Object Oriented Reuse: Experience in Developing a Framework for Speech Recognition Applications

Savitha Srinivasan
IBM Corporation/Research Division
650 Harry Road
San Jose, CA 95120-6099, USA
Tel: 1-408-927-1430
E-mail: savitha@almaden.ibm.com

John Vergo
IBM Corporation/Research Division
30 Saw Mill River Road
Hawthorne, New York 10532, USA
Tel: 1-914-784-7035
E-mail: jvergo@watson.ibm.com

ABSTRACT
The development of highly interactive software systems with complex user interfaces has become increasingly common, where prototypes are often used as a vehicle for demonstrating visions of innovative systems. Given this trend, it is important for new technology to be based on flexible architectures that do not mandate the understanding of all complexities inherent in a system. In this context, we share our experience in developing an object oriented framework for a specific new technology, i.e. speech recognition. We describe the benefits of the object oriented paradigm rich with design patterns, which provide a natural way to model complex concepts and capture system relationships effectively, along with achieving a high level of software reuse.

Keywords
Object oriented design, application frameworks, speech applications

INTRODUCTION
Speech recognition based computing is already a reality[6,10,19]. The domain in which speech recognition may be applicable can vary from highly interactive user interfaces for creating text from speech, to telephony applications which may have no graphical user interface. Systems which support the synergistic use of speech and direct manipulation using keyboard or mouse hold the appeal of improved usability by providing a more natural interface to the computer. The medical, legal and journalism industries have been using speech recognition[19] to create their documents in order to eliminate transcription time and improve productivity.

Although speech recognition is a reality today, it is being used in specific domains primarily for the purpose of document creation. Other applications such as data entry systems or multiple choice form fill-in are also suitable candidates for speech input. Advances in technology are making significant progress towards the goal of any individual being able to speak naturally to a computer on any topic, and be understood accurately. However, we are not there yet. Even continuous speech recognition with high accuracy requires the user to speak clearly, enunciate each syllable properly, and to have one's thoughts in order before starting. Factors inhibiting the pervasive use of speech technology today include, the lack of availability of general purpose high accuracy continuous speech recognition, lack of systems that support the synergistic use of speech input with other forms of input and challenges associated with designing speech user interfaces for increased user productivity despite speech recognition inaccuracies.

After an initial assessment of a problem domain appears to be suitable for the application of speech recognition, a true understanding of the advantages and limitations of speech can be obtained only by prototyping the system. The recent trend of prototyping as a means to evaluate the quality of a solution at an early stage[1] as opposed to the traditional life-cycle approach, leads to a requirement for an environment that enables the rapid integration of speech recognition technology. The emergence of the object oriented paradigm provides an important enabling technology for reuse of large components such as speech recognition software. This has resulted in the evolution of object oriented frameworks[4,5,9,14,18] defined to be a set of collaborating object classes that embody an abstract design in order to provide solutions for a family of related problems. A framework is considered to be a semi-complete application which contains certain fixed aspects common to all applications in the problem domain, along with certain variable aspects, unique to each application in the problem domain. What differentiates one framework application from another is the manner in which the variable aspects are defined.
The object oriented paradigm has been touted as a means of improving software quality and programmer productivity. In particular, frameworks are believed to facilitate sound software designs that lend themselves to reuse due to the systemic use of object oriented features such as data abstraction, inheritance and dynamic binding. Different levels of reuse such as reuse-in-the-small, reuse-in-the-medium and reuse-in-the-large have been identified[11]. Most object oriented systems exhibit reuse-in-the-small by the usage of inheritance and object composition. Reuse-in-the-medium is exemplified by framework applications which contain object class interactions, or micro-architectures[5]. Reuse-in-the-large refers to the highest form of reuse where application objects which are independent systems are reused, where the system that reuses the framework may not have been developed based on frameworks, or even in the same language.

