information via the web is a critical part of this project. We have overhauled our web site to showcase this project and to make information more readily accessible to our users. The URL for the project is:

<http://www.isip.msstate.edu/projects/speech/>

This web site contains some novel features that are described below.

#### C.1. Project Web Site

We have designed and implemented a uniform look and feel for the web site, as shown in Figure 6. The hierarchy is designed to make it easy for users to access the software, educational resources, and the on-line job submission facility. Most of the web pages are implemented using server-side includes that provide a uniform look and feel for all pages. We have also implemented a search capability using a public domain SQL database package. Records in this database are currently entered manually using a web-based interface. Our attempts at generating the database automatically produces unacceptable results (personal AltaVista was the best tool we looked at, but does not exist as a Unix package currently).

We have made it easy for people to contact us for support by providing a single point of email contact: [help@isip.msstate.edu](mailto:help@isip.msstate.edu). We typically have been able to respond in less than one hour to most requests for help, though traffic has been fairly light thus far. Incoming requests to help are reviewed by the project manager and assigned to the appropriate student worker for resolution.

We have also added a facility for archival of all mail messages sent to our project-specific email alias: [asr@isip.msstate.edu](mailto:asr@isip.msstate.edu). The URL for this archive is [http://www.isip.msstate.edu/data/mailling\\_lists/](http://www.isip.msstate.edu/data/mailling_lists/). Any message to this list is archived and added to the web page by a process that runs nightly using a mail processing tool (Monarch) that generates a threaded display. This archive has proven to be extremely useful when new members join the list. Eventually, we will add an FAQ to the web site to complement the information in the mailing list archives.

We also automatically track downloads of our software. The statistics on who is downloading our software can be viewed at the following URL: <http://www.isip.msstate.edu/data/statistics/web/>. This page tracks hits on a user-specified set of web pages, allowing us to do a thorough analysis of people accessing our web pages and downloading our software.

#### C.2. Documentation

We have begun building on-line documentation for our foundation classes. An example of this documentation [16] is shown in Figure 7. To the left, we have the entire class index in a scrollable window. To the right, we have the documentation for a particular class. Each major heading, such as 'MAIN,' has an overview of the library or set of libraries. The classes are grouped by their position in the hierarchy. Eventually, we will need to supply a search engine for random access to these pages.

Each individual web page is organized similar to a Unix man page, with the appropriate modifications to account for the fact that these are classes instead of library functions. The source code for the class is directly linked to the web page, making it easy for users to study the source code and the documentation simultaneously. The pages are structured as follows:

Section Name	Description	Links Provided
	class name	class header file

Synopsis	broad overview of the class N/A		
Quick Start	a working example	N/A	
Description	brief description of the goals we had in designing the class		N/A
Dependencies	other classes included in the header files and required for compilation		corresponding classes on which this class is dependent
Public Methods	user interface (also shows methods required for all classes)		source code for each method
Public Constants	constants available for general use	N/A	
Protected Data	information for programmers on the internal data		class header file
Private Methods	methods used internally in the class		source code for each method
Examples	simple examples how to use the code	N/A	
Notes	other information relevant to users and programmers	N/A	

Table 2. An overview of the information contained in a typical page documenting a class.

Pages for utilities, applications, and toolkits will follow the same format. A searchable database is also under development to support random access to these pages.

### C.3. Educational Java Applets

We have made a strategic commitment to developing Java applications because of Java's inherent portability. However, this has been a mixed blessing because the Java language and associated toolkits are constantly changing. On top of that, each release of Java seems to have serious bugs that get in the way of developing robust applications. The net effect is that our programming efficiency in Java has been quite low, and our progress on educational applets has been hampered by these problems. Java's lack of portability seems to be an industry-wide problem at the moment.

There are two strategic issues with Java programming. First, there is the GUI, or application interface. Java previously provided the Abstract Window Toolkit (AWT) as its standard interface. We developed a number of applications around this interface [17]. Unfortunately, this interface was recently obsoleted, and replaced with Swing [18]. We spent time this year training our Java programmers on the Swing interface, and porting our existing applications to this new interface. Swing is still somewhat buggy and working around these bugs has been a time-consuming process. We have, however, managed to release several new applets under Swing. An example of one such applet, which teaches the principles of digital filter design, is shown in Figure 8. We would like all our applets to have a common interface. Hence, some amount of retooling of existing applets was necessary.

To make matters worse, Netscape's latest releases of its browser are not fully compliant with Swing. Netscape recently seems to consistently lag Sun Microsystems on support of Java. Hence, users must download a plug-in from Sun to get true Java and Swing compliance. This appears to be the best solution at the moment, Netscape's commitment to retooling its browser appears to be questionable. We provide detailed instructions on our web site [19] for how to download the package and install it in several different configurations. More importantly, we have also programmed our applets to probe the user's browser, detect that this plug-in is missing, and prompt the user with a message indicating the plug-in download instructions.

Despite our Java retooling problems, we have been able to develop two new applets. The first is the digital filtering applet mentioned previously, and shown in Figure 8. In this applet, a user can design a filter using several predefined algorithms involving well-known filter prototypes. The user can also draw a desired frequency response, and let the applet design the corresponding filter. The applet provides details on the actual design, including filter coefficients, frequency and phase response, and a pole/zero analysis. This applet is targeted towards split-level DSP courses, and undergraduate signals and systems classes.

A second applet involves demonstration of fundamental concepts in pattern classification. Users can select pre-stored data sets that highlight the differences between common classification schemes such as principal components analysis, linear discriminant analysis, and Euclidean distance. Users can also optionally enter their own data sets. Classifiers can then be trained on this data, and the results depicted in terms of classification regions. This applet demonstrates several statistical normalization principles used in the feature extraction component of a speech recognizer. It will be useful for graduate courses in pattern recognition, speech recognition, and digital signal processing. It has not been formally released because there are several Java problems with the user interface. We expect this applet to be released in the first quarter of the second year of this project.

We have also begun building a much more ambitious demo that is essentially a port of our Tk/Tcl demonstration of the search algorithm used in the recognizer. This demo has been available for some time as part of the recognition toolkit. It requires the Tk/Tcl toolkit on the platform running the demo, as well as a port of the recognizer. We have demonstrated this application on Windows as well as Unix machines. Our approach in Java was to port the C++ code for the recognizer, and retool the interface. Unfortunately, this turned out to be a much more ambitious effort than planned, for some of the reasons described above. Hence, we decided to better learn how to implement more straightforward applets in Swing first. In the second year of this project, we will return to the problem of providing a Java-based graphical tutorial of how a speech recognition search engine works.

### C.4. Remote Job Submission

One of the truly unique capabilities that we added to the web site this year was the ability to submit a speech recognition job over the Internet to our servers. The interface for this facility is shown in Figure 9. The page can be reached by clicking on the 'Experiments' button on the project main page, or directly from the following URL: <http://www.isip.msstate.edu/projects/experiments/>. The page contains a CPU monitor on the upper left that shows the status of our compute servers, a dialog box on the bottom that is used to interact with the user, and windows to the

right that provide status on active jobs and access to the results produced by the job. After a job is submitted, users can view the results on-line via a URL, or have the results transferred via email.

The current implementation is an initial prototype that will be refined in the coming year. It is certainly not robust and not as graphical as we would like. There are two important features included in the current system. First, users can run a canned experiment and obtain detailed information about how the recognizer analyzed the data. This is useful for comparing performance, replicating well-known results, or learning how the algorithm processes data. Second, users can supply their own audio file via a URL. This is useful if one wants to compare the performance of several systems on the same data. Eventually, we will provide more support for editing data graphically, and interacting with parameters of the system. For the moment, most interactions are done via text boxes, and only a limited set of parameters can be modified.

#### D. Summary

The first year of this project has been productive and much of the groundwork to support the subsequent years of the project has been laid. From a human resources standpoint, the funding from this project has allowed the recruitment, training and development of four promising undergraduate students (two of whom plan to pursue graduate degrees in our department under ISIP's direction), two M.S. students (who plan to continue for a Ph.D.), and one Ph.D. student. All but one of these students will remain on this project until its conclusion. In addition to making fundamental contributions to the project in the first year, all have been trained on our strict software engineering paradigm. Since this project has strong synergy with a related project focusing on core technology development, we are able to leverage many resources and much infrastructure from that project. One of our most senior graduate students has transitioned from the core technology project to this project, and will manage technical aspects of this project in the second year.

The second year of the program provides for a professional staff position to manage the routine operations of the project, particularly support and web site development. We have recruited a senior engineer for this position. This individual has an M.S. degree in Computer Science, and over 20 years of experience in software engineering and computing systems in both industry and academia. For the past several years, he has been the computer systems administrator for another college on campus. Prior to that, he has operated a small consulting company that developed business management software for hospitals. Since he is already a university employee, he has begun interacting with our group on a volunteer basis so that he can familiarize himself with our operation.

This staff position has three primary duties: quality control, web site development, and support. He will directly supervise the students working on the project, and be responsible for software releases, bug fixes, and updates. A near-term goal is to implement the GNU configure [20] distribution paradigm into our system, so that our software will automatically configure itself upon installation. A secondary immediate goal will be to implement problem-tracking software so that all messages to our help line will receive proper prioritization and attention.

The second year of this project is a pivotal year in that we will hold our first set of workshops. In January 2000, we will hold a one-day industrial forum in which we conduct a formal design review of the system, and solicit feedback from the participants on desired enhancements for the coming year. This workshop is tentatively scheduled for January 6-7. We hope to have a mixture of senior professionals from industry and academia (with more of an emphasis on industrial participation). The program will most likely consist of a half-day of design reviews and demos, followed by a half-day of discussion about recommended enhancements to the system. We expect to develop a clear plan of action from this meeting in an attempt to focus our development towards things of interest to the general community.

Our first summer training workshop is also tentatively scheduled for May 21-27. For this workshop, we will invite approximately 12 graduate students (and perhaps senior undergraduates) to spend one week in our lab learning about our system. Travel expenses will be paid for these students. The agenda will most likely consist of morning lectures and demonstrations followed by afternoon laboratories. Initial feedback on this has been very positive, with several sites suggesting they would subsidize attendance by their professionals rather than have their people miss the event. Our facilities can accommodate 12 students comfortably, with a reasonable ratio of students to staff, and adequate access to computing equipment. If interest exceeds this limit, we will investigate alternative facilities. However, our tendency for the first training workshop is to keep it small and focused, so that it can proceed as smoothly as possible.

