BACKGROUND OF THE ART

Understanding and processing natural language, as either spoken or written by humans, has long been a goal in the field of artificial intelligence. As computers have been programmed of late to perform awe-inspiring feats, such as defeating the world's best human chess master in his game, other skills exhibited by humans are still seemingly beyond the reach of even the most powerful computers. Although a small child may not be able to play chess, that child typically has the facility to process and understand its native tongue. Computers, on the other hand, have yet to exhibit any significant level of mastery in the realm of natural language processing.

One attempt at simulating natural language skills is the virtual robot, or BOT. A BOT may use a scripting language that matches input sentences from a user against input templates of keywords. An input template might, for example, take a group of related keywords and lump them together for the purposes of responding. Thus, words like "father, mother, brother, sister" might be grouped together for a response that relied on the concept of "family". In addition to recognizing familiar words, a scripting language capable of recognizing the ways these words are used in the sentence, and of tracking context across sentences, enables an associated processing model to track and respond to a wide variety of utterances. Generally, the process model that makes use of a scripting language will have a default response if none of the keyword templates matches the input sentence. Thus the BOT always has a response.

Script programs may be written by human designers having little or no formal programming experience. For the purposes of the discussion, some characteristics of a scripting language are described in Backus Normal form below. There are 4 characteristics of a scripting language of interest in this discussion: (1) topics, (2) subjects, (3) conditions, and (4) pattern lists. A topic is used to process user statements.

```
Topic <string> is
   <Tstatement>*
EndTopic
```

There are three types of topics for the purposes of this discussion: standard, default, and sequence. Standard topics are used to recognize and respond to utterances. If more than one standard topic matches an utterance, then computational mechanisms can be used to prioritize and then select one to apply in constructing a response. Two such mechanisms are specificity, which is based on the information content in the utterance, and recency, which is based on which subject most recently uttered is associated with a topic. Because standard topics are used to respond to utterances, they are executed first. Default topics respond when standard topics cannot respond. There are two types of default topics. One default topic type is associated with a standard topic, so it is related to the subject of the standard topic. Another default topic type is a universal default, sometimes called a "last line of defense", which will respond in the event that no other standard or default topic can respond. Default topics are tested in the order in which they appear in the program and the first applicable default is used to build the response. Sequence topics are used to recognize and respond to utterance sequences when the utterances are connected by how the user responds. Sequence topics are executed only when explicitly accessed in a SwitchTo statement, which is a form of redirection. Sequence topics have the lowest priority of the three topic types described.

The body of each topic is a list of conditional blocks. These conditional blocks are executed in the order found in the topic. If the condition of a conditional block is false, execution goes on to the next conditional block in the topic, or to the next topic if there are no further conditional blocks. If the condition is true, the commands and conditional blocks inside the block are executed, and further behavior of the program is dependent on the keyword which ends the conditional block. If it ends with Done, execution ceases until the next input occurs. If it ends with Continue, execution continues with the next conditional block in the topic, or the next topic if there are no further conditional blocks. If it ends with NextTopic, the rest of the current topic is skipped and execution continues with the next topic.

```
<SubjectList> = Subjects <string> [,<string>]*;
```

The top level of a topic may contain one or more Subjects statements. Each asserts that the given subjects are subjects of the topic. If a non-IF command within the body of the topic is executed, all topics which share at least one Subject with the topic are brought to the front of the "focus of attention." Focus of attention generally refers those things a person is currently thinking of. In this context, topics sharing a subject match part of what the user uttered and are used as the first sorting mechanism for topic selection.

```
<Condition> = If <conditionpatlist> Then |
              IfHeard <patlist> Then |
              IfHeard <pat> [and <pat>]* [and not <pat>]* |
              IfRecall <memlist> Then |
              IfRecall <memref> [and <memref>]* [and not <memref>]* |
              IfDontRecall <memlist> Then |
              IfDontRecall <memref> [and <memref>]* |
              IfChance <chance> Then |
              IfChance Then |
              Always

<patlist>  = <pat> [, <pat>]*  | <symbol>
```

A pattern list is anything that evaluates to a list of strings. It can be either the name of a PatternList object or a list of patterns separated by commas.

A virtual robot generally embodies a particular universe of discourse reflective of the subject matter of interest -- e.g. a BOT developed to converse about personal computers should "know" something about computers and their peripherals. The development of such a BOT employs the scripting language to recognize aspects of the subject matter and respond with appropriate content. Often these "script programs" (or scripts) are written in an action-response type style wherein the actual language supplied by the user embodies an "action" to which the "response" is written into the script program itself.

Scripts are generally written by a "BOT administrator" (human or otherwise) by defining a list of “categories” in which the BOT will be well conversant. Categories may comprise "topics" that are recognizable by a runtime executive. Topics, in turn, may comprise patterns or words that are matched against the stream of input communication (in either spoken or written or any other suitable form of communication) from the user.

The main drawback with constructing virtual BOTs by a list of categories is that the topics developed cannot provide complete coverage of all subjects in the universe of discourse. The result is that the BOT responds with the universal default. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with the default. A related drawback is that the universal default response generally provides insufficient guidance to the user as to their original input: it doesn't provide information regarding why the input "confused" the BOT, and it doesn't provide a knowledgeable response to the input.

The BOT development task must recognize that the level of quality and value evidenced by users is not judged merely in discrete terms, but rather by the overall impression that they get from their interaction with the BOT and by their level of satisfaction with the information the BOT provides.

Thus, there is a need in the art to have a means for easily designing and creating virtual BOTs that enables the BOT to effectively respond to arbitrary utterances with knowledge regardless of the number of topics implemented, guides the user toward providing utterances that will move the user closer to the information they seek, provides the user with information about what in the user's utterance confused the bot when that occurs, and performs these tasks within a framework the eases the maintanence and entendability of the BOT's capabilities.

SUMMARY OF THE INVENTION

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a suitable operating environment for the purposes of the present invention.

Figure 2 depicts the topic types used to present a typical implementation of the present invention.

Figure 3 depicts developed topics in a universe of discourse.

Figure 4 depicts a procedural decomposition of a program into its computational subtasks.

Figure 5 depicts a collection of conceptual domains which describe a universe of discourse.

Figure 6 depicts a uniform distribution of defaults in a universe of discourse.

Figure 7 depicts a set of developed topics in a uniform distribution of defaults.

Figure 8 expands the family view of a conceptual domain as shown in Figure 5.

Figure 9 depicts domain topics for the conceptual domain as found in Figure 8.

Figure 10 depicts the cauterization problem.

Figure 11 depicts the cauterization solution for a domain's children.

Figure 12 depicts the cauterization solution for a domain's parent and siblings.

Figure 13 depicts the tiebreaker problem.

DETAILED DESCRIPTION OF THE INVENTION

I. OVERVIEW AND GENERAL ARCHITECTURE

The term "robot" is used interchangably with "BOT" throughout the remainder of this writeup. For the purposes of this writeup, both "BOT" and "robot" refer to any program which interacts with a human user in some fashion, and should not be assumed to refer only to physically embodied robots. The term "supervisor" is used interchangably with "administrator". The term "domain" is used to represent a body of knowledge that comprises a component of a universe of discourse that the "BOT" will be conversant with. The term "hierarchy" will be used to describe the collection of domains that represents the universe of discourse that the "BOT" will be designed to converse about. It should be noted that the term hierarchy should not be construed to mean the collection data type. Many means of representing the collection are known to those skilled in the art, and the functionality of the collection is not dependent on the data type used to implement it.

Referring now to Figure 1, the operating environment in which the present invention applies is depicted. The environment can be characterized generally into three partitions: front end 102; BOT processor 100; and back end 104. Front end 102 is generally the environment in which a human user 116 consults a virtual BOT interface 114 via a computer 112 that may be connected to the BOT processor via a communications link, such as through a server connected to the Internet or alternatively directly connected to BOT processor 100. It will be appreciated that many other means of connection to a BOT processor 100 are well known to those skilled in the art and that the present invention should not be limited to any particular aspects of the general operating environment as disclosed herein.

Typically, human user 116 connects to a site whose interface of first impression is a virtual BOT interface 114. The advantage for the site developer is that human user 116 may have a help or information

request that is easily handled via BOT interface 114. Today, it is not uncommon to find sites having a list of FAQs ("Frequently Asked Questions") that serve this purpose of handling very low level user concerns and questions. However, for more advanced questions or interactions with the site, virtual BOTs will become increasing popular.

In the operating environment that hosts the embodiment of the present invention, BOT interface 114 is an instantiation of a process that is spawned by BOT processor 100 via connection 110. BOT processor 100 itself may comprise connection 110; runtime executive process 106, compiler 107, and a set of BOT programs 108. As users 116 log onto a site having BOT processor 100 via connection 110, runtime executive 106 executes an interaction routine that guides the discussion that occurs between user 116 and BOT processor 100. Typically, a two way communications dialogue occurs between user 116 and BOT processor 100 wherein user 116 may ask questions, make declarative statements and other normal communications patterns that humans typify. For the purposes of the present invention, “communications” is to be very broadly interpreted. Indeed, suitable communications could be in the form of written or spoken language, graphics, URL’s or the like that may be passed to and from a user to an automatic interface program, such as the present invention.

In turn, runtime executive 106 parses the statements and questions generated by the user and responds according to a set of BOT programs 108. As will be discussed in greater detail, BOT programs 108 are typically created at the back end 104 as a set of "scripts" that the BOT processor will tend to engage in with user 116. For example, if the site using BOT processor 100 is a site for a reseller of personal computers, then BOT processor 100 should be designed to handle questions and discussions concerning personal computers and their peripherals in general. Thus, the back end 104 will generate scripts that will guide the discussion concerning many computer-related topics. These script programs 108 are then compiled by compiler 107 and the compiled code is incorporated into runtime executive 106.

As the two-way discussions between user 116 and runtime executive 106 continue, it is generally desirable to engage in quality control of BOT processor 100. This quality control is provided at back end 104 via feedback loop comprising a transcript of dialogues 118 and backtrace and state information 120 of the BOT processor 100; a supervisor 122 and editor 124. As transcripts develop over the course of interacting with a user, the text of these transcripts are stored, together with the state of the runtime executive and backtrace of execution through the runtime executive code. This information forms the basis for accurately diagnosing the runtime executive and for debugging its performance. Such information may be stored electronically in a storage media or could be printed out in human readable form.

Supervisor 122 analyzes the information at 118 and 120 with an eye towards optimizing the performance of the runtime executive. Typically, supervisor 122 could be another human, deciding if the semantics captured by the system needs to be upgraded in response to a dialog transcript that has occurred. If so, supervisor 122 could optionally invoke an editor 124 to edit the programs that represent the semantic framework of the runtime executive. These programs would then be re-compiled and incorporated into the runtime executive.

Although Figure 1 gives a general description of various operating environments in which virtual BOTs may exist, it will be appreciated that many other operating environments are obvious to those skilled in the art and that the scope of the present invention should not be so limited to the exemplary descriptions as given

above.

II. BOT DEVELOPMENT

In general, BOT user 116 knows what kind of information he seeks, but not necessarily how to articulate his request. Functionally, the implementation of the art described herein provides user 116 with the ability to interact with BOT 100 on any level of abstraction directly associated with the universe of discourse. BOT 100 should be able to achieve this either with a single response to a very specific question that immediately identifies the users' needs, or through dialog in response to a series of increasingly specific queries guided by BOT 100 through the BOT programs 108. The mechanism is independent of the order in which topics are developed.

A. CURRENT MECHANISM

As mentioned in the background, BOT programs 108, or scripts, are generally written by BOT administrator 122 by defining a list of categories in which BOT 100 will be well conversant. These categories generally come from BOT administrator 122, FAQs, user input, and other sources associated with the universe of discourse. BOTs developed this way suffer from two problems. First, BOT 100's ability to answer questions is dependent on the number of topics developed, and the finite number of topics developed by BOT administrator 122 cannot completely cover all aspects of the organization associated with the universe of discourse. Thus, it is highly likely that queries made by BOT user 116 will result in "default" responses. Second, the development of BOT 100 follows specific needs/interests of administrator 122, rather than a systematic, coherant approach. Thus there is no coherence between the topics or their defaults.

Figure 3 illustrates the problem that arises when BOT programs incompletely cover the universe of discourse. Universe of discourse 300 is comprised of those topics 302 that have been developed. Each topic 302 potentially has its own default 304. The content areas remaining when all topic-related content areas are covered are responded to by universal default 306, which will be "hit" by all off-topic user queries. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with a default. Since the defaults 304 are related to the topics, rather than to each other, or to the universe of discourse, hitting defaults 304 or 306 provides the user with little information of value that would assist in continuing the conversation. There is thus an inherant qualitative problem in developing BOTS from a list of topics. Since BOT 100 is a conversational agent, its value derives entirely from how well it interacts with users 116, where the word "well" may be defined in terms of knowlege content BOT 100 conveys, its friendliness, how easily it is "confused", and how much interaction is required for user 116 to find what is sought. If user 116 asks questions that confuse BOT 100, then BOT 100 is seen as "stupid" and its value is diminished.

The embodiment of the current invention implements BOT 100 by designing a framework of defaults in such a way that it is impossible for BOT 100 to be asked a question that cannot be answered. The structural foundation for this framework, and the interacting mechanisms associated with it, comprise the art described in the remaining sections.

B. APPLICABLE SOFTWARE DESIGN/DEVELOPMENT TOOLS

BOT programs 108 can be thought of as a program in the traditional computer engineering framework. Each can be decomposed to increasingly specific components.

Figure 4 illustrates a typical procedural decomposition, because in both the result can be decomposed to increasingly specific tasks. In the traditional framework, program 402 is decomposed into constituent functional components, or tasks. The "Initialize Objects" 404, "Input Data" 406, "Do Calculations" 408, and "Output Data" 410 tasks are independent functional tasks that, together, comprise program 402. The functional components interact through data objects. Each functional component (404-410) is itself decomposed to those functional tasks that comprise it. For example, the "Do Calculations" 408 task is decomposed to "Perform Analysis" 412 and "Continue Test" 414 subtasks. The decomposition continues to a point where the designer/developer is satisfied that source code can be developed. The tasks at this level are called "terminal" tasks, because they aren't further decomposed. In the software development task, intermediate tasks, such as "Do Calculations" 408, are used to organize the functions of their subcomponents, and their effects are otherwise not seen. Terminal tasks implement the actual functionality of program 402.

Once a procedural decomposition is designed, a program can be implemented without developing any of the terminal functionality. A problem that often arises in software design and development is the tendency for developers to develop the terminal functionality without first implementing the supporting framework depicted by program structure 400. This problem is analogous to developing BOT topics without a supporting content framework. The difference is that the BOT can still function, though its function is severely hampered, whereas a program cannot. Software engineering has a tool, called a function stub, that enables the developer to implement the framework of program 402 without implementing the terminal functionality. The stub is a structure that adheres to the input/output requirements of program 402's design, but implements none of the functionality. For intermediate tasks, stubs comprise the appropriate function calls to subcomponent tasks. Using this approach, the software developer can implement a large program without implementing any of the functionality, and then replace the empty stubs with the actual code that implements the design algorithms for a particular task.

