**Annual Report for Period:** 08/1999 - 08/2000 **Submitted on:** 05/03/2000

**Principal Investigator:** Picone, Joseph . **Award ID:** 9809300

**Organization:** Mississippi State Univ

CARE: Internet-Accessible Speech Recognition Technology

## Project Participants

**Senior Personnel**

**Name:** Picone, Joseph
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**

**Name:** Chapman, William
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) Software engineer responsible for design, implementation, and support of our software.

**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

**Name:** Duncan, Richard
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/98) programmer responsible for low-level foundation classes
(08/99) as a graduate student, now responsible for signal processing portions of the system.

**Name:** Hamaker, Jonathan
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) Lead developer for the production speech recognition system, including search algorithms and decoder architectures.

**Name:** Srivastava, Shivali
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) junior programmer assigned to the development of speech recognition-related foundation classes such as acoustic modeling and transcriptions.

**Undergraduate Student**

**Name:** Rogers, Jason
**Worked for more than 160 Hours:** No
**Contribution to Project:**
(08/99) Technical publishing

**Name:** Vogel, Jennifer
**Worked for more than 160 Hours:** No
**Contribution to Project:**
(08/99) basic Unix programming and data entry

**Name:** Laundre, David
**Worked for more than 160 Hours:** No
**Contribution to Project:**
(08/99) Java programmer

**Name:** Kay, David
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) Web designer

**Name:** King, Rick
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) Java programmer

**Name:** Robinson, Antonio
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) entry-level programmer and data entry

**Name:** Mohammadi-Aragh, Mahnas
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
Mahnas Jean Mohammadi-Aragh

**Name:** Foote, Joey
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) entry-level programmer and data entry

**Name:** George, Cedric
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) entry-level programming and data entry

**Name:** Alphonso, Issac
**Worked for more than 160 Hours:** Yes
**Contribution to Project:**
(08/99) software engineer specializing in Java and GUI's

**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

## Organizational Partners

**Department of Defense**
The Department of Defense has been a consistent consumer of the technology developed in this project, and regularly gives us feedback on design issues. We support them as an alpha test site since they are typically the first to use our system. Though we have separate funding from DoD, their impact on the production system has been important.

## Other Collaborators or Contacts

We maintain several web-based resources geared towards collaboration. This includes a mailing list for project-related announcements that has grown to over 170 participants. These users provide feedback on design questions in addition to bug reports and other such support requests that arise from use of the software.

## Activities and Findings

**Project Activities and Findings: (See PDF version submitted by PI at the end of the report)**
==========
08/15/99 - 08/14/00: RESEARCH AND EDUCATIONAL ACTIVITIES

In the second year of this project, we focused our efforts in five major areas:
À Production System Release: the first release of the production speech recognition system, based on our modular libraries, is scheduled for July 1.
À Hosted two workshops: a software design review held in January 2000, and a one-week training workshop held in May 2000.
À Software engineering: upgraded our distribution to use the autoconf facility, added an automated report tracking system to our on-line support, created a multi-platform support facility.
À Foundation classes: added algorithms and other signal processing building blocks, introduced classes to handle acoustic models, search algorithms, and knowledge sources, and released a front-end that allows arbitrary algorithms to be implemented using a graphical user interface.
À Java Applets: enhanced our pattern recognition applet with several important new features, including generation of arbitrary data sets, clustering, and visualization of decision surfaces.
The workshops appear to be extremely successful as demand has far surpassed our original estimates for enrollment (and taxed our facilities). The number of serious users of the recognition system is continually growing. It is becoming a challenge to provide same-day response to most support requests, particularly given the wide range of experience levels from the users.

See the attached pdf file for the entire report.

==========
08/15/98 - 08/14/99: 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.

See the attached pdf file for the entire report.

**Project Training and Development: (See PDF version submitted by PI at the end of the report)**
08/15/99 - 08/14/00: MAJOR FINDINGS
In the second year of this project, we introduced two pivotal activities in the project: annual workshops and a rigorous software distribution process. The workshop activities are proceeding smoothly. In January 2000, we hosted nine visitors from several foreign countries (China, Finland), government agencies (FBI, DoD), and industrial sites (IBM, MITRE, Lincoln Labs). We reviewed the goals of the research program, the architecture of the system, provided several demonstrations, and collected feedback on features and capabilities needed in future versions of the system. Several collaborations resulted from this meeting, including an audio indexing opportunity with George Tech, and invited talks at IBM. More collaborations are planned. At the time this report was written, we are completing plans for the May 2000 workshop, which will include 25 participants, of which 20 are graduate students. Demand for the workshop was strong - we tripled the number of participants over what was originally budgeted. We turned away approximately 10 potential participants due to space and resource limitations.
One surprise in the second year of the project has been the growing importance of supporting the Linux operating system, and the difficulties in doing so. Through experience, we have learned that despite using the same compiler and software (GNU gcc, make, etc.) but a different flavor of Unix (Sun Solaris x86), we cannot guarantee robust releases for Linux users. Hence, we spent some time enhancing our ability to achieve platform independence across a wide range of Unix platforms. We now routinely run our releases through a suite of systems before actually making the release. We also have encountered a large demand for Windows-based ports of our system. Currently, we do this through the use of a Unix-like shell available under Windows (Cygnus' cygwin tools). This has also increased the support overhead in making releases of our system. Despite the demand for a native Windows port, we are not assigning this a high priority until the core system is stable. This is primarily because there is a lack of standardization of C++ compilers and development environments, therefore making it hard to support both environments simultaneously when the software base is changing rapidly.
With the addition of a full-time staff person, it has been possible to expand our on-line support activities. We now handle approximately 5 serious support requests per day. Many support requests involve hand-holding inexperienced users on basic computing issues - we are still struggling with how to deal with these in a timely manner. The remainder involve unexpected program crashes that require extensive diagnosis. We have implemented an automated problem tracking system to make sure such requests are properly tracked. We also provide an ability for users to upload their data to us, so that we can replicate their problems. The majority of serious problems seem to relate to compiler-dependent problems (for example, a bug which did not show up on one version of an operating system, but is fatal on a different architecture with a different compiler). This also exposed the critical need for an in-house multi-platform evaluation facility.
Last but not least, we have made extensive progress enhancing basic functionality of the system. We have developed a generalized hierarchical search engine that is the first such system of its type. We provide an ability to decode speech using either networks for N-gram language models. Users can supply either type of model, or both, at any level of the system. For example, it is possible to constrain the search using N-grams of parts of speech, as well as N-grams of words. We have also developed a front-end that allows users to configure the signal processing portions of the system without writing any code - algorithms can be specified using a GUI-oriented tool that automatically schedules the necessary operations required.
08/15/98 - 08/14/99: 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:**

==========

08/15/99 to 08/14/00:

- Workshop:

  Students attending our one-week summer workshop receive 5 half-day lectures on theory and implementation details, and then spend afternoons in the lab acquiring hands-on skills. Students are led through some lab exercises by a senior graduate student, and then supervised by lab instructors on more open-ended assignments. They work from individual workstations, and can see the lab instructor's work on a projection system at the front of the class. This format has been very successful for training students on our software.

==========

08/15/98 to 08/14/99:

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 us 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.

**Outreach Activities:**

==========

08/15/99 to 08/14/00:

- Workshops:

As mentioned elsewhere, the two workshops we host annually are a major component of our outreach activities. Based on our attendance lists, we seem to be achieving our goal of attracting people new to the details of speech recognition technology.

- Undergraduates:

We have been successful at recruiting more underrepresented groups to work in entry-level capacities as undergraduate hourly workers.

==========

08/15/98 to 08/14/99:

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. 84, vol. 16, (1999). ) Published
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. 48, vol. 16, (1999). ) Published

## Books or Other One-time Publications

Vishwanath Mantha, "A New Look at the SWITCHBOARD Corpus", (2000). *Thesis*, Submitted
Bibliography: Mississippi State University

Aravind Ganapathiraju, "Support Vector Machines for Speech Recognition", (). *Thesis*, Submitted
Bibliography: Mississippi State University

J. Picone, C. Atkeson and I. Alphonso, "Harnessing High Bandwidth: Applications in Speech Recognition", (2000). *Conference Presentation*, Published
Bibliography: presented at the Spring 2000,
Internet2 Member Meeting, Washington, DC, USA, March 2000

P.J. Price and J. Picone, "Automatic Speech Recognition: Better Than Text?", (2000). *Conference presentation*, Published
Bibliography: presented at the AAAS Annual Meeting and Science Innovation
Exposition, Washington, D.C., USA, February 2000

G. Doddington, A.Ganapathiraju, J. Picone and Y. Wu, "Adding Word Duration Information to Bigram Language Models", (1999). *Conference publication*, Published
Bibliography: presented at IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, Colorado, USA, December 1999.

N. Deshmukh, A. Ganapathiraju, J. Hamaker, J. Picone and M. Ordowski, "A Public Domain Speech-to-Text System", (1999). *conference paper*, Published
Bibliography: Proceedings of the 6th
European Conference on Speech Communication and Technology, vol. 5, pp. 2127-2130, Budapest, Hungary, September 1999.

## 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

**Product Type:** Software (or netware)
**Product Description:**
Object-oriented speech recognition software built from a hierarchy
of general purpose modules including math, data structures, and signal processing. Makes extensive use of C++ and templates.