#### E. REFERENCES

- [1] R.A. Cole, et al, 'Multilingual Access and Retrieval using Communicative Interface Agents (MARCIA),' submitted to Multilingual Information Access and Management: Call for International Research Cooperation, National Science Foundation, June 1999.
- [2] N. Deshmukh, A. Ganapathiraju and J. Picone, 'Hierarchical Search for Large Vocabulary Conversational Speech Recognition,' to appear in IEEE Signal Processing Magazine, September 1999.
- [3] J. Picone, 'Continuous Speech Recognition Using Hidden Markov Models,' IEEE ASSP Magazine, vol. 7, no. 3, pp. 26-41, July 1990.
- [4] Y. Wu, A. Ganapathiraju, and J. Picone, 'Baum-Welch Reestimation of Hidden Markov Models,' [http://www.isip.msstate.edu/publications/reports/isip\\_lvcsr/1999/baum\\_welch/report\\_061599.pdf](http://www.isip.msstate.edu/publications/reports/isip_lvcsr/1999/baum_welch/report_061599.pdf), Mississippi State University, Mississippi State, Mississippi, USA, May 1999.
- [5] N. Deshmukh, A. Ganapathiraju, J. Hamaker, J. Picone and M. Ordowski, 'A Public Domain Speech-to-Text System,' to appear in Proceedings of the 6th European Conference on Speech Communication and Technology, Budapest, Hungary, September 1999.
- [6] H. Hermansky, 'Perceptual Linear Predictive (PLP) Analysis of Speech.' Journal of the Acoustical Society of America, vol. 4, pp. 1738-1752, 1990.
- [7] W.M. Fisher, et al, 'Data Selection for Broadcast News CSR Evaluations,' presented at the DARPA Broadcast News Transcription and Understanding Workshop, Lansdowne, Virginia, U.S.A., February 1998.
- [8] I. Alphonso, N. Deshmukh, and J. Picone, <http://www.isip.msstate.edu/projects/speech/software/transcriber/index.html>, Mississippi State University, Mississippi State, Mississippi, USA, May 1999.
- [9] P. Bothner, et al, 'Welcome to the GCC Project!,' <http://egcs.cygnus.com>, June 1999.
- [10] J. Picone, 'Managing Software Complexity in Signal Processing Research,' Proceedings of the IEEE International Conference on

Acoustics, Speech and Signal Processing, pp. III-41-III-44, Minneapolis, Minnesota, USA, April 1993.

- [11] J. Fiscus, et al, 'SPeech Quality Assurance (SPQA) Package Version 2.3 AND Speech File Manipulation Software (SPHERE) Package Version 2.5,' [ftp://jaguar.ncsl.nist.gov/pub/spqa\\_2.3+sphere\\_2.5.tar.Z](ftp://jaguar.ncsl.nist.gov/pub/spqa_2.3+sphere_2.5.tar.Z), National Institute of Standards and Technology, Gaithersburg, Maryland, USA, June 1999.
- [12] W.F. Tichy, 'RCS - A System for Version Control,' Software - Practice & Experience, vol. 15, no. 7, pp. 637-654, July 1985.
- [13] 'Concurrent Versions System (CVS),' <http://www.cyclic.com/cyclic-pages/howget.html>, Cyclic Software, Washington, D.C., USA, June 1999.
- [14] R. Duncan, 'Software Version Control System,' <http://www.isip.msstate.edu/projects/speech/education/tutorials/cvs/index.html>, Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [15] I. Alphonso, 'CVS Anonymous Download Instructions,' [http://www.isip.msstate.edu/projects/speech/support/info/cvs\\_instructions.html](http://www.isip.msstate.edu/projects/speech/support/info/cvs_instructions.html), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [16] S. Balakrishnama and N. Deshmukh, 'ISIP Software Documentation,' [http://www.isip.msstate.edu/projects/speech/education/tutorials/isip\\_env/](http://www.isip.msstate.edu/projects/speech/education/tutorials/isip_env/), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [17] J. Shaffer, J. Hamaker and J. Picone, 'Visualization of Signal Processing Concepts,' Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, pp. III-1853-III-1856, Seattle, Washington, USA, 1998.
- [18] E. Eckstein, M. Loy, and D. Wood, Java Swing, O'Reilly and Associates, Cambridge, Massachusetts, USA, 1998.
- [19] R. Duncan, 'Java Plug-In Installation Instructions,' [http://www.isip.msstate.edu/projects/speech/support/info/java\\_instructions.html](http://www.isip.msstate.edu/projects/speech/support/info/java_instructions.html), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [20] D. MacKenzie and B. Elliston, [http://www.gnu.org/manual/autoconf-2.13/html\\_chapter/autoconf\\_toc.html](http://www.gnu.org/manual/autoconf-2.13/html_chapter/autoconf_toc.html), Free Software Foundation, Boston, Massachusetts, USA, July 1999.

## **Research Findings: (See PDF version submitted by PI at the end of the report)**

### **MAJOR FINDINGS**

Since the main component of this project is the development and dissemination of speech recognition technology, we did not expect to generate a significant list of technology-related research accomplishments in the first year of the project. Nevertheless, we have begun some interesting research as peripheral activities. These research topics include the use of Support Vector Machines for improved acoustic modeling, the study of the influence of context-sensitive word duration models on conversational speech recognition performance (a step in the direction of introducing prosodics into the speech recognition problem), and implementation of a new segmental Baum-Welch training algorithm (preliminary results for these approaches look promising; detailed results should be available by December 1999). The fact that such research can be easily performed with our system supports our contention that the system is extensible.

With respect to the core technology component of the program, we believe we have delivered a decoder that is extremely efficient for conversational speech recognition, and is competitive with state-of-the-art. Decoding time and memory requirements are within the reach of standard PC-class computers. This is important in the context of this program because it will increase access to this technology by allowing smaller research labs to be able to use the system with fairly modest computing environments. To move to larger domains than conversational speech, such as broadcast news, we have developed a dynamic language modeling capability that caches large language models to decrease physical memory requirements. We have also demonstrated that porting of the system to any gcc compliant platform is fairly easy. The only outstanding issue is wide character support (Unicode) under Linux. Once Linux compilers catch up (expected in Fall'99), our cross-platform support problems should be minimal.

Foundation class development has proceeded using a model similar to

Java, but adapted to the demands of speech research. We have found it extremely useful to abstract the user from the details of the operating system through the use of our system classes. These handle all low-level interactions with the operating system, and centralize many tasks such as memory management, file management and I/O. The next level above the system classes, the math classes, provide the user with basic data type building blocks. Here we have followed an STL model, and have demonstrated that a mixture of templates and fixed classes are an optimal way to compromise between the needs of low-level programmers to see physical data types (such as short integers) and the needs of high-level programmers to be able to build generic math objects (for example, a matrix of signals). Templates have only become practical with recent releases of C++ compilers.

Web-based dissemination of project information has proven to be a mixed bag. Unfortunately, a significant percentage of people interested in our technology and resources appear to still have limited Internet bandwidth and access. Hence, the demand for small distributions that can be downloaded via slow modems still exists. This severely limits what we are able to accomplish in the way of on-line documentation, interactive applets, and distribution of toolkits including enough data to run a reasonable experiment. Our anonymous CVS server has been very useful in that it allows users only to download pieces of the code that have changed - thereby reducing the amount of data one needs to download to remain current.

The remote job submission facility, though extremely unique and impressive, is not receiving the initial traffic we had expected. Users still seem to prefer to download the package and build the demos on their local machines. We hope to improve the visibility of this facility by enhancing and streamlining the user interface in the next year of this project.

### **Research Training:**

Students working on this project develop skills in four major areas:

- core technology: speech recognition

In the first year of this project, students have been exposed to all major components of a speech recognition system: search algorithms, acoustic modeling, training, and lexicon development.

- software engineering: concurrent software development

Our students are trained to use a state-of-the-art concurrent development package (CVS). We make heavier use of this than most, and students quickly learn to grapple with the realities of merging code from several developers.

Students are also exposed to a strict procedure for software design and development that includes design reviews, code reviews, diagnostic testing, memory and format checking, multiple platform portability testing, documentation, and release.

- programming languages: C++, perl, and Tk/Tcl

Since our primary programming language is C++, students are quickly trained to be experts in C++. While we emphasize that code be clean, simple, and easy to read, students nevertheless learn about subtle issues of the language, including memory management and compiler optimization.

Students also perform some of their work in perl and Tk/Tcl. Tk/Tcl is particularly useful for developing high performance GUI-oriented applications. Perl is used primarily to massage data into our standard formats.

- web programming languages: HTML and Java

All students must deliver documentation and presentations in html, and hence quickly become proficient web programmers. A significant portion of our project involves the development of Java applets. Our undergraduates, in particular, get a heavy exposure to Java.

### **Education and Outreach:**

We normally invite high school students to participate in our research as part of a special program created with a local math and sciences school in our area. In the first year of this project, we had one high school student spend a semester programming a Java application. He was an outstanding student who actually produced useful code during his tenure in our group - one of our better experiences with high school students.

### **Journal Publications**

N. Deshmukh, A. Ganapathiraju, and J. Picone, "Hierarchical Search for Large Vocabulary Conversational Speech Recognition", *IEEE Signal Processing Magazine*, p. , vol. , (1999). ) Accepted

J. Picone, J. Hamaker, R. Brown, R.A. Cole and J.H.L. Hansen, "Modern DSP: The Story of Three Greek Philosophers", *IEEE Signal Processing Magazine*, p. , vol. , (1999). ) Accepted

### **Books or Other One-time Publications**

### **Web/Internet Sites**

#### **URL(s):**

<http://www.isip.msstate.edu/projects/speech>

#### **Description:**

This highly interactive web site has been developed as part of the project to disseminate all information about the project. It contains software, publications, Java applets, and even a remote job submission testbed.

### **Other Specific Products**

### **Contributions**

#### **Contributions within Discipline:**

Our contributions in the first year of this project primarily

impact the fields of speech recognition, human language technology, and digital signal processing. Our major accomplishments are as follows:

- Development of ISIP's Foundation Classes (IFCs)
- Creation of a Comprehensive Web Site
- Java Applets
- Remote Job Submission Facility
- Human Resources and Outreach

These are described in detail in various sections below.