The art described herein conceptually makes use of the hierarchical/procedural decomposition and function stub to develop a BOT that has an organizational structure and is developed around a hierarchical framework of defaults. The analog to program 402's functional component is called a "domain". The analog to program 402's functional decomposition is a content decomposition called a domain "hierarchy." The analog to the function stub is the "domain default." Regardless of the user's query, or the degree of topic development, the BOT developed with a domain hierarchy and domain defaults can answer the user and even direct the user toward what they seek.

C. HIERARCHICAL DECOMPOSITION OF THE UNIVERSE OF DISCOURSE

Every universe of discourse can be described as a grouping of different content components that can be

hierarchically decomposed. Using a virtual robot to provide information about a particular universe of discourse, one must understand how that universe is decomposed and design the robot around the associated hierarchy.

Figure 5 depicts a domain hierarchy for "Company" 502. The decomposition represents the sum total of all information about the company, which is decomposed into four informational and functional components, "Information" 504, "Products" 506, "Services" 508, and "Sales" 510. The "Products" component 506 is shown further decomposed into three components, "Widgets" 512, "Flingys" 514, and "Gadgets" 516. The pictured decomposition is typical, but unimportant. What is important is that the combination of components completely describes the organization, analogous to the relationship of the functional components (404-412) and program 402 in Figure 4.

Each component 502-516 in hierarchy 500 is called a "domain." A domain represents an informational aspect of the organization. This type of hierarchy is called a "component" hierarchy, in that the domains under the top, or "root," domain (i.e., domain 502) are considered components of domain 502. Thus, “Information” 504 is a component of “Company” 502. The reverse is not true (i.e., “Company" 502 is not a component of “Information” 504). A terminal domain is one which has no subdomains. Just as a program’s functionality is implemented in its terminal components, BOT 100s specific content is implemented in its terminal domains. The overall hierarchical decomposition 500 is called a domain "hierarchy."

D. DOMAIN TOPICS

The domain analog to the function stub is a mechanism that allows BOT 100 to respond in lieu of topic development associated with a terminal domain. The domain stub is effectively a default response to a query. It is thus referred to as a domain default.

Figure 6 illustrates the universe of discourse 300 for BOT 100 when it is developed using a domain hierarchy and domain defaults (i.e., stubs), as opposed to the largely vacant universe of discourse 300 depicted in Figure 3. The universe of discourse 300 is now completely divided into non-overlapping areas 602 that represent the domains in hierarchy 500. Two types of queries can be made to BOT 100 in lieu of terminal topic development: (1) relevant and unspecific, or (2) irrelevant. A "relevant" query is one which hits a domain in the universe of discourse (i.e., a domain in the hierarchy). A "specific" query is one that would hit a terminal domain topic. Thus a relevant but unspecific query should hit an intermediate domain in the hierarchy. An "irrelevant" query is one which misses the domain associated with the current focus of attention or, in the worst case, the universe of discourse (i.e., hits no domains in the hierarchy). Despite the fact that no topics are shown in Figure 6, BOT 100 will respond in a coherent manner to relevant but unspecific queries as well as irrelevant queries.

Figure 7 illustrates that topics 302 could be developed for any terminal domain, in any order, once the domain hierarchy and its defaults are in place.

The analogy to a program decomposition and a function stub breaks down with a virtual robot in three

ways: (1) the user/program interaction is different than a user/BOT interaction, (2) the relationship between program components is different than the relationship between hierarchy domains, and (3) the function stub responds differently than the domain default. In a program, a user has very specific points where input is allowed, and very specific points were output is produced, whereas BOT user 116 could possibly interact with BOT 100 at any domain as mentioned above. Moreover, the program user would have no use for output at intermediate points, since all that is necessary is the functional implementation at component terminals. The domains in a virtual bot hierarchy are always related by content, so every domain from root domain 502 to a particular terminal domain can provide a viable degree of content for the terminal domain. As a result, a query directed at "Widgets" 512, if made in a vague way, could be responded to by "Products" 506, or even "Company" 502. Finally, the function stub response may or may not have a value, but the domain default always has a language response.

For example, if the terminal domain is "Widgets" 512, then domain "stubs" are required for each of the "Company" 502, "Products" 506, and "Widgets" 512 domains, because user 116 could make a query regarding "Widgets" 512 that could hit domain defaults at either "Products" 506 or "Company" 502, depending on the degree query specificity. Any of the following queries/comments could be issued by user 116 with respect to widgets:

Q1 - "Do you have any widgets?"
Q2 - "What kind of widgets do you have?"
Q3 - "Widgets"
Q4 - "What products do you have?"
Q5 - "What do you have?"
Q6 - "Do penguins live here?"

The so-called domain "stubs", or "defaults", could respond to these queries without any specific information regarding widgets being developed. Such responses would look like the following:

Q1 - "Do you have any widgets?"
A1 - "Yes, we have a number of widgets, would you like to see a listing, would you like information about a particular type of widget, or would you like to see our inventory?"
Q2 - "What kind of widgets do you have?"
A2 - "We have red, green and blue widgets. Would you like a feature comparison chart or information about a particular model?"
Q3 - "Widgets"
A3 - "Widgets are one of our products. They are used in many industrial applications. We also manufacture and sell flingys and gadgets."
Q4 - "What products do you have?"
A4 - "We manufacture and sell widgets, flingys, and gadgets. Would you like to know more about any of these, or are you interested in our inventory of any of them."
Q5 - "What do you do?"
A5 - "I am glad you asked that. We are a small manufacturing company. We design and build the best widgets, flingys and gadgets money can buy. Would you like more information on a particular product?"
Q6 - "Do penguins live here?"
A6 - "I’m sory, I do not understand what you mean. Weren’t we just talking about our company’s products?

Notice that all of the queries are a bit unspecific, and that there are different types of responses to the different types of queries. In query Q1, it is unclear whether user 116 is seeking information about whether the organization produces widgets, what kind of widgets, or whether user 116 wants to know how many are in stock. The answer must be a combination of what would be at a terminal domain and information obtained from the "Sales" 510 domain. In query Q2, the request is really for an enumeration of widget types, but it is vague because what the user wants to know about widgets is unspecified. The response should provide both the information requested and provide options for viewing that information. Query Q3 only mentions the term "Widgets" itself, and so BOT 100 can only assume that user 116 seeks to know more about widgets, or maybe what role they play in the organization. Query Q4 is more general, in that it refers to all of the products this company has. The response is, again, an enumeration of product types, and should look similiar to the answer to query Q1. Query Q5 is extremely general, but the response still enables user 116 to provide a new query the will advance closer to the widget information sought. Finally, query 6 is irrelevant to the current conversation. The response shows that BOT 100 is confused, but also attempts to help user 116 by reminding user 116 about the previous topic of conversation.

Domain Family Relationships and Topic Types

A function stub may return 0 or more different object values. Similarly, there is variety in the types of default responses BOT 100 can make. Unlike the function stub, the robot can only say one thing at a time. Thus there is a need to have more than one type of default associated with each domain. The number and type of domain defaults is based on the conceptual/informational relationships a domain has in the domain hierarchy.

Figure 8 illustrates the family relationships for the "Products" 506 domain. There are three kinds of relationships with respect to a domain: (1) it has a single parent 802, (2) it has perhaps many siblings 804, and (3) it has perhaps many children 808. A domain's parent represents the more abstract domain of which domain 806 is a component. The parent 802 to domain 806 is represented in Figure 8 with "Company" 502. A domain's siblings share the same parent, and hence the same level of abstraction, with the domain in the hierarchy. A sibling 804 to domain 806 is represented in Figure 8 with "Information" 504. The connection between domain 806, its parent 802, and its child 808 is shown with a solid line owing to the direct relationships between them. The connection between domain 806 and sibling 804 is shown as a dotted line because there is no direct connection between the two. Their relationship exists because they are both components of parent 802. The other sibling domains would be "Services" 508 and "Sales" 510. A domain's children constitute the information categories that comprise the domain, as components. A child 808 of domain 806 is represented in Figure 8 with "Flingys" 514. The other children of domain 806 are "Widgets" 512 and "Gadgets" 516.

Domain family relationships are important in BOT 100 because they are directly associated with types of queries and responses that can be made on a domain. There are three query/response types directly associated with family relationships. A child query/response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on domain 806's subcomponents. Queries Q2 and Q4 represent child queries, because the request is for enumeration of domain 806's children. A sibling query/ response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on the domain 806's definition or relationship to its siblings. Query Q3 represents a sibling query since it only mentions the domain. The response talks about what the domain ("Widgets" 512) means and how it relates

to parent 802 with respect to its siblings. A parent query/response is associated with queries that are irrelevant to the previous query domain 806. Typically, when a query hits a domain topic, the focus of attention is set on that domain. When user 116 makes query Q6 after having made query Q5, the focus is set on the domain topic that provided the Q5 response A5, namely "Products" 506. Since query Q6 is irrelevant to "Products" 506, the response informs user 116 that BOT 100 is confused. In addition, the response reminds user 116 what the last domain was, and its parent ("Company" 502, parent 802), so as to help the user clarify the next query. Since the family query/response types provide information about the location of domain 806 in domain hierarchy 500, they can be used as a navigational aid for assisting user 116 in clarifying requests to BOT 100. This mechanism thus satisfies one of the requirements of BOT 100 design.

Figure 9 illustrates that family components and family-related query/answer types are implemented with three independent domain defaults. The child default 902 focuses on domain children. The sibling default 904 focuses on the domain siblings. The parent default 904 focuses on the domain parent. Together, the child, sibling, and parent defaults comprise the domain defaults. Domain defaults are typically implemented as standard topics, and so in bot development they are called domain topics.

Domain topics (i.e., defaults) have an order of precedence based on the desire to ‘move’ user 116 closer to a domain terminal, in which a standard topic can be used to fulfill their needs. Child 902 and sibling 904 topics respond to relevent (i.e., in universe of discourse 300) utterances, so they have a higher precedence than parent topics 906, which respond to irrelevant utterances. Child topics 902 preferred over sibling topics 904 because they point downward, by talking about domain children 808. These are most likely to be asked and most likely to provide user 116 with appropriate guidance. Sibling topics 904 are the next most preferred, because they are still relevant, and they talk about domain siblings 804. Parent topics 906 are the least preferred, and point to the parent domain 802. As will be disclosed below, the typical implementation mechanism will cause the domain topics to be executed in a way that maintains this precedence ordering.

III BOT IMPLEMENTATION

The operating environment that hosts the current embodiment of the present invention uses Neuroscript to implement BOT programs 108 and Neuroserver to implement BOT processor 100. It will appreciated by those skilled in the art that implementations of the current invention need not be made using Neuroscript or Neuroserver, and that other computational mechanisms could be employed. Domain hierarchy 500 is implemented with a file structure that mirrors the domains of hierarchy 500. This is typically done for ease of organization and maintanence purposes. Every domain in the hierarchy represents a directory by the same name in the file system. Both default and non-default topics are implemented in files. Domain (i.e., default) topics are implemented in a file in the domain-name directory. Non-default topics are implemented in a file in domain terminals. Thus domain terminals have two files, one each for default and non-default topics. Domain (i.e., default) topics are implemented with Neuroscript using standard topics 220. Standard topics are also used to implement non-default responses. As a result, computational mechanisms are required to distinguish and select between the two topic types. Three such mechanisms, focus of attention, specificity, and recency, have already been described. Terminal topics will be more specific than any domain topics. Domains that are deeper in the hierarchy will be more specific than domains higher in the hierarchy. In addition, as will be disclosed below, child topics will be more specific than sibling topics, which will be more specific than parent topics. These mechanisms thus work in concert to satisfy the requirements of BOT development. Below is a description of the implementations of the child 902, sibling 904, and parent 906

domain topics, followed by a description of implementation mechanisms used to maintain domain hierarchies.

A. CHILD TOPICS

Child topics 902 are triggered by description or fact based queries to BOT 100. For example, if user 116 makes query Q7, BOT 100 recognizes it as a description question, in that it basically asks for a description of what is available. Description questions can generally be answered with an enumeration. If user 116 makes query Q8, BOT 100 recognizes it as a factual question. Factual questions can generally be answered yes or no, but the response to query Q8 must be used for both query types, so the yes answer is implicit rather than explicit.

Q7 - "What educational programs do you have?"

A7 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

Q8 - "Do you have an employee reimbursement programs?"

A8 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

In Neuroscript, fact and description question types are parsed using ?FactQuestion and ?DescriptionQuestion, respectively. In queries Q7 and Q8 the child topic 902 responds. In both cases, the question is unspecific, because user 116 does't actually ask for information on a specific program, so a terminal standard topic cannot respond. But BOT 100 can help user 116 out, by elaborating what kinds of educational programs the company has. User 116 can now clarify their request, and BOT 100 doesn't look stupid. The following is an example illustrating the Neuroscript for an "Education" child topic that would respond to queries such as Q7 and Q8:

```
Topic "Random description or fact question about Education" is
Subjects "Education";
    If (?DescriptionQuestion contains DOM_EDUCATION) or
       (?FactQuestion contains DOM_EDUCATION)
    Then
        SayToConsole "Trace -- Education, A answer";
        Example "what kind of ducks swim in Education pool?";
        Say "Talk about Education and, in particular, " +
            "with respect to the children: Science Engineering Business";
    Done
EndTopic
```

The specificity of the topic is based on the combined specificity of ?DescriptionQuestion and ?FactQuestion. Note that the subject of the topic, "Education", is the same as the domain 806 name. The pattern list (as described in the background section), DOM_EDUCATION, is initially implemented with a single element by the name of domain 806. During BOT 100 development, DOM_EDUCATION is extended to include synonyms for education, such as "training." Domain pattern lists must be carefully implemented so that domains don't clash, since clashes reduce the ability of BOT 100 to answer queries

with the correct domain topic.

B. SIBLING TOPICS

Sibling topics 904 are triggered by direct reference based queries to BOT 100. For example, if user 116 makes a query such as Q9 or Q10, BOT 100 recognizes it as a direct reference to domain 806 and responds in two ways: (1) pseudo definition, and (2) information about siblings, as exemplified by the response to queries Q9 and Q10.

Q9 - "Education"

A9 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

Q10 - "Does employee education play a role in advancement at your company?"