**Sharing Information:**
The software has been placed in the public domain and can be
downloaded from our web site.

**Product Type:** Teaching aids
**Product Description:**
Java applets that teach fundamentals of signal processing
and pattern recognition.

**Sharing Information:**
The software has been placed in the public domain and can be
run or downloaded from our web site.

## Contributions

**Contributions within Discipline:**

==========

08/15/99 to 08/14/00:

Our contributions in the second year of this project primarily impact the fields of speech recognition, human language technology, and digital signal processing. Our major accomplishments are as follows:

- Production System Release:
  -> first release of the production speech recognition based on our modular libraries is scheduled for July 1.

- Hosted two workshops:
  -> a software design review held in January 2000 that included a one-day training session
  -> a one-week training workshop to be held in May 2000 that will include 25 participants representing 6 countries, 17 universities, one government agency and one company. Twenty of the participants are graduate students.

- Software engineering:
  -> added an automated report tracking system to our on-line support
  -> upgraded our distribution to use the autoconf facility: a standard way of distribution Unix software in which the distribution automatically configures itself.
  -> enhanced our ability to do multi-platform testing (we now benchmark our releases on Sun Sparc, Sun Solaris x86, Linux, and Windows before making a release), and bug detection (we make extensive use of professional strength debugging tools).

- Foundation classes:
  -> refined the existing core mathematics classes
  -> added data structures, algorithms, and other signal processing building blocks.
  -> introduced classes to handle acoustic models, search algorithms, and knowledge sources.
  -> released a production front-end that allows arbitrary algorithms to be implemented using a graphical user interface

- Java Applets:
  -> enhanced our pattern recognition applet with several important new features, including generation of arbitrary data sets, clustering, and visualization of decision surfaces.

- On-Line Support:
  -> we are averaging approximately 5 serious support requests per day, and support a user group that has grown to over 170 participants. Support is definitely becoming a time-consuming issue.

==========

08/15/98 to 08/14/99:

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/mailing_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 Human Resource Development:**
==========
08/15/99 to 08/14/00:

Our workshops are an excellent example of the outreach activities in this project. Most of the participants represent schools not prominent in the field of speech recognition. We have at least one underrepresented university participating in our May 2000 workshop. Demand for the workshop was so great that we increased the size from 8 participants (originally budgeted) to 25 participants in the first year of the summer workshop.

We also significantly improved participation of undergraduate students from underrepresented groups in our research project in the second year of the program.

==========
08/15/98 to 08/14/99:

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:**
==========
08/15/99 to 08/14/00:

- Workshops:

We have developed extensive on-line documentation for the two workshops we host. All presentation materials are available from the web. Related coursework (such as an updated set of notes for a speech recognition course) is also available from the web.

- Tutorials:

Now that our software base has been stabilizing, we have begun developing extensive turnkey scripts that run canned experiments on important applications (such as conversational speech recognition). These scripts are extremely important in that they show users how to implement subtle details of the technology.

==========

08/15/98 to 08/14/99:

- 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:** None

**Change in Objectives or Scope:** None

**Unobligated funds:** less than 20 percent of current funds

**Animal, Human Subjects, Biohazards:** None

## 08/15/99 — 08/14/00: RESEARCH AND EDUCATIONAL ACTIVITIES

In the second year of this project, we focused our efforts in five major areas:

- **Production System Release**: the first release of the production speech recognition system, based on our modular libraries, is scheduled for July 1.
- **Hosted two workshops**: a software design review held in January 2000, and a one-week training workshop held in May 2000.
- **Software engineering**: upgraded our distribution to use the autoconf facility, added an automated report tracking system to our on-line support, created a multi-platform support facility.
- **Foundation classes**: added algorithms and other signal processing building blocks, introduced classes to handle acoustic models, search algorithms, and knowledge sources, and released a front-end that allows arbitrary algorithms to be implemented using a graphical user interface.
- **Java Applets**: enhanced our pattern recognition applet with several important new features, including generation of arbitrary data sets, clustering, and visualization of decision surfaces.

The workshops appear to be extremely successful as demand has far surpassed our original estimates for enrollment (and taxed our facilities). The number of serious users of the recognition system is continually growing. It is becoming a challenge to provide same-day response to most support requests, particularly given the wide range of experience levels from the users.

### A. Production System Release

An overview of a typical speech recognition system is shown in Figure 1. There are three main components to this system: signal processing, language modeling, and search. We have had a prototype system in release now for over one year. This system was recently evaluated as part of DoD's yearly evaluation cycle [1] — an important step towards gaining wider acceptance of the system as a state of the art system. We are now nearing the first major release of our production system that is built from the ISIP foundation classes. We currently have many of the core pieces implemented, including the signal processing section (described later), acoustic modeling, and a prototype hierarchical search engine that was demonstrated at our January workshop. Language modeling classes are currently under development and nearing completion. Integration of these classes into a system has begun, and is expected to be completed by mid-summer.

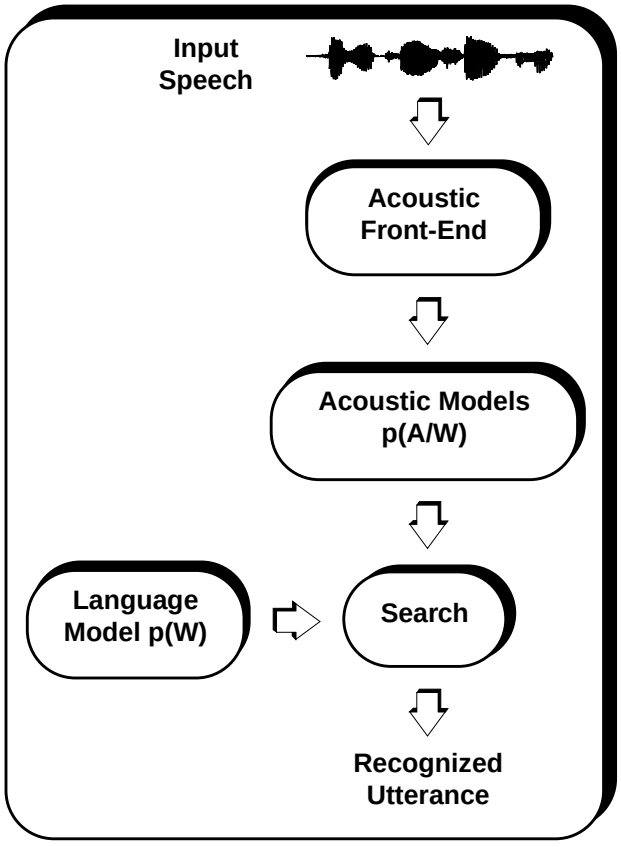


Figure 1. A typical speech recognition system.

Novel aspects of this system include a generalized hierarchical search engine, shown in Figure 2, and a flexible approach to

signal processing that allows new algorithms to be implemented using a CASE-based approach involving a graphical user interface. The search engine is crucial to this system, in that it is by far the most complex and unwieldy component. A clean implementation that provides users reasonable programming hooks into all levels of the process is very important. The generalized approach below still requires significant work with respect to efficiency and memory resources. We are using the prototype system currently in release to develop these details, and then transferring that knowledge into the production system.

Pieces of the production system, particularly the foundation classes, have been in release since November 1999. The most current versions of the code are also available from our anonymous CVS server. The foundation classes are slowly stabilizing as we add more upper-level functionality and expose more bugs. The C++ language definition and implementation has recently begun to stabilize, making many things possible using the latest version of the compilation tools. This in turn has allowed us to change several aspects of our class designs. We believe we have reached convergence on most major aspects of the system design, and now plan to remain backwardly compatible with subsequent releases. We have also developed tools to automatically convert data formats between the prototype and production systems, thereby allowing users to leverage features of both systems while the latter is under development.