- Development of ISIP's Foundation Classes (IFCs)

The foundation classes include general mathematics (scalars, vectors, matrices), data structures (linked lists, binary trees) and other useful abstractions (command line parsing, database management). We have completed implementation of the math classes.

The abstractions we use for these build upon ideas promoted in the ANSI C++ standard template library, but also add important features required for speech recognition research and technology development, such as explicit control of the data size of an integral type.

Several interesting software engineering practices were implemented, including internal diagnostics that automatically test a class. For example, by simply typing 'make diagnose', a test program is generated for a class, which can be run, debugged, checked for memory leaks, etc. This is proving to be an invaluable tool for guaranteeing the quality of the code.

- Creation of a Comprehensive Web Site

The entire project can be viewed from a web site created to support this project. The URL is:

<http://www.isip.msstate.edu/projects/speech>

This site includes a place to download software, educational information such as tutorials, applets, technical reports, application toolkits, some Java applets demonstrating core concepts, and a remote job submission facility described below.

We have implemented a facility to manage and distribute our software using a package called Concurrent Versions System (CVS). This allows users to download our production code via an anonymous CVS server (similar to ftp) that automatically updates their code as revisions are made. CVS is generally considered to be state-of-the-art in software management.

We have also implemented web pages that maintain an archive of our mailing lists used for the project. These archives are located at

[http://www.isip.msstate.edu/data/mailling\\_lists](http://www.isip.msstate.edu/data/mailling_lists)

and are automatically updated daily.

#### **Contributions to Other Disciplines:**

The software being developed within the foundation classes is intended to be a general purpose testbed for signal processing applications beyond speech recognition and human language engineering. The Java applets are of general educational use for undergraduate engineering.

#### **Contributions to Education and Human Resources:**

- Human Resources and Outreach

This grant has directly supported the equivalent of four full-time graduate students and several undergraduates. Undergraduate students have made major contributions in web site development, Java programming, and speech system tool development.

We have also interacted with one high school student in this program. This student developed the first version of a Swing-based Java applet that is an enhancement of an existing applet. He graduated in Spring'99 and is pursuing a degree in computer science.

#### **Contributions to Science and Technology Infrastructure:**

- Java Applets

We have upgraded our existing set of Java signal processing applets to a new interface available in Java called SWING. This is the latest attempt by Java developers to provide a standard high-level interface to applications programmers. The previous interface we used is being obsoleted. Hence, it was necessary to make this step.

We also introduced two new applets. The first applet teaches users about digital filter design. This applet also served as our initial testbed for SWING. The second applet demonstrates pattern classification. Users can create data sets, and classify them using a host of classifiers. This is still under development.

- Remote Job Submission Facility

One development we are most proud of, and somewhat ahead of schedule on, is an applet that allows users to submit speech recognition experiments over the Internet. This applet is central to our vision of Internet-based educational resources. Users can choose an experiment, configure various parameters related to the experiment, and submit the job. The job is distributed to our bank of servers, and results are returned via the web and/or email. Users can supply their own audio data to the recognizer.

This applet is still in the preliminary stages of development, but appears to be quite promising. A major

application of this applet will be to allow users to run the system in debug mode and obtain results which they can use to benchmark their own algorithms.

**Beyond Science and Engineering:**

All technology developed in this project is available via the web and is public domain. Industry as well as academia are free to use this technology with no restrictions. We currently have several industrial partners planning to use the code, and have developed one supporting toolkit that is in production use at a company.

**Special Requirements**

**Special reporting requirements:**

**Change in Objectives or Scope:**

**Unobligated Funds:**

**Animal, Human Subjects, Biohazards:**

**Categories for which nothing is reported:**

Partner Organizations

Other Collaborators

Any Book

Any Product

Change in Objectives or Scope: None

Special Requirements for Annual Progress Report: None

Change in Objectives or Scope: None

Unobligated funds: None

Animal, human subjects, biohazards: None

## RESEARCH AND EDUCATIONAL ACTIVITIES

In the first year of this project, we focused our efforts in three major areas:

- **Core Technology:** extensions of the speech recognition system required to enhance its appeal to our customer base (driven by customer feedback);
- **Foundation Classes:** building blocks such as vectors, matrices, and data structures that simplify and standardize the development of higher-level classes;
- **Web-Based Information:** a comprehensive and informative web site that constitutes a central point of contact for everything related to the project.

We have seen interest in the project grow as evidenced by the fact our mailing list has grown to 150 participants, and we have received several serious inquiries about collaborations based on our system (one of which resulted in participation in a joint NSF/EU proposal [1]). Major milestones for the first year of the project included the release of a fully functional speech recognition system (including feature extraction and training), and the development of a remote job submission capability that lets users submit jobs to our system over the Internet.

### A. Core Technology

State-of-the-art speech recognition technology is the foundation upon which this project is built. We must not lose sight of the importance of this core technology to this project. This technology is being developed in a parallel effort funded by the Department of Defense. The speech recognition system developed in this effort has improved dramatically since the start of our NSF project in August'98. We briefly review the enhancements made to the system in the first year of this project, and then discuss the impact this has made on our efforts within this project. Next, we describe some enhancements made to the system to broaden its appeal to our customer base, and review some initial attempts at cross-platform portability.

#### A.1. System Status

In the past year, three important capabilities have been added to the speech recognition system we have been developing: feature extraction, word graph generation, and Hidden Markov Model (HMM) training. Feature extraction is the process by which the speech signal is converted to a sequence of vectors that serve as input to the recognition system (a continuous density HMM system). This is often called the acoustic front-end. Our approach was to initially replicate an industry-standard front-end consisting of mel-spaced cepstral features [2] and their first and second-order derivatives. This is one step that allows users to duplicate results obtained with other commercial and proprietary systems. This capability was delivered as part of a general front-end capability summarized in Figure 1.

A second key feature added to the system was the ability to generate word graphs. Speech recognition experiments are time-consuming primarily because of the large language models used in the decoding portion [2] of a system (these large language models are desirable because they maximize performance). Hence, to save time on subsequent experiments, a word graph is constructed that represents most plausible, or highly probable, hypotheses that can be generated by this network. This graph is then rescored with new acoustic or language models depending on the nature of the research. This network can be quickly rescored — a process that often runs at

least ten times faster than the process required to generate the network. Because of these time savings, word graph rescoring is an extremely popular method of doing speech research for conversational speech recognition, and is therefore a modality that must be supported for a system to be widely accepted.

This word graph generation feature was added to the system in early 1999, and required significant changes to the way the decoder manages the search process. Fortunately, the current implementation represents a vast improvement in the architecture and performance of the system [2]. Though it took longer than expected to add this feature to the system (we struggled with this for 6 months), the final implementation is extremely clean and efficient, and will have a positive impact on subsequent generations of the system. The system now supports more decoding modes than any commercially available system, and more than most proprietary systems as well.

A third essential feature that was added to the system this year was HMM training [3]. This essentially provided closure on the speech recognition system in that users are now able to build real systems from scratch (previously, one had to borrow some component of the system from another source). We first implemented an algorithm known as Viterbi training, in which only the best state sequence through a network is considered. This is an elegant algorithm in that it is efficient, fast, and consistent with a formal languages view of the speech recognition problem. This allowed us to develop the proper control structures and code infrastructure quickly. Once this was complete, we added Baum-Welch training [4], which is more popular in state-of-the-art systems.

The current system is described in great detail in an upcoming publication [2]. This is one of the first such publications to provide a tutorial on the details of search algorithms, and is being published in a journal that emphasizes education of entry-level graduate students in signal processing. It is our hope that the time spent on this publication will pay off as more researchers are made aware of the availability of this system and project. Our system is also represented at an upcoming major speech conference [5] that will include a panel discussion on the merits of public domain speech technology.

## **A.2. System Enhancements**

In any such project intended to develop public domain code, it is important to adapt to the changing needs of your user community. In the past few years, several major sites in speech research have shifted to the use of features based on an algorithm known as perceptual linear prediction [6]. This technique in some cases has been shown to deliver small improvements in performance. It is an important feature to include if such sites are attempting to replicate their best systems with our public domain system. We completed an initial implementation of this algorithm this year, and are in the process of integrating it into our front-end architecture. A summary of this approach is shown in Figure 2.

Another feature that was requested by several sites this year was the ability to switch language models in the middle of the recognition process. This feature is useful for two reasons: (1) it enables the development of command and control applications that switch between sub-grammars depending on what words are recognized (context-sensitive language models or menus); (2) it allows the recognition system to dynamically recurse through language models at runtime, rather than compile all language models into one big network. The first point impacts the development

of voice-driven menu systems and real-time demonstrations. As a word or phrase is recognized, a new language model can be loaded, providing a small set of words or phrases as the next choice. This strategy is the same as that used by most systems to allow acceleration of desktop menus with voice commands.

A second, and equally compelling, reason to consider this feature is the development of systems that consist of a hierarchy of grammars (for example, sentences in terms of words, words in terms of syllables, syllables in terms of phones, etc.). Many existing systems will compile such a system into one large network. While this is sometimes attractive for efficiency reasons, for research flexibility it is often better to process this hierarchy of networks at runtime (“on the fly”). This allows users to change one small module of the system (for example, allowing a word to be represented multiple ways) without recompiling the entire system.

Implementation of this capability requires use of an approach called “caching.” The system must only activate those portions of the network that represent active words, or words used in the recent past, and leave the remainder of the language model stored on disk. Fortunately, by implementing this strategy, we can handle much larger language models, such as those used in the broadcast news [7] and audio data mining research fields. Since there is currently a shift towards such applications in various research communities including NSF and DARPA, we feel it is important to provide a solution to this problem. Broadcast news language models tend to be large (because they deal with lots of words that occur infrequently) and cannot be managed in memory as a single unit. Our conversational speech recognition system could not handle such a large language model due to the number of bigram entries contained in this model.

In addition to these algorithmic enhancements, several supporting tools were developed to facilitate language modeling. These include a grammar compiler that accepts regular expressions as input and outputs a finite state machine description used by our system, and a grammar compaction algorithm that reduces the size of a word graph without compromising performance. We also developed several display utilities [8] that allow users to look at speech data and analyze its frequency content. We have interacted with several industrial sites on the development of these tools. In fact, one commercial site is making extensive use of this tool in a production data collection capacity, and hired one of our undergraduate programmers for the summer to customize this tool to better suit their unique needs.