With the design of the speech framework, we strive to achieve all three forms of reuse for speech applications. We describe the evolution of a simple, yet powerful speech interface through an object oriented design where the distinction between the classic software engineering phases of analysis, design and implementation are less apparent because of the common object abstractions used in all three phases. The design process converts the description from the analysis phase into an object oriented framework in the implementation phase.

In this paper, we first define terms and describe basic speech recognition technology concepts. We then articulate the problem and explain how the object oriented speech framework addresses many of the issues. While we describe the technical underpinnings of such a system, we do not focus on design criteria and tradeoffs related to speech user interface design. The paper is intended to present a bridge between object oriented design groups and user interface groups by demonstrating a complex, but real solution to a real-world interface development problem. We conclude with a description of lessons learned as the framework evolved with each new speech application.

The framework was originally developed in conjunction with the MedSpeak/Radiology[10] application and is based on IBM's Continuous Speech Recognition[7] technology.

**SPEECH RECOGNITION CONCEPTS**

Speech recognition may be used in one of two ways in an application. The first is aimed at text entry or document creation applications such as dictation for electronic mail or word processing, and is referred to as dictation. The second is targeted for transaction processing and data entry systems and is referred to as command and control or navigation. Central concepts to speech applications are vocabularies and vocabulary sizes. A vocabulary defines the set of words or phrases that can be translated by a speech engine and is one of the resources that the engine uses to process spoken input. Speech grammars are an extension of the single words or simple phrases supported by vocabularies. They are a structured collection of words and phrases bound together by rules that define the set of speech streams that can be recognized by the speech engine at a given point in time. The speech engine matches the acoustic models to the speech input to words in the vocabularies; therefore, only words in the vocabulary are capable of being recognized.

Words in a vocabulary are recognized based entirely on their pronunciation or how they sound. Pronunciations are the possible phonetic representations of a word. Words can have multiple pronunciations; for example, "the" will have at least two pronunciations, "thee" and "thuh." Punctuation symbols are often associated with several different verbal representations; for example, the symbol "." may have the pronunciations "period", "point", "dot", and so on[7].

Word-usage models are another resource used by the speech engine to provide statistical information on word sequences. They assist the speech engine in decoding speech by biasing the output towards high probability word sequences. Within a particular application domain such as radiology, perplexity is a measurement of the number of equally likely word choices given a sequence of words[3,12,17]. Together, vocabularies and word-usage models are used in the selection of the best match for a word or phrase by the speech engine.

**DESIGN CHALLENGES**

Our primary objective in the design of the object oriented framework was to hide complexities associated with speech recognition technology and to provide a flexible, application level speech interface for diverse development environments. In particular, we wanted to support a spectrum of applications ranging from those that wished to use a very simple speech interface with minimal engine interaction, to others that wished to be highly interactive with the user in order to exploit advanced speech recognition functions. We wanted to attain the highest form of reuse, which translated into the framework not mandating the use of any particular development environment or class library.

A secondary objective was to develop an infrastructure flexible enough to accommodate the integration of synergistic input technologies such as pen or gaze input for improved command and control recognition. The framework in its present form, supports IBM's speech recognition programming interface[7]. We attempt to abstract the key concepts associated with speech recognition technology such that replacing IBM's technology with a different speech recognition technology does not change the framework's interface.
Having stated our mission, the analysis phase of the problem domain translated into expressing the requirements in terms of objects. In doing this, we had to capture complexities at different levels; those at the recognition technology level, the application level and development environment level. While we faced many of the described issues in the specific applications we developed, we believe that other speech applications will pose similar problems in one form or other.

Recognition Technology Issues
Speech engines communicate with client applications in an inherently asynchronous mode. Speech engines that do not require asynchronous message processing may require some event handling in the application to dispatch messages. The handshaking associated with multiple asynchronous engine calls must be handled by the application.

In order to reduce perplexity (and hence, increase accuracy), it is desirable to limit the size of the enabled vocabulary for speech systems. The application must assume all responsibility for defining and managing these vocabularies.