## B. Outreach Via Workshops

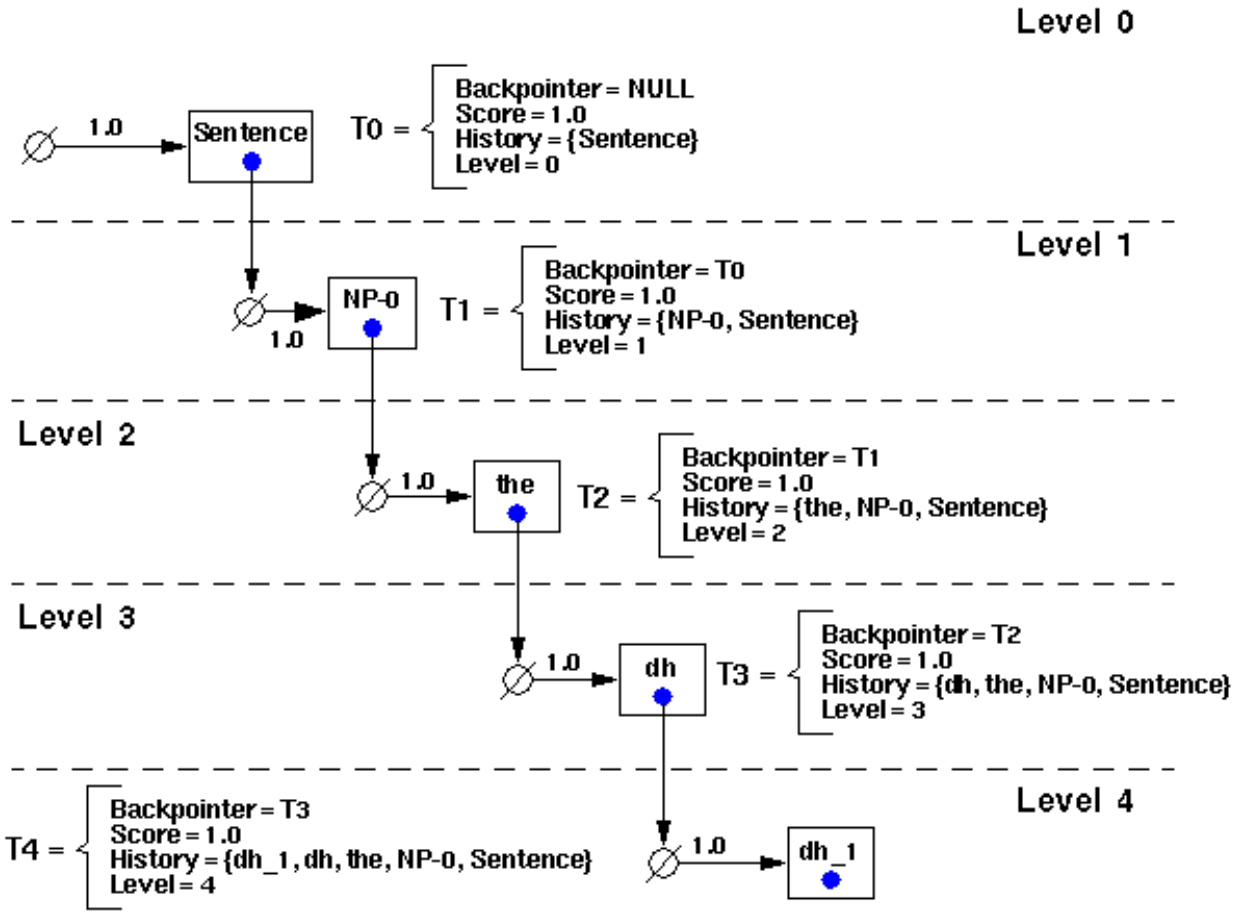


Figure 2. An overview of a generalized search engine that allows users to implement speech recognition systems as a hierarchy of knowledge sources.

In January 2000, we hosted our first annual design review workshop. The primary function of this workshop was to review our software design. A second important goal was to solicit the community for input on new features. We arranged this workshop to be a two-day event. The first day was devoted to an overview of the system, including demos, and general discussion. The second day consisted of a training session where we walked users through our on-line tutorial. A complete archive of the workshop, including presentation slides, is available on-line at the workshop web site [2]. Participants received notebooks containing handouts of the lecture notes, as well as a CDROM containing all software and instructional materials used at the workshop.

Attendance at this workshop was slightly lower than expected: 9 participants from several foreign countries (China, Finland, Korea), government agencies (FBI, DoD), and industrial sites (IBM, MITRE, Lincoln Labs). This was partially due to the time at which the workshop was held (January 5) and concerns about residual Y2K problems. Nevertheless, it was a very productive workshop in that users were able to build an entire large vocabulary continuous speech recognition (LVCSR) system during the training session, and left feeling very good about the software. Several collaborations resulted from the workshop, including invited talks at IBM, a collaboration with Georgia Tech on the classroom of the future [3], and potential collaborations with MITRE on various DoD-related speech recognition applications.

Feedback from this workshop was generally positive. Some examples are shown in Figure 3. We were particularly interested in thoughts about the summer workshop, and action items for the following year. A summary of the discussion about future plans is available [4] on the web. Given the diverse group of participants, it was hard to form a consensus on the priorities of these items. However, generally speaking, there were no surprises relative to our current plans.

In May 2000, we will offer our first extended training workshop [5], which is geared towards entry-level graduate students. Travel funds are provided to encourage graduate students from underrepresented institutions to attend. The program [6] for this week-long workshop combines morning lectures on theory with afternoon laboratories focused on skill-building. The morning lectures are split into two parts: fundamental theory and applications to speech recognition (explanations of how the theory is actually implemented in a system). The laboratories involve skill-building projects ranging from basic recognition foundation class programming to conversational speech recognition system development. Participants literally leave the workshop with a toolkit to run some of today's important research tasks, and should be able to make programming-level modifications to the system.

Twenty-four participants, including 18 graduate students, will attend the first workshop. Nineteen institutions from seven countries are represented, including by design a broad range of U.S. universities. Established research groups such as MIT, Rutgers and University of Colorado at Boulder are represented. Underrepresented universities such as North Carolina A&T are also participating. Further, universities less prominent in speech recognition research, such as University of Houston, University of Denver, and Old Dominion University, are represented by graduate students early in their Ph.D. programs. Hence, we are well along towards our goal of increasing access to speech technology by incorporating underrepresented groups. In fact, workshop enrollment was three times what was originally expected, and our acceptance rate was approximately 66% of the applicants.

We plan to broadcast live still images from this workshop on the conference web site [5] this year using a networked camera. Next year, we will attempt a live Internet broadcast using facilities

1. Things I liked: Clearly structured presentation of the system. Presentations covered most aspects of the system. Code-testing (e.g. the diagnose() function) was stressed. Demo. Instrumental in inspiring confidence in the system.

2. Things that need improvement: In the demo, most of my time was spent typing in long pathnames. I suggest writing a (one-line) shell script for each step, and simply allowing the user to read the pathnames.

3. Things in a summer training workshop: Make a list of use-scenarios (e.g. word-lattice-construction; lattice-rescoring; 1-pass decoding; viterbi-training; EM; segmentation; alignment; speaker adaptation; system-extension with new acoustic models like SVMs, etc.) and go over how to do each one.

4. Things to do differently: I would spend a bit more time on some of the tougher algorithmic issues, e.g. how exactly right-word extensions work for cross-word context dependence, and what happens if you want to look (say) 5 phones to the right. And when traces are extended, presumably there are some circumstances where two traces can be merged; how exactly is this handled? Basically, I'd like to get a more explicit sense of what the toughest issues were, and how the system handles them. Perhaps a 1/2 hour on this.

Overall, I thought it was an excellent review.

1. Things you liked about the workshop:
The workshop went well, and met my goals:
a. get to know the ISIP group to facilitate collaboration
b. get the software up and running
c. get a feel for where the project is going
d. articulate my needs

2. Things that need improvement: Hmmm, having trouble thinking of practical changes....
Maybe a tutorial on "current issues in speech recognition"

3. Things you would like to see in the summer training workshop: I plan to send students not experienced in speech recognition, so tutorials would be useful.

4. Things you would like to see us do differently for the design review:
see 2.

I appreciate it very much that I had the opportunity to come to MSU to learn automatic speech recognition (asr). I have done survey on the availability of asr software which I can apply LDC DARPA-TIMIT data (on cd-rom) for training an asr system. But I did not have any one until I talked Mr. Dave Graff of LDC who suggested that I talked to you.

The hospitality you extended to me is sincerely appreciated. I have done some paper-reading on speech recognition years ago. But this is the first time I am trying to use it for data systems. The need at my organization is speaker-independent asr.

Most of the telephone data is conversational, I will try to work on Switchboard-type of data, hopefully in the summer.

I tended to think that it would be a nice thing for a user if the input and modules can be simplified. I am an engineer at my job; the real end-users may not be engineers.

Figure 3. Examples of feedback collected from SRSDR'00. Comments about topics for the extended summer workshop we extremely helpful and have been incorporated into the program.

available at MS State (at no charge to the project). All materials developed for the workshop, including laboratory exercises, will be posted on the web site. We have already had several requests for these materials from people who cannot attend the workshop.

## C. Software Engineering

With every release of our system, we accumulate more experience with the challenges of supporting research software in a Unix environment, where things tend to be less standardized. With the addition of a full-time software engineer this year, we were able to address these issues in a much more powerful manner. One of the most popular distribution mechanisms for software in Unix is an automatic configuration system developed by the GNU organization. This system, known by various names such as configure and autoconf, automatically searches the system during installation for required software packages, and configures software accordingly. A typical installation procedure consists of the following sequence of commands:

- tar xvf isip_proto_v5.3.tar — unpacks the software distribution
- cd isip_proto_v5.3: — enters top-level directory
- configure --prefix=/usr/local/isip — configures the software and sets the installation directory
- make — compiles and links the software
- make install — installs the software

This deceptively simple procedure has taken years of refinement for Unix systems, and involves locating many important tools (gcc, perl, Tk/tcl, shells, etc.), and deciding how best to build the software given the local system's capabilities. In recent years, installation using this approach has become fairly smooth under a multitude of Unix systems.

The overhead cost in adopting this form of installation procedure is high. The tools to do this are not trivial. This year, we finally mastered this software and incorporated this facility into our releases of the prototype system. This should resolve most support issues we have dealt with involving system incompatibilities, and definitely minimizes the effort required by users to install the system (since everything is automatic). Migrating our previous installation procedure required a major overhaul, but was clearly well worth the effort.