A10 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

The idea of the sibling topic 904 is that user 116 has mentioned domain 806 by name. In some cases, as in query Q9, perhaps only the domain name is provided in the query. In other cases, such as in query Q10, the query may be complex, but the only thing recognizable by BOT 100 is the direct reference. BOT 100 cannot be expected to direct user 116 downward, but what it can do is provide a little information about domain 806, and to provide user 116 with some information about how domain 806 fits into the parent domain 802 by talking about its sibling domains 804. The response to query Q9 indicates that, if user 116 is confused, providing some information about how education fits into the company's benefits programs might help them to restate their query with greater clarity. Also notice, in the response to query Q10, that a direct reference can lead to inappropriate responses.

```
Topic "User mentions Education, by itself" is
Subjects "Education";
    If Heard DOM_EDUCATION
    then
        Example "Education";
        SayToConsole "trace -- Education, B answer";
        Say "We have educational support for employees wishing to, " +
            "further their professional growth in degree-granting "  +
            "programs. We also have training programs that lead to " +
            "certification rather than degree objectives.";
    Done
EndTopic
```

Once again, it is clear from the Neuroscript of the sibling topic 904 that the domain 806 name is the subject of the topic. In Neuroscript, direct references are parsed using the "If Heard" mechanism. In the sibling topic, the "If Heard" mechanism is applied to the domain pattern list, DOM_EDUCATION. "If Heard" is also less specific than either "?DescriptionQuestion" or "?FactQuestion", so the sibling topic is less likely to be selected than the child topic, but, being more general, it is more likely to match a relevant utterance.

C. PARENT TOPICS

Parent topics 906 are triggered by queries to BOT 100 that have nothing to do with the current focus of attention. For example, if user 116 previously made a query about education, then education is currently the focus of attention. If user 116 then makes a query such as Q11, BOT 100 recognizes that the subject of the query isn't education, and so is irrelevant. When the query is irrelevant, the parent topic 906 responds, as shown in the response to query Q11 below:

Q11 - "Can I paraglide off the cliff in your back yard?"

A11 - "I am confused, what you have said is either too complicated for me to understand, or I cannot see the relationship to our last topic, which was Education or Services."

There are any number of possibilities of why user 116's current query is unrecognizable, and BOT 100 simply isn't smart enough to decide what to do. As such, parent topic 906 tells user 116 that it doesn't understand what the user said, and reminds user 116 that the last topic of conversation was about current domain 806, but focuses the discussion on the parent domain 802.

```
Topic "We are baffled, but the last topic was Education" is
Subjects "Education";
    If Focused
    Then
        SayToConsole "Trace -- Education, C response";
        When Focused Example "Do you go sledding on winter mornings?";
        Say "I am confused, what you have said is either too complicated " +
            "for me to understand, or I cannot see the relationship to "   +
            "our last topic, which was Education or Services.";
        Focus Subjects "Services";
    Done
EndTopic
```

In this final example, it is seen that domain 806 name is again the topic subject, though it is no longer associated with the test or the example. Domain 806 is again referenced in the response, along with the parent domain 802, but the parent domain 802 is added to the focus list. In cases of non-relevance, such as query Q11 above, it is important to show user 116 that BOT 100 isn't giving up on them, and so the "backing up" response is viable.

Domain topics perform the same task for every domain 806 in domain hierarchy 500, so the structure of the each of the three domain topics is identical except that the domain 806 name changes from topic to topic. In the current implementation of the art, an iterative process is used for creating and maintaining domain topics.

D. DOMAIN INTERACTIONS

The connectivity in a domain hierarchy 500 results in inter-domain interactions which affect domain topic content. Two such interactions are important to note: (1) changes to the number of domains 806 in hierarchy 500, and (2) interactions between the domain 806 names across major branches of hierarchy 500.

Domain Hierarchy Content Changes

The structure of domain hierarchy 500 imposes requirements on the content of domain topics when family members change. The normal development of BOT 100 sees three types of change: (1) add domain 806 to hierarchy 500, (2) remove domain 806 from hierarchy 500, and (3) cauterize domain 806 in hierarchy 500. In the first two cases, the number of components under parent 802 changes, and this changes the child topic 902 and sibling topic 904 say statements. For example, in Figure 5, if two new domains are added under "Services" 508, called "Education" and "Training", then the child topic 902 for Services domain 508 must be edited to include the "Education" and "Training" domains. In addition, the "Education" domain sibling topic 904 has to be edited to include the "Training" domain, and vice versa.

Domain Cauterization

Two scenarios exist wherein domain hierarchy 500 is unable to provide accurate support in universe of discourse 300. The first scenario is when BOT administrator 122 chooses to restrict the universe of discourse with respect to a more generic domain hierarchy. For example, in hierarchy 500, BOT administrator 122 may choose to disable the domain defaults for the "Services" 508 domain without changing the structure of hierarchy 500. The second scenario is when BOT administrator 122 chooses not to restrict the universe of discourse, but the topic development for a domain subtree (all domains below a particular domain) is incomplete. Using the same example above, during BOT 100 testing, perhaps some of the non-default topics under "Services" 508 are incomplete, so rather than have BOT 100 respond incoherently, BOT administrator 122 again disables the subtree.

Figure 10 depicts a "cauterization", which is the operation performed in both scenarios. A cauterization recursively replaces the domain topic say statements with a single statement that refers back to the root domain in the cauterization. In this example, the cauterization subtree is shown as a shadowed triangle.

Figure 11 shows that each domain topic inside the shadowed triangle refers back to the "Products" domain. "Products" domain 506 is being cauterized, so it is called the "cauterization root" domain. The cauterization say statement is basically the same for each topic type inside the triangle, and will be illustrated with a child topic 902 for the "Widgets" 512 domain:

```
Topic "Random description or fact question about Widgets" is
Subjects "Widgets";
    If (?DescriptionQuestion contains DOM_WIDGETS) or
       (?FactQuestion contains DOM_WIDGETS)
    Then
```

```
        SayToConsole "Trace -- Widgets, A answer";
        Example "what kind of ducks swim in Widgets pool?";
        Say "I'm not trained to talk about Products at this " +
            "time, sorry.";
    Done
EndTopic
```

Regardless of which domain topic in the cauterization root's subtree matches user 116's query, the response should be the same. They all point back to "Products" 506.

Figure 12 depicts domain 806's family members that are affected by a cauterization. When one or more of many child domains 806 under a parent 802 is cauterized, then the parent topic 906 must be modified to remove the domain names from the child topic 902, and to remove the domain name as a sibling in the sibling topic 904. In the current implementation of the art, domain cauterization can automatically be performed, reversed, and reperformed using a different cauterization root domain, over an over again, without adversely affecting the coherence and continuity of the hierarchy.

Domain Tiebreakers

There is a certain degree of redundancy in any well-designed domain hierarchy. Many instances can arise in the development of hierarchy 500 where domain topic subjects and pattern lists in different hierarchy branches "clash," meaning that they will contain one or more of the same values. In Neuroscript, only one topic can respond at a time, so only one of perhaps many clashing topics will have its say statement executed. In cases where domain clashes are identified, BOT 100 should be designed to implement what is termed in the art as a "tiebreaker." The tiebreaker redirects control to a sequence topic 240. The sequence topic directs user 116 to select from the different subjects possible, the content for which comes from the say statements of the affected topics.

Figure 13 illustrates an example tiebreaker scenario between two domains relating to "Sales," one of which is a component of the "Information" subtree, while the other is a component in the "Sales" subtree. In each case the meaning of "Sales" is slightly different, but BOT 100 would match on the word "sales", and so "sales" would be a member of each of the pattern lists for topics in these domains. The sequence topic informs user 116 that BOT 100 knows about "Sales" in a number of contexts, and asks user 116 to select one and continue. This way, it becomes clear that BOT 100 recognizes that user 116 is making a general query but perhaps isn't aware that the query can be responded to in different ways. The sequence topic is intended to help user 116 obtain an answer to the query without making assumptions as to which of the domains the query references. The say statements in the tiebreaker are typically copied from the child and sibling domain topics (902 and 904) directly, though they need not be. The following Neuroscript shows how the domain tiebreaker for a child topic is implemented.

```
Topic "Tiebreaker A for PatternList SALES" is
Subjects "Information Sales", "Company Sales";
    If Heard "sale#"
```

```
        Then
            SwitchTo "Sequence Tiebreaker A for PatternList Sales";
    Done
EndTopic

Sequence Topic "Sequence Tiebreaker A for PatternList Sales" is
    Always
        SayToConsole "Trace -- Sales Tiebreaker, A sequence answer";
        Say "I know about <I>Information</I> Sales and <I>Company</I> " +
            "Sales. Which would you like to know more about?";

    WaitForResponse;
        If Heard DOM_INFORMATION
            Then
               SayToConsole "Trace -- Sales Tiebreaker, A information answer";
             Say "I can tell you about what our overall sales were, what our " +
                 "sales per product type were, and what our sales for "       +
                    "particular products were.";
                Focus Subjects "FINANCIALSALES";
            Done
        If Heard OUTREACH
            Then
                SayToConsole "Trace -- Sales Tiebreaker, A outreach answer";
              Say "I can tell you about things like how our sales department " +
                    "works, and what they have in mind for the future.";
                Focus Subjects "OUTREACHSALES";
            Done
        Otherwise Always
           SayToConsole "Trace -- Sales Tiebreaker, A otherwise always answer";
            Say "I thought we were talking about sales, but anyway...";
            InterruptSequence;
        Done
    SwitchBack
EndTopic
```

The first thing to notice in the tiebreaker is that it always has multiple subjects, whereas in other topics it is possible, but not necessary, to have multiple subjects. Second, the recognition mechanism for the tiebreaker is an "If Heard" mechanism. If executed, control is switched to the sequence topic. The sequence topic requests user 116 to select between the two clashing subjects and waits for user 116's response. If user 116 selects one of the pattern lists, then the response is taken from the associated domain's topics, and that domain's subject is focuses. If neither of the pattern lists is heard, then a tiebreaker default is issued as a response and the sequence is terminated.

IV. THE PROCESS OF IMPLEMENTING A HIERARCHY-BASED BOT

1. Create a hierarchical description of the organization being modeled
2. Create a file system architecture that mirrors the domain hierarchy
3. Create domain topics for the hierarchy

4. Create domain pattern lists for the hierarchy
5. Perform any necessary cauterizations
6. Create any necessary tiebreakers
7. Proceed with standard topic development in domain terminals
8. Develop standard topic pattern lists
9. Cauterize/uncauterize as necessary

CLAIMS

1.

2.

3.

4.

5.

ABSTRACT 57

BACKGROUND OF THE ART

Understanding and processing natural language, as either spoken or written by humans, has long been a goal in the field of artificial intelligence. As computers have been programmed of late to perform awe-inspiring feats, such as defeating the world's best human chess master in his game, other skills exhibited by humans are still seemingly beyond the reach of even the most powerful computers. Although a small child may not be able to play chess, that child typically has the facility to process and understand its native tongue. Computers, on the other hand, have yet to exhibit any significant level of mastery in the realm of natural language processing.

One attempt at simulating natural language skills is the virtual robot, or BOT. A BOT may use a scripting language that matches input sentences from a user against input templates of keywords. An input template might, for example, take a group of related keywords and lump them together for the purposes of responding. Thus, words like "father, mother, brother, sister" might be grouped together for a response that relied on the concept of "family". In addition to recognizing familiar words, a scripting language capable of recognizing the ways these words are used in the sentence, and of tracking context across sentences, enables an associated processing model to track and respond to a wide variety of utterances. Generally, the process model that makes use of a scripting language will have a default response if none of the keyword templates matches the input sentence. Thus the BOT always has a response.

Script programs may be written by human designers having little or no formal programming experience. For the purposes of the discussion, some characteristics of a scripting language are described in Backus Normal form below. There are 4 characteristics of a scripting language of interest in this discussion: (1) topics, (2) subjects, (3) conditions, and (4) pattern lists. A topic is used to process user statements.

```
Topic <string> is
   <Tstatement>*
EndTopic
```

There are three types of topics for the purposes of this discussion: standard, default, and sequence. Standard topics are used to recognize and respond to utterances. If more than one standard topic matches an utterance, then computational mechanisms can be used to prioritize and then select one to apply in constructing a response. Two such mechanisms are specificity, which is based on the information content in the utterance, and recency, which is based on which subject most recently uttered is associated with a topic. Because standard topics are used to respond to utterances, they are executed first. Default topics respond when standard topics cannot respond. There are two types of default topics. One default topic type is associated with a standard topic, so it is related to the subject of the standard topic. Another default topic type is a universal default, sometimes called a "last line of defense", which will respond in the event that no other standard or default topic can respond. Default topics are tested in the order in which they appear in the program and the first applicable default is used to build the response. Sequence topics are used to recognize and respond to utterance sequences when the utterances are connected by how the user responds. Sequence topics are executed only when explicitly accessed in a SwitchTo statement, which is a form of redirection. Sequence topics have the lowest priority of the three topic types described.

The body of each topic is a list of conditional blocks. These conditional blocks are executed in the order found in the topic. If the condition of a conditional block is false, execution goes on to the next conditional block in the topic, or to the next topic if there are no further conditional blocks. If the condition is true, the commands and conditional blocks inside the block are executed, and further behavior of the program is dependent on the keyword which ends the conditional block. If it ends with Done, execution ceases until the next input occurs. If it ends with Continue, execution continues with the next conditional block in the topic, or the next topic if there are no further conditional blocks. If it ends with NextTopic, the rest of the current topic is skipped and execution continues with the next topic.

```
<SubjectList> = Subjects <string> [,<string>]*;
```

The top level of a topic may contain one or more Subjects statements. Each asserts that the given subjects are subjects of the topic. If a non-IF command within the body of the topic is executed, all topics which share at least one Subject with the topic are brought to the front of the "focus of attention." Focus of attention generally refers those things a person is currently thinking of. In this context, topics sharing a subject match part of what the user uttered and are used as the first sorting mechanism for topic selection.

```
<Condition> = If <conditionpatlist> Then |
              IfHeard <patlist> Then |
              IfHeard <pat> [and <pat>]* [and not <pat>]* |
              IfRecall <memlist> Then |
              IfRecall <memref> [and <memref>]* [and not <memref>]* |
              IfDontRecall <memlist> Then |
              IfDontRecall <memref> [and <memref>]* |
              IfChance <chance> Then |
              IfChance Then |
              Always

<patlist>  = <pat> [, <pat>]*  | <symbol>
```

A pattern list is anything that evaluates to a list of strings. It can be either the name of a PatternList object or a list of patterns separated by commas.

A virtual robot generally embodies a particular universe of discourse reflective of the subject matter of interest -- e.g. a BOT developed to converse about personal computers should "know" something about computers and their peripherals. The development of such a BOT employs the scripting language to recognize aspects of the subject matter and respond with appropriate content. Often these "script programs" (or scripts) are written in an action-response type style wherein the actual language supplied by the user embodies an "action" to which the "response" is written into the script program itself.

Scripts are generally written by a "BOT administrator" (human or otherwise) by defining a list of “categories” in which the BOT will be well conversant. Categories may comprise "topics" that are recognizable by a runtime executive. Topics, in turn, may comprise patterns or words that are matched against the stream of input communication (in either spoken or written or any other suitable form of communication) from the user.

The main drawback with constructing virtual BOTs by a list of categories is that the topics developed cannot provide complete coverage of all subjects in the universe of discourse. The result is that the BOT responds with the universal default. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with the default. A related drawback is that the universal default response generally provides insufficient guidance to the user as to their original input: it doesn't provide information regarding why the input "confused" the BOT, and it doesn't provide a knowledgeable response to the input.

