Date: 1/12/01 Express Mail Label No. EL530287373US

Inventor(s): Steven I. Ross and Robert C. Armes

Attorney's Docket No.: 1280.2002-000

SYSTEM AND METHOD FOR DETERMINING UTTERANCE CONTEXT IN A MULTI-CONTEXT SPEECH APPLICATION

Goals and Objectives:

After the enthusiastic reception of Lotus' integration of ViaVoice speech recognition and synthesis technology into WordPro and 1-2-3, a scoping committee was charged with charting a course for future speech related developments. After much research, discussion, and the conducting of a focus group, the committee established the following goals:

Allow task-specific speech. Vocabulary specific to particular documents should be recognized when working with those documents.
Provide for "natural" interaction. The user should have a variety of intuitive ways to access speech functionality.
Permit cross-application speech. It should be possible to talk to background applications.
Make consistent dictation with voice control available where it makes sense. Word processing, text fields, etc.
Allow the user to speak visible text. Voice access to menus and especially dialog boxes should be possible.

In addition, several architectural objectives were established:

Provide an extensible architecture that will allow for growth in power and sophistication over time.
Minimize impact on existing applications.
Support limited dialog between the user and the system.

To meet these goals and objectives, the committee proposed the development of the *SpeechCenter.*

Overview

The SpeechCenter is an independent application which serves as a focal point for all speech communication with the user. The SpeechCenter provides speech services for a number of desktop applications, including the important task of deciding the target application for a particular utterance. This is in contrast to the conventional practice of individually speech-enabling applications, which ultimately leaves the user with a stable of voice-enabled applications, each fighting for the microphone, and each oblivious to the other.

By centralizing our speech services, we gain a number of advantages:

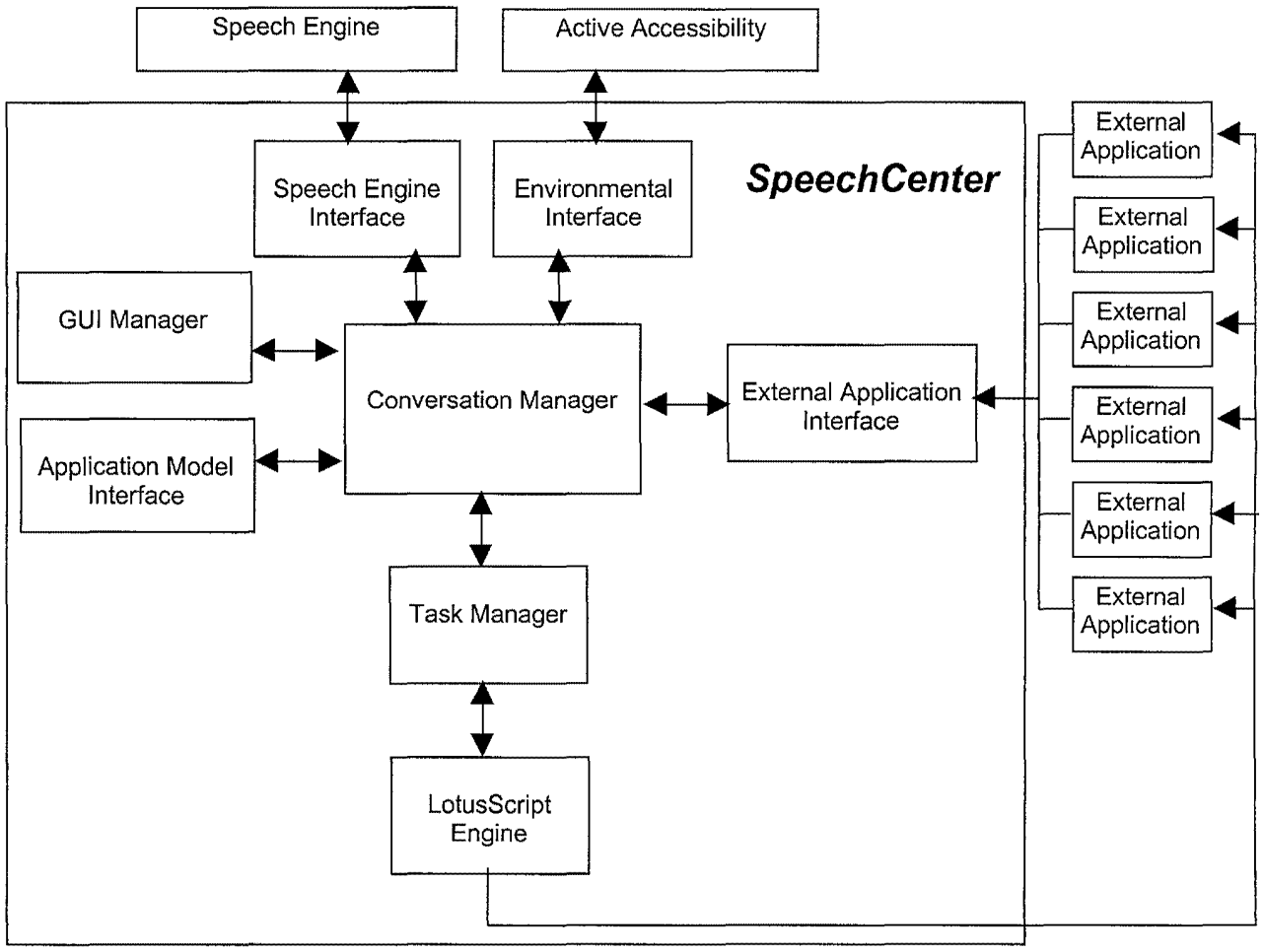
We avoid the time, expense, and footprint of duplicating speech services in multiple applications.