### **A.3. Cross-Platform Portability**

We have begun to address issues related to portability across different computing platforms. Our plan remains to develop code under the Solaris operating system on two platforms — Sun Sparc and Intel PC (Solaris x86). Within these environments, we use a common compiler, gcc, provided by the Free Software Foundation (GNU). This compiler is supported across a wide range of platforms, including Microsoft Windows’9X and Windows NT. Since an ANSI standard for C++ has only recently been adopted, and most implementations of C++ are still not compatible (this likely will continue for a few years even after the adoption of the standard), using a common compiler across all platforms is the best way to guarantee portability. Currently, we periodically release a Windows version by porting the GNU environment to Windows, and compiling the code under gcc and the bash shell. This vastly simplifies the Windows porting effort.

Perhaps the fastest growing user population for our system is the Linux community. Fortunately, the backbone of the Linux system is GNU software and the gcc compiler in particular. In fact,

GNU recently turned over development of its compiler, gcc, to an organization known as EGCS [9], which is a collection of free software advocates who have been informally developing compiler extensions to gcc for years. EGCS is responsible for all subsequent releases of gcc (the two have effectively merged) and is the compiler delivered on most production releases of the Linux operating system (RedHat for example). Hence, conformance to gcc standards covers a large portion of both the Windows and Unix markets, yet minimizes portability issues.

However, the Linux version of gcc distributed by EGCS currently lags the Sun Solaris version in one important dimension: wide character support. One of the founding principles of our system is that languages be handled in native format using Unicode character encoding. The current ANSI C/C++ standards support a wide character encoding with a collection of new functions. These must be supported by a runtime library, which Sun Solaris provides, but production releases of Linux do not. Hence, the current release of our code will not compile under Linux. We expect this problem to be resolved within the next three months. It doesn't make sense to provide an interim work around, because anything we do will be quickly obsolete. It is expected that EGCS will do a much better job in the future of tracking standards in C++.

## **B. Foundation Classes**

The most critical part of the first year of this project was to lay the proper foundation upon which all other software can be written. This is perhaps the most difficult part of any technology-specific project — balancing the efficiency and simplicity of the target application with extensibility and flexibility needed by future research. We believe this is the major strength of our project — the environment is object-oriented from the ground up and designed to be neutral to any particular algorithmic approach. Fortunately, we have been able to leverage pre-existing code developed for a much less ambitious project [10] involving speech recognition. However, most of this code needed revamping based on the changes in the C++ language definition and gcc compiler capabilities.

The hierarchy of classes is shown in Figure 10. Our goal in the first year of this project was to deliver everything through the DSP libraries, which we refer to as the ISIP foundation classes (IFCs). In the second year of the project, we begin the “Great Convergence” as we rewrite the speech recognition system using these IFCs. We expect this task to be completed by the end of 1999. This latter version of the system will be the basis for our training workshops which will begin in the summer of 2000.

### **B.1 Integral Types**

At the bottom of our class pyramid is a header file that defines all low-level data types available to the user. These are known as the integral types, and in many ways mirror the syntax of the C programming language. Our current integral types file is shown in Figure 4. This deceptively simple file was the result of much hand-wringing about two competing design philosophies. The C programming language was designed to be somewhat architecture independent. For example, the datatype “int” could be 16-bits long on one machine, and 32-bits long on another. A C program written properly would work on both machines regardless of the specific implementation. On the other hand, in signal processing, we often need access to specific data types. For example, speech signals are stored as 16-bit integers, and need to be represented as such in C. The problem in C is that a 16-bit integer doesn't really exist. Instead, one can use a

“short int” and must assume that an integer is 32 bits long.

There is a growing movement within the C++ community to support data types such as `int32` for a 32-bit integer. In fact, Microsoft seems to be leaning this way with its Visual C++. The integral types file shown in Figure 4 represents a synthesis of the two approaches. Programmers used to constructs such as “long” have these available. However, the scalar classes, to be discussed later, are defined to be of specific sizes, and hence use the size-specific data types. In the future, as architectures and compilers expand to 64-bits and 128-bits, our code will be backwardly compatible with no modifications, because the types are tied to a specific number of bytes. By providing new types, such as “double double” or “long long,” we can also accommodate the new wider formats.

## B.2. System and I/O Classes

Our goal in the design of our system is to abstract the user from details of the operating system. The system library serves this function by encapsulating all operating system specific activities, such as file I/O and character string processing. Both libraries represent a significant improvement over the previous implementation of this environment [10]. The System library supports low-level operations such as file management (opening, closing, reading, writing), character processing (Unicode support), error handling and error notification. All of these require interacting with operating system-specific C functions, and hence must be centralized and abstracted from user-level code.

The I/O library contains a novel approach to file formats. All files in our environment are represented as Signal Object Files (Sof) [10]. Sof alleviates the need for users to read and write data manually (formatting of data tends to be one of the most time-consuming aspects of speech research). All objects know how to read/write from/to a Sof object. In fact, higher-level objects just recurse through their hierarchy of objects for I/O. Sof is simply a smart indexing scheme that keeps track of what objects are stored in a file. It allows multiple instances of an object (a String object can be written several times to the file), and handles ASCII or binary files transparently. It is also machine-independent in that users need not worry about byte-ordering or floating point representations across platforms — this is handled automatically within Sof.

A centralized data storage strategy is essential in speech research, and other data-intensive research areas as well. There is nothing speech-specific about Sof. Sophisticated database management strategies have yet to provide a clean solution for researchers, so the tendency has been to use proprietary formats or unstructured formats (ASCII). Most database packages do not want to deal with byte-formatted data, such as an MPEG audio or video stream, and do not allow partial I/O of these types of data (e.g. retrieve only the middle three seconds of an audio file). There are some public domain file formats being supported within the community [11], but these do not interface well to C++ and have certain limitations in terms of the flexibility (only one instance of an object can be written to a file). Sof is an important part of our overall strategy to ease the programming burden of our users.

## B.3 Math Classes

As shown in Figure 3, the next step up from our low-level IFCs are the math classes. This is actually the first level we expect application developers to interact with — users should not use the system classes directly, and should only use the I/O classes for programming I/O into new

class definitions. The math classes consist of scalars, vectors, and matrices. Their implementation follows that of the C++ Standard Template Libraries (STL), with two important exceptions. Our data types are allowed to be size-specific. For example, a Long is always a 32-bit integer, a VectorLong is always a vector of 32-bit integers, etc. This gives the programmer complete control of data sizes. Also, our matrix class is a vector of vectors, where each vector can have a different dimension. This costs very little in overhead, and makes the matrix class much more useful as a container class.

We began implementing the math classes with a simple strategy that did not rely on templates. This was mainly due to a legacy of gcc not supporting a useful model of templates. Gcc was based on an inclusion model in which all code related to a template had to be included in the header file. Obviously, for the system of the scale we are talking about, this is impractical. Fortunately, with the release of version 2.8.X of the gcc compiler, appropriate hooks have been provided to allow source and header files to be separated for template definitions. A header file for a template version of a scalar class is shown in Figure 5. Our benchmarks on code implemented with and without headers seems to show that the template approach is every bit as efficient as the non-template version, and requires much less time to compile. It is a win-win situation thus far. This is summarized in Table 1 below.

Description	Non-Template	Template
Scalar Long:		
executable (kB)	564	566
memory usage (kB)	1484	1484
runtime (ms)	10.9	11.4
compilation time (secs)	31.8	18.6
VectorLong:		
executable (kB)	708	713
memory usage (kB)	1592	1596
runtime (ms)	12.9	13.1
compilation time (secs)	76.8	66.1
MatrixLong:		
executable (kB)	4590	4605
memory usage (kB)	1640	1648
runtime (ms)	17.6	17.9
compilation time (secs)	99.4	76.9

Table 1. A comparison of template and non-template implementations of the math classes. Templates appear to finally be competitive with traditional code.

In the first year of this project, we have completed implementation of the math classes based on our new template approach. This was somewhat of a paradigm shift for us, and hence required some additional training of our programmers. This approach will pay great dividends when we move to higher-level libraries such as data structures, where one expects a template capability. With templates, users can build generic lists, hash tables, etc. of whatever object they desire. This is an extremely important capability for C++ programming. The only drawback of our approach is that the template implementation is specific to gcc, since there exists no clear standard for implementing templates in the ANSI C++ specification. We continually monitor standards activities to see how we can improve the portability of our software.

## B.4 Concurrent Versions System (CVS) and Anonymous CVS Servers

One of the interesting aspects of our project is that truly concurrent development is being done by a large number of programmers (between six and eight people work on the system continuously). This places great stress on most non-commercial software management systems. Commercial packages, on the other hand, are expensive (typically \$1K per seat) and hamper our ability to do concurrent development in a distributed fashion as we describe below. Hence, we spent some time this year converting our software management environment from the popular revision control system (RCS) [12] to a state-of-the-art package called Concurrent Versions System (CVS) [12].

CVS allows multiple users to check out the same code, make revisions, and check it back in. The tool automatically merges the code to produce what it thinks is the correct version. CVS also allows one to manage utilities, classes, libraries, etc., all within the same framework. It allows users to check out an entire software tree as a single unit, rather than manage this as individual files as is done in RCS. Unfortunately, this is not as simple as it sounds, and there is a steep learning curve for CVS. Hence, we spent some time developing wrappers for common CVS functions so that users were protected from the details of CVS [14]. This is important because CVS is unwieldy at times, and can corrupt files if not used carefully. However, we are now routinely using it in production, with satisfactory results.

One of the strongest arguments for using CVS is the capability it has for doing distributed software development. We have implemented an anonymous CVS server that functions much like an ftp server. Users can log into this server, and grab a snapshot of the code currently under development, and do concurrent development if necessary. This is a fairly new capability introduced into CVS in the last year, and is being used by several public domain software projects. In our case, it is an extremely useful capability because it allows users to update a portion of the environment as we make incremental changes, rather than download the entire environment each time we make a small change. Since our final environment will be large, the anonymous CVS distribution strategy allows users to maintain a current copy of the environment without repeatedly doing massive downloads. It also offers the potential for others to contribute to the environment. An on-line tutorial [15] on how to use our anonymous CVS server is available on the web.