As mentioned previously, support activities are requiring an increasing amount of time. Hence, it became clear that we needed to install a formal method of support request tracking. We have installed a public domain system called RT — Request Tracker [7]. This is a powerful system that has most of the standard features included in such packages: ticket numbers, time-stamping of requests, automatic acknowledgements, queues, and resolution tracking. RT is popular, particularly within our university. We are able to leverage other installations on campus, and enjoy excellent technical support on the package from other campus organizations. RT has been extremely useful in managing our support line. For example, we are now able to generate automated reports on the timeliness of our service. Any email to our support line, help@isip.msstate.edu, is automatically routed to the RT system and acknowledged. Our goal is to provide a reasonable response to each request within a 24 hour period when support staff are on-site. Our software engineering staff position manages this system as part of his job responsibilities.

A third step we have taken to improve the quality of our distributions involves the development of a multi-platform and multi-OS environment to check releases for compilation and run-time problems. We purchased a Pentium workstation and installed multiple operating systems: Sun

Solaris x86, Windows NT, and Linux. We also have Sun Sparc Solaris machines in our lab. Further, we recently acquired an IBM AIX machine as a donation from IBM. These machines are used to check every release before it is actually made available to the public. Though we tend to be very methodical in our debugging and validation methods, we have found instances where software will pass validation runs on all but one of the operating systems. Such problems are subtle and rarely flagged by compilers (even though we use a common compiler across all machines). Though such multi-platform checks are time-consuming, they are necessary if one wants to avoid problems. Linux support, in particular, has recently become a critical requirement.

Along similar lines, we have also recently acquired professional strength code development tools for Unix. Previously, we have been relying on a public domain code checker — dmalloc. Unfortunately, software of the complexity we are developing breaks most public domain tools. Such tools are not able to properly diagnose and isolate problems. Hence, we now have access to two professional quality development tools offered by Rational Software. Even these tools do not catch 100% of the problems observed in our code, primarily due to the complications that arise from the use of many levels of C++ templates. However, such tools are often able to help us resolve problems in minutes rather than hours, and have greatly increased productivity.

Using these tools, we were able to isolate and fix a number of memory bugs in our current releases. Some of these were quite subtle and took hours of run-time to reproduce. However, our current releases are now free of all known memory bugs, and are vastly improved over previous releases. The software has been checked on a much wider range of tasks as well.

## D. Foundation Classes

The foundation classes, upon which all higher-level software is built, continue to grow in terms of their breadth and depth. We currently support the following libraries in our class hierarchy:

- system (i.e., Console, MemoryManager)
- input/output (i.e., Signal Object File, Sof Parser)
- math (i.e., Scalars, Vectors, and Matrices)
- data structures (i.e., Linked Lists, Hash Tables)
- shell (i.e., CommandLine, Filename)
- multimedia (AudioFile)
- statistics (GaussianModel, StatisticalModel)
- algorithms (Cepstrum, Linear Prediction)
- signal processing (FrontEnd, Features)
- pattern recognition (PCA, LDA)
- automatic speech recognition (Recognizer).

This year, we have focused on higher-level libraries such as Statistics, which provides statistical models for each state in our acoustic models, and Data Structures, which provides graph objects used in the search engine.

Our recent focus has been the development of the signal processing portion of the system. An overview of the tool we have developed to provide users an easy way to build signal processing systems is shown in Figure 4. The users have at their disposal any of the tools available in our Algorithms library. For example, an industry-standard front-end uses a Fourier Transform operation, a Cepstrum calculation, time derivatives of feature vectors, and a special type of normalization algorithm. Each of these modules is available as a class under the Algorithms library. Each class has a special set of methods that interface to the application builder, known as

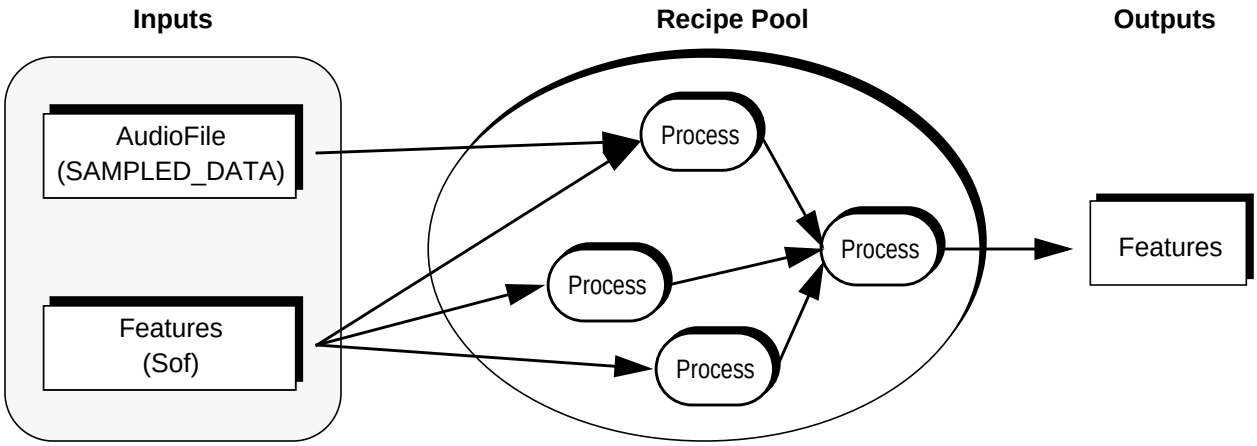


Figure 4. An overview of a CASE-based tool that implements signal processing algorithms using a GUI-oriented tool. Users essentially flow-chart an algorithm as a graph of recipes. The system automatically schedules operations to implement the desired algorithm.

the Transform class (the corresponding utility is isip_transform).

Our Transform class combines the user's algorithm specifications by building a graph depicting the sequence of algorithms to be applied to the signal, as shown above in Figure 4. Such block diagram type approaches to signal processing have been popular for a number of years in signal processing. Transform is a very powerful class in that it allows users to mask, combine, and postprocess measurements of the signal in arbitrary ways. Since the internal structure of this class is somewhat complicated, a graphical user interface is essential. Rather than have users edit a parameter file containing information about the algorithms and their interconnections, users can manipulate this representation using graphical tool. We decided to implement this in Java to make it as portable as possible. Our first release of this tool will coincide with the summer workshop.

We have also made significant progress towards the development of the production recognition system by implementing statistical modeling aspects of the system. The decoder portion of a speech recognition system can be regarded as a hierarchy of graphs [8]. The leaf nodes of the lowest level of this hierarchy are states in a hidden Markov model. Our StatisticalModel class implements a generalized state, which can be an arbitrary mixture of distributions — Gaussians, Exponentials, Laplacians, etc. Mixtures of Gaussian distributions are most popular in speech recognition today; exponential models are becoming increasingly popular for some aspects of the problem (they are rooted in maximum entropy theory).

We have begun tying these together to build a full-fledged recognition system. We have also created conversion utilities that transform outputs of the prototype system into formats accepted by the new system. This is an important capability since it allows us to interface the two systems and maintain some level of backward compatibility. It also makes the development cycle more efficient, since we can incrementally test isolated components of the new system on large enough tasks to expose subtle bugs.

## E. Java Applets

As we described in our last report, the first year of the java programming phase of our project focused on stabilizing our approach to Java. Since the language was undergoing dramatic

changes, it was hard to modularize our applet development. The net result was that progress was slow. Fortunately, in the second year of this effort, Java has stabilized considerably, and we have been able to push ahead on using Java in our development environment. We have also had time to expand the capabilities of our existing applets.

A good example of this is our pattern recognition applet. This year, we were able to greatly enhance its educational value by augmenting basic capabilities with more visualization. An example of this is shown in Figure 5. Users can specify the parameters of a Gaussian distribution, and can view the support regions for these distributions in the original data space (along with the resulting decision regions). In Figure 6, we show two new data sets that were added by one of our sophomore undergraduate programmers. These data sets pose interesting challenges for classification algorithms since they are not linearly separable (they can't be separated by decision regions formed by hyperplanes). We have plans to include algorithms that can handle such data — for example, an algorithm based on support vector machines which is currently under development in our research group. The pattern recognition applet was used extensively in our Fundamentals of Speech Recognition course this semester.

Another novel feature added to this applet was the ability to classify the data using two popular clustering algorithms — KMEANS and Linde-Buzo-Gray (LBG). These algorithms iteratively reestimate cluster centers, and form decision regions based on nearest neighbor calculation with respect to these cluster centers. A Java applet is an ideal forum in which to learn about such algorithms because you can see the decision regions evolve with each iteration. This is demonstrated in Figure 7. Users select the number of clusters they want to use, and the number of iterations to be performed, and can then step through each iteration of the algorithm. Classification results are displayed in the description box to the right. In Spring '01, we plan to use this applet extensively in a pattern recognition course.

We have also begun implementation of several new applets. One which we are very excited about is an applet that helps students visualize search algorithms. This was motivated by a visit to MS State's 3D immersive visualization environment known as the COVE during the January design review, and viewing a demo where one walks along the edge of a mountain. We are attempting to create such a visualization of the search space during recognition using standard Java components (as opposed to some of the experimental virtual reality engines that are not quite standard yet). Further, we are developing a basic digital signal processing applet demonstrating the sampling process for signals. This is intended to be used in our undergraduate Signals and Systems course. Most of this work is being carried out by our undergraduate programmers, and represents a nice, non-critical path in which they can contribute to our research program.