Express Mail Label No. EL530287373US

We achieve consistent functionality across applications. Dictation will have the same capabilities in all applications that support it, for example.
As technology improves, we can incorporate it into the SpeechCenter once, instead of into each application individually.
We can allow cross-application speech. You can be working with 1-2-3, and ask "When is my next appointment?" and have that information retrieved from Notes or Organizer, or "What is the IBM stock price" and have that retrieved from SmartCenter, or "Do I have any new messages?" and have the answer supplied by Notes.

Applications don't even have to be running to be playing in this game. Say "Take a letter", and WordPro could be started up if it isn't already running.
We can use speech to create an overall Environment which melds the capabilities of a number of applications on the user's desktop. These applications do not have to necessarily be Lotus applications. Any application that supports a robust automation UI can participate.

High Level Architecture



Meta Platforms, Inc. - Exhibit 1016
Page 2 of 16

The SpeechCenter consists of the following modules:

Speech Engine Interface
This module encapsulates the details of communicating with the speech engine, isolating the SpeechCenter from the speech engine specifics. While we certainly expect to make use of IBM's ViaVoice speech engine, we would be able to easily replace it with a better alternative if one came along.

Environmental Interface
The SpeechCenter needs to keep in touch with what is happening on the user's computer. Changes in window focus, dialogs popping up and being dismissed, and applications launching and exiting must all be monitored in order to interpret the meaning of voice commands. Initial implementation will be to use Microsoft Active Accessibility to provide this information, but again flexibility to change this or incorporate additional information sources is desirable. In order to fully take advantage of MSAA, some applications will need to make changes. 1-2-3, for example, implements its dialogs through UIMS, which is not MSAA-enabled, therefore without additional effort, 1-2-3 dialogs would not be speech-enabled through the SpeechCenter.

LotusScript Engine
The SpeechCenter controls applications by executing script against them. The LotusScript engine provides three particularly valuable capabilities:
The script engine supports cross-application scripting via OLE automation or through imported DLLs.
It is capable of executing arbitrary strings representing well formed LotusScript statements. This enables the SpeechCenter to easily compose calls to operations and invoke them.
Of course, the LotusScript environment also allows the definition of new subroutines and functions that combine the primitive functionality provided by applications into actions that more closely correspond to those that a user might talk about.

We will take advantage of these capabilities and leverage our investment in LotusScript by incorporating it into the SpeechCenter. While the SpeechCenter will be a script-enabled application, this does not mean that the applications that it controls need to be script-enabled. So long as an application provides an OLE automation or DLL interface, it will be controllable by the SpeechCenter.

Task Manager
The Task Manager controls script execution through the LotusScript engine. It provides the capability to proceed with multiple execution requests to simultaneously, queues up additional script commands for busy applications, and tracks the progress of the execution, informing the clients when execution of a script is in progress or has completed.

External Application Interface
For the most part, the SpeechCenter can operate without any modifications to the applications it controls, but in some circumstances, it may be desirable to allow the applications to communicate information directly back to the SpeechCenter. An interface will be provided to support this kind of push-back of information. This will be needed to allow applications to load custom grammars, or define task specific vocabulary. The External Application Interface will also allow applications to explicitly tap into the SpeechCenter for speech recognition and synthesis services. The SpeechChannel which provided speech services for SmartSuite R9 applications will be converted to use the External Application Interface, thus providing continued support of our initial speech offerings.

Meta Platforms, Inc. - Exhibit 1016
Page 3 of 16

Application Model Interface
The power of the SpeechCenter derives from the fact that it has significant knowledge about the applications it controls. Without this knowledge, it would be limited to providing little more than simplistic menu based command and control services, which is not a compelling use of voice technology. Instead, it has a detailed model of what a user might say to a particular application, and how to respond. That knowledge is provided individually on an application by application basis, and is incorporated into the SpeechCenter through this module.

GUI Manager
Even though the SpeechCenter operates primarily through a speech interface, there will still be some cases of graphical user interface interaction with the user. Recognition feedback, dictation correction, and preference setting are all cases where traditional GUI interface elements may be desirable. The GUI Manager abstracts the details of exactly how these services are implemented, and provides an abstract interface to the rest of the SpeechCenter.

Conversation Manager
The central component of the SpeechCenter integrates the information from all the other modules. In reality, the conversation manager is not really a component, but really the *internals* of the SpeechCenter. Isolated by the outer modules from speech engine and operating system dependencies, it is abstract and portable.

When an utterance is recognized, the conversation manager combines an analysis of the utterance with information on the state of the desktop and remembered context from previous recognitions to determine the intended target of the utterance. The utterance is then translated into the appropriate LotusScript calls and dispatched to the target application. The conversation manager is also responsible for controlling when dictation functionality is active, based on the context determined by the Environmental Interface.