The BOT development task must recognize that the level of quality and value evidenced by users is not judged merely in discrete terms, but rather by the overall impression that they get from their interaction with the BOT and by their level of satisfaction with the information the BOT provides.

Thus, there is a need in the art to have a means for easily designing and creating virtual BOTs that enables the BOT to effectively respond to arbitrary utterances with knowledge regardless of the number of topics implemented, guides the user toward providing utterances that will move the user closer to the information they seek, provides the user with information about what in the user's utterance confused the bot when that occurs, and performs these tasks within a framework the eases the maintanence and entendability of the BOT's capabilities.

SUMMARY OF THE INVENTION

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a suitable operating environment for the purposes of the present invention.

Figure 2 depicts the topic types used to present a typical implementation of the present invention.

Figure 3 depicts developed topics in a universe of discourse.

Figure 4 depicts a procedural decomposition of a program into its computational subtasks.

Figure 5 depicts a collection of conceptual domains which describe a universe of discourse.

Figure 6 depicts a uniform distribution of defaults in a universe of discourse.

Figure 7 depicts a set of developed topics in a uniform distribution of defaults.

Figure 8 expands the family view of a conceptual domain as shown in Figure 5.

Figure 9 depicts domain topics for the conceptual domain as found in Figure 8.

Figure 10 depicts the cauterization problem.

Figure 11 depicts the cauterization solution for a domain's children.

Figure 12 depicts the cauterization solution for a domain's parent and siblings.

Figure 13 depicts the tiebreaker problem.

DETAILED DESCRIPTION OF THE INVENTION

I. OVERVIEW AND GENERAL ARCHITECTURE

The term "robot" is used interchangably with "BOT" throughout the remainder of this writeup. For the purposes of this writeup, both "BOT" and "robot" refer to any program which interacts with a human user in some fashion, and should not be assumed to refer only to physically embodied robots. The term "supervisor" is used interchangably with "administrator". The term "domain" is used to represent a body of knowledge that comprises a component of a universe of discourse that the "BOT" will be conversant with. The term "hierarchy" will be used to describe the collection of domains that represents the universe of discourse that the "BOT" will be designed to converse about. It should be noted that the term hierarchy should not be construed to mean the collection data type. Many means of representing the collection are known to those skilled in the art, and the functionality of the collection is not dependent on the data type used to implement it.

Referring now to Figure 1, the operating environment in which the present invention applies is depicted. The environment can be characterized generally into three partitions: front end 102; BOT processor 100; and back end 104. Front end 102 is generally the environment in which a human user 116 consults a virtual BOT interface 114 via a computer 112 that may be connected to the BOT processor via a communications link, such as through a server connected to the Internet or alternatively directly connected to BOT processor 100. It will be appreciated that many other means of connection to a BOT processor 100 are well known to those skilled in the art and that the present invention should not be limited to any particular aspects of the general operating environment as disclosed herein.

Typically, human user 116 connects to a site whose interface of first impression is a virtual BOT interface 114. The advantage for the site developer is that human user 116 may have a help or information

request that is easily handled via BOT interface 114. Today, it is not uncommon to find sites having a list of FAQs ("Frequently Asked Questions") that serve this purpose of handling very low level user concerns and questions. However, for more advanced questions or interactions with the site, virtual BOTs will become increasing popular.

In the operating environment that hosts the embodiment of the present invention, BOT interface 114 is an instantiation of a process that is spawned by BOT processor 100 via connection 110. BOT processor 100 itself may comprise connection 110; runtime executive process 106, compiler 107, and a set of BOT programs 108. As users 116 log onto a site having BOT processor 100 via connection 110, runtime executive 106 executes an interaction routine that guides the discussion that occurs between user 116 and BOT processor 100. Typically, a two way communications dialogue occurs between user 116 and BOT processor 100 wherein user 116 may ask questions, make declarative statements and other normal communications patterns that humans typify. For the purposes of the present invention, “communications” is to be very broadly interpreted. Indeed, suitable communications could be in the form of written or spoken language, graphics, URL’s or the like that may be passed to and from a user to an automatic interface program, such as the present invention.

In turn, runtime executive 106 parses the statements and questions generated by the user and responds according to a set of BOT programs 108. As will be discussed in greater detail, BOT programs 108 are typically created at the back end 104 as a set of "scripts" that the BOT processor will tend to engage in with user 116. For example, if the site using BOT processor 100 is a site for a reseller of personal computers, then BOT processor 100 should be designed to handle questions and discussions concerning personal computers and their peripherals in general. Thus, the back end 104 will generate scripts that will guide the discussion concerning many computer-related topics. These script programs 108 are then compiled by compiler 107 and the compiled code is incorporated into runtime executive 106.

As the two-way discussions between user 116 and runtime executive 106 continue, it is generally desirable to engage in quality control of BOT processor 100. This quality control is provided at back end 104 via feedback loop comprising a transcript of dialogues 118 and backtrace and state information 120 of the BOT processor 100; a supervisor 122 and editor 124. As transcripts develop over the course of interacting with a user, the text of these transcripts are stored, together with the state of the runtime executive and backtrace of execution through the runtime executive code. This information forms the basis for accurately diagnosing the runtime executive and for debugging its performance. Such information may be stored electronically in a storage media or could be printed out in human readable form.

Supervisor 122 analyzes the information at 118 and 120 with an eye towards optimizing the performance of the runtime executive. Typically, supervisor 122 could be another human, deciding if the semantics captured by the system needs to be upgraded in response to a dialog transcript that has occurred. If so, supervisor 122 could optionally invoke an editor 124 to edit the programs that represent the semantic framework of the runtime executive. These programs would then be re-compiled and incorporated into the runtime executive.

Although Figure 1 gives a general description of various operating environments in which virtual BOTs may exist, it will be appreciated that many other operating environments are obvious to those skilled in the art and that the scope of the present invention should not be so limited to the exemplary descriptions as given

above.

II. BOT DEVELOPMENT

In general, BOT user 116 knows what kind of information he seeks, but not necessarily how to articulate his request. Functionally, the implementation of the art described herein provides user 116 with the ability to interact with BOT 100 on any level of abstraction directly associated with the universe of discourse. BOT 100 should be able to achieve this either with a single response to a very specific question that immediately identifies the users' needs, or through dialog in response to a series of increasingly specific queries guided by BOT 100 through the BOT programs 108. The mechanism is independent of the order in which topics are developed.

A. CURRENT MECHANISM

As mentioned in the background, BOT programs 108, or scripts, are generally written by BOT administrator 122 by defining a list of categories in which BOT 100 will be well conversant. These categories generally come from BOT administrator 122, FAQs, user input, and other sources associated with the universe of discourse. BOTs developed this way suffer from two problems. First, BOT 100's ability to answer questions is dependent on the number of topics developed, and the finite number of topics developed by BOT administrator 122 cannot completely cover all aspects of the organization associated with the universe of discourse. Thus, it is highly likely that queries made by BOT user 116 will result in "default" responses. Second, the development of BOT 100 follows specific needs/interests of administrator 122, rather than a systematic, coherant approach. Thus there is no coherence between the topics or their defaults.

Figure 3 illustrates the problem that arises when BOT programs incompletely cover the universe of discourse. Universe of discourse 300 is comprised of those topics 302 that have been developed. Each topic 302 potentially has its own default 304. The content areas remaining when all topic-related content areas are covered are responded to by universal default 306, which will be "hit" by all off-topic user queries. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with a default. Since the defaults 304 are related to the topics, rather than to each other, or to the universe of discourse, hitting defaults 304 or 306 provides the user with little information of value that would assist in continuing the conversation. There is thus an inherant qualitative problem in developing BOTS from a list of topics. Since BOT 100 is a conversational agent, its value derives entirely from how well it interacts with users 116, where the word "well" may be defined in terms of knowlege content BOT 100 conveys, its friendliness, how easily it is "confused", and how much interaction is required for user 116 to find what is sought. If user 116 asks questions that confuse BOT 100, then BOT 100 is seen as "stupid" and its value is diminished.

The embodiment of the current invention implements BOT 100 by designing a framework of defaults in such a way that it is impossible for BOT 100 to be asked a question that cannot be answered. The structural foundation for this framework, and the interacting mechanisms associated with it, comprise the art described in the remaining sections.

B. APPLICABLE SOFTWARE DESIGN/DEVELOPMENT TOOLS

BOT programs 108 can be thought of as a program in the traditional computer engineering framework. Each can be decomposed to increasingly specific components.

Figure 4 illustrates a typical procedural decomposition, because in both the result can be decomposed to increasingly specific tasks. In the traditional framework, program 402 is decomposed into constituent functional components, or tasks. The "Initialize Objects" 404, "Input Data" 406, "Do Calculations" 408, and "Output Data" 410 tasks are independent functional tasks that, together, comprise program 402. The functional components interact through data objects. Each functional component (404-410) is itself decomposed to those functional tasks that comprise it. For example, the "Do Calculations" 408 task is decomposed to "Perform Analysis" 412 and "Continue Test" 414 subtasks. The decomposition continues to a point where the designer/developer is satisfied that source code can be developed. The tasks at this level are called "terminal" tasks, because they aren't further decomposed. In the software development task, intermediate tasks, such as "Do Calculations" 408, are used to organize the functions of their subcomponents, and their effects are otherwise not seen. Terminal tasks implement the actual functionality of program 402.

Once a procedural decomposition is designed, a program can be implemented without developing any of the terminal functionality. A problem that often arises in software design and development is the tendency for developers to develop the terminal functionality without first implementing the supporting framework depicted by program structure 400. This problem is analogous to developing BOT topics without a supporting content framework. The difference is that the BOT can still function, though its function is severely hampered, whereas a program cannot. Software engineering has a tool, called a function stub, that enables the developer to implement the framework of program 402 without implementing the terminal functionality. The stub is a structure that adheres to the input/output requirements of program 402's design, but implements none of the functionality. For intermediate tasks, stubs comprise the appropriate function calls to subcomponent tasks. Using this approach, the software developer can implement a large program without implementing any of the functionality, and then replace the empty stubs with the actual code that implements the design algorithms for a particular task.

The art described herein conceptually makes use of the hierarchical/procedural decomposition and function stub to develop a BOT that has an organizational structure and is developed around a hierarchical framework of defaults. The analog to program 402's functional component is called a "domain". The analog to program 402's functional decomposition is a content decomposition called a domain "hierarchy." The analog to the function stub is the "domain default." Regardless of the user's query, or the degree of topic development, the BOT developed with a domain hierarchy and domain defaults can answer the user and even direct the user toward what they seek.

C. HIERARCHICAL DECOMPOSITION OF THE UNIVERSE OF DISCOURSE

Every universe of discourse can be described as a grouping of different content components that can be

hierarchically decomposed. Using a virtual robot to provide information about a particular universe of discourse, one must understand how that universe is decomposed and design the robot around the associated hierarchy.

Figure 5 depicts a domain hierarchy for "Company" 502. The decomposition represents the sum total of all information about the company, which is decomposed into four informational and functional components, "Information" 504, "Products" 506, "Services" 508, and "Sales" 510. The "Products" component 506 is shown further decomposed into three components, "Widgets" 512, "Flingys" 514, and "Gadgets" 516. The pictured decomposition is typical, but unimportant. What is important is that the combination of components completely describes the organization, analogous to the relationship of the functional components (404-412) and program 402 in Figure 4.

Each component 502-516 in hierarchy 500 is called a "domain." A domain represents an informational aspect of the organization. This type of hierarchy is called a "component" hierarchy, in that the domains under the top, or "root," domain (i.e., domain 502) are considered components of domain 502. Thus, “Information” 504 is a component of “Company” 502. The reverse is not true (i.e., “Company" 502 is not a component of “Information” 504). A terminal domain is one which has no subdomains. Just as a program’s functionality is implemented in its terminal components, BOT 100s specific content is implemented in its terminal domains. The overall hierarchical decomposition 500 is called a domain "hierarchy."

D. DOMAIN TOPICS

The domain analog to the function stub is a mechanism that allows BOT 100 to respond in lieu of topic development associated with a terminal domain. The domain stub is effectively a default response to a query. It is thus referred to as a domain default.

Figure 6 illustrates the universe of discourse 300 for BOT 100 when it is developed using a domain hierarchy and domain defaults (i.e., stubs), as opposed to the largely vacant universe of discourse 300 depicted in Figure 3. The universe of discourse 300 is now completely divided into non-overlapping areas 602 that represent the domains in hierarchy 500. Two types of queries can be made to BOT 100 in lieu of terminal topic development: (1) relevant and unspecific, or (2) irrelevant. A "relevant" query is one which hits a domain in the universe of discourse (i.e., a domain in the hierarchy). A "specific" query is one that would hit a terminal domain topic. Thus a relevant but unspecific query should hit an intermediate domain in the hierarchy. An "irrelevant" query is one which misses the domain associated with the current focus of attention or, in the worst case, the universe of discourse (i.e., hits no domains in the hierarchy). Despite the fact that no topics are shown in Figure 6, BOT 100 will respond in a coherent manner to relevant but unspecific queries as well as irrelevant queries.

Figure 7 illustrates that topics 302 could be developed for any terminal domain, in any order, once the domain hierarchy and its defaults are in place.

The analogy to a program decomposition and a function stub breaks down with a virtual robot in three

ways: (1) the user/program interaction is different than a user/BOT interaction, (2) the relationship between program components is different than the relationship between hierarchy domains, and (3) the function stub responds differently than the domain default. In a program, a user has very specific points where input is allowed, and very specific points were output is produced, whereas BOT user 116 could possibly interact with BOT 100 at any domain as mentioned above. Moreover, the program user would have no use for output at intermediate points, since all that is necessary is the functional implementation at component terminals. The domains in a virtual bot hierarchy are always related by content, so every domain from root domain 502 to a particular terminal domain can provide a viable degree of content for the terminal domain. As a result, a query directed at "Widgets" 512, if made in a vague way, could be responded to by "Products" 506, or even "Company" 502. Finally, the function stub response may or may not have a value, but the domain default always has a language response.

For example, if the terminal domain is "Widgets" 512, then domain "stubs" are required for each of the "Company" 502, "Products" 506, and "Widgets" 512 domains, because user 116 could make a query regarding "Widgets" 512 that could hit domain defaults at either "Products" 506 or "Company" 502, depending on the degree query specificity. Any of the following queries/comments could be issued by user 116 with respect to widgets:

Q1 - "Do you have any widgets?"
Q2 - "What kind of widgets do you have?"
Q3 - "Widgets"
Q4 - "What products do you have?"
Q5 - "What do you have?"
Q6 - "Do penguins live here?"