Recognition errors may occur for several reasons. Variations in pronunciation, dictating out of vocabulary words and unusual word usage models are all causes for errors. Speech applications usually provide error correction functions to meet two objectives; the first is to create an accurate document, and the second is to use the speech engine correction functionality to increase the accuracy of subsequent dictations[3,10]. While the first objective may be met by direct manipulation of the displayed text, the second objective requires the updating of a pronunciation, the vocabulary and the word usage model. Applications usually support both modes of error correction. This leads to the problem of maintaining synchronization between the data maintained by the speech engine and the actual text displayed to the user.

Application Level Issues
Speech engines provide recognition technology level functionality; applications must build their own infrastructure to direct and dispatch messages from the engine. Concepts of window level commands versus widget level commands must be architected by the application. Providing a mechanism to decouple the interaction of the speech engine events from application specific events is the responsibility of the application.

Since speech programming interfaces tend to reflect engine level functionality, it may be necessary to perform a sequence of engine calls to achieve application level functionality. This sequence of calls must adhere to the handshaking protocol dictated by the engine. For example, change of state from command and control mode to dictation mode requires several engine calls to be made.

Development Environment Issues
Existing recognition technologies support programming interfaces for specific development environments and require the application to be developed in one of the supported environments. This means that speech applications must comply with the language and environment requirements of the speech engine. An application that must run in an environment different from one of the supported environments may not incorporate speech technology. For example, many engines support C language programming interfaces only. An application that must run in an interpretive environment such as Tool Command Language(Tcl) or Java may not have access to speech technology. For those interpretive environments that do support a speech programming interface, applications must still deal with engine level and application level issues.

RELATED WORK
The role of object oriented frameworks as a means of reusing analysis, design and code has been discussed at length in the literature. They are deemed to be particularly important for the development of open systems, where not only functionality, but also architecture must be reused across a family of related applications[4,16]. Prior research identifies phases in framework development, documentation of frameworks and also provides design guidelines for tailororable frameworks. It also reports on framework composition, framework development methods, usage and maintainence related issues. More recently, the importance of design patterns in frameworks has been stressed. The role of patterns in alleviating software complexity in the domain analysis, design and maintainence phases has been discussed[2,16].

Specific software architectures have been described for multiple interaction techniques such as the synergistic use of speech and gesture[13] where a generic fusion engine that can be embedded in a multi-agent architecture modeling technique and the fusion algorithm is described. Another specific application incorporates speech recognition into a video indexing and searching application during the library creation and library exploration phases[6].

At the recognition technology level, there are several proprietary speech recognition technology providers that provide programming interfaces for speech[19]. Many of these are accompanied by sample programs that illustrate the usage of C language application programming interfaces(APIs). The JavaSpeech API is designed to enable developers to incorporate speech recognition and speech synthesis into Java applications[8].
To summarize, significant research has been done related to the design, evolution and documentation of object oriented frameworks as an effective object oriented technique. Notable research projects that incorporate speech recognition technology in specific environments have been described. Various speech recognition technology providers support development tools which demonstrate the integration of speech technology in applications. To the best of our knowledge, we have not seen a pragmatic effort that learns from the object oriented framework research, applies it to the domain of speech recognition and evolves a reusable framework through the repeated development of real-world applications.

OBJECT ORIENTED FRAMEWORK DESIGN

The speech framework is a semi-complete application where the responsibilities of a system are partitioned among its fixed and variable components. The fixed component, or the reusable design for a speech application is expressed as a set of collaborating (C++) object classes that abstract speech interface concepts and encapsulate engine level complexity. Many of the objects used in the final design are those that we identified in the analysis phase, in order to articulate the requirements. We evolved the behavioral object collaborations using design patterns and distinguished between abstract framework classes and concrete application classes over the design process. Each new speech application is an instantiation of the framework where the variable component is proportional to application specific features only, and not to the total complexity of the problem domain.