> At this level of detail, this architecture leaves much unspecified. There are a range of implementations with varying features and capabilities that would correspond to this high level architecture. This is intentional. One of our goals is to provide a flexible and expandable architecture that can grow in power and functionality as our technology and infrastructure improves. It is our intention to implement the SpeechCenter architecture in two initial phases. Phase 1 will focus on providing initial implementations of all the components, roughing in the SpeechCenter infrastructure. While most of the initial module implementations will provide subsets of the intended functionality, the implementation of the Conversation Manager in Phase 1 will be a particularly simple implementation of the basic SpeechCenter infrastructure. Phase 2 will focus primarily on providing a more sophisticated and powerful conversation manager.

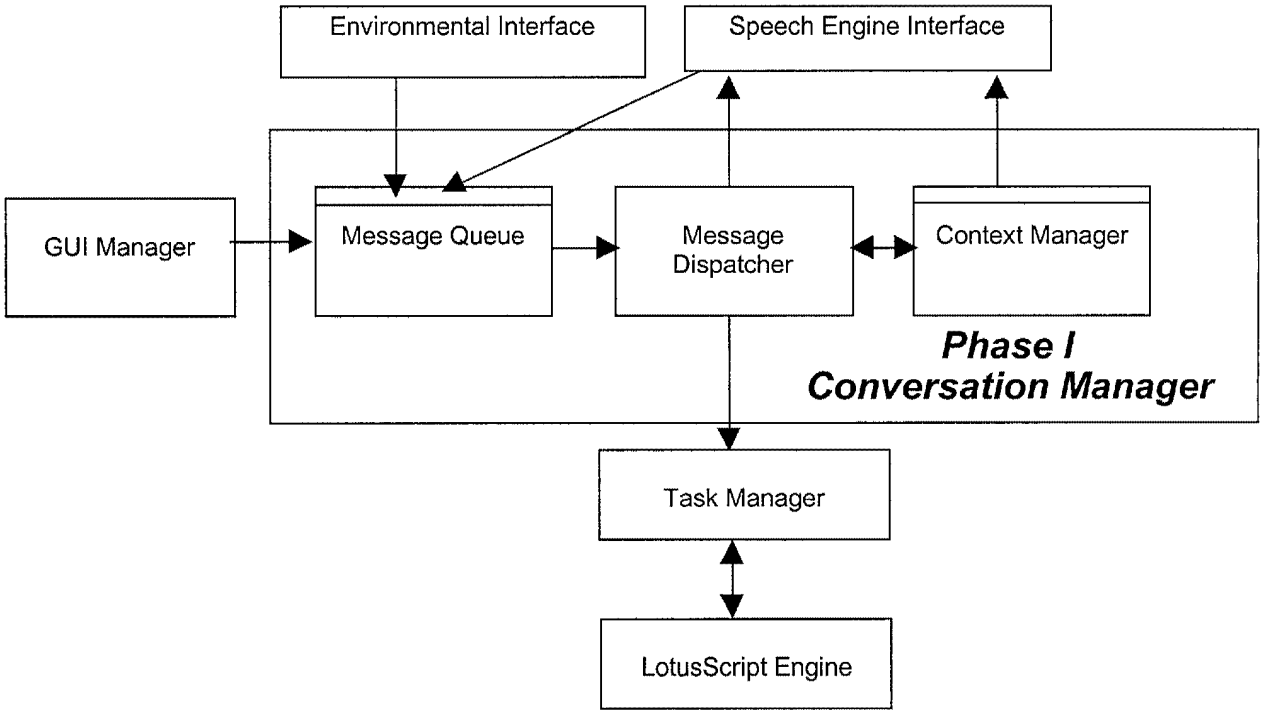
Phase 1 Implementation - The Prototype

The initial implementation of the SpeechCenter will make use of many of the tools, such as the translation mechanism, that we have already developed for dealing with voice. The goal is to provide multiple-application support relatively quickly. Toward that end, we will define a foreground and background grammar for each application, along with the associated translations to LotusScript, exactly as we did for the SmartSuite R9 speech support. The SpeechCenter will track speech focus in the Environmental interface with the aid of some rudimentary MSAA support, mostly tracking window focus. One grammar from each application will be active simultaneously, but since there will always be overlap between what

Meta Platforms, Inc. - Exhibit 1016
Page 4 of 16

the different application grammars will accept, the SpeechCenter will not necessarily dispatch a command back to the application whose grammar recognized it. Instead, when an utterance is recognized, it will be tested against each application's grammar to see if the grammar will accept it. The order of testing will be based on a *dynamic speech focus priority.* The first application's grammar which will accept the utterance will then be used for translation, and the command dispatched to the corresponding application. The speech focus priority will be established by maintaining an ordered list of applications. Whenever an application gains window focus, it will move to the head of the list. Likewise, whenever an application which is not the top priority application is chosen as the target for a speech command, *it* will move to the head of the list. In this way, the last application that the user touched or talked to will get first crack at interpreting the next user utterance, and the other applications will be ordered in a most-recently-accessed way.

Phase 1 Conversation Manager Architecture



The diagram above represents the structure of the phase 1 conversation manager. A central message queue receives asynchronous messages from the external interfaces as well as from internal modules. These messages are then processed and acted upon. The context manager maintains the priority and state of the various grammars in the system. Focus change messages from the environmental interface will cause the context manager to reshuffle module priorities and grammar states. Recognition messages from the speech engine interface will be tested by the context manager against the active grammars in priority order. When a successful match is found, the corresponding translation will be dispatched to the LotusScript Engine for execution, and the priority of the matching grammar raised.