The so-called domain "stubs", or "defaults", could respond to these queries without any specific information regarding widgets being developed. Such responses would look like the following:

Q1 - "Do you have any widgets?"
A1 - "Yes, we have a number of widgets, would you like to see a listing, would you like information about a particular type of widget, or would you like to see our inventory?"
Q2 - "What kind of widgets do you have?"
A2 - "We have red, green and blue widgets. Would you like a feature comparison chart or information about a particular model?"
Q3 - "Widgets"
A3 - "Widgets are one of our products. They are used in many industrial applications. We also manufacture and sell flingys and gadgets."
Q4 - "What products do you have?"
A4 - "We manufacture and sell widgets, flingys, and gadgets. Would you like to know more about any of these, or are you interested in our inventory of any of them."
Q5 - "What do you do?"
A5 - "I am glad you asked that. We are a small manufacturing company. We design and build the best widgets, flingys and gadgets money can buy. Would you like more information on a particular product?"
Q6 - "Do penguins live here?"
A6 - "I’m sory, I do not understand what you mean. Weren’t we just talking about our company’s products?

Notice that all of the queries are a bit unspecific, and that there are different types of responses to the different types of queries. In query Q1, it is unclear whether user 116 is seeking information about whether the organization produces widgets, what kind of widgets, or whether user 116 wants to know how many are in stock. The answer must be a combination of what would be at a terminal domain and information obtained from the "Sales" 510 domain. In query Q2, the request is really for an enumeration of widget types, but it is vague because what the user wants to know about widgets is unspecified. The response should provide both the information requested and provide options for viewing that information. Query Q3 only mentions the term "Widgets" itself, and so BOT 100 can only assume that user 116 seeks to know more about widgets, or maybe what role they play in the organization. Query Q4 is more general, in that it refers to all of the products this company has. The response is, again, an enumeration of product types, and should look similiar to the answer to query Q1. Query Q5 is extremely general, but the response still enables user 116 to provide a new query the will advance closer to the widget information sought. Finally, query 6 is irrelevant to the current conversation. The response shows that BOT 100 is confused, but also attempts to help user 116 by reminding user 116 about the previous topic of conversation.

Domain Family Relationships and Topic Types

A function stub may return 0 or more different object values. Similarly, there is variety in the types of default responses BOT 100 can make. Unlike the function stub, the robot can only say one thing at a time. Thus there is a need to have more than one type of default associated with each domain. The number and type of domain defaults is based on the conceptual/informational relationships a domain has in the domain hierarchy.

Figure 8 illustrates the family relationships for the "Products" 506 domain. There are three kinds of relationships with respect to a domain: (1) it has a single parent 802, (2) it has perhaps many siblings 804, and (3) it has perhaps many children 808. A domain's parent represents the more abstract domain of which domain 806 is a component. The parent 802 to domain 806 is represented in Figure 8 with "Company" 502. A domain's siblings share the same parent, and hence the same level of abstraction, with the domain in the hierarchy. A sibling 804 to domain 806 is represented in Figure 8 with "Information" 504. The connection between domain 806, its parent 802, and its child 808 is shown with a solid line owing to the direct relationships between them. The connection between domain 806 and sibling 804 is shown as a dotted line because there is no direct connection between the two. Their relationship exists because they are both components of parent 802. The other sibling domains would be "Services" 508 and "Sales" 510. A domain's children constitute the information categories that comprise the domain, as components. A child 808 of domain 806 is represented in Figure 8 with "Flingys" 514. The other children of domain 806 are "Widgets" 512 and "Gadgets" 516.

Domain family relationships are important in BOT 100 because they are directly associated with types of queries and responses that can be made on a domain. There are three query/response types directly associated with family relationships. A child query/response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on domain 806's subcomponents. Queries Q2 and Q4 represent child queries, because the request is for enumeration of domain 806's children. A sibling query/ response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on the domain 806's definition or relationship to its siblings. Query Q3 represents a sibling query since it only mentions the domain. The response talks about what the domain ("Widgets" 512) means and how it relates

to parent 802 with respect to its siblings. A parent query/response is associated with queries that are irrelevant to the previous query domain 806. Typically, when a query hits a domain topic, the focus of attention is set on that domain. When user 116 makes query Q6 after having made query Q5, the focus is set on the domain topic that provided the Q5 response A5, namely "Products" 506. Since query Q6 is irrelevant to "Products" 506, the response informs user 116 that BOT 100 is confused. In addition, the response reminds user 116 what the last domain was, and its parent ("Company" 502, parent 802), so as to help the user clarify the next query. Since the family query/response types provide information about the location of domain 806 in domain hierarchy 500, they can be used as a navigational aid for assisting user 116 in clarifying requests to BOT 100. This mechanism thus satisfies one of the requirements of BOT 100 design.

Figure 9 illustrates that family components and family-related query/answer types are implemented with three independent domain defaults. The child default 902 focuses on domain children. The sibling default 904 focuses on the domain siblings. The parent default 904 focuses on the domain parent. Together, the child, sibling, and parent defaults comprise the domain defaults. Domain defaults are typically implemented as standard topics, and so in bot development they are called domain topics.

Domain topics (i.e., defaults) have an order of precedence based on the desire to ‘move’ user 116 closer to a domain terminal, in which a standard topic can be used to fulfill their needs. Child 902 and sibling 904 topics respond to relevent (i.e., in universe of discourse 300) utterances, so they have a higher precedence than parent topics 906, which respond to irrelevant utterances. Child topics 902 preferred over sibling topics 904 because they point downward, by talking about domain children 808. These are most likely to be asked and most likely to provide user 116 with appropriate guidance. Sibling topics 904 are the next most preferred, because they are still relevant, and they talk about domain siblings 804. Parent topics 906 are the least preferred, and point to the parent domain 802. As will be disclosed below, the typical implementation mechanism will cause the domain topics to be executed in a way that maintains this precedence ordering.

III BOT IMPLEMENTATION

The operating environment that hosts the current embodiment of the present invention uses Neuroscript to implement BOT programs 108 and Neuroserver to implement BOT processor 100. It will appreciated by those skilled in the art that implementations of the current invention need not be made using Neuroscript or Neuroserver, and that other computational mechanisms could be employed. Domain hierarchy 500 is implemented with a file structure that mirrors the domains of hierarchy 500. This is typically done for ease of organization and maintanence purposes. Every domain in the hierarchy represents a directory by the same name in the file system. Both default and non-default topics are implemented in files. Domain (i.e., default) topics are implemented in a file in the domain-name directory. Non-default topics are implemented in a file in domain terminals. Thus domain terminals have two files, one each for default and non-default topics. Domain (i.e., default) topics are implemented with Neuroscript using standard topics 220. Standard topics are also used to implement non-default responses. As a result, computational mechanisms are required to distinguish and select between the two topic types. Three such mechanisms, focus of attention, specificity, and recency, have already been described. Terminal topics will be more specific than any domain topics. Domains that are deeper in the hierarchy will be more specific than domains higher in the hierarchy. In addition, as will be disclosed below, child topics will be more specific than sibling topics, which will be more specific than parent topics. These mechanisms thus work in concert to satisfy the requirements of BOT development. Below is a description of the implementations of the child 902, sibling 904, and parent 906

domain topics, followed by a description of implementation mechanisms used to maintain domain hierarchies.

A. CHILD TOPICS

Child topics 902 are triggered by description or fact based queries to BOT 100. For example, if user 116 makes query Q7, BOT 100 recognizes it as a description question, in that it basically asks for a description of what is available. Description questions can generally be answered with an enumeration. If user 116 makes query Q8, BOT 100 recognizes it as a factual question. Factual questions can generally be answered yes or no, but the response to query Q8 must be used for both query types, so the yes answer is implicit rather than explicit.

Q7 - "What educational programs do you have?"

A7 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

Q8 - "Do you have an employee reimbursement programs?"

A8 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

In Neuroscript, fact and description question types are parsed using ?FactQuestion and ?DescriptionQuestion, respectively. In queries Q7 and Q8 the child topic 902 responds. In both cases, the question is unspecific, because user 116 does't actually ask for information on a specific program, so a terminal standard topic cannot respond. But BOT 100 can help user 116 out, by elaborating what kinds of educational programs the company has. User 116 can now clarify their request, and BOT 100 doesn't look stupid. The following is an example illustrating the Neuroscript for an "Education" child topic that would respond to queries such as Q7 and Q8:

```
Topic "Random description or fact question about Education" is
Subjects "Education";
    If (?DescriptionQuestion contains DOM_EDUCATION) or
       (?FactQuestion contains DOM_EDUCATION)
    Then
        SayToConsole "Trace -- Education, A answer";
        Example "what kind of ducks swim in Education pool?";
        Say "Talk about Education and, in particular, " +
            "with respect to the children: Science Engineering Business";
    Done
EndTopic
```

The specificity of the topic is based on the combined specificity of ?DescriptionQuestion and ?FactQuestion. Note that the subject of the topic, "Education", is the same as the domain 806 name. The pattern list (as described in the background section), DOM_EDUCATION, is initially implemented with a single element by the name of domain 806. During BOT 100 development, DOM_EDUCATION is extended to include synonyms for education, such as "training." Domain pattern lists must be carefully implemented so that domains don't clash, since clashes reduce the ability of BOT 100 to answer queries

with the correct domain topic.

B. SIBLING TOPICS

Sibling topics 904 are triggered by direct reference based queries to BOT 100. For example, if user 116 makes a query such as Q9 or Q10, BOT 100 recognizes it as a direct reference to domain 806 and responds in two ways: (1) pseudo definition, and (2) information about siblings, as exemplified by the response to queries Q9 and Q10.

Q9 - "Education"

A9 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

Q10 - "Does employee education play a role in advancement at your company?"

A10 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

The idea of the sibling topic 904 is that user 116 has mentioned domain 806 by name. In some cases, as in query Q9, perhaps only the domain name is provided in the query. In other cases, such as in query Q10, the query may be complex, but the only thing recognizable by BOT 100 is the direct reference. BOT 100 cannot be expected to direct user 116 downward, but what it can do is provide a little information about domain 806, and to provide user 116 with some information about how domain 806 fits into the parent domain 802 by talking about its sibling domains 804. The response to query Q9 indicates that, if user 116 is confused, providing some information about how education fits into the company's benefits programs might help them to restate their query with greater clarity. Also notice, in the response to query Q10, that a direct reference can lead to inappropriate responses.

```
Topic "User mentions Education, by itself" is
Subjects "Education";
    If Heard DOM_EDUCATION
    then
        Example "Education";
        SayToConsole "trace -- Education, B answer";
        Say "We have educational support for employees wishing to, " +
            "further their professional growth in degree-granting "  +
            "programs. We also have training programs that lead to " +
            "certification rather than degree objectives.";
    Done
EndTopic
```

Once again, it is clear from the Neuroscript of the sibling topic 904 that the domain 806 name is the subject of the topic. In Neuroscript, direct references are parsed using the "If Heard" mechanism. In the sibling topic, the "If Heard" mechanism is applied to the domain pattern list, DOM_EDUCATION. "If Heard" is also less specific than either "?DescriptionQuestion" or "?FactQuestion", so the sibling topic is less likely to be selected than the child topic, but, being more general, it is more likely to match a relevant utterance.

C. PARENT TOPICS

Parent topics 906 are triggered by queries to BOT 100 that have nothing to do with the current focus of attention. For example, if user 116 previously made a query about education, then education is currently the focus of attention. If user 116 then makes a query such as Q11, BOT 100 recognizes that the subject of the query isn't education, and so is irrelevant. When the query is irrelevant, the parent topic 906 responds, as shown in the response to query Q11 below:

Q11 - "Can I paraglide off the cliff in your back yard?"

A11 - "I am confused, what you have said is either too complicated for me to understand, or I cannot see the relationship to our last topic, which was Education or Services."

There are any number of possibilities of why user 116's current query is unrecognizable, and BOT 100 simply isn't smart enough to decide what to do. As such, parent topic 906 tells user 116 that it doesn't understand what the user said, and reminds user 116 that the last topic of conversation was about current domain 806, but focuses the discussion on the parent domain 802.

```
Topic "We are baffled, but the last topic was Education" is
Subjects "Education";
    If Focused
    Then
        SayToConsole "Trace -- Education, C response";
        When Focused Example "Do you go sledding on winter mornings?";
        Say "I am confused, what you have said is either too complicated " +
            "for me to understand, or I cannot see the relationship to "   +
            "our last topic, which was Education or Services.";
        Focus Subjects "Services";
    Done
EndTopic
```

In this final example, it is seen that domain 806 name is again the topic subject, though it is no longer associated with the test or the example. Domain 806 is again referenced in the response, along with the parent domain 802, but the parent domain 802 is added to the focus list. In cases of non-relevance, such as query Q11 above, it is important to show user 116 that BOT 100 isn't giving up on them, and so the "backing up" response is viable.

Domain topics perform the same task for every domain 806 in domain hierarchy 500, so the structure of the each of the three domain topics is identical except that the domain 806 name changes from topic to topic. In the current implementation of the art, an iterative process is used for creating and maintaining domain topics.

D. DOMAIN INTERACTIONS

The connectivity in a domain hierarchy 500 results in inter-domain interactions which affect domain topic content. Two such interactions are important to note: (1) changes to the number of domains 806 in hierarchy 500, and (2) interactions between the domain 806 names across major branches of hierarchy 500.

Domain Hierarchy Content Changes

The structure of domain hierarchy 500 imposes requirements on the content of domain topics when family members change. The normal development of BOT 100 sees three types of change: (1) add domain 806 to hierarchy 500, (2) remove domain 806 from hierarchy 500, and (3) cauterize domain 806 in hierarchy 500. In the first two cases, the number of components under parent 802 changes, and this changes the child topic 902 and sibling topic 904 say statements. For example, in Figure 5, if two new domains are added under "Services" 508, called "Education" and "Training", then the child topic 902 for Services domain 508 must be edited to include the "Education" and "Training" domains. In addition, the "Education" domain sibling topic 904 has to be edited to include the "Training" domain, and vice versa.

Domain Cauterization

Two scenarios exist wherein domain hierarchy 500 is unable to provide accurate support in universe of discourse 300. The first scenario is when BOT administrator 122 chooses to restrict the universe of discourse with respect to a more generic domain hierarchy. For example, in hierarchy 500, BOT administrator 122 may choose to disable the domain defaults for the "Services" 508 domain without changing the structure of hierarchy 500. The second scenario is when BOT administrator 122 chooses not to restrict the universe of discourse, but the topic development for a domain subtree (all domains below a particular domain) is incomplete. Using the same example above, during BOT 100 testing, perhaps some of the non-default topics under "Services" 508 are incomplete, so rather than have BOT 100 respond incoherently, BOT administrator 122 again disables the subtree.

Figure 10 depicts a "cauterization", which is the operation performed in both scenarios. A cauterization recursively replaces the domain topic say statements with a single statement that refers back to the root domain in the cauterization. In this example, the cauterization subtree is shown as a shadowed triangle.