Figure 1 represents a high level view of a speech framework application. The speech engine and application are independent processes where the application communicates with the speech engine to define vocabularies and the speech engine in turn, communicates asynchronous events such as recognized text, commands and phrases to the application.

To maximize flexibility and reuse, we have adopted a layered approach in the framework. The first layer, the core framework, encapsulates the speech engine functionality in abstract classes, independent of any specific graphical user interface (GUI) class library. A GUI speech recognition application usually involves the use of a GUI class library; therefore, the second layer extends the core framework for different GUI environments in order to provide seamless speech recognition functionality. The GUI framework extensions provide tight integration by collaborating with the classes in the GUI library. In this paper, we focus on the speech abstractions created in the core framework rather than the clear separation of functionality between the core framework and its GUI extensions.

Each major speech component modeled by an abstract class in the core framework is realized in the application by the creation of a concrete derived instance of the abstract class. Due to the object oriented features of inheritance and dynamic binding, the concrete instances encapsulate large parts of functionality and control flow in the application, provided by the abstract classes. Application specific logic must be performed by the concrete realizations of the abstract classes. While the core framework internally consists of more than fifty classes, the speech engine functionality is provided by four core interface classes, each of which models a major speech component to provide the interactive interface functionality. The partitioning of the framework into distinct classes that support kernel functionality and interactive interface functionality, allows for a flexible system architecture where different aspects may be allowed to vary independently.

Figure 2 shows the core framework classes and the flow of recognition events between the speech engine and the concrete derived classes instantiated by the speech application. The SpeechClient class is responsible for maintaining engine connections, enabling vocabularies and receiving window level recognition events; the SpeechText class for edit control level recognition events; the VocabularyManager class for vocabulary definition functions and the SpeechObserver class for observing speech engine status changes. When the application starts up, an instance of the SpeechClient encapsulates speech engine initialization, vocabulary and grammar definition. The user may then begin speaking words from any of the enabled vocabularies into the microphone. The speech engine processes the audio stream and communicates recognized commands and phrases to the SpeechClient class and recognized text to the SpeechText class. Speech engine status is maintained within the framework and broadcast to registered SpeechObservers.

Thus, the abstract classes in the core framework provide the flexible system composition where their collaborations provide the infrastructure to abstract, direct, dispatch and receive speech engine events at a level that is meaningful to the application. The concrete instances of these abstract classes created in the application by instantiating a specific
framework extension must process recognition events to implement application specific behavior.

**DESIGN PATTERNS USED**

We describe some aspects of the framework design by identifying concrete realizations of design patterns that describe successful solutions to known software engineering problems and facilitate the direct reuse of design and code. We have used patterns to guide the creation of abstractions in the design phases, necessary to accommodate future changes and yet maintain architectural integrity. These abstractions help decouple the major components of the system so that each may vary independently, thereby making the framework more resilient.

In developing this framework, we have studied and applied a combination of well known object oriented patterns along with variations of patterns that focus on efficient, reliable, concurrent and distributed programming[15]. The object oriented patterns reflect, among others, principles of good software engineering and object oriented design, such as achieving reuse by favoring object composition or delegation over class inheritance, decoupling the user interface from the computational component of an application and programming to an interface as opposed to an implementation. In this section, we list some of the strategic object oriented patterns employed, the issues they addressed and their impact on the evolution of the framework.

**Facade/Singleton**

We defined a Facade class object, used internally by the framework to provide a unified, abstract interface to a set of interfaces supported by the speech subsystem. The Facade class is not exposed in the framework’s public interface. However, the core abstract classes, such as, SpeechText, SpeechClient, VocabularyManager and SpeechObserver communicate with the speech subsystem by sending the requests to the Facade object, which forwards the requests to the appropriate subsystem objects. The Facade class internally consists of various classes that assume responsibility for distinct functional areas of the speech subsystem. The Facade also implements the Singleton pattern where it guarantees a single instance of the speech subsystem for each client process.