## B.5 Software Quality Control

Maintaining quality software releases in a rapidly developing environment is always a challenge, especially when dealing with student programmers. In fact, this is one reason we will add a staff member to the project next year. We have been continually refining our software quality control process. Three important facilities were added to our environment to enhance this process. First, we adopted a public domain memory checking system known as *dmalloc* as a routine part of our software development cycle. Dmalloc checks for memory leaks and memory related programming errors. Though not an industrial strength product (such as Pure Software's Purify), *dmalloc* runs on all our supported platforms and does a good job of catching memory problems. It has made a big difference in the quality of our code. Typical released code that used to have two or three memory problems per class now is released with virtually no defects.

A second important step we have taken to improve the quality of our software is to introduce a diagnostic method in each class. As a programmer implements a class, a diagnose method is

provided that exercises all methods in the class, and produces predetermined output. We have integrated this into our make facility as well: the “make diagnose” option automatically generates a test program that uses this method. This facility, coupled with dmalloc, allows the programmer to do a fairly complete test and verification of the class before it is released.

Finally, we have instituted a web-based checklist facility that programmers use to make sure they complete all steps required of a release. As a programmer works through a class, the checklist is updated with each major step. The checklist includes items such as initial design, design review, implementation, diagnostics, debugging, dmalloc, documentation, cross-platform check, and release. Pertinent information, such as the programmer’s name and date of completion, are automatically generated and stored in database. Everything is done via the web and an SQL database interface, which makes it extremely easy for the project manager to track progress.

## C. Web-Based Information

Dissemination of information via the web is a critical part of this project. We have overhauled our web site to showcase this project and to make information more readily accessible to our users. The URL for the project is:

<http://www.isip.msstate.edu/projects/speech/>

This web site contains some novel features that are described below.

### C.1. Project Web Site

We have designed and implemented a uniform look and feel for the web site, as shown in Figure 6. The hierarchy is designed to make it easy for users to access the software, educational resources, and the on-line job submission facility. Most of the web pages are implemented using server-side includes that provide a uniform look and feel for all pages. We have also implemented a search capability using a public domain SQL database package. Records in this database are currently entered manually using a web-based interface. Our attempts at generating the database automatically produces unacceptable results (personal AltaVista was the best tool we looked at, but does not exist as a Unix package currently).

We have made it easy for people to contact us for support by providing a single point of email contact: *help@isip.msstate.edu*. We typically have been able to respond in less than one hour to most requests for help, though traffic has been fairly light thus far. Incoming requests to help are reviewed by the project manager and assigned to the appropriate student worker for resolution.

We have also added a facility for archival of all mail messages sent to our project-specific email alias: *asr@isip.msstate.edu*. The URL for this archive is [http://www.isip.msstate.edu/data/mailling\\_lists/](http://www.isip.msstate.edu/data/mailling_lists/). Any message to this list is archived and added to the web page by a process that runs nightly using a mail processing tool (Monarch) that generates a threaded display. This archive has proven to be extremely useful when new members join the list. Eventually, we will add an FAQ to the web site to complement the information in the mailing list archives.

We also automatically track downloads of our software. The statistics on who is downloading our software can be viewed at the following URL: <http://www.isip.msstate.edu/data/statistics/web/>. This page tracks hits on a user-specified set of web pages, allowing us to do a thorough analysis of people accessing our web pages and downloading our software.

## C.2. Documentation

We have begun building on-line documentation for our foundation classes. An example of this documentation [16] is shown in Figure 7. To the left, we have the entire class index in a scrollable window. To the right, we have the documentation for a particular class. Each major heading, such as "MAIN," has an overview of the library or set of libraries. The classes are grouped by their position in the hierarchy. Eventually, we will need to supply a search engine for random access to these pages.

Each individual web page is organized similar to a Unix man page, with the appropriate modifications to account for the fact that these are classes instead of library functions. The source code for the class is directly linked to the web page, making it easy for users to study the source code and the documentation simultaneously. The pages are structured as follows:

Section	Description	Links Provided
Name	class name	class header file
Synopsis	broad overview of the class	N/A
Quick Start	a working example	N/A
Description	brief description of the goals we had in designing the class	N/A
Dependencies	other classes included in the header files and required for compilation	corresponding classes on which this class is dependent
Public Methods	user interface (also shows methods required for all classes)	source code for each method
Public Constants	constants available for general use	N/A
Protected Data	information for programmers on the internal data	class header file
Private Methods	methods used internally in the class	source code for each method
Examples	simple examples how to use the code	N/A
Notes	other information relevant to users and programmers	N/A

Table 2. An overview of the information contained in a typical page documenting a class.

Pages for utilities, applications, and toolkits will follow the same format. A searchable database is also under development to support random access to these pages.

## C.3. Educational Java Applets

We have made a strategic commitment to developing Java applications because of Java's inherent portability. However, this has been a mixed blessing because the Java language and associated toolkits are constantly changing. On top of that, each release of Java seems to have serious bugs that get in the way of developing robust applications. The net effect is that our programming efficiency in Java has been quite low, and our progress on educational applets has been hampered

by these problems. Java's lack of portability seems to be an industry-wide problem at the moment.

There are two strategic issues with Java programming. First, there is the GUI, or application interface. Java previously provided the Abstract Window Toolkit (AWT) as its standard interface. We developed a number of applications around this interface [17]. Unfortunately, this interface was recently obsoleted, and replaced with Swing [18]. We spent time this year training our Java programmers on the Swing interface, and porting our existing applications to this new interface. Swing is still somewhat buggy and working around these bugs has been a time-consuming process. We have, however, managed to release several new applets under Swing. An example of one such applet, which teaches the principles of digital filter design, is shown in Figure 8. We would like all our applets to have a common interface. Hence, some amount of retooling of existing applets was necessary.

To make matters worse, Netscape's latest releases of its browser are not fully compliant with Swing. Netscape recently seems to consistently lag Sun Microsystems on support of Java. Hence, users must download a plug-in from Sun to get true Java and Swing compliance. This appears to be the best solution at the moment, Netscape's commitment to retooling its browser appears to be questionable. We provide detailed instructions on our web site [19] for how to download the package and install it in several different configurations. More importantly, we have also programmed our applets to probe the user's browser, detect that this plug-in is missing, and prompt the user with a message indicating the plug-in download instructions.

Despite our Java retooling problems, we have been able to develop two new applets. The first is the digital filtering applet mentioned previously, and shown in Figure 8. In this applet, a user can design a filter using several predefined algorithms involving well-known filter prototypes. The user can also draw a desired frequency response, and let the applet design the corresponding filter. The applet provides details on the actual design, including filter coefficients, frequency and phase response, and a pole/zero analysis. This applet is targeted towards split-level DSP courses, and undergraduate signals and systems classes.

A second applet involves demonstration of fundamental concepts in pattern classification. Users can select pre-stored data sets that highlight the differences between common classification schemes such as principal components analysis, linear discriminant analysis, and Euclidean distance. Users can also optionally enter their own data sets. Classifiers can then be trained on this data, and the results depicted in terms of classification regions. This applet demonstrates several statistical normalization principles used in the feature extraction component of a speech recognizer. It will be useful for graduate courses in pattern recognition, speech recognition, and digital signal processing. It has not been formally released because there are several Java problems with the user interface. We expect this applet to be released in the first quarter of the second year of this project.

We have also begun building a much more ambitious demo that is essentially a port of our Tk/Tcl demonstration of the search algorithm used in the recognizer. This demo has been available for some time as part of the recognition toolkit. It requires the Tk/Tcl toolkit on the platform running the demo, as well as a port of the recognizer. We have demonstrated this application on Windows as well as Unix machines. Our approach in Java was to port the C++ code for the recognizer, and retool the interface. Unfortunately, this turned out to be a much more ambitious effort than planned, for some of the reasons described above. Hence, we decided to better learn how to implement more straightforward applets in Swing first. In the second year of this project, we will

return to the problem of providing a Java-based graphical tutorial of how a speech recognition search engine works.

#### **C.4. Remote Job Submission**

One of the truly unique capabilities that we added to the web site this year was the ability to submit a speech recognition job over the Internet to our servers. The interface for this facility is shown in Figure 9. The page can be reached by clicking on the "Experiments" button on the project main page, or directly from the following URL: <http://www.isip.msstate.edu/projects/experiments/>. The page contains a CPU monitor on the upper left that shows the status of our compute servers, a dialog box on the bottom that is used to interact with the user, and windows to the right that provide status on active jobs and access to the results produced by the job. After a job is submitted, users can view the results on-line via a URL, or have the results transferred via email.

The current implementation is an initial prototype that will be refined in the coming year. It is certainly not robust and not as graphical as we would like. There are two important features included in the current system. First, users can run a canned experiment and obtain detailed information about how the recognizer analyzed the data. This is useful for comparing performance, replicating well-known results, or learning how the algorithm processes data. Second, users can supply their own audio file via a URL. This is useful if one wants to compare the performance of several systems on the same data. Eventually, we will provide more support for editing data graphically, and interacting with parameters of the system. For the moment, most interactions are done via text boxes, and only a limited set of parameters can be modified.

#### **D. Summary**

The first year of this project has been productive and much of the groundwork to support the subsequent years of the project has been laid. From a human resources standpoint, the funding from this project has allowed the recruitment, training and development of four promising undergraduate students (two of whom plan to pursue graduate degrees in our department under ISIP's direction), two M.S. students (who plan to continue for a Ph.D.), and one Ph.D. student. All but one of these students will remain on this project until its conclusion. In addition to making fundamental contributions to the project in the first year, all have been trained on our strict software engineering paradigm. Since this project has strong synergy with a related project focusing on core technology development, we are able to leverage many resources and much infrastructure from that project. One of our most senior graduate students has transitioned from the core technology project to this project, and will manage technical aspects of this project in the second year.

The second year of the program provides for a professional staff position to manage the routine operations of the project, particularly support and web site development. We have recruited a senior engineer for this position. This individual has an M.S. degree in Computer Science, and over 20 years of experience in software engineering and computing systems in both industry and academia. For the past several years, he has been the computer systems administrator for another college on campus. Prior to that, he has operated a small consulting company that developed business management software for hospitals. Since he is already a university employee, he has begun interacting with our group on a volunteer basis so that he can familiarize himself with our

operation.