Figure 11 shows that each domain topic inside the shadowed triangle refers back to the "Products" domain. "Products" domain 506 is being cauterized, so it is called the "cauterization root" domain. The cauterization say statement is basically the same for each topic type inside the triangle, and will be illustrated with a child topic 902 for the "Widgets" 512 domain:

```
Topic "Random description or fact question about Widgets" is
Subjects "Widgets";
    If (?DescriptionQuestion contains DOM_WIDGETS) or
       (?FactQuestion contains DOM_WIDGETS)
    Then
```

```
        SayToConsole "Trace -- Widgets, A answer";
        Example "what kind of ducks swim in Widgets pool?";
        Say "I'm not trained to talk about Products at this " +
            "time, sorry.";
    Done
EndTopic
```

Regardless of which domain topic in the cauterization root's subtree matches user 116's query, the response should be the same. They all point back to "Products" 506.

Figure 12 depicts domain 806's family members that are affected by a cauterization. When one or more of many child domains 806 under a parent 802 is cauterized, then the parent topic 906 must be modified to remove the domain names from the child topic 902, and to remove the domain name as a sibling in the sibling topic 904. In the current implementation of the art, domain cauterization can automatically be performed, reversed, and reperformed using a different cauterization root domain, over an over again, without adversely affecting the coherence and continuity of the hierarchy.

Domain Tiebreakers

There is a certain degree of redundancy in any well-designed domain hierarchy. Many instances can arise in the development of hierarchy 500 where domain topic subjects and pattern lists in different hierarchy branches "clash," meaning that they will contain one or more of the same values. In Neuroscript, only one topic can respond at a time, so only one of perhaps many clashing topics will have its say statement executed. In cases where domain clashes are identified, BOT 100 should be designed to implement what is termed in the art as a "tiebreaker." The tiebreaker redirects control to a sequence topic 240. The sequence topic directs user 116 to select from the different subjects possible, the content for which comes from the say statements of the affected topics.

Figure 13 illustrates an example tiebreaker scenario between two domains relating to "Sales," one of which is a component of the "Information" subtree, while the other is a component in the "Sales" subtree. In each case the meaning of "Sales" is slightly different, but BOT 100 would match on the word "sales", and so "sales" would be a member of each of the pattern lists for topics in these domains. The sequence topic informs user 116 that BOT 100 knows about "Sales" in a number of contexts, and asks user 116 to select one and continue. This way, it becomes clear that BOT 100 recognizes that user 116 is making a general query but perhaps isn't aware that the query can be responded to in different ways. The sequence topic is intended to help user 116 obtain an answer to the query without making assumptions as to which of the domains the query references. The say statements in the tiebreaker are typically copied from the child and sibling domain topics (902 and 904) directly, though they need not be. The following Neuroscript shows how the domain tiebreaker for a child topic is implemented.

```
Topic "Tiebreaker A for PatternList SALES" is
Subjects "Information Sales", "Company Sales";
    If Heard "sale#"
```

```
        Then
            SwitchTo "Sequence Tiebreaker A for PatternList Sales";
    Done
EndTopic

Sequence Topic "Sequence Tiebreaker A for PatternList Sales" is
    Always
        SayToConsole "Trace -- Sales Tiebreaker, A sequence answer";
        Say "I know about <I>Information</I> Sales and <I>Company</I> " +
            "Sales. Which would you like to know more about?";

    WaitForResponse;
        If Heard DOM_INFORMATION
            Then
               SayToConsole "Trace -- Sales Tiebreaker, A information answer";
             Say "I can tell you about what our overall sales were, what our " +
                 "sales per product type were, and what our sales for "       +
                    "particular products were.";
                Focus Subjects "FINANCIALSALES";
            Done
        If Heard OUTREACH
            Then
                SayToConsole "Trace -- Sales Tiebreaker, A outreach answer";
              Say "I can tell you about things like how our sales department " +
                    "works, and what they have in mind for the future.";
                Focus Subjects "OUTREACHSALES";
            Done
        Otherwise Always
           SayToConsole "Trace -- Sales Tiebreaker, A otherwise always answer";
            Say "I thought we were talking about sales, but anyway...";
            InterruptSequence;
        Done
    SwitchBack
EndTopic
```

The first thing to notice in the tiebreaker is that it always has multiple subjects, whereas in other topics it is possible, but not necessary, to have multiple subjects. Second, the recognition mechanism for the tiebreaker is an "If Heard" mechanism. If executed, control is switched to the sequence topic. The sequence topic requests user 116 to select between the two clashing subjects and waits for user 116's response. If user 116 selects one of the pattern lists, then the response is taken from the associated domain's topics, and that domain's subject is focuses. If neither of the pattern lists is heard, then a tiebreaker default is issued as a response and the sequence is terminated.

IV. THE PROCESS OF IMPLEMENTING A HIERARCHY-BASED BOT

1. Create a hierarchical description of the organization being modeled
2. Create a file system architecture that mirrors the domain hierarchy
3. Create domain topics for the hierarchy

4. Create domain pattern lists for the hierarchy
5. Perform any necessary cauterizations
6. Create any necessary tiebreakers
7. Proceed with standard topic development in domain terminals
8. Develop standard topic pattern lists
9. Cauterize/uncauterize as necessary

CLAIMS

1.

2.

3.

4.

5.

ABSTRACT 57

BACKGROUND OF THE ART

Understanding and processing natural language, as either spoken or written by humans, has long been a goal in the field of artificial intelligence. As computers have been programmed of late to perform awe-inspiring feats, such as defeating the world's best human chess master in his game, other skills exhibited by humans are still seemingly beyond the reach of even the most powerful computers. Although a small child may not be able to play chess, that child typically has the facility to process and understand its native tongue. Computers, on the other hand, have yet to exhibit any significant level of mastery in the realm of natural language processing.

One attempt at simulating natural language skills is the virtual robot, or BOT. A BOT may use a scripting language that matches input sentences from a user against input templates of keywords. An input template might, for example, take a group of related keywords and lump them together for the purposes of responding. Thus, words like "father, mother, brother, sister" might be grouped together for a response that relied on the concept of "family". In addition to recognizing familiar words, a scripting language capable of recognizing the ways these words are used in the sentence, and of tracking context across sentences, enables an associated processing model to track and respond to a wide variety of utterances. Generally, the process model that makes use of a scripting language will have a default response if none of the keyword templates matches the input sentence. Thus the BOT always has a response.

Script programs may be written by human designers having little or no formal programming experience. For the purposes of the discussion, some characteristics of a scripting language are described in Backus Normal form below. There are 4 characteristics of a scripting language of interest in this discussion: (1) topics, (2) subjects, (3) conditions, and (4) pattern lists. A topic is used to process user statements.

```
Topic <string> is
   <Tstatement>*
EndTopic
```

There are three types of topics for the purposes of this discussion: standard, default, and sequence. Standard topics are used to recognize and respond to utterances. If more than one standard topic matches an utterance, then computational mechanisms can be used to prioritize and then select one to apply in constructing a response. Two such mechanisms are specificity, which is based on the information content in the utterance, and recency, which is based on which subject most recently uttered is associated with a topic. Because standard topics are used to respond to utterances, they are executed first. Default topics respond when standard topics cannot respond. There are two types of default topics. One default topic type is associated with a standard topic, so it is related to the subject of the standard topic. Another default topic type is a universal default, sometimes called a "last line of defense", which will respond in the event that no other standard or default topic can respond. Default topics are tested in the order in which they appear in the program and the first applicable default is used to build the response. Sequence topics are used to recognize and respond to utterance sequences when the utterances are connected by how the user responds. Sequence topics are executed only when explicitly accessed in a SwitchTo statement, which is a form of redirection. Sequence topics have the lowest priority of the three topic types described.

The body of each topic is a list of conditional blocks. These conditional blocks are executed in the order found in the topic. If the condition of a conditional block is false, execution goes on to the next conditional block in the topic, or to the next topic if there are no further conditional blocks. If the condition is true, the commands and conditional blocks inside the block are executed, and further behavior of the program is dependent on the keyword which ends the conditional block. If it ends with Done, execution ceases until the next input occurs. If it ends with Continue, execution continues with the next conditional block in the topic, or the next topic if there are no further conditional blocks. If it ends with NextTopic, the rest of the current topic is skipped and execution continues with the next topic.

```
<SubjectList> = Subjects <string> [,<string>]*;
```

The top level of a topic may contain one or more Subjects statements. Each asserts that the given subjects are subjects of the topic. If a non-IF command within the body of the topic is executed, all topics which share at least one Subject with the topic are brought to the front of the "focus of attention." Focus of attention generally refers those things a person is currently thinking of. In this context, topics sharing a subject match part of what the user uttered and are used as the first sorting mechanism for topic selection.

```
<Condition> = If <conditionpatlist> Then |
              IfHeard <patlist> Then |
              IfHeard <pat> [and <pat>]* [and not <pat>]* |
              IfRecall <memlist> Then |
              IfRecall <memref> [and <memref>]* [and not <memref>]* |
              IfDontRecall <memlist> Then |
              IfDontRecall <memref> [and <memref>]* |
              IfChance <chance> Then |
              IfChance Then |
              Always

<patlist>  = <pat> [, <pat>]*  | <symbol>
```

A pattern list is anything that evaluates to a list of strings. It can be either the name of a PatternList object or a list of patterns separated by commas.

A virtual robot generally embodies a particular universe of discourse reflective of the subject matter of interest -- e.g. a BOT developed to converse about personal computers should "know" something about computers and their peripherals. The development of such a BOT employs the scripting language to recognize aspects of the subject matter and respond with appropriate content. Often these "script programs" (or scripts) are written in an action-response type style wherein the actual language supplied by the user embodies an "action" to which the "response" is written into the script program itself.

Scripts are generally written by a "BOT administrator" (human or otherwise) by defining a list of “categories” in which the BOT will be well conversant. Categories may comprise "topics" that are recognizable by a runtime executive. Topics, in turn, may comprise patterns or words that are matched against the stream of input communication (in either spoken or written or any other suitable form of communication) from the user.

The main drawback with constructing virtual BOTs by a list of categories is that the topics developed cannot provide complete coverage of all subjects in the universe of discourse. The result is that the BOT responds with the universal default. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with the default. A related drawback is that the universal default response generally provides insufficient guidance to the user as to their original input: it doesn't provide information regarding why the input "confused" the BOT, and it doesn't provide a knowledgeable response to the input.

The BOT development task must recognize that the level of quality and value evidenced by users is not judged merely in discrete terms, but rather by the overall impression that they get from their interaction with the BOT and by their level of satisfaction with the information the BOT provides.

Thus, there is a need in the art to have a means for easily designing and creating virtual BOTs that enables the BOT to effectively respond to arbitrary utterances with knowledge regardless of the number of topics implemented, guides the user toward providing utterances that will move the user closer to the information they seek, provides the user with information about what in the user's utterance confused the bot when that occurs, and performs these tasks within a framework the eases the maintanence and entendability of the BOT's capabilities.

SUMMARY OF THE INVENTION

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a suitable operating environment for the purposes of the present invention.

Figure 2 depicts the topic types used to present a typical implementation of the present invention.

Figure 3 depicts developed topics in a universe of discourse.

Figure 4 depicts a procedural decomposition of a program into its computational subtasks.

Figure 5 depicts a collection of conceptual domains which describe a universe of discourse.

Figure 6 depicts a uniform distribution of defaults in a universe of discourse.

Figure 7 depicts a set of developed topics in a uniform distribution of defaults.

Figure 8 expands the family view of a conceptual domain as shown in Figure 5.

Figure 9 depicts domain topics for the conceptual domain as found in Figure 8.

Figure 10 depicts the cauterization problem.

Figure 11 depicts the cauterization solution for a domain's children.

Figure 12 depicts the cauterization solution for a domain's parent and siblings.

Figure 13 depicts the tiebreaker problem.

DETAILED DESCRIPTION OF THE INVENTION

I. OVERVIEW AND GENERAL ARCHITECTURE

The term "robot" is used interchangably with "BOT" throughout the remainder of this writeup. For the purposes of this writeup, both "BOT" and "robot" refer to any program which interacts with a human user in some fashion, and should not be assumed to refer only to physically embodied robots. The term "supervisor" is used interchangably with "administrator". The term "domain" is used to represent a body of knowledge that comprises a component of a universe of discourse that the "BOT" will be conversant with. The term "hierarchy" will be used to describe the collection of domains that represents the universe of discourse that the "BOT" will be designed to converse about. It should be noted that the term hierarchy should not be construed to mean the collection data type. Many means of representing the collection are known to those skilled in the art, and the functionality of the collection is not dependent on the data type used to implement it.

Referring now to Figure 1, the operating environment in which the present invention applies is depicted. The environment can be characterized generally into three partitions: front end 102; BOT processor 100; and back end 104. Front end 102 is generally the environment in which a human user 116 consults a virtual BOT interface 114 via a computer 112 that may be connected to the BOT processor via a communications link, such as through a server connected to the Internet or alternatively directly connected to BOT processor 100. It will be appreciated that many other means of connection to a BOT processor 100 are well known to those skilled in the art and that the present invention should not be limited to any particular aspects of the general operating environment as disclosed herein.

Typically, human user 116 connects to a site whose interface of first impression is a virtual BOT interface 114. The advantage for the site developer is that human user 116 may have a help or information

request that is easily handled via BOT interface 114. Today, it is not uncommon to find sites having a list of FAQs ("Frequently Asked Questions") that serve this purpose of handling very low level user concerns and questions. However, for more advanced questions or interactions with the site, virtual BOTs will become increasing popular.

In the operating environment that hosts the embodiment of the present invention, BOT interface 114 is an instantiation of a process that is spawned by BOT processor 100 via connection 110. BOT processor 100 itself may comprise connection 110; runtime executive process 106, compiler 107, and a set of BOT programs 108. As users 116 log onto a site having BOT processor 100 via connection 110, runtime executive 106 executes an interaction routine that guides the discussion that occurs between user 116 and BOT processor 100. Typically, a two way communications dialogue occurs between user 116 and BOT processor 100 wherein user 116 may ask questions, make declarative statements and other normal communications patterns that humans typify. For the purposes of the present invention, “communications” is to be very broadly interpreted. Indeed, suitable communications could be in the form of written or spoken language, graphics, URL’s or the like that may be passed to and from a user to an automatic interface program, such as the present invention.

In turn, runtime executive 106 parses the statements and questions generated by the user and responds according to a set of BOT programs 108. As will be discussed in greater detail, BOT programs 108 are typically created at the back end 104 as a set of "scripts" that the BOT processor will tend to engage in with user 116. For example, if the site using BOT processor 100 is a site for a reseller of personal computers, then BOT processor 100 should be designed to handle questions and discussions concerning personal computers and their peripherals in general. Thus, the back end 104 will generate scripts that will guide the discussion concerning many computer-related topics. These script programs 108 are then compiled by compiler 107 and the compiled code is incorporated into runtime executive 106.

As the two-way discussions between user 116 and runtime executive 106 continue, it is generally desirable to engage in quality control of BOT processor 100. This quality control is provided at back end 104 via feedback loop comprising a transcript of dialogues 118 and backtrace and state information 120 of the BOT processor 100; a supervisor 122 and editor 124. As transcripts develop over the course of interacting with a user, the text of these transcripts are stored, together with the state of the runtime executive and backtrace of execution through the runtime executive code. This information forms the basis for accurately diagnosing the runtime executive and for debugging its performance. Such information may be stored electronically in a storage media or could be printed out in human readable form.