The use of the Facade and Singleton were straightforward applications of the patterns, given the context of the problem they addressed in the framework. However, they address some of the recognition technology issues we faced by abstracting speech concepts so as to hide differences in philosophy adopted by different speech recognition technology providers. For example, the speech engine that we developed the framework for, requires the engine to be in a halted state when a vocabulary is enabled. The mechanics of halting the speech engine, enabling the vocabulary and then restarting the speech engine were encapsulated in the Facade object.

**Adapter**

We used the class Adapter pattern to create a reusable speech class that cooperates with other existing and unrelated graphical user interface classes that do not have a speech interface. One way of implementing a speech enabled GUI window class is to create a new class that inherits from the abstract SpeechClient class and a specific GUI window class. Similarly, a speech enabled edit control class may be implemented by creating a new class that inherits from the abstract SpeechText class and a GUI edit control class.

The use of this pattern has been critical to the design and evolution of the framework and has been responsible for the
level of reuse achieved. It enabled us to encapsulate the speech aware behavior in abstract framework classes such that existing GUI classes did not have to be modified in order to exhibit speech aware behavior. The SpeechClient abstraction supports GUI window level functionality such as change of state from dictation mode to command mode, change of vocabularies in a given state of the window etc. The SpeechText abstraction supports GUI edit control level functionality by maintaining synchronization between the speech engine data and the text displayed to the user. In addition, the Adapter pattern also addresses a key development environment issue of supporting speech functionality in a non C++ environment because the abstract speech components do not contain GUI components and yet provide speech aware behavior.

Observer
We used the Observer[5] pattern to implement the notification mechanism in order to observe speech engine changes and reflect them in the user interface as appropriate. Having partitioned the system into a collection of cooperating classes such as SpeechClients and SpeechTexts, these objects needed to maintain consistency with the state of the speech engine. The Observer pattern implements the publish-subscribe mechanism where each instance of the SpeechObserver class registers itself to be notified of engine status changes through a broadcast mechanism implemented within the framework. The introduction of this class addresses some of the application level issues by simplifying the accurate maintenance of speech engine state in the application.

ITERATIVE DESIGN: FRAMEWORK EVOLUTION THROUGH REUSE
We describe our experience in developing different speech applications using the framework, the impact of the experience on the framework design and the level of reuse achieved in each case. In all cases, we were able to reuse the core framework entirely since the domain knowledge and application abstractions were encapsulated in the core framework.

Radiology Dictation Application
This application allows radiologists to create, edit and manage reports using real-time continuous speech recognition[10]. The primary objective of the application being report creation using speech to text, it takes advantage of dictation related speech framework functionality such as audio playback and error correction. It uses the command and control mode in a limited manner to issue simple voice commands and navigate between screens. We developed this application by extending the core framework to work with the XVT(tm) GUI class library.

At the time we developed this application, we had defined the concept of SpeechClient to abstract window level speech behavior. The need to provide a SpeechText class was apparent when we realized the complexity associated with maintaining the list of dictated words in an edit control, playing back related audio and performing error correction in the application. This lead to the creation of an abstract SpeechText class to provide speech aware edit control functionality. Likewise, the problems associated with maintaining the speech engine state across different windows lead to the definition of the SpeechObserver class.

Pathology Dictation Application
This application is used by pathologists to create, edit and manage reports similar to the Radiology Dictation application, but tailored to the pathology work flow. To develop this application, we extended the core framework to work with the IBM VisualAge(tm) GUI class library.

Our Adapter approach to providing speech aware behavior in GUI class library independent classes was reinforced by the relative ease with which we were able to extend the core framework for different GUI environments.