This staff position has three primary duties: quality control, web site development, and support. He will directly supervise the students working on the project, and be responsible for software releases, bug fixes, and updates. A near-term goal is to implement the GNU configure [20] distribution paradigm into our system, so that our software will automatically configure itself upon installation. A secondary immediate goal will be to implement problem-tracking software so that all messages to our help line will receive proper prioritization and attention.

The second year of this project is a pivotal year in that we will hold our first set of workshops. In January 2000, we will hold a one-day industrial forum in which we conduct a formal design review of the system, and solicit feedback from the participants on desired enhancements for the coming year. This workshop is tentatively scheduled for January 6-7. We hope to have a mixture of senior professionals from industry and academia (with more of an emphasis on industrial participation). The program will most likely consist of a half-day of design reviews and demos, followed by a half-day of discussion about recommended enhancements to the system. We expect to develop a clear plan of action from this meeting in an attempt to focus our development towards things of interest to the general community.

Our first summer training workshop is also tentatively scheduled for May 21-27. For this workshop, we will invite approximately 12 graduate students (and perhaps senior undergraduates) to spend one week in our lab learning about our system. Travel expenses will be paid for these students. The agenda will most likely consist of morning lectures and demonstrations followed by afternoon laboratories. Initial feedback on this has been very positive, with several sites suggesting they would subsidize attendance by their professionals rather than have their people miss the event. Our facilities can accommodate 12 students comfortably, with a reasonable ratio of students to staff, and adequate access to computing equipment. If interest exceeds this limit, we will investigate alternative facilities. However, our tendency for the first training workshop is to keep it small and focused, so that it can proceed as smoothly as possible.

## E. REFERENCES

- [1] R.A. Cole, *et al*, "Multilingual Access and Retrieval using Communicative Interface Agents (MARCIA)," submitted to Multilingual Information Access and Management: Call for International Research Cooperation, National Science Foundation, June 1999.
- [2] N. Deshmukh, A. Ganapathiraju and J. Picone, "Hierarchical Search for Large Vocabulary Conversational Speech Recognition," to appear in *IEEE Signal Processing Magazine*, September 1999.
- [3] J. Picone, "Continuous Speech Recognition Using Hidden Markov Models," *IEEE ASSP Magazine*, vol. 7, no. 3, pp. 26-41, July 1990.
- [4] Y. Wu, A. Ganapathiraju, and J. Picone, "Baum-Welch Reestimation of Hidden Markov Models," [http://www.isip.msstate.edu/publications/reports/isip\\_lvcsr/1999/baum\\_welch/report\\_061599.pdf](http://www.isip.msstate.edu/publications/reports/isip_lvcsr/1999/baum_welch/report_061599.pdf), Mississippi State University, Mississippi State, Mississippi, USA, May 1999.
- [5] N. Deshmukh, A. Ganapathiraju, J. Hamaker, J. Picone and M. Ordowski, "A Public Domain Speech-to-Text System," to appear in *Proceedings of the 6<sup>th</sup> European Conference on Speech Communication and Technology*, Budapest, Hungary, September 1999.
- [6] H. Hermansky, "Perceptual Linear Predictive (PLP) Analysis of Speech." *Journal of the Acoustical Society of America*, vol. 4, pp. 1738-1752, 1990.
- [7] W.M. Fisher, *et al*, "Data Selection for Broadcast News CSR Evaluations," presented at the *DARPA Broadcast News Transcription and Understanding Workshop*, Lansdowne, Virginia, U.S.A., February 1998.
- [8] I. Alphonso, N. Deshmukh, and J. Picone, <http://www.isip.msstate.edu/projects/speech/software/transcriber/index.html>, Mississippi State University, Mississippi State, Mississippi, USA, May 1999.
- [9] P. Bothner, *et al*, "Welcome to the GCC Project!," <http://egcs.cygnus.com>, June 1999.
- [10] J. Picone, "Managing Software Complexity in Signal Processing Research," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. III-41-III-44, Minneapolis, Minnesota, USA, April 1993.
- [11] J. Fiscus, *et al*, "Speech Quality Assurance (SPQA) Package Version 2.3 AND Speech File Manipulation Software (SPHERE) Package Version 2.5," [ftp://jaguar.ncsl.nist.gov/pub/spqa\\_2.3+sphere\\_2.5.tar.Z](ftp://jaguar.ncsl.nist.gov/pub/spqa_2.3+sphere_2.5.tar.Z), National Institute of Standards and Technology, Gaithersburg, Maryland, USA, June 1999.
- [12] W.F. Tichy, "RCS — A System for Version Control," *Software — Practice & Experience*, vol. 15, no. 7, pp. 637-654, July 1985.
- [13] "Concurrent Versions System (CVS)," <http://www.cyclic.com/cyclic-pages/howget.html>, Cyclic Software, Washington, D.C., USA, June 1999.

- [14] R. Duncan, "Software Version Control System," <http://www.isip.msstate.edu/projects/speech/education/tutorials/cvs/index.html>, Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [15] I. Alphonso, "CVS Anonymous Download Instructions," [http://www.isip.msstate.edu/projects/speech/support/info/cvs\\_instructions.html](http://www.isip.msstate.edu/projects/speech/support/info/cvs_instructions.html), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [16] S. Balakrishnama and N. Deshmukh, "ISIP Software Documentation," [http://www.isip.msstate.edu/projects/speech/education/tutorials/isip\\_env/](http://www.isip.msstate.edu/projects/speech/education/tutorials/isip_env/), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [17] J. Shaffer, J. Hamaker and J. Picone, "Visualization of Signal Processing Concepts," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. III-1853-III-1856, Seattle, Washington, USA, 1998.
- [18] E. Eckstein, M. Loy, and D. Wood, *Java Swing*, O'Reilly and Associates, Cambridge, Massachusetts, USA, 1998.
- [19] R. Duncan, "Java Plug-In Installation Instructions," [http://www.isip.msstate.edu/projects/speech/support/info/java\\_instructions.html](http://www.isip.msstate.edu/projects/speech/support/info/java_instructions.html), Mississippi State University, Mississippi State, Mississippi, USA, June 1999.
- [20] D. MacKenzie and B. Elliston, [http://www.gnu.org/manual/autoconf-2.13/html\\_chapter/autoconf\\_toc.html](http://www.gnu.org/manual/autoconf-2.13/html_chapter/autoconf_toc.html), Free Software Foundation, Boston, Massachusetts, USA, July 1999.

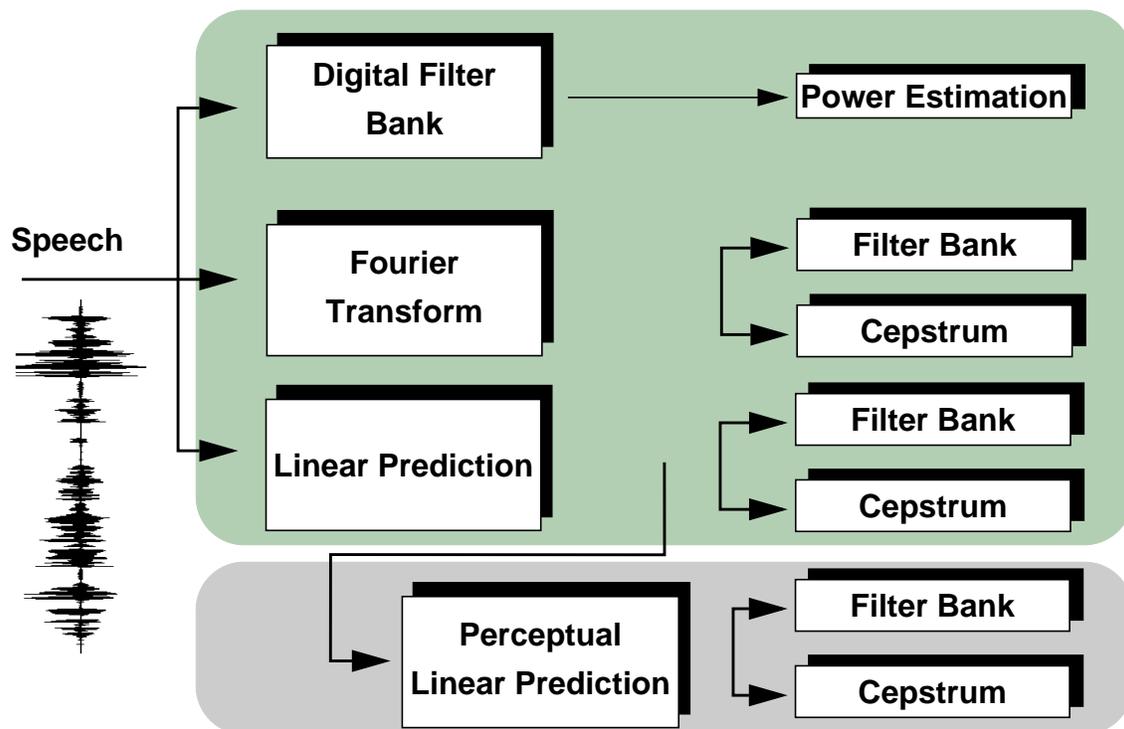


Figure 1. An overview of the front-end portion of the speech recognition system. Two popular analysis techniques, mel-spaced cepstrum and perceptual linear prediction, are supported in the system. Other approaches based on frame-based analysis techniques can be easily added.