Supervisor 122 analyzes the information at 118 and 120 with an eye towards optimizing the performance of the runtime executive. Typically, supervisor 122 could be another human, deciding if the semantics captured by the system needs to be upgraded in response to a dialog transcript that has occurred. If so, supervisor 122 could optionally invoke an editor 124 to edit the programs that represent the semantic framework of the runtime executive. These programs would then be re-compiled and incorporated into the runtime executive.

Although Figure 1 gives a general description of various operating environments in which virtual BOTs may exist, it will be appreciated that many other operating environments are obvious to those skilled in the art and that the scope of the present invention should not be so limited to the exemplary descriptions as given

above.

II. BOT DEVELOPMENT

In general, BOT user 116 knows what kind of information he seeks, but not necessarily how to articulate his request. Functionally, the implementation of the art described herein provides user 116 with the ability to interact with BOT 100 on any level of abstraction directly associated with the universe of discourse. BOT 100 should be able to achieve this either with a single response to a very specific question that immediately identifies the users' needs, or through dialog in response to a series of increasingly specific queries guided by BOT 100 through the BOT programs 108. The mechanism is independent of the order in which topics are developed.

A. CURRENT MECHANISM

As mentioned in the background, BOT programs 108, or scripts, are generally written by BOT administrator 122 by defining a list of categories in which BOT 100 will be well conversant. These categories generally come from BOT administrator 122, FAQs, user input, and other sources associated with the universe of discourse. BOTs developed this way suffer from two problems. First, BOT 100's ability to answer questions is dependent on the number of topics developed, and the finite number of topics developed by BOT administrator 122 cannot completely cover all aspects of the organization associated with the universe of discourse. Thus, it is highly likely that queries made by BOT user 116 will result in "default" responses. Second, the development of BOT 100 follows specific needs/interests of administrator 122, rather than a systematic, coherant approach. Thus there is no coherence between the topics or their defaults.

Figure 3 illustrates the problem that arises when BOT programs incompletely cover the universe of discourse. Universe of discourse 300 is comprised of those topics 302 that have been developed. Each topic 302 potentially has its own default 304. The content areas remaining when all topic-related content areas are covered are responded to by universal default 306, which will be "hit" by all off-topic user queries. Such responses are considered "misses", because the BOT demonstrates "holes" in its knowledge of the universe of discourse when it is forced to respond with a default. Since the defaults 304 are related to the topics, rather than to each other, or to the universe of discourse, hitting defaults 304 or 306 provides the user with little information of value that would assist in continuing the conversation. There is thus an inherant qualitative problem in developing BOTS from a list of topics. Since BOT 100 is a conversational agent, its value derives entirely from how well it interacts with users 116, where the word "well" may be defined in terms of knowlege content BOT 100 conveys, its friendliness, how easily it is "confused", and how much interaction is required for user 116 to find what is sought. If user 116 asks questions that confuse BOT 100, then BOT 100 is seen as "stupid" and its value is diminished.

The embodiment of the current invention implements BOT 100 by designing a framework of defaults in such a way that it is impossible for BOT 100 to be asked a question that cannot be answered. The structural foundation for this framework, and the interacting mechanisms associated with it, comprise the art described in the remaining sections.

B. APPLICABLE SOFTWARE DESIGN/DEVELOPMENT TOOLS

BOT programs 108 can be thought of as a program in the traditional computer engineering framework. Each can be decomposed to increasingly specific components.

Figure 4 illustrates a typical procedural decomposition, because in both the result can be decomposed to increasingly specific tasks. In the traditional framework, program 402 is decomposed into constituent functional components, or tasks. The "Initialize Objects" 404, "Input Data" 406, "Do Calculations" 408, and "Output Data" 410 tasks are independent functional tasks that, together, comprise program 402. The functional components interact through data objects. Each functional component (404-410) is itself decomposed to those functional tasks that comprise it. For example, the "Do Calculations" 408 task is decomposed to "Perform Analysis" 412 and "Continue Test" 414 subtasks. The decomposition continues to a point where the designer/developer is satisfied that source code can be developed. The tasks at this level are called "terminal" tasks, because they aren't further decomposed. In the software development task, intermediate tasks, such as "Do Calculations" 408, are used to organize the functions of their subcomponents, and their effects are otherwise not seen. Terminal tasks implement the actual functionality of program 402.

Once a procedural decomposition is designed, a program can be implemented without developing any of the terminal functionality. A problem that often arises in software design and development is the tendency for developers to develop the terminal functionality without first implementing the supporting framework depicted by program structure 400. This problem is analogous to developing BOT topics without a supporting content framework. The difference is that the BOT can still function, though its function is severely hampered, whereas a program cannot. Software engineering has a tool, called a function stub, that enables the developer to implement the framework of program 402 without implementing the terminal functionality. The stub is a structure that adheres to the input/output requirements of program 402's design, but implements none of the functionality. For intermediate tasks, stubs comprise the appropriate function calls to subcomponent tasks. Using this approach, the software developer can implement a large program without implementing any of the functionality, and then replace the empty stubs with the actual code that implements the design algorithms for a particular task.

The art described herein conceptually makes use of the hierarchical/procedural decomposition and function stub to develop a BOT that has an organizational structure and is developed around a hierarchical framework of defaults. The analog to program 402's functional component is called a "domain". The analog to program 402's functional decomposition is a content decomposition called a domain "hierarchy." The analog to the function stub is the "domain default." Regardless of the user's query, or the degree of topic development, the BOT developed with a domain hierarchy and domain defaults can answer the user and even direct the user toward what they seek.

C. HIERARCHICAL DECOMPOSITION OF THE UNIVERSE OF DISCOURSE

Every universe of discourse can be described as a grouping of different content components that can be

hierarchically decomposed. Using a virtual robot to provide information about a particular universe of discourse, one must understand how that universe is decomposed and design the robot around the associated hierarchy.

Figure 5 depicts a domain hierarchy for "Company" 502. The decomposition represents the sum total of all information about the company, which is decomposed into four informational and functional components, "Information" 504, "Products" 506, "Services" 508, and "Sales" 510. The "Products" component 506 is shown further decomposed into three components, "Widgets" 512, "Flingys" 514, and "Gadgets" 516. The pictured decomposition is typical, but unimportant. What is important is that the combination of components completely describes the organization, analogous to the relationship of the functional components (404-412) and program 402 in Figure 4.

Each component 502-516 in hierarchy 500 is called a "domain." A domain represents an informational aspect of the organization. This type of hierarchy is called a "component" hierarchy, in that the domains under the top, or "root," domain (i.e., domain 502) are considered components of domain 502. Thus, “Information” 504 is a component of “Company” 502. The reverse is not true (i.e., “Company" 502 is not a component of “Information” 504). A terminal domain is one which has no subdomains. Just as a program’s functionality is implemented in its terminal components, BOT 100s specific content is implemented in its terminal domains. The overall hierarchical decomposition 500 is called a domain "hierarchy."

D. DOMAIN TOPICS

The domain analog to the function stub is a mechanism that allows BOT 100 to respond in lieu of topic development associated with a terminal domain. The domain stub is effectively a default response to a query. It is thus referred to as a domain default.

Figure 6 illustrates the universe of discourse 300 for BOT 100 when it is developed using a domain hierarchy and domain defaults (i.e., stubs), as opposed to the largely vacant universe of discourse 300 depicted in Figure 3. The universe of discourse 300 is now completely divided into non-overlapping areas 602 that represent the domains in hierarchy 500. Two types of queries can be made to BOT 100 in lieu of terminal topic development: (1) relevant and unspecific, or (2) irrelevant. A "relevant" query is one which hits a domain in the universe of discourse (i.e., a domain in the hierarchy). A "specific" query is one that would hit a terminal domain topic. Thus a relevant but unspecific query should hit an intermediate domain in the hierarchy. An "irrelevant" query is one which misses the domain associated with the current focus of attention or, in the worst case, the universe of discourse (i.e., hits no domains in the hierarchy). Despite the fact that no topics are shown in Figure 6, BOT 100 will respond in a coherent manner to relevant but unspecific queries as well as irrelevant queries.

Figure 7 illustrates that topics 302 could be developed for any terminal domain, in any order, once the domain hierarchy and its defaults are in place.

The analogy to a program decomposition and a function stub breaks down with a virtual robot in three

ways: (1) the user/program interaction is different than a user/BOT interaction, (2) the relationship between program components is different than the relationship between hierarchy domains, and (3) the function stub responds differently than the domain default. In a program, a user has very specific points where input is allowed, and very specific points were output is produced, whereas BOT user 116 could possibly interact with BOT 100 at any domain as mentioned above. Moreover, the program user would have no use for output at intermediate points, since all that is necessary is the functional implementation at component terminals. The domains in a virtual bot hierarchy are always related by content, so every domain from root domain 502 to a particular terminal domain can provide a viable degree of content for the terminal domain. As a result, a query directed at "Widgets" 512, if made in a vague way, could be responded to by "Products" 506, or even "Company" 502. Finally, the function stub response may or may not have a value, but the domain default always has a language response.

For example, if the terminal domain is "Widgets" 512, then domain "stubs" are required for each of the "Company" 502, "Products" 506, and "Widgets" 512 domains, because user 116 could make a query regarding "Widgets" 512 that could hit domain defaults at either "Products" 506 or "Company" 502, depending on the degree query specificity. Any of the following queries/comments could be issued by user 116 with respect to widgets:

Q1 - "Do you have any widgets?"
Q2 - "What kind of widgets do you have?"
Q3 - "Widgets"
Q4 - "What products do you have?"
Q5 - "What do you have?"
Q6 - "Do penguins live here?"

The so-called domain "stubs", or "defaults", could respond to these queries without any specific information regarding widgets being developed. Such responses would look like the following:

Q1 - "Do you have any widgets?"
A1 - "Yes, we have a number of widgets, would you like to see a listing, would you like information about a particular type of widget, or would you like to see our inventory?"
Q2 - "What kind of widgets do you have?"
A2 - "We have red, green and blue widgets. Would you like a feature comparison chart or information about a particular model?"
Q3 - "Widgets"
A3 - "Widgets are one of our products. They are used in many industrial applications. We also manufacture and sell flingys and gadgets."
Q4 - "What products do you have?"
A4 - "We manufacture and sell widgets, flingys, and gadgets. Would you like to know more about any of these, or are you interested in our inventory of any of them."
Q5 - "What do you do?"
A5 - "I am glad you asked that. We are a small manufacturing company. We design and build the best widgets, flingys and gadgets money can buy. Would you like more information on a particular product?"
Q6 - "Do penguins live here?"
A6 - "I’m sory, I do not understand what you mean. Weren’t we just talking about our company’s products?

Notice that all of the queries are a bit unspecific, and that there are different types of responses to the different types of queries. In query Q1, it is unclear whether user 116 is seeking information about whether the organization produces widgets, what kind of widgets, or whether user 116 wants to know how many are in stock. The answer must be a combination of what would be at a terminal domain and information obtained from the "Sales" 510 domain. In query Q2, the request is really for an enumeration of widget types, but it is vague because what the user wants to know about widgets is unspecified. The response should provide both the information requested and provide options for viewing that information. Query Q3 only mentions the term "Widgets" itself, and so BOT 100 can only assume that user 116 seeks to know more about widgets, or maybe what role they play in the organization. Query Q4 is more general, in that it refers to all of the products this company has. The response is, again, an enumeration of product types, and should look similiar to the answer to query Q1. Query Q5 is extremely general, but the response still enables user 116 to provide a new query the will advance closer to the widget information sought. Finally, query 6 is irrelevant to the current conversation. The response shows that BOT 100 is confused, but also attempts to help user 116 by reminding user 116 about the previous topic of conversation.

Domain Family Relationships and Topic Types

A function stub may return 0 or more different object values. Similarly, there is variety in the types of default responses BOT 100 can make. Unlike the function stub, the robot can only say one thing at a time. Thus there is a need to have more than one type of default associated with each domain. The number and type of domain defaults is based on the conceptual/informational relationships a domain has in the domain hierarchy.

Figure 8 illustrates the family relationships for the "Products" 506 domain. There are three kinds of relationships with respect to a domain: (1) it has a single parent 802, (2) it has perhaps many siblings 804, and (3) it has perhaps many children 808. A domain's parent represents the more abstract domain of which domain 806 is a component. The parent 802 to domain 806 is represented in Figure 8 with "Company" 502. A domain's siblings share the same parent, and hence the same level of abstraction, with the domain in the hierarchy. A sibling 804 to domain 806 is represented in Figure 8 with "Information" 504. The connection between domain 806, its parent 802, and its child 808 is shown with a solid line owing to the direct relationships between them. The connection between domain 806 and sibling 804 is shown as a dotted line because there is no direct connection between the two. Their relationship exists because they are both components of parent 802. The other sibling domains would be "Services" 508 and "Sales" 510. A domain's children constitute the information categories that comprise the domain, as components. A child 808 of domain 806 is represented in Figure 8 with "Flingys" 514. The other children of domain 806 are "Widgets" 512 and "Gadgets" 516.

Domain family relationships are important in BOT 100 because they are directly associated with types of queries and responses that can be made on a domain. There are three query/response types directly associated with family relationships. A child query/response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on domain 806's subcomponents. Queries Q2 and Q4 represent child queries, because the request is for enumeration of domain 806's children. A sibling query/ response is associated with queries that are relevant to universe of discourse 300, unspecific, and based on the domain 806's definition or relationship to its siblings. Query Q3 represents a sibling query since it only mentions the domain. The response talks about what the domain ("Widgets" 512) means and how it relates

to parent 802 with respect to its siblings. A parent query/response is associated with queries that are irrelevant to the previous query domain 806. Typically, when a query hits a domain topic, the focus of attention is set on that domain. When user 116 makes query Q6 after having made query Q5, the focus is set on the domain topic that provided the Q5 response A5, namely "Products" 506. Since query Q6 is irrelevant to "Products" 506, the response informs user 116 that BOT 100 is confused. In addition, the response reminds user 116 what the last domain was, and its parent ("Company" 502, parent 802), so as to help the user clarify the next query. Since the family query/response types provide information about the location of domain 806 in domain hierarchy 500, they can be used as a navigational aid for assisting user 116 in clarifying requests to BOT 100. This mechanism thus satisfies one of the requirements of BOT 100 design.

Figure 9 illustrates that family components and family-related query/answer types are implemented with three independent domain defaults. The child default 902 focuses on domain children. The sibling default 904 focuses on the domain siblings. The parent default 904 focuses on the domain parent. Together, the child, sibling, and parent defaults comprise the domain defaults. Domain defaults are typically implemented as standard topics, and so in bot development they are called domain topics.