VRAD: Visual Rapid Application Development
Yet another mission of the speech framework was to be included in an all-encompassing object oriented programming framework, where the development paradigm is to layout the application in a visual tool, set the control's design-time properties and write event handlers as short blocks of code to be executed in response to events from the user or other controls. We found the requirements of such a framework to be more stringent in terms of adhering to the overall framework guidelines, different from our previous experience where the speech framework was used to develop a single application at a time.

For example, certain object oriented programming principles such as hiding implementation classes, the use of surrogate objects and most importantly, reflecting the client's view only to the extent needed, were stressed. Though we had incorporated these features to varying extents in the framework, we modified the public interface in order to fully adhere to these guidelines. A specific example of a design change made to accommodate one consideration was to create a surrogate SpeechSession object which hides the hidden master, the Facade object, where it had previously been a public framework class.

Socket Interface Applications
The next application that we developed was in the domain of using speech recognition to annotate videos during an ingest phase of a digital video library system, similar to the Informedia[6] system. The requirements of such an application were a subset of the core framework functions
since this application relies on the command and control mode for annotation of videos. The environment in which this application was developed was an interpretive, non C++ environment. We decided to extend the VRAD extension of the core framework by providing a thin socket interface which supported limited speech commands. Simple string interfaces such as Start and Stop were supported by this framework, where Start turned the recognition process on and Stop turned the recognition process off.

We provided this functionality by developing a VRAD socket application based on the VRAD framework which supported a socket interface. We extended the SpeechClient to create a SpeechSocketClient which performed the read and write operations on a socket connection. The run time environment of the actual speech application was not relevant to the socket interface application since communication was through sockets.

This framework application provided us with the maximum reuse, where a specific framework application was used to provide speech recognition services. The modular design embodied in the framework classes allowed the creation of a new interface to the speech engine with relative ease. The speech engine functionality we chose to provide was limited to receiving recognized commands and there was no support for application level dispatching of recognition events; however, it met the requirements of the application adequately.

LESSONS LEARNED

Despite the fact that the framework applications developed are not too diverse, we believe that several lessons can be learned from our experience with the development of such a framework. The lack of diversity is more due to the limited applicability of speech recognition technology rather than limitations of the framework. Figure 3 summarizes different aspects of our experience for the various framework applications developed.

We observe that with each new application, the number of framework interface classes grew to encapsulate distinct functional areas, thereby exemplifying the principle of modularization to conquer complexity. Not all abstractions in the problem domain were captured during the initial analysis phase. Some framework applications lead to the creation of new speech interface abstractions necessary in the analysis and design phases. As a result, the framework realized more design patterns allowing for greater resilience. We were able to progressively abstract more domain knowledge into the core framework, thereby minimizing the required GUI class library specific extensions. The following lessons can be drawn from our experience with developing speech framework applications.

What we would do the same

Identify object abstractions in problem domain early. Using a common set of abstractions across the analysis, design and implementation phases lead to better communication and a more efficient technique for handling changes as the system evolved.

The framework designer is also the framework user. We believe that our being the designer and the user for the initial applications contributed to our understanding of the requirements of the framework, and our striking the right balance between flexibility, extensibility and tailorable. It helped us define, to what extent the framework should realize variable components, or hot-spots[14], so as to provide flexibility at the cost of having to understand more of the framework, thereby making it less tailorable.

Develop a framework in the context of at least three real applications. The design of the framework was simplified because the task of predicting future uses for the framework had well defined boundaries, given the real applications we were building. This enabled us to extend the design for realistic requirements of other applications in the problem domain.

Empty design patterns in the communication of micro-architectures or object collaborations. The use of cataloged design patterns to specify micro-architectures provided a common vocabulary for design and facilitated the direct reuse of design and code. It helped reduce system complexity by naming and defining abstractions, thereby reducing the framework's learning time.

Continue to evolve the framework through reuse. The speech framework benefited from each new application developed. Hence, an effective way for a framework to evolve is by its repeated use in developing diverse applications in the problem domain.