Finally, we have begun some collaborations with George Institute of Technology and MS State's College of Engineering to use our job submission applet to do audio indexing of classroom lectures. This application was presented at a recent Internet 2 conference [3], and is an application enabled by the vast bandwidth potential of Internet 2. It will become more feasible when we expand our compute serving resources in the third year of this project. In this application, audio from a lecture is shipped to our system for automatic transcription via recognition, and time-alignment. The results are then transferred back to a database which can be searched by students ("Show me all the lectures about Fourier Transform."). While we are in the early stages of the development of this capability, it is a good example of the burgeoning interest in audio indexing and audio mining.

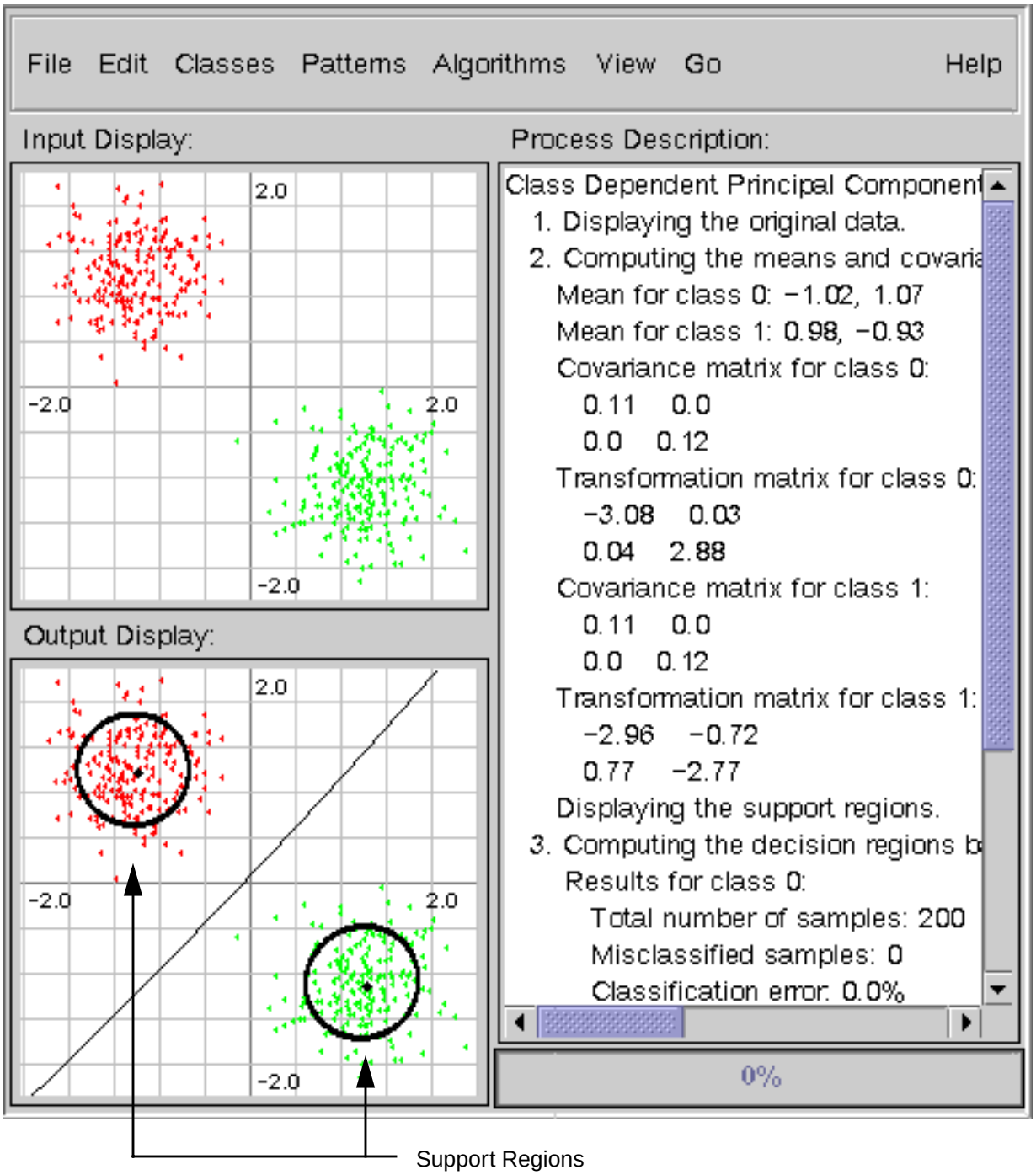


Figure 5. An example of enhanced visualization capabilities in our pattern recognition applet. We now allow users to specify the parameters of Gaussian distributed data through dialog boxes. In the case above, two Gaussian distributions were generated with means of [-1,1] and [1,-1] respectively. Principal Components Analysis (PCA) was chosen as the classification algorithm. Step-by-step output from the PCA approach is shown in the scrolling box on the right. Once the means and covariances are computed in Step 2, the support regions for the distributions are displayed in the lower left. These are crucial to understanding how algorithms such as PCA transform data. Other enhancements to this applet include the addition of new clustering algorithms, and an interactive status bar (lower right) that provides feedback for users when time-consuming operations are being performed.

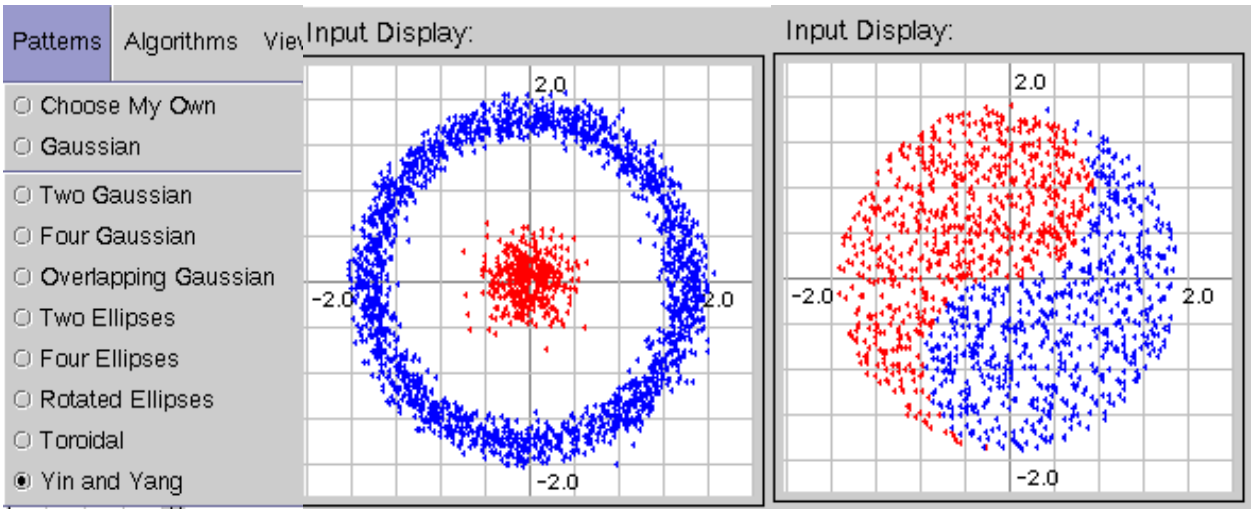


Figure 6. Two new data sets that pose challenging problems for classification algorithms have been added by one of our undergraduate programmers.