Domain topics (i.e., defaults) have an order of precedence based on the desire to ‘move’ user 116 closer to a domain terminal, in which a standard topic can be used to fulfill their needs. Child 902 and sibling 904 topics respond to relevent (i.e., in universe of discourse 300) utterances, so they have a higher precedence than parent topics 906, which respond to irrelevant utterances. Child topics 902 preferred over sibling topics 904 because they point downward, by talking about domain children 808. These are most likely to be asked and most likely to provide user 116 with appropriate guidance. Sibling topics 904 are the next most preferred, because they are still relevant, and they talk about domain siblings 804. Parent topics 906 are the least preferred, and point to the parent domain 802. As will be disclosed below, the typical implementation mechanism will cause the domain topics to be executed in a way that maintains this precedence ordering.

III BOT IMPLEMENTATION

The operating environment that hosts the current embodiment of the present invention uses Neuroscript to implement BOT programs 108 and Neuroserver to implement BOT processor 100. It will appreciated by those skilled in the art that implementations of the current invention need not be made using Neuroscript or Neuroserver, and that other computational mechanisms could be employed. Domain hierarchy 500 is implemented with a file structure that mirrors the domains of hierarchy 500. This is typically done for ease of organization and maintanence purposes. Every domain in the hierarchy represents a directory by the same name in the file system. Both default and non-default topics are implemented in files. Domain (i.e., default) topics are implemented in a file in the domain-name directory. Non-default topics are implemented in a file in domain terminals. Thus domain terminals have two files, one each for default and non-default topics. Domain (i.e., default) topics are implemented with Neuroscript using standard topics 220. Standard topics are also used to implement non-default responses. As a result, computational mechanisms are required to distinguish and select between the two topic types. Three such mechanisms, focus of attention, specificity, and recency, have already been described. Terminal topics will be more specific than any domain topics. Domains that are deeper in the hierarchy will be more specific than domains higher in the hierarchy. In addition, as will be disclosed below, child topics will be more specific than sibling topics, which will be more specific than parent topics. These mechanisms thus work in concert to satisfy the requirements of BOT development. Below is a description of the implementations of the child 902, sibling 904, and parent 906

domain topics, followed by a description of implementation mechanisms used to maintain domain hierarchies.

A. CHILD TOPICS

Child topics 902 are triggered by description or fact based queries to BOT 100. For example, if user 116 makes query Q7, BOT 100 recognizes it as a description question, in that it basically asks for a description of what is available. Description questions can generally be answered with an enumeration. If user 116 makes query Q8, BOT 100 recognizes it as a factual question. Factual questions can generally be answered yes or no, but the response to query Q8 must be used for both query types, so the yes answer is implicit rather than explicit.

Q7 - "What educational programs do you have?"

A7 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

Q8 - "Do you have an employee reimbursement programs?"

A8 - "We have an employee reimbursement program for degree programs. We also have in-house training for non-degree certification programs."

In Neuroscript, fact and description question types are parsed using ?FactQuestion and ?DescriptionQuestion, respectively. In queries Q7 and Q8 the child topic 902 responds. In both cases, the question is unspecific, because user 116 does't actually ask for information on a specific program, so a terminal standard topic cannot respond. But BOT 100 can help user 116 out, by elaborating what kinds of educational programs the company has. User 116 can now clarify their request, and BOT 100 doesn't look stupid. The following is an example illustrating the Neuroscript for an "Education" child topic that would respond to queries such as Q7 and Q8:

```
Topic "Random description or fact question about Education" is
Subjects "Education";
    If (?DescriptionQuestion contains DOM_EDUCATION) or
       (?FactQuestion contains DOM_EDUCATION)
    Then
        SayToConsole "Trace -- Education, A answer";
        Example "what kind of ducks swim in Education pool?";
        Say "Talk about Education and, in particular, " +
            "with respect to the children: Science Engineering Business";
    Done
EndTopic
```

The specificity of the topic is based on the combined specificity of ?DescriptionQuestion and ?FactQuestion. Note that the subject of the topic, "Education", is the same as the domain 806 name. The pattern list (as described in the background section), DOM_EDUCATION, is initially implemented with a single element by the name of domain 806. During BOT 100 development, DOM_EDUCATION is extended to include synonyms for education, such as "training." Domain pattern lists must be carefully implemented so that domains don't clash, since clashes reduce the ability of BOT 100 to answer queries

with the correct domain topic.

B. SIBLING TOPICS

Sibling topics 904 are triggered by direct reference based queries to BOT 100. For example, if user 116 makes a query such as Q9 or Q10, BOT 100 recognizes it as a direct reference to domain 806 and responds in two ways: (1) pseudo definition, and (2) information about siblings, as exemplified by the response to queries Q9 and Q10.

Q9 - "Education"

A9 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

Q10 - "Does employee education play a role in advancement at your company?"

A10 - "We have degree and certification educational programs here. We also have stock participation, health insurance, paid holidays and retirement benefits for employees."

The idea of the sibling topic 904 is that user 116 has mentioned domain 806 by name. In some cases, as in query Q9, perhaps only the domain name is provided in the query. In other cases, such as in query Q10, the query may be complex, but the only thing recognizable by BOT 100 is the direct reference. BOT 100 cannot be expected to direct user 116 downward, but what it can do is provide a little information about domain 806, and to provide user 116 with some information about how domain 806 fits into the parent domain 802 by talking about its sibling domains 804. The response to query Q9 indicates that, if user 116 is confused, providing some information about how education fits into the company's benefits programs might help them to restate their query with greater clarity. Also notice, in the response to query Q10, that a direct reference can lead to inappropriate responses.

```
Topic "User mentions Education, by itself" is
Subjects "Education";
    If Heard DOM_EDUCATION
    then
        Example "Education";
        SayToConsole "trace -- Education, B answer";
        Say "We have educational support for employees wishing to, " +
            "further their professional growth in degree-granting "  +
            "programs. We also have training programs that lead to " +
            "certification rather than degree objectives.";
    Done
EndTopic
```

Once again, it is clear from the Neuroscript of the sibling topic 904 that the domain 806 name is the subject of the topic. In Neuroscript, direct references are parsed using the "If Heard" mechanism. In the sibling topic, the "If Heard" mechanism is applied to the domain pattern list, DOM_EDUCATION. "If Heard" is also less specific than either "?DescriptionQuestion" or "?FactQuestion", so the sibling topic is less likely to be selected than the child topic, but, being more general, it is more likely to match a relevant utterance.

C. PARENT TOPICS

Parent topics 906 are triggered by queries to BOT 100 that have nothing to do with the current focus of attention. For example, if user 116 previously made a query about education, then education is currently the focus of attention. If user 116 then makes a query such as Q11, BOT 100 recognizes that the subject of the query isn't education, and so is irrelevant. When the query is irrelevant, the parent topic 906 responds, as shown in the response to query Q11 below:

Q11 - "Can I paraglide off the cliff in your back yard?"

A11 - "I am confused, what you have said is either too complicated for me to understand, or I cannot see the relationship to our last topic, which was Education or Services."

There are any number of possibilities of why user 116's current query is unrecognizable, and BOT 100 simply isn't smart enough to decide what to do. As such, parent topic 906 tells user 116 that it doesn't understand what the user said, and reminds user 116 that the last topic of conversation was about current domain 806, but focuses the discussion on the parent domain 802.

```
Topic "We are baffled, but the last topic was Education" is
Subjects "Education";
    If Focused
    Then
        SayToConsole "Trace -- Education, C response";
        When Focused Example "Do you go sledding on winter mornings?";
        Say "I am confused, what you have said is either too complicated " +
            "for me to understand, or I cannot see the relationship to "   +
            "our last topic, which was Education or Services.";
        Focus Subjects "Services";
    Done
EndTopic
```

In this final example, it is seen that domain 806 name is again the topic subject, though it is no longer associated with the test or the example. Domain 806 is again referenced in the response, along with the parent domain 802, but the parent domain 802 is added to the focus list. In cases of non-relevance, such as query Q11 above, it is important to show user 116 that BOT 100 isn't giving up on them, and so the "backing up" response is viable.

Domain topics perform the same task for every domain 806 in domain hierarchy 500, so the structure of the each of the three domain topics is identical except that the domain 806 name changes from topic to topic. In the current implementation of the art, an iterative process is used for creating and maintaining domain topics.

D. DOMAIN INTERACTIONS

The connectivity in a domain hierarchy 500 results in inter-domain interactions which affect domain topic content. Two such interactions are important to note: (1) changes to the number of domains 806 in hierarchy 500, and (2) interactions between the domain 806 names across major branches of hierarchy 500.

Domain Hierarchy Content Changes

The structure of domain hierarchy 500 imposes requirements on the content of domain topics when family members change. The normal development of BOT 100 sees three types of change: (1) add domain 806 to hierarchy 500, (2) remove domain 806 from hierarchy 500, and (3) cauterize domain 806 in hierarchy 500. In the first two cases, the number of components under parent 802 changes, and this changes the child topic 902 and sibling topic 904 say statements. For example, in Figure 5, if two new domains are added under "Services" 508, called "Education" and "Training", then the child topic 902 for Services domain 508 must be edited to include the "Education" and "Training" domains. In addition, the "Education" domain sibling topic 904 has to be edited to include the "Training" domain, and vice versa.

Domain Cauterization

Two scenarios exist wherein domain hierarchy 500 is unable to provide accurate support in universe of discourse 300. The first scenario is when BOT administrator 122 chooses to restrict the universe of discourse with respect to a more generic domain hierarchy. For example, in hierarchy 500, BOT administrator 122 may choose to disable the domain defaults for the "Services" 508 domain without changing the structure of hierarchy 500. The second scenario is when BOT administrator 122 chooses not to restrict the universe of discourse, but the topic development for a domain subtree (all domains below a particular domain) is incomplete. Using the same example above, during BOT 100 testing, perhaps some of the non-default topics under "Services" 508 are incomplete, so rather than have BOT 100 respond incoherently, BOT administrator 122 again disables the subtree.

Figure 10 depicts a "cauterization", which is the operation performed in both scenarios. A cauterization recursively replaces the domain topic say statements with a single statement that refers back to the root domain in the cauterization. In this example, the cauterization subtree is shown as a shadowed triangle.

Figure 11 shows that each domain topic inside the shadowed triangle refers back to the "Products" domain. "Products" domain 506 is being cauterized, so it is called the "cauterization root" domain. The cauterization say statement is basically the same for each topic type inside the triangle, and will be illustrated with a child topic 902 for the "Widgets" 512 domain:

```
Topic "Random description or fact question about Widgets" is
Subjects "Widgets";
    If (?DescriptionQuestion contains DOM_WIDGETS) or
       (?FactQuestion contains DOM_WIDGETS)
    Then
```

```
        SayToConsole "Trace -- Widgets, A answer";
        Example "what kind of ducks swim in Widgets pool?";
        Say "I'm not trained to talk about Products at this " +
            "time, sorry.";
    Done
EndTopic
```

Regardless of which domain topic in the cauterization root's subtree matches user 116's query, the response should be the same. They all point back to "Products" 506.

Figure 12 depicts domain 806's family members that are affected by a cauterization. When one or more of many child domains 806 under a parent 802 is cauterized, then the parent topic 906 must be modified to remove the domain names from the child topic 902, and to remove the domain name as a sibling in the sibling topic 904. In the current implementation of the art, domain cauterization can automatically be performed, reversed, and reperformed using a different cauterization root domain, over an over again, without adversely affecting the coherence and continuity of the hierarchy.

Domain Tiebreakers

There is a certain degree of redundancy in any well-designed domain hierarchy. Many instances can arise in the development of hierarchy 500 where domain topic subjects and pattern lists in different hierarchy branches "clash," meaning that they will contain one or more of the same values. In Neuroscript, only one topic can respond at a time, so only one of perhaps many clashing topics will have its say statement executed. In cases where domain clashes are identified, BOT 100 should be designed to implement what is termed in the art as a "tiebreaker." The tiebreaker redirects control to a sequence topic 240. The sequence topic directs user 116 to select from the different subjects possible, the content for which comes from the say statements of the affected topics.

Figure 13 illustrates an example tiebreaker scenario between two domains relating to "Sales," one of which is a component of the "Information" subtree, while the other is a component in the "Sales" subtree. In each case the meaning of "Sales" is slightly different, but BOT 100 would match on the word "sales", and so "sales" would be a member of each of the pattern lists for topics in these domains. The sequence topic informs user 116 that BOT 100 knows about "Sales" in a number of contexts, and asks user 116 to select one and continue. This way, it becomes clear that BOT 100 recognizes that user 116 is making a general query but perhaps isn't aware that the query can be responded to in different ways. The sequence topic is intended to help user 116 obtain an answer to the query without making assumptions as to which of the domains the query references. The say statements in the tiebreaker are typically copied from the child and sibling domain topics (902 and 904) directly, though they need not be. The following Neuroscript shows how the domain tiebreaker for a child topic is implemented.

```
Topic "Tiebreaker A for PatternList SALES" is
Subjects "Information Sales", "Company Sales";
    If Heard "sale#"
```

```
        Then
            SwitchTo "Sequence Tiebreaker A for PatternList Sales";
    Done
EndTopic

Sequence Topic "Sequence Tiebreaker A for PatternList Sales" is
    Always
        SayToConsole "Trace -- Sales Tiebreaker, A sequence answer";
        Say "I know about <I>Information</I> Sales and <I>Company</I> " +
            "Sales. Which would you like to know more about?";

    WaitForResponse;
        If Heard DOM_INFORMATION
            Then
               SayToConsole "Trace -- Sales Tiebreaker, A information answer";
             Say "I can tell you about what our overall sales were, what our " +
                 "sales per product type were, and what our sales for "       +
                    "particular products were.";
                Focus Subjects "FINANCIALSALES";
            Done
        If Heard OUTREACH
            Then
                SayToConsole "Trace -- Sales Tiebreaker, A outreach answer";
              Say "I can tell you about things like how our sales department " +
                    "works, and what they have in mind for the future.";
                Focus Subjects "OUTREACHSALES";
            Done
        Otherwise Always
           SayToConsole "Trace -- Sales Tiebreaker, A otherwise always answer";
            Say "I thought we were talking about sales, but anyway...";
            InterruptSequence;
        Done
    SwitchBack
EndTopic
```

The first thing to notice in the tiebreaker is that it always has multiple subjects, whereas in other topics it is possible, but not necessary, to have multiple subjects. Second, the recognition mechanism for the tiebreaker is an "If Heard" mechanism. If executed, control is switched to the sequence topic. The sequence topic requests user 116 to select between the two clashing subjects and waits for user 116's response. If user 116 selects one of the pattern lists, then the response is taken from the associated domain's topics, and that domain's subject is focuses. If neither of the pattern lists is heard, then a tiebreaker default is issued as a response and the sequence is terminated.

IV. THE PROCESS OF IMPLEMENTING A HIERARCHY-BASED BOT

1. Create a hierarchical description of the organization being modeled
2. Create a file system architecture that mirrors the domain hierarchy
3. Create domain topics for the hierarchy

4. Create domain pattern lists for the hierarchy
5. Perform any necessary cauterizations
6. Create any necessary tiebreakers
7. Proceed with standard