What we would do differently

Adopt formal design guidelines to framework development. In developing the framework, we had to predict the future uses of the software and to incorporate the requirements for these possible future applications into the current design. We had to strike the right balance between flexibility and tailorable, which are often at odds with each other. We achieved this, initially, without following any particular methodology. Adopting formal design guidelines which support interoperability, distribution and extensibility in framework development could possibly have eased the design process. For example, we reused many design patterns initially without understanding the internal protocol and functionality. Starting with a catalog of predesigned micro-architectures expressed by design patterns may lead to an improved design. This may be achieved by using it as a blueprint to introduce a new design and as a reference to
<table>
<thead>
<tr>
<th>OO Framework</th>
<th>Radiology Dictation</th>
<th>Pathology Dictation</th>
<th>VRAD Applications</th>
<th>Socket Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Extensions</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Employed Patterns</td>
<td>Adapter, Service Configurator</td>
<td>Facade/Singleton, Service Configurator</td>
<td>Facade/Singleton, Adapter, Service Configurator, Observer, Asynchronous Completion Token</td>
<td>Facade/Singleton, Adapter, Service Configurator, Observer, Asynchronous Completion Token</td>
</tr>
<tr>
<td>Speech Functions</td>
<td>Dictation Command</td>
<td>Dictation Command</td>
<td>Dictation Command Grammars</td>
<td>Command Grammars</td>
</tr>
<tr>
<td>Speech Functionality Distributed Over Core Framework and the GUI Extensions</td>
<td>75% + 25%</td>
<td>80% + 20%</td>
<td>90% + 10%</td>
<td>97% + 3%</td>
</tr>
</tbody>
</table>

Figure 3: Framework Applications Summary

ensure that class redesign does not affect behavioral collaborations.

Converge towards multiple fine-grained integrated frameworks and away from a single, large monolithic framework. Recent trends in framework based application development have been towards the use of composition of multiple frameworks to meet the requirements of applications. We have not necessarily violated this guideline yet, but as we add new features such as supporting multimodal input and natural language understanding, we would adhere to this philosophy. One example in the existing system would be to explore the benefits of supporting command mode and dictation mode using multiple frameworks.

CONCLUSION
We have described our experience with developing an object oriented framework for speech applications designed to simplify the integration of speech technology in applications. While not every subtlety associated with recognition technology may have been completely encapsulated, we believe that we achieved positive results with respect to our first objective of providing an open architecture with a simplified speech interface resulting in all three forms of reuse for speech applications. The reduced time to develop the latter framework applications endorsed the productivity gains obtained, despite the initial learning curve associated with the framework. The ability to rapidly integrate new technology such as speech recognition, is all the more important, given the recent trend of prototyping to evaluate the quality of a software solution at an early stage.

With respect to our second objective of building an infrastructure to support the integration of synergistic technologies, we can only say, that we believe we have introduced sufficient abstractions in the framework to meet this requirement. Future work involving the integration of multimodal inputs and natural language understanding must be done to validate this claim.

However, we do believe that the contribution of this paper is in consciously applying variations of emerging object oriented framework concepts to the problem domain of speech recognition applications, resulting in a flexible, coherent, reusable infrastructure for speech applications. The most tangible accomplishment is that of abstracting speech interface concepts using the object oriented paradigm so as to enable the rapid integration of speech recognition technology in applications.

Also, this iterative approach to object oriented design and development is by no means restricted to the domain of speech applications. It is just as applicable to any problem domain where the commonalities in a family of related applications can be effectively captured in a framework.
ACKNOWLEDGMENTS
We thank Dewayne Perry and Jacob Ukelson for their review and helpful comments.

REFERENCES
5. E. Gamma, R. Helm, R. Johnson and J. Vliissides. Design Patterns: Elements of Reusable Object Oriented Software. Reading, MA: Addison-Wesley, 1995.
7. IBM Corp., VoiceType Dictation, VoiceType Developers Toolkit. 1996.