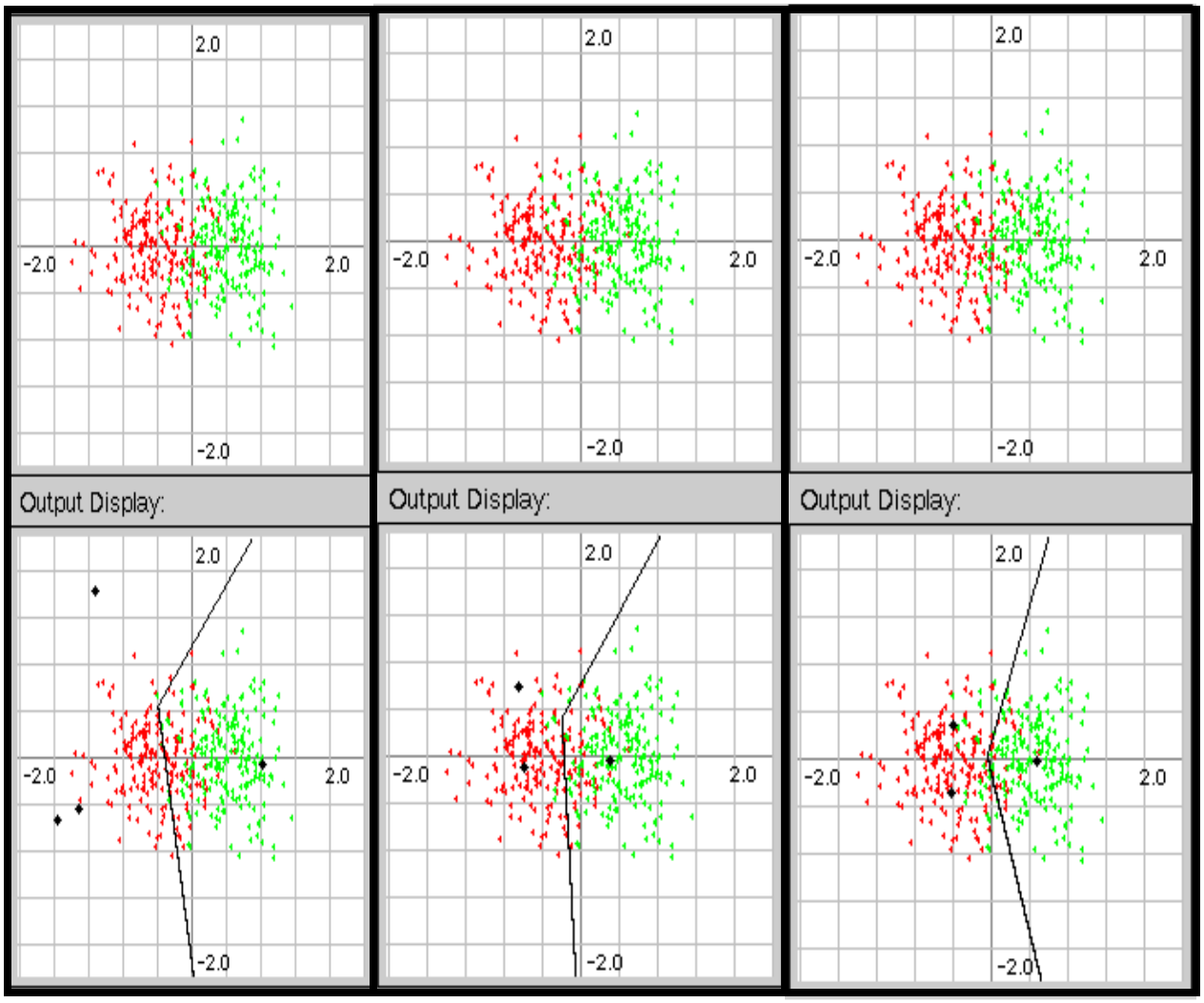


Figure 7. An example of a clustering algorithm in which users see the decision regions evolve.

## F. REFERENCES

[1] R. Sundaram, A. Ganapathiraju, J. Hamaker and J. Picone, "ISIP Public Domain LVCSR System," to be presented at the Speech Transcription Workshop, The University of Maryland University College, College Park, Maryland, USA, May 2000.

[2] J. Picone and W.C. Chapman, "Speech Recognition System Design Review," *http://www.isip.msstate.edu/conferences/srsdr00*, Mississippi State University, Mississippi State, Mississippi, USA, January 2000.

[3] J. Picone, C. Atkeson and I. Alphonso, "Harnessing High Bandwidth: Applications in Speech Recognition," presented at the Spring 2000, Internet2 Member Meeting, Washington, DC, USA, March 2000.

[4] J. Picone, "Summary of SRSDR'00," *http://www.isip.msstate.edu/conferences/srsdr00/technical_program/session_08/index.html*, Mississippi State University, Mississippi State, Mississippi, USA, May 2000.

[5] J. Picone, "Speech Recognition System Training Workshop," *http://www.isip.msstate.edu/conferences/srstw00*, Mississippi State University, Mississippi State, Mississippi, USA, January 2000.

[6] J. Picone, "Workshop Program," *http://www.isip.msstate.edu/conferences/srstw00/html/program.html*, Mississippi State University, Mississippi State, Mississippi, USA, May 2000.

[7] J. Vincent, "Request Tracker," *http://www.fsck.com/projects/rt/*, March 2000.

[8] N. Deshmukh, A. Ganapathiraju and J. Picone, "Hierarchical Search for Large Vocabulary Conversational Speech Recognition," *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 84-107, September 1999.

## 08/15/98 — 08/14/99: 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 loose site 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 system 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 front-end. Our approach was to initially replicate an industry-standard front-end consisting of mel-spaced cepstral features [8] 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 decoding portion [8] 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 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 [8]. 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 [11]. 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 from the system from another source). We first implemented an algorithm known as Viterbi decoding, 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 [12], which is more popular in state-of-the-art systems.

The current system is described in great detail in an upcoming publication [8]. 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 [13] 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 [14]. 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-senstitive 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 is the strategy used by most systems that allow you to accelerate your 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 [15] 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 are dealing 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 [16] 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 suite 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 [17], 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 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 preexisting code developed for a much less ambitious project [18] 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 18. 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 we 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 and the integral types, and in many ways mirrors 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, you 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 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 [18]. 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) [18]. 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 an Sof file. 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 don’t want to deal with byte-formatted, such as an MPEG audio or video stream, and don’t allow partial I/O of these types of data (retrieve only the middle three seconds of this audio file). There are some public domain file formats being supported within the community [19], 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 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 ms | 11.4 ms |
| 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) | 1640K | 1648K |
| 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, user 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 no clear standard for how to implement templates in the ANSI C++ specification. We continually monitor standards activities to see how we can improve our portability.

## 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) [20] to a state-of-the-art package called Concurrent Versions System (CVS) [20].

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 you 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 [22]. 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 [23] tutorial 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 our 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 other such programmer 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 are 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: “make diagnose” 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 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/mailing_lists*. Any message to this list is archived and added to the web page by a process that runs nightly using 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 who is 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 [24] 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 appopriate 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 [25]. Unfortunately, this interface was recently obsoleted, and replaced with Swing [26]. 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 installation instructions on our web site [27] 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 this plug-in is missing, and prompt the user with a message indicating what to do to download the plug-in.

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 prestored data sets that highlight the differences between common classification schemes such as principle 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 signal to feature vector conversion process in speech recognition. 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 experiments 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 you want 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 models. 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 in the much of the groundwork to support the subsequent years of the project has 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 MS 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 [28] 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 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

[9] 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.

[10] N. Deshmukh, A. Ganapathiraju and J. Picone, "Hierarchical Search for Large Vocabulary Conversational Speech Recognition," to appear in *IEEE Signal Processing Magazine*, September 1999.

[11] J. Picone, "Continuous Speech Recognition Using Hidden Markov Models," *IEEE ASSP Magazine*, vol. 7, no. 3, pp. 26-41, July 1990.

[12] 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*, Mississippi State University, Mississippi State, Mississippi, USA, May 1999.

[13] N. Deshmukh, A. Ganapathiraju, J. Hamaker, J. Picone and M. Ordowski, "A Public Domain Speech-to-Text System," to be presented at the 6th European Conference on Speech Communication and Technology, Budapest, Hungary, September 1999.

[14] H. Hermansky, "Perceptual Linear Predictive (PLP) Analysis of Speech." *Journal of the Acoustical Society of America*, vol. 4, pp. 1738-1752, 1990.

[15] 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.

[16] 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.

[17] P. Bothner, *et al*, "Welcome to the GCC Project!," *http://egcs.cygnus.com*, June 1999.

[18] 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.

[19] 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, National Institute of Standards and Technology, Gaithersburg, Maryland, USA, June 1999.

[20] W.F. Tichy, "RCS--A System for Version Control," *Software--Practice & Experience*, vol. 15, no. 7, pp. 637-654, July 1985.

[21] "Concurrent Versions System (CVS)", *http://www.cyclic.com/cyclic-pages/howget.html*, Cyclic Software, Washington, D.C., USA, June 1999.

[22] 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.

[23] I. Alphonso, “CVS Anonymous Download Instructions,” *http://www.isip.msstate.edu/projects/speech/support/info/cvs_instructions.html*, Mississippi State University, Mississippi State, Mississippi, USA, June 1999.

[24] S. Balakrishnama and N. Deshmukh, “ISIP Software Documentation,” *http://www.isip.msstate.edu/projects/speech/education/tutorials/isip_env,* Mississippi State University, Mississippi State, Mississippi, USA, June 1999.

[25] ICASSP applets paper

[26] E. Eckstein, M. Loy, and D. Wood, *Java Swing*, O’Reilly and Associates, Cambridge, Massachusetts, USA, 1998.

[27] R. Duncan, “Java Plug-In Installation Instructions,” *http://www.isip.msstate.edu/projects/speech/support/info/java_instructions.html,* Mississippi State University, Mississippi State, Mississippi, USA, June 1999.

[28] D. MacKenzie and B. Elliston, *http://www.gnu.org/manual/autoconf-2.13/html_chapter/autoconf_toc.html*, Free Software Foundation, Boston, Massachusetts, USA, July 1999.