Meta Platforms, Inc. - Exhibit 1016
Page 5 of 16

Phase 1 is intended to serve as a first order implementation, but is not intended to be a shipping product. The phase 1 implementation of the SpeechCenter will allow the same sorts of capabilities that we provided for 1-2-3 in R9, without any changes to the supported applications so long as they support an automation or DLL interface. Still, there are some drawbacks to this approach.

There will be no support for consistency across the different product grammars.
There will be a great deal of redundancy across the product grammars. This will make changing them in a consistent fashion very difficult.
There will be no support for automatic dialog between the user and the system to ascertain missing information. These kinds of conversations can only be engineered in a limited ad-hoc fashion in script on a case-by-case basis.
There is no support for maintaining any sense of the state of the conversation. The system treats every utterance as a totally independent event.

While the translation mechanism allows the system to convert utterances into action, there just isn't enough information there for the system to do any kind of reasoning about what is going on. For this reason, the phase 1 approach by itself is insufficient for achieving our goals for the SpeechCenter.

Phase 2 Implementation

To address these issues, the SpeechCenter architecture must have at least a crude model of the structure of the domains that it is dealing with. By providing the system with more knowledge about the domain, we can give it the power to perform a crude form of reasoning about the utterances that it receives and how to best respond to them.

The first step toward this goal is to provide an application independent *foundation domain model,*which will consist of common classes in the SpeechCenter that can be shared and extended by the applications. These classes will define commonly used concepts such as people, organizations, objects, attributes, actions, events, time, location, and number, along with base grammatical constructs that define ways that these items can be combined into sentences. Individual applications can extend this base model by adding their own elements to the base classes, or defining new subclasses of these classes and new sentence forms to work with them. This will address the redundancy and consistency issues, and will also make it much easier to develop grammars for individual applications, because we'll be building them on a solid foundation rather than starting from scratch.

These base classes go beyond defining a common grammar library, in that they provide the beginnings of a semantic model that the SpeechCenter can work with. The classes in our original grammars were just uninterpreted lists of words that had no real meaning to the system other than that imposed by their translation to LotusScript, but our new built-in base classes can have built-in meaning, in that the system can have a crude model about what kinds of entities populate the different built-in classes, and what kinds of operations can be performed upon them. We can further empower this model by explicitly representing the available LotusScript functions, their parameters, prerequisites, and effects. Rather than translating directly between utterance and action, we can then translate the utterance into an internal representation that captures the meaning of the utterance, and can then be further processed by simple matching and automated inference procedures. This will enable the system to respond to ambiguity and missing information in utterances, carry on simple dialogs, and generally behave in a more intelligent fashion.

These kinds of capabilities will be simple and modest at first. It is critical that we avoid becoming too ambitious, especially in the early phases of the project. It is unrealistic to aspire to achieving full natural language capabilities in a short time with limited resources. Researchers around the world have been working on this problem for more than thirty years, and it is still not solved. We can make use of these

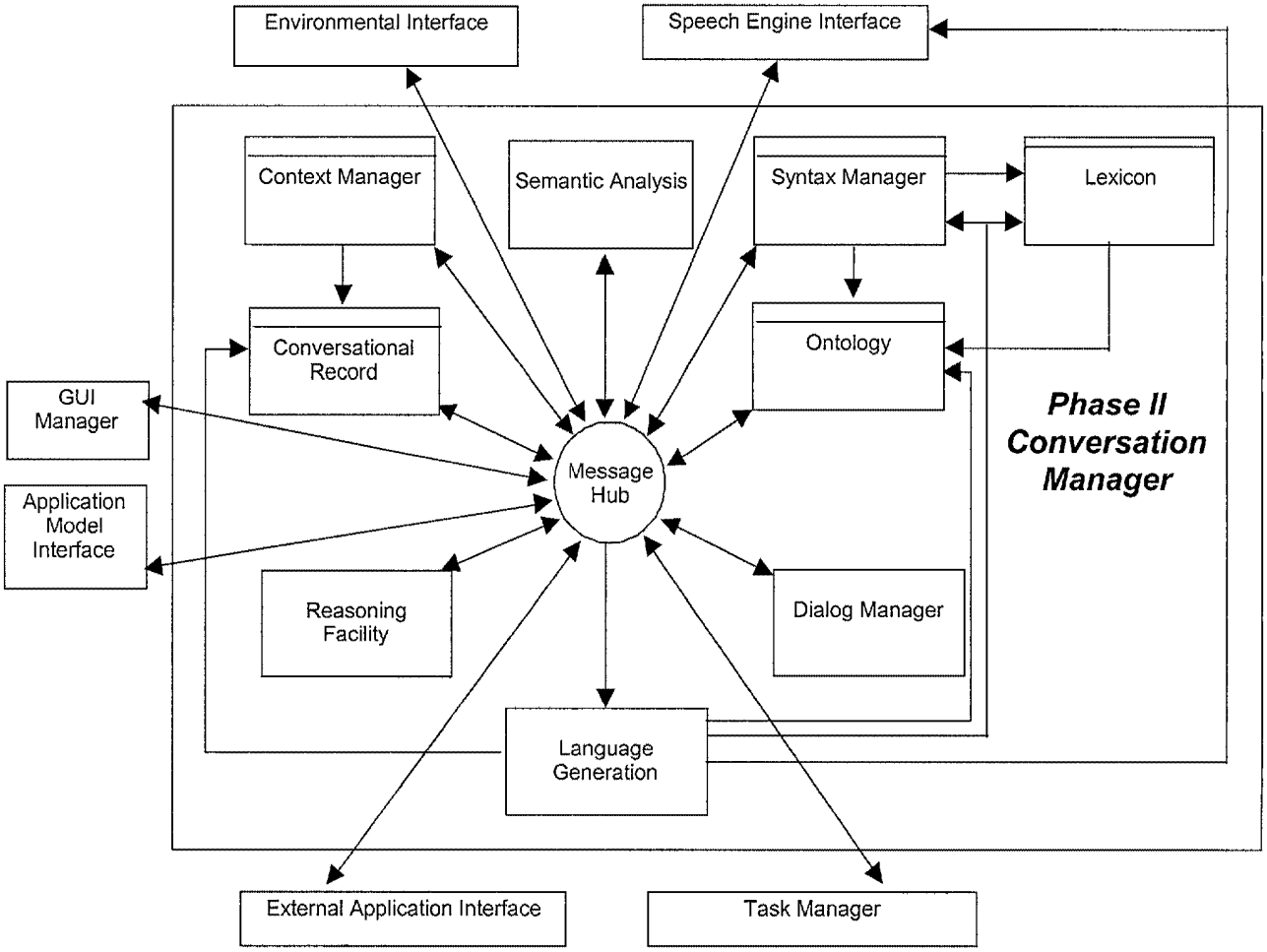
Meta Platforms, Inc. - Exhibit 1016
Page 6 of 16