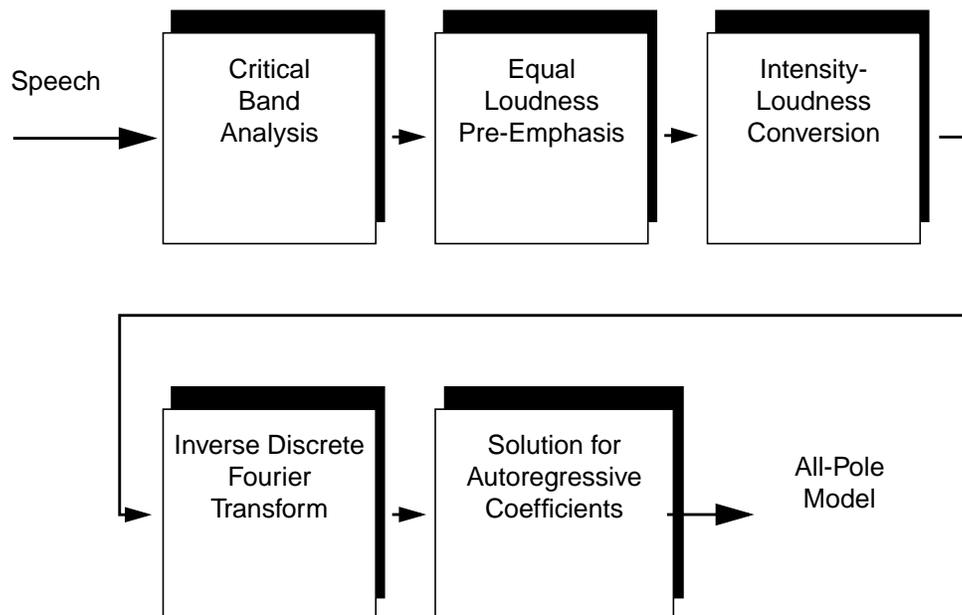


Figure 2. An overview of perceptual linear prediction (PLP) analysis. This front-end has become increasingly popular in recent years.



Any research in the area of signal processing requires the development of large applications in a relatively short period of time. Unfortunately, research commonly suffers from a creative backlog due to rewriting of common functions and the time spent in debugging such things as file I/O. It would be ideal to have a large, hierarchical software environment which can support advanced research in signal processing. The ISIP Foundation Classes (IFCs) and software environment are designed to meet this need, providing everything from complex data structures to an abstract file I/O interface. This software environment starts from the lowest, system level classes, and culminates in a state of the art public domain large vocabulary speech recognition system.

Some significant features of the ISIP software environment include

- unicode compatibility and wide character support to allow multilingual applications
- abstract interface for file i/o
- well-equipped library of DSP functions
- advanced mathematical classes to provide linear algebra and matrix operations

The hierarchical structure of the software environment is as follows:

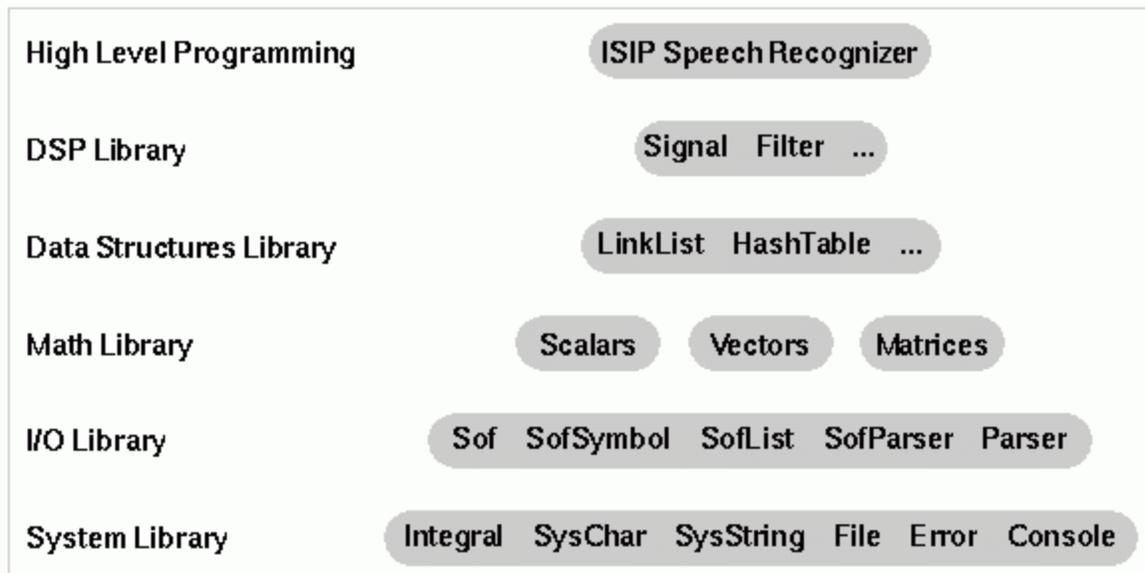


Figure 3. An overview of the hierarchy of ISIP classes. The system and I/O libraries are new additions to the class structure. The math and data structure libraries make extensive use of templates.

```
// file: $isip/class/system/Integral/IntegralTypes.h
// version: $Id: IntegralTypes.h,v 1.4 1999/07/12 18:29:29 duncan Exp $
//

// system include file
//
#include <wchar.h>

// this is the basic isip environment include file. all Integral types
// are defined in this file. these are also implemented as C++ classes.
// all software must be built upon these basic types.
...
typedef void* voidp;
typedef signed char boolean;
typedef unsigned char byte;
typedef wchar_t unichar;
typedef unsigned short int ushort;
typedef unsigned long int ulong;
typedef unsigned long long int ullong;
//typedef short int short;
//typedef long int long;
typedef long long int llong;
//typedef float float;
//typedef double double;
typedef unsigned char byte8;
typedef unsigned short int ushort16;
typedef unsigned long int uint32;
typedef unsigned long long int uint64;
typedef short int int16;
typedef long int int32;
typedef long long int int64;
typedef float float32;
typedef double float64;
```

Figure 4. The integral types define the fundamental building blocks of the ISIP environment. We have taken an approach that requires these types to be a fixed number of bytes.

```

// Scalar: our template scalar class
//
template<class T>
class Scalar {

protected:

    // internal data
    //
    T value_d;

public:

    // required static methods:
    //
    static String& name();

    // required methods:
    // no setDebug required
    //
    boolean debug(unichar* message);
    T size();

    // initialization and release methods.
    boolean init();
    boolean release();

    // destructors/constructors
    //
    ~Scalar();
    Scalar();
    Scalar(Scalar& arg);
    Scalar(T arg);

    // former in-line methods
    //
    operator T();

    Scalar& operator=( T arg);

    // get methods
    //
    boolean get(Scalar& arg);
    boolean get(T& arg);

    // assignment methods
    //
    boolean assign(T arg);

    // mathematical functions
    //
    T min(T arg);
    T min(T arg_1, T arg_2);

    T max(T arg);

    T max(T arg_1, T arg_2);

    T abs();
    T abs(T arg);

    T sign();
    T sign(T arg);

    T factorial();
    T factorial(T arg);

    // useful for DSP
    //
    T limit(T min, T max);
    T limit(T min, T max, T val);

    T limitHard(T thresh, T new_val);
    T limitHard(T thresh, T new_val, T arg);

    T centerClip(T min, T max);
    T centerClip(T min, T max, T arg);

private:

public:

    // define the class name
    //
    static const unichar CLASS_NAME[] = L"Scalar";

    // define the default value(s) of the class data
    //
    static const T DEF_VALUE = (T)0;
    static const T DEF_RAND_MIN = (T)0;

    // default arguments to methods
    //
    static const long NEGATIVE = (long)-1;
    static const long POSITIVE = (long)1;

    static const long ERR = (long)20666;

};

// all classes need to inherit Scalar
//
template class Scalar<long>;
//template class Scalar<short>;

// end of include file
//
#endif

```

Figure 5. A template class definition for a scalar object. This template is used to build classes such as Long, Short, and Float.

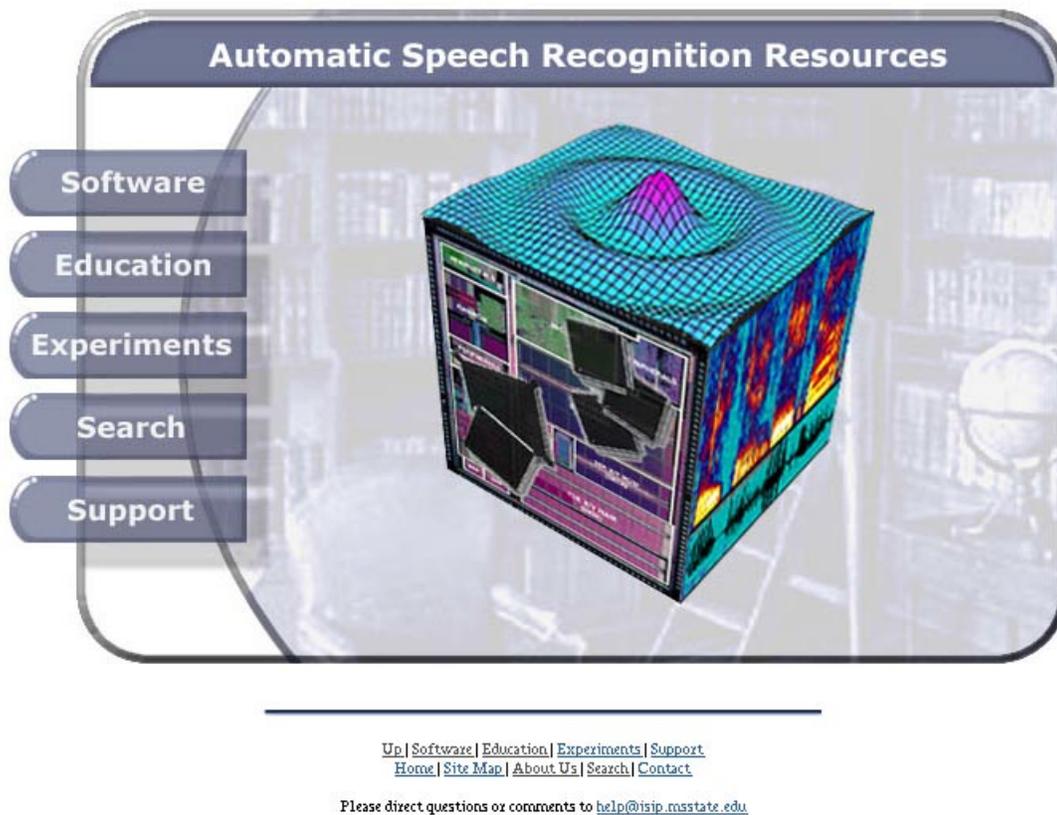


Figure 6. A new set of web pages have been created to support the project. These have been designed to provide easy access to the web site. The choices to the left of the image mirror the physical organization of the web site.