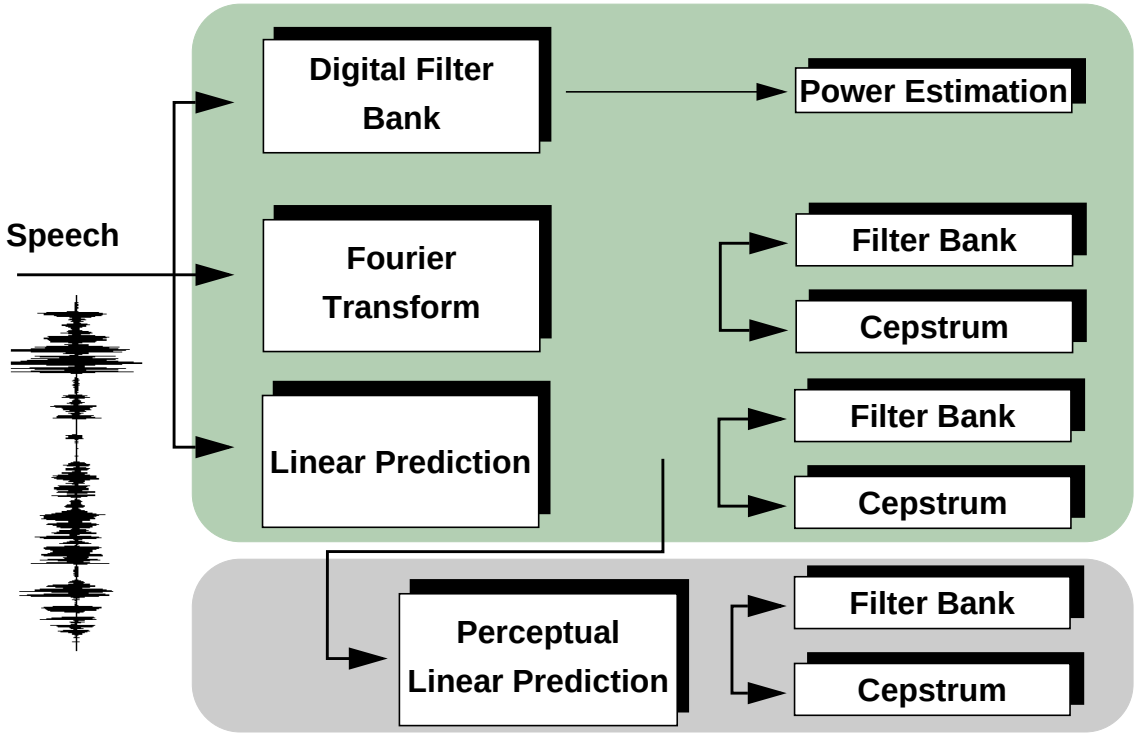


Figure 1. An overview of the front-end portion of the speech recognition system. Two popular analysis techniques, mel-spaced cepstrum and perceptual linear prediction, are supported in the system. Other approaches based on frame-based analysis techniques can be easily added.

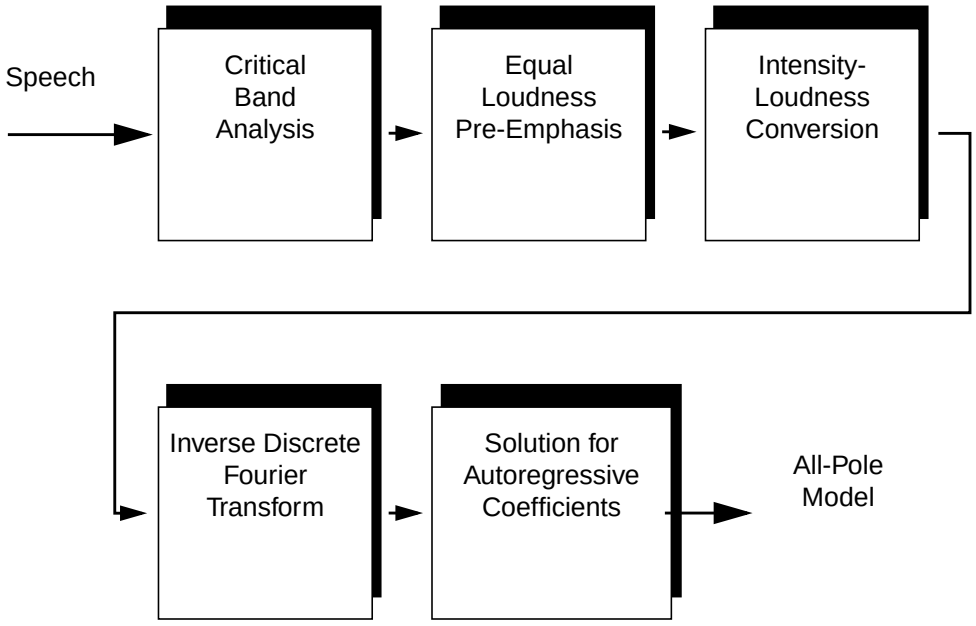


Figure 2. An overview of perceptual linear prediction (PLP) analysis. This front-end has become increasingly popular in recent years.


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:

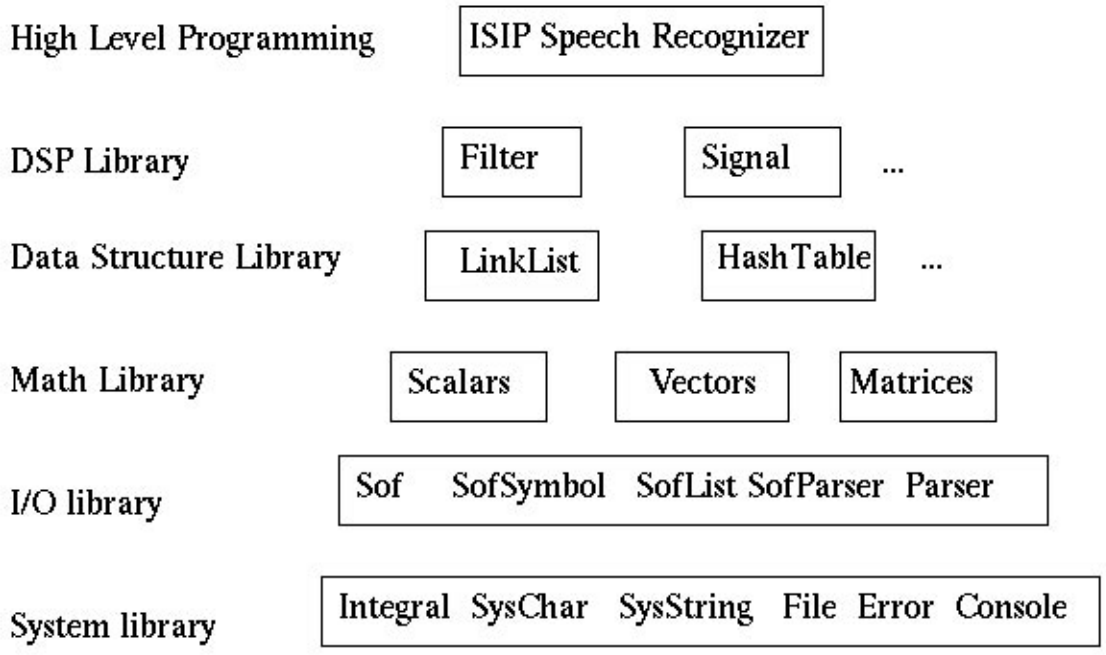


Figure 3. An overview of the hierarchy of ISIP classes. The system and I/O libraries are new additions to the class structure. The math and data structure libraries make extensive use of templates.

```
// 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.

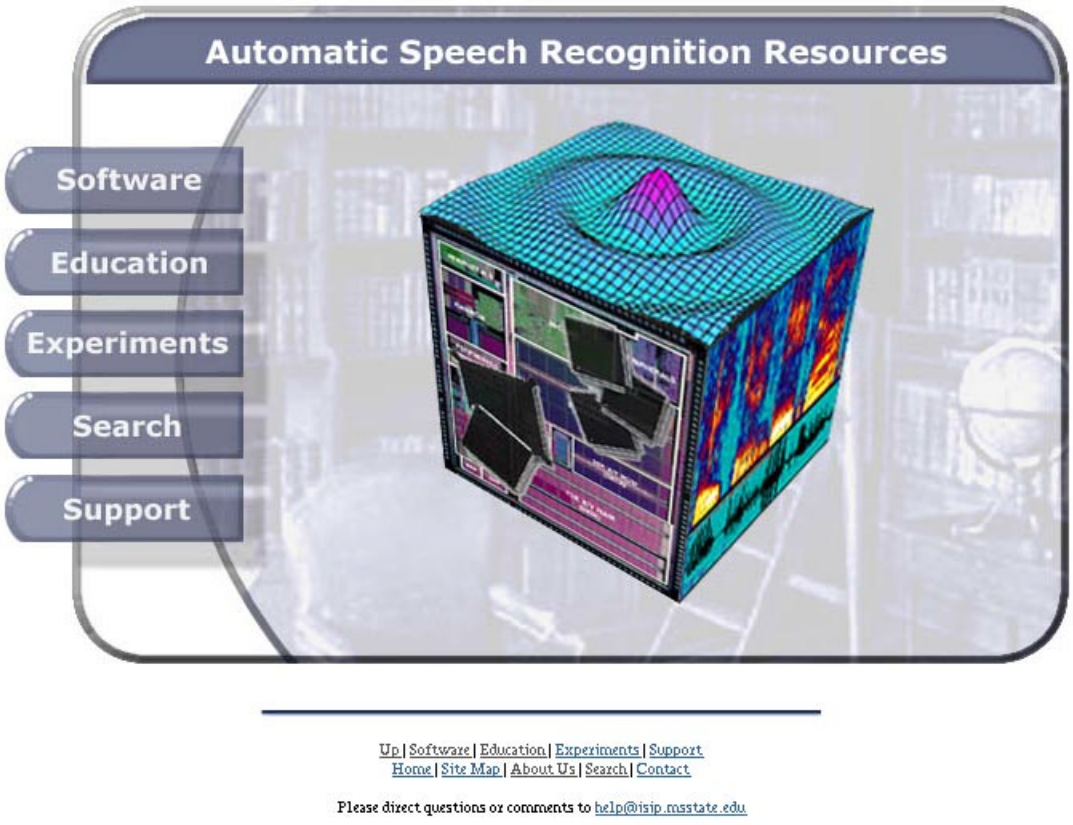


Figure 6. A new set of web pages have been created to support the project. These have been designed to provide easy access to the web site. The choices to the left of the image mirror the physical organization of the web site.


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.

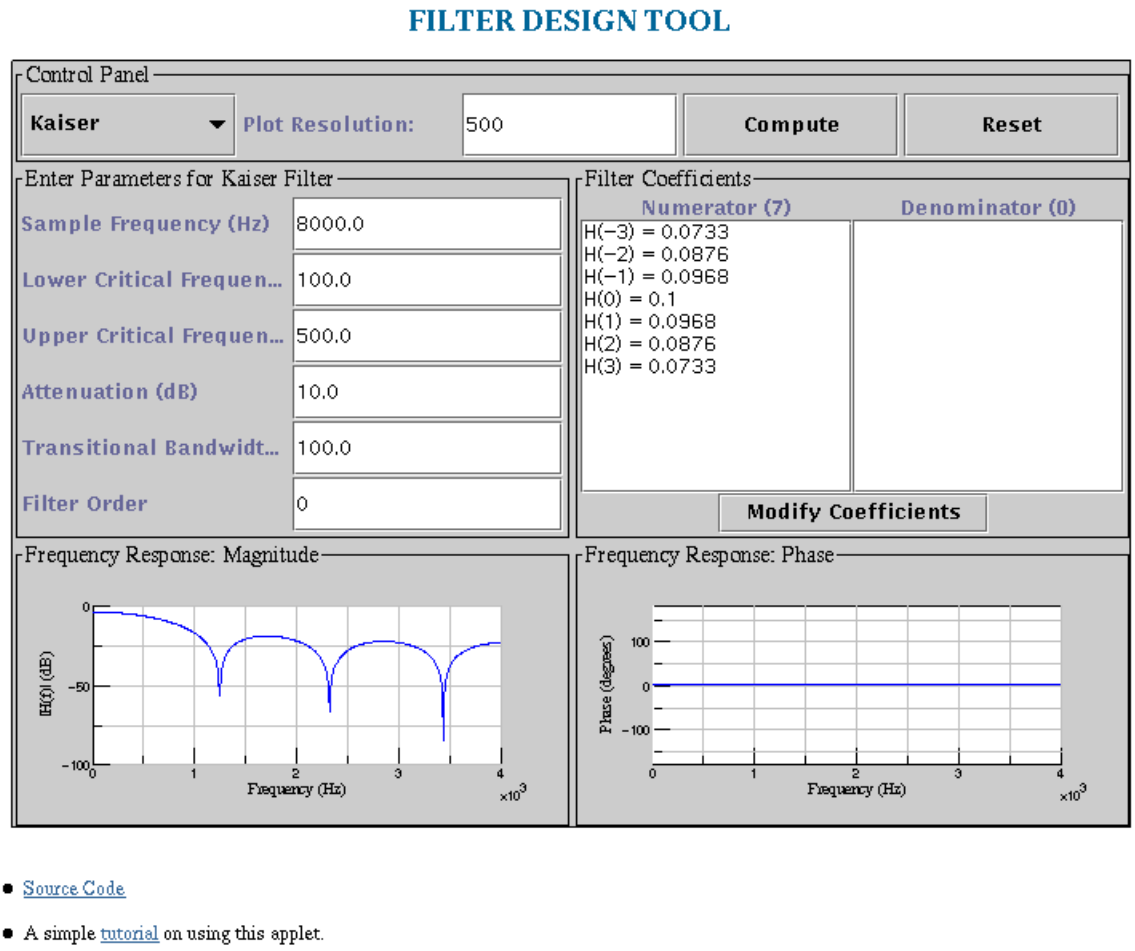


Figure 8. An example of a Java Swing applet that demonstrates the concept of digital filter design. Swing has been a mixed blessing. While some aspects of GUI programming are nicely abstracted, other aspects, such as interactions between grid boxes and event handlers, have been problematic.

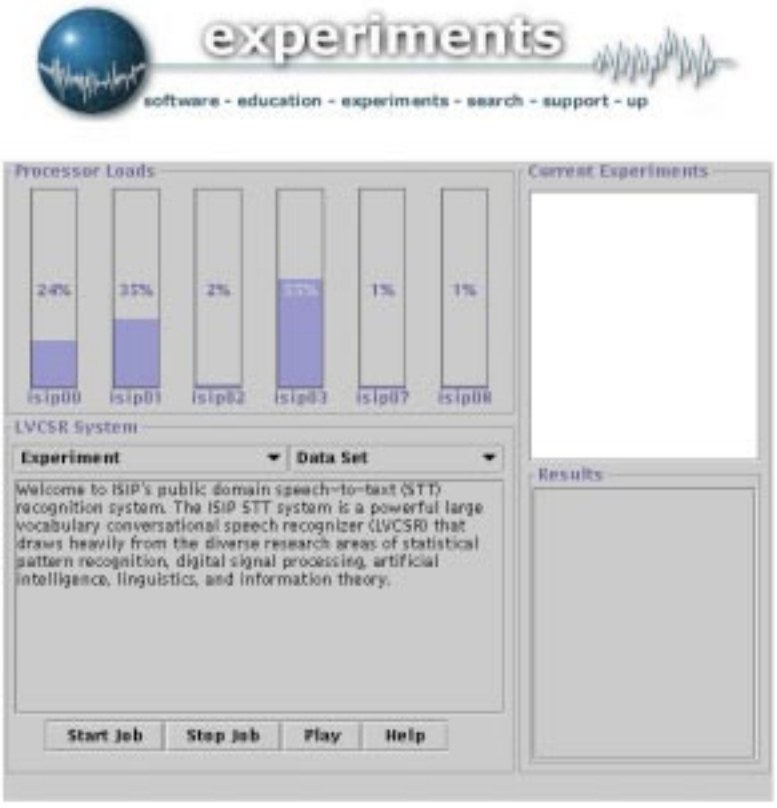


Figure 9. A Java applet that allows users to submit speech recognition jobs remotely to a bank of compute servers. Users can run canned experiments, or supply their own audio data. Parameters for the experiment can be specified via dialog boxes. Results are emailed to the user, and can be examined directly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

## 08/15/99 — 08/14/00: MAJOR FINDINGS

In the second year of this project, we introduced two pivotal activities in the project: annual workshops and a rigorous software distribution process. The workshop activities are proceeding smoothly. In January 2000, we hosted nine visitors from several foreign countries (China, Finland), government agencies (FBI, DoD), and industrial sites (IBM, MITRE, Lincoln Labs). We reviewed the goals of the research program, the architecture of the system, provided several demonstrations, and collected feedback on features and capabilities needed in future versions of the system. Several collaborations resulted from this meeting, including an audio indexing opportunity with George Tech, and invited talks at IBM. More collaborations are planned. At the time this report was written, we are completing plans for the May 2000 workshop, which will include 25 participants, of which 20 are graduate students. Demand for the workshop was strong — we tripled the number of participants over what was originally budgeted. We turned away approximately 10 potential participants due to space and resource limitations.

One surprise in the second year of the project has been the growing importance of supporting the Linux operating system, and the difficulties in doing so. Through experience, we have learned that despite using the same compiler and software (GNU gcc, make, etc.) but a different flavor of Unix (Sun Solaris x86), we cannot guarantee robust releases for Linux users. Hence, we spent some time enhancing our ability to achieve platform independence across a wide range of Unix platforms. We now routinely run our releases through a suite of systems before actually making the release. We also have encountered a large demand for Windows-based ports of our system. Currently, we do this through the use of a Unix-like shell available under Windows (Cygnus' cygwin tools). This has also increased the support overhead in making releases of our system. Despite the demand for a native Windows port, we are not assigning this a high priority until the core system is stable. This is primarily because there is a lack of standardization of C++ compilers and development environments, therefore making it hard to support both environments simultaneously when the software base is changing rapidly.

With the addition of a full-time staff person, it has been possible to expand our on-line support activities. We now handle approximately 5 serious support requests per day. Many support requests involve hand-holding inexperienced users on basic computing issues — we are still struggling with how to deal with these in a timely manner. The remainder involve unexpected program crashes that require extensive diagnosis. We have implemented an automated problem tracking system to make sure such requests are properly tracked. We also provide an ability for users to upload their data to us, so that we can replicate their problems. The majority of serious problems seem to relate to compiler-dependent problems (for example, a bug which did not show up on one version of an operating system, but is fatal on a different architecture with a different compiler). This also exposed the critical need for an in-house multi-platform evaluation facility.

Last but not least, we have made extensive progress enhancing basic functionality of the system. We have developed a generalized hierarchical search engine that is the first such system of its type. We provide an ability to decode speech using either networks for N-gram language models. Users can supply either type of model, or both, at any level of the system. For example, it is possible to constrain the search using N-grams of parts of speech, as well as N-grams of words. We have also developed a front-end that allows users to configure the signal processing portions of the system without writing any code — algorithms can be specified using a GUI-oriented tool that automatically schedules the necessary operations required.

## 08/15/98 — 08/14/99: 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.