research results to build a system that can carry on limited, but useful, natural language dialogs, and design our system so that we will have laid down the foundation for increasing sophistication and power in the future.

Google Ex. 1016 p. 7 of 16

Meta Platforms, Inc. - Exhibit 1016
Page 7 of 16

Phase 2 Conversation Manager Architecture



The diagram above represents the structure of the phase 2 conversation manager. This structure is basically an expansion of the phase 1 conversation manager structure. Each of the functional modules (indicated by plain boxes) and data abstraction modules (indicated by boxes with a bar across the top) will be implemented in a simple and straightforward manner initially. Increased sophistication and complexity in these modules later will allow for greater capabilities in the future. Even the initial implementation of this architecture will allow far more natural interactions than the Phase 1 implementation.

The Phase 2 Conversation Manager will consist of the following modules:

Message Hub
The message hub is simply the combination of the message queue and message dispatcher modules from the phase 1 architecture. It provides a way for the various modules to communicate asynchronous results. The central message dispatcher in the message hub has special purpose code for handling each type of message that it might receive, and calls on services in other modules to respond to the message. Modules are not restricted to communication through the hub. They are free to call upon services provided by other modules when appropriate.

Context Manager

Meta Platforms, Inc. - Exhibit 1016
Page 8 of 16

The context manager module is carried over from the original Phase 1 conversation manager. It keeps track of the targets of previous commands, factors in changes in the desktop environment, and uses this information to determine the target of new commands.

Domain Model

All applications have an implicit model of the corner of the world that they inhabit. This model guides the design of the UI and the functionality of the program. The problem with an implicit model is that it is all in the mind of the designers and developers, and so is often not thoroughly or consistently implemented in the product. Furthermore, since the model is not represented in the product, the product cannot act in accordance with the model's principles, explain its behavior in terms of the model, or otherwise be helpful to the user in explaining how it works. Our system will have an explicit, but extremely basic, model of the world which will serve as a foundation for language understanding and reasoning. Some of the basic concepts that the system will model are:

Things	A basic category that includes all others
Agents	Animate objects, people, organizations, computer programs
Objects	Inanimate objects, including documents and their sub-objects
Locations	Places in the world, within the computer, the network, and within documents
Time	Includes dates, as well as time of day.
Actions	Things that agents can do to alter the state of the world
Attributes	Characteristics of things, such as color, author, etc.
Events	An action that has occurred, will occurred, or is occurring over a span of time.

These concepts are described in the portion of the domain model known as the ontology. The ontology represents the classes of interest in the domain and their relationships to one another. Classes may be be defined as being subclasses of existing classes, for example. Attributes can be defined for particular classes, which associate entities that are members of these classes with other entities in other classes. For example, *person* class might support a *height* attribute whose value is a member of the *number* class. *Height* is therefore a relation which maps from its domain class, *person,* to its range class, *number.*

Although the ontology represents the semantic structure of the domain, it says nothing about the language used to speak about the domain. That information is contained within the syntax specification. The base syntax specification contained in the foundation domain model defines a class of simple, natural language-like sentences that specify how these classes are linked together to form assertions, questions, and commands. For example, given that we have defined classes for these basic concepts, we can define a simple form of command as:

```
template command(action)
<command> = <action> thing(action.patient)? manner(action)*.
```

Based on the ontology definitions of actions and their patients (the thing acted upon by an action) and on the definition of the thing and manner templates, this small piece of grammar specification would cover a wide range of commands such as "move down" and "send this file to Kathy".