**MAIN**

**CLASSES**

**SYSTEM**

- [Integral](#)
- [MemoryManager](#)
- [SysChar](#)
- [SysString](#)
- [Error](#)
- [File](#)
- [Console](#)

**I/O**

- [Sof](#)
- [SofList](#)
- [SofParser](#)
- [SofSymbolTable](#)

**MATH**

**SCALAR**

- [Boolean](#)
- [Byte](#)
- [Short](#)
- [Ushort](#)
- [Long](#)
- [Ulong](#)
- [Llong](#)
- [Ullong](#)
- [Float](#)
- [Double](#)
- [String](#)

**VECTOR**

- [VectorLong](#)

**MATRIX**

- [MatrixLong](#)

**UTILITIES**

**SCRIPTS**

**documentation**

classes - utilities - scripts - speech - search - up

[name](#) [synopsis](#) [start](#) [description](#) [dependencies](#) [public](#) [constants](#) [protected](#) [private](#) [examples](#) [notes](#)

## Console

**name:** [Console](#)

**synopsis:**

```
gcc [flags ...] file ... -l /isip/tools/lib/$ISIP_BINARY/lib_system.a
#include <Console.h>

~Console();
Console();
static boolean open(SysString& filename, long mode = File::APPEND_ONLY);
static boolean broadcast(unichar* str);
static boolean close();
```

**quick start:**

```
Console cons;
boolean global_error = Integral::FALSE;

Boolean status = cons.open(L"out.txt");

if (status == Integral::FALSE) {
    cons.put(L"this file does not exist");
}

if (global_error == Integral::TRUE) {
    cons.broadcast(L"out.txt is being created");
}

cons.close();
```

**description:**

Console class controls messages (errors and debugging information) which programmers might want to send to stdout. This class does not add new data. Modularity of this class provides user to control debugging of higher and lower level class with separate consoles. The user can save the error and debugging messages in a separate log file and use it for extensive debugging purpose.

**dependencies:**

- [SysString](#)
- [Integral](#)
- [File](#)

**public methods:**

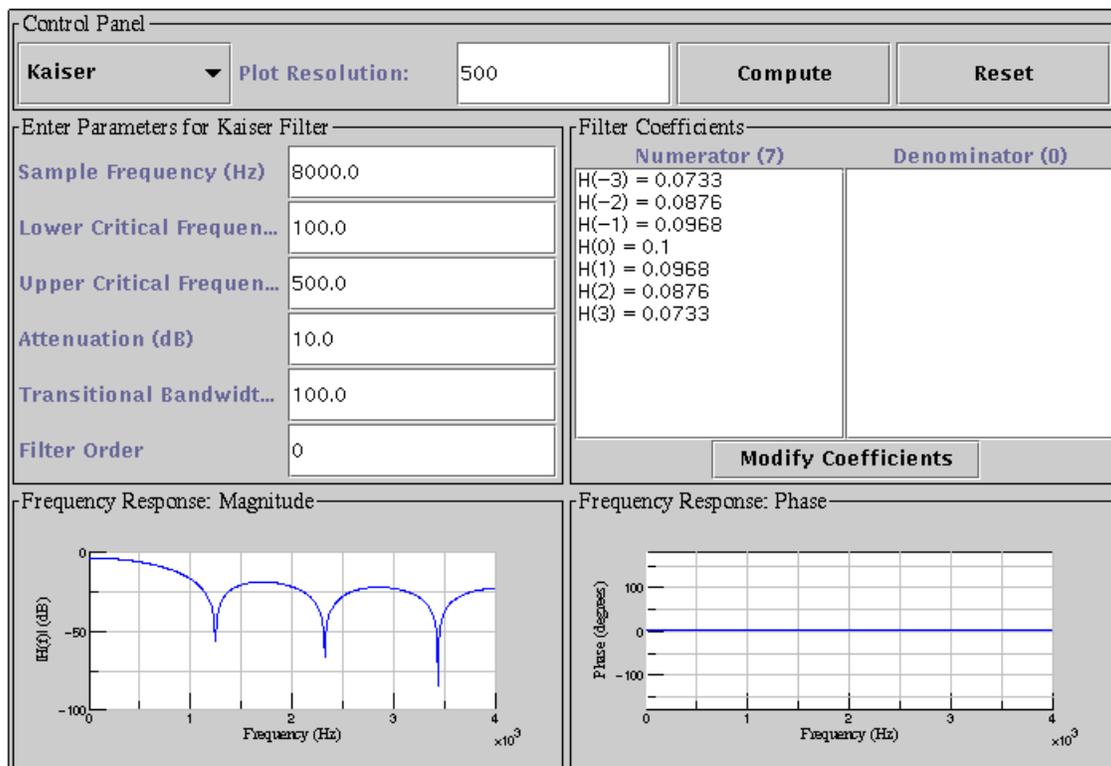
- required static method for the filename operation

```
static String& name();
```

- required static method for the diagnose operation

Figure 7. An example page of documentation for the IFCs. The code is directly linked to the page, making it easy for users to view the code while studying the documentation.

## FILTER DESIGN TOOL



- [Source Code](#)
- A simple [tutorial](#) on using this applet.

Figure 8. An example of a Java Swing applet that demonstrates the concept of digital filter design. Swing has been a mixed blessing. While some aspects of GUI programming are nicely abstracted, other aspects, such as interactions between grid boxes and event handlers, have been problematic.

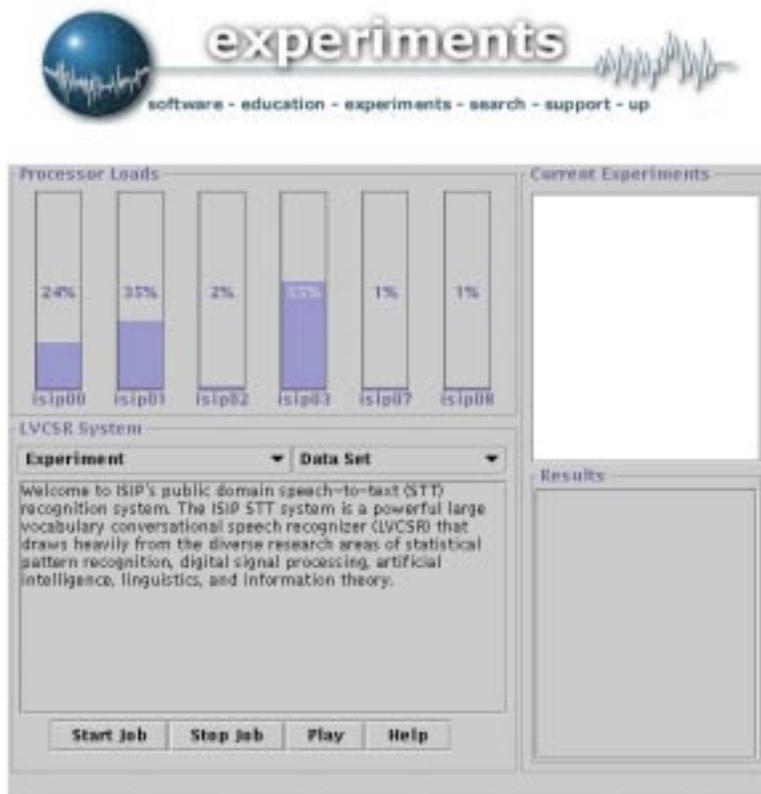


Figure 9. A Java applet that allows users to submit speech recognition jobs remotely to a bank of compute servers. Users can run canned experiments, or supply their own audio data. Parameters for the experiment can be specified via dialog boxes. Results are emailed to the user, and can be examined directly on the web site via links provided in the dialog boxes to the right.

## MAJOR FINDINGS

Since the main component of this project is the development and dissemination of speech recognition technology, we did not expect to generate a significant list of technology-related research accomplishments in the first year of the project. Nevertheless, we have begun some interesting research as peripheral activities. These research topics include the use of Support Vector Machines for improved acoustic modeling, the study of the influence of context-sensitive word duration models on conversational speech recognition performance (a step in the direction of introducing prosodics into the speech recognition problem), and implementation of a new segmental Baum-Welch training algorithm (preliminary results for these approaches look promising; detailed results should be available by December 1999). The fact that such research can be easily performed with our system supports our contention that the system is extensible.

With respect to the core technology component of the program, we believe we have delivered a decoder that is extremely efficient for conversational speech recognition, and is competitive with state-of-the-art. Decoding time and memory requirements are within the reach of standard PC-class computers. This is important in the context of this program because it will increase access to this technology by allowing smaller research labs to be able to use the system with fairly modest computing environments. To move to larger domains than conversational speech, such as broadcast news, we have developed a dynamic language modeling capability that caches large language models to decrease physical memory requirements. We have also demonstrated that porting of the system to any gcc compliant platform is fairly easy. The only outstanding issue is wide character support (Unicode) under Linux. Once Linux compilers catch up (expected in Fall'99), our cross-platform support problems should be minimal.

Foundation class development has proceeded using a model similar to Java, but adapted to the demands of speech research. We have found it extremely useful to abstract the user from the details of the operating system through the use of our system classes. These handle all low-level interactions with the operating system, and centralize many tasks such as memory management, file management and I/O. The next level above the system classes, the math classes, provide the user with basic data type building blocks. Here we have followed an STL model, and have demonstrated that a mixture of templates and fixed classes are an optimal way to compromise between the needs of low-level programmers to see physical data types (such as short integers) and the needs of high-level programmers to be able to build generic math objects (for example, a matrix of signals). Templates have only become practical with recent releases of C++ compilers.

Web-based dissemination of project information has proven to be a mixed bag. Unfortunately, a significant percentage of people interested in our technology and resources appear to still have limited Internet bandwidth and access. Hence, the demand for small distributions that can be downloaded via slow modems still exists. This severely limits what we are able to accomplish in the way of on-line documentation, interactive applets, and distribution of toolkits including enough data to run a reasonable experiment. Our anonymous CVS server has been very useful in that it allows users only to download pieces of the code that have changed — thereby reducing the amount of data one needs to download to remain current.

The remote job submission facility, though extremely unique and impressive, is not receiving the initial traffic we had expected. Users still seem to prefer to download the package and build the demos on their local machines. We hope to improve the visibility of this facility by enhancing and streamlining the user interface in the next year of this project.