To describe a new application to the conversation manager we will construct sentences in a language that describe the kinds of objects, attributes, and operations that the application makes available. To the extent that these objects and classes fit into the built-in domain model hierarchy, the existing grammatical constructs will apply to them as well. So, if an application provides an operation for, say, printing. It could specify:

```
print is a kind of action.
file is a patient of print.
```

and commands such as "print this file" would be available with no further syntax specification required.

Meta Platforms, Inc. - Exhibit 1016
Page 9 of 16

The description of an application can also introduce additional grammatical constructs that provide more specialized sentence forms for the new classes introduced . In this way, we provide a little model of the "world" inside this program, and a way to talk about it. Each supported program will have its own domain model included in its associated "application module description" (apm) file.

The SpeechCenter has a rudimentary built-in notion of what an "action" is. An "action" is something that an agent can do in order to achieve some change in the state of the world. The SpeechCenter has at its disposal a set of actions that it can perform itself. These are a subclass of the class of all actions that it knows about, and are known as operations. Operations are implemented as LotusScript functions. New operations can be added to the SpeechCenter by providing a definition of the function, and a set of domain model rules that describe the prerequisites and effects of the operation.

By providing the system with what is in effect "machine readable documentation" on its functions, we make it possible for it to choose which functions to call in order to achieve its goals. As an example, the user might ask the system to "Create an appointment with Mark tomorrow". Searching through its available rules it will find one that states that it can create an appointment. Examining the rule description, it will find that it calls a function which needs as parameters a person, date, time, and place. It will then set up goals to fill in these parameters, based on the information already available. The goal of finding the date will result in the location of another rule which will invoke a function that can calculate a date based on the relative date "tomorrow" information. The goal of finding a person will result in the location of a rule that will invoke a function which will attempt to disambiguate a person's full name from their first name. The goal of finding the time will not be satisfiable by any rules that the SpeechCenter knows about, and so a question to the user will be generated to get the information needed. Once all the required information is assembled, the appointment creation function will be called and the appointment scheduled.

One of the most important aspects of the domain model is that it is explicitly represented and accessible to the system. Therefore, it can be referred to for help purposes and explanation generation, as well as being much more flexible and customizable than traditional programs.

Syntax Manager

Grammatical specifications will be used to define the language that the SpeechCenter understands. The "foundation domain model" will contain a set of grammatical specifications that will define base classes such as numbers, dates, assertions, commands and questions. These specifications will be in an annotated form of BNF, that will be further processed by the syntax manager rather than being passed on directly to the speech engine. For example, we would like to support a grammatical specification for asserting a property for an object in our base grammar. In conventional BNF, it might take the form:

```
<statement> = <article> <attribute> of <object> is <value>.
```

This would allow us to create sentences like "The color of A1 is red" or "The age of Tom is 35". The BNF doesn't quite capture the desired meaning, however, because it doesn't relate the set of legal attributes to specific type of the object, and it doesn't relate the set of legal values to the particular attribute in question. We don't want to include in our grammar "The age of Tom is red", for example. Likewise, we would not want to allow sentences that specify attributes of objects that do not possess those attributes. To capture this distinction in BNF, we'd have to create separate definitions for each type of attribute, and separate sets of attributes for each type of object. Rather than force the person who specifies the grammar to do this, we will accept more general specifications in the form of *syntax templates,*which will then be processed by the syntax manager module, and the more specific BNF definitions will be created automatically. The syntax template version of the above statement would look something like this:

```
template statement(object)
attribute = object%monoattributes
<statement> = <article> <attribute> of <object> is <attribute.range>.
```


Meta Platforms, Inc. - Exhibit 1016
Page 10 of 16

This template tells the syntax manager how to take this more general syntax specification and turn it into conventional BNF based on the ontological information in the domain model. The grammatical specification is very tightly bound to the domain model ontology. The ontology provides meaning to the grammatical specifications, and the grammatical specifications determine what form statements about the objects defined in the ontology may take.

Given a syntax specification and an ontology, the syntax manager generates a BNF grammar which can be used by the speech engine to guide recognition. The grammar is annotated with translation information which can be used to convert an utterance recognized by the grammar to a set of script calls to the frame building functions of the sematics analysis module.

Lexicon
The lexicon implements a dictionary of all the words known to the system. The lexicon links each word to all the information known about that word, including ontology classes that it may belong to, and the various syntactic forms that the word might take.

Semantic Analysis
Rather than converting directly from recognized utterance to LotusScript action as we did in Phase 1, the conversation manager will convert the utterance into an intermediate form that is more amenable to processing. The translation process will initially convert recognized utterances into sequences of script calls to frame-building functions via the same translation facility that we now use to translate spoken commands directly into LotusScript. When these functions are executed, they will build frames within the Semantic Analysis module which serve as an initial semantic representation of the utterance. The frames are then processed into a series of attribute-object-value triples, which we call propositions. Frame to triple translation is mostly a matter of filling in references to containing frames. These triples are stored in memory, and provide the raw material upon which the reasoning facility operates. A sentence such as "make this column green" would be translated to a frame structure by a series of calls like these:

```
Begin("command")
AssociateValue("action")
Begin("action")
   AssociateClass("make")
   AssociateValue("patient")
   Begin("thing")
      AssociateClass("column")
   End("thing")
   AssociateValue("destination")
   AssociateParameter("green")
End("action")
End("command")
```

After the frame representation of the sentence is constructed, it is converted into a series of propositions, which are primarily attribute-object-value triples. A triple *X Y Z* can be read as "*The X of Y is Z*". The triples derived from the above frame representation are shown below. The words with numbers appended to them represent anonymous objects introduced by the system.

```
Class Command-1 Command
Class Action-1 Action
Action Command-1 Action-1
Class Thing-1 Column
Patient Action-1 Thing-1
Destination Action-1 Green
```

The set of triples generated from the sentence serve as the instructions to the reasoning facility, described below. Note that while much has been made explicit at this point, not everything has. The reasoning facility will still have to determine which column to operate upon, for example.

Meta Platforms, Inc. - Exhibit 1016
Page 11 of 16

Reasoning Facility
The reasoning facility performs the reasoning process for the conversation manager. It is a goal-directed rule based system composed of an inference engine, memory, rule base and agenda. Rules consist of some number of condition propositions and some number of action propositions. Each rule represents a valid inference step that the reasoning facility can take in the associated domain. A rule states that when the condition propositions are satisified, then the action propositions can be concluded. Both condition and action propositions can contain embedded script function calls, allowing the rules to interact with both external applicatons and other SpeechCenter components. Goals are created in response to user requests, and may also be created by the inference engine itself. A goal will be a proposition that may contain a variable for one or more of its elements. The system will then attempt to find or derive a match for that proposition, and find values for any variables. To do so, it will scan through the rules registered in the rule base, looking for ones whose actions unify with the goal. Once a matching rule has been found, the rule's conditions must be satisifed. These become new goals for the inference engine to achieve, based on the content of the memory and the conversational record. When no appropriate operations can be found to satisfy a goal, a question to the user will be generated. For phase 2, the reasoning facility will be primarily concerned with the determination of how to achieve the goals derived from the user's questions and commands.

Conversational Record
Conversational speech is full of implicit and explicit references back to people and objects that were mentioned earlier. To understand these sentences, we need to look at the conversational record, and find the missing information. Each utterance is indexed in the conversational record, along with the results of its semantic analysis. The information is eventually purged from the conversational record when it is no longer relevant to active goals and some period of time has elapsed.

For example, after having said, "Create an appointment with Mark at 3 O'clock tomorrow", a user might say "Change that to 4 O'clock." The system will establish that a time attribute of *something* is changing, but will need to refer back to the conversational record to find the appointment object whose time attribute is changing. Usually, the most recently mentioned object that fits the requirements will be chosen, but in some cases the selection of the proper referent is more complex, and involves the goal structure of the conversation.

Dialog Manager
The dialog manager serves as a sort of traffic cop for information flowing back and forth between the reasoning facility and the user. Questions generated by the reasoning facilility as well as answers derived to user questions and unsolicited announcements by the system are all processed by the dialog manager. The dialog manager also is responsible for managing question-answering grammars, and converting incomplete answers generated by the user into a form digestible by the reasoning facility.

The dialog manager has the responsibility for deciding whether a SpeechCenter-generated response should be visible or audible. It also decides whether the response can be presented immediately, or whether it must ask permission first. If an operation is taking more than a few seconds, the dialog manager generates an indication to the user that the operation is in progress.

Language Generation
When questions or responses to the user are derived by the Inference engine, they must be translated back into Natural Language. Ultimately we would like to take maximal advantage of the knowledge stored in the syntax manager, domain model, lexicon, and conversational record in order to generate natural language output. Our initial implementation, however, will necessarily be much more simple and ad-hoc. We will generate language from the same syntax templates we use for recognition, or from additional templates provided specifically for language generation. Rules in the rule base will decide which template is appropriate for the language generation task at hand.

Phase 3

Meta Platforms, Inc. - Exhibit 1016
Page 12 of 16

There is little point in detailed speculation about changes to the SpeechCenter beyond phase 2. Undoubtedly, our experience in implementing and using the phase 2 system will suggest all sorts of new directions and improvements. Nevertheless, I'd like to point out a few possible directions for further development.

Grammar-based recognition will always have the undesirable quality of inflexibility. There is no way to introduce a new word into the vocabulary by voice, since the system, by definition, can't recognize words that it doesn't know about. If we could use dictation as the input medium, then this problem would be eliminated, but new problems would be introduced. We would have to extend and improve our parsing capabilities, and be prepared to deal with extra-grammatical utterances. In addition, we would have to contend with the likelihood of reduced recognition accuracy, due to the reduction in constraints that comes with the dictation model. It may be possible to use our explicit representation of grammatical information in the syntax manager, along with ViaVoice alternative word guesses and scores to approximate the accuracy of grammar-based recognition while retaining the flexibility of dictation-based recognition.

To increase the range of utterances that we can handle, we will have to make the system more aware of word morphology, so that it can handle different tenses, endings and prefixes for words in the lexicon.

As the flexibility of our natural language capabilities improves, we will have to be prepared to deal with more ambiguity in the input. Particular words may have many possible senses and meanings, and a single utterance could have multiple interpretations. Using context and the conversational record may help to disambiguate utterances in some circumstances, but there will always be cases where the system will have to ask the user to clarify the meaning. Of course, there will also be circumstances where the system can find no interpretations because it isn't aware of some of the words or word senses that the user is employing. Enabling the system to carry on a dialog about sentences that it doesn't understand, and then improve its knowledge so that it can understand such sentences in the future would make for a system that could be painlessly customized by the user.

Inference capabilities can be improved to allow other forms of reasoning besides just goal-directed rule processing. The grammar can be extended to allow users to speak utterances which result in new rules and formulae being entered into the system in such a way as they will affect future reasoning. The system will then be teachable in at least a limited sense.

Improving the amount and variety of knowledge that the SpeechCenter contains will enable it to carry on more natural dialog with the user, and perform more powerful and useful inferences. At some point, we can consider incorporating knowledge of the user's domain, including goals, plans, tasks, and processes. This will enable the system to have a better understanding of why a user might be asking a particular question or giving a particular command, an expectation of what might be coming next, and ultimately the ability to take on a more active and helpful role.

In addition to improving the conversation manager, we can consider developing new applications that take advantage of its functionality. Knowledge management would be an excellent candidate, for example. Additional applications could be seen as providing new *capabilities* to the desktop environment created by the SpeechCenter.

Meta Platforms, Inc. - Exhibit 1016
Page 13 of 16

In speech recognition systems for which multiple application contexts coexist simultaneously, determining the target context for a particular utterance can be difficult if more than one context will accept that utterance. For example, on a computer desktop, there could be several speech-enabled applications active, and the system must determine the target application for each utterance. Though some utterances might be understandable to only certain applications, other utterances might be understandable by more than one application, or even by all of them. This invention provides a means for assigning an application context to each utterance.

Our invention prioritizes the contexts based on recency of relevant access, and targets an utterance to the highest priority context that will accept it. This provides a seamless cross-context capability.

Existing desktop speech systems either limit the available context to the application with window focus, or allow multiple applications to share a global context, provided that their language models do not overlap. Our approach allows users to speak to contexts corresponding to background applications, and gracefully handles overlapping context language models.

Meta Platforms, Inc. - Exhibit 1016
Page 14 of 16

Bibliography

Allen, James, Natural Language Processing, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA (1995)

Bateman, John A., *Upper Modelling. A general organization of knowledge for natural language processing"* Workshop on Standards for Knowledge Representation Systems, Santa Barbara. (March, 1990)

Blank, Glen David, *A Finite and Real-Time Processor for Natural Language,*in Communications of the ACM, Volume 32, No. 10, (October, 1989)

Charniak, Eugene, and McDermott, Drew V. Artificial Intelligence. Addison-Wesley Publishing Company, Reading Massachusetts (1987)

Door, Bonnie J., *Principle-Based Parsing for Machine Translation,* Massachusetts Institute of Technology Artificial Intelligence Laborartory, A.I. Memo No. 947 (December, 1987)

Franz, Alexander, *Ambiguity Resolution in Natural Language Parsing,* Carnegie Mellon University Center for Machine Translation Memo 92-135, (May, 1992)

Goodman, Gary, and Reddy, Raj, *Alernative Control Structures for Speech Understanding Systems,*in Trends in Speech Recognition, Wayne A. Lea, Editor, Prentice-Hall, Inc. Englewood Cliffs, NJ (1980)

Hayes, Philip J., Hauptmann, Alexander, G. Carbonell, Jaime G., Tomita, Masuru. *Parsing Spoken Language: A Semantic Case-Frame Approach.* Carnegie Mellon University Center for Machine Translation Memo 87-103, (January, 1987)

Hayes-Roth, Frederick, *Syntax, Semantics, and Pragmatics in Speech Understanding Systems,*in Trends in Speech Recognition, Wayne A. Lea, Editor, Prentice-Hall, Inc. Englewood Cliffs, NJ (1980)

IBM, *Speech Manager Application Programming Interface Developers Guide,* IBM ViaVoice SDK for Windows (August 1998)

Katz, Boris, *A Three-Step Procedure for Language Generation,* Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Memo No. 599 (December, 1980)

Katz, Boris, and Levin, Beth, *Exploiting Lexical Regularities in Designing Natural Language Systems,* Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Memo No. 1041 (April, 1988)

Katz, Boris, *From Sentence Processing to Information Access on the World Wide Web,*AAAI Spring Symposium on Natural Language Processing for the World Wide Web, Stanford University, Stanford CA (1997).

Levin, Beth, *Mapping Sentences to Case Frames* Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Working Paper No. 143 (March, 1977)

Martin, William A. *Natural Language and the Computer Representation of Knowledge,* Unpublished (1980).

Microsoft Corporation, *Microsoft Speech 4.0 Software Development Kit,* (1995-1998).

McAllester David, and Givan Robert, *Natural Language Syntax and First Order Inference,* Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Memo No.1176 (October, 1989).

Rich, Charles and Sidner, Candance L., COLLAGEN: *When Agents Collaborate With People*, Lotus Technical Report 96-12, February, 1997.

Meta Platforms, Inc. - Exhibit 1016
Page 15 of 16

Tano, Shun'ichi, *Natural Language Understanding System using Text-based Inference.* Carnegie Mellon University Center for Machine Translation Memo 91-124, (April, 1991)

Winston, Patrick Henry, Artificial Intelligence, Addison-Wesley Publishing Company, Reading Massacusetts (1977)

Meta Platforms, Inc. - Exhibit 1016
Page 16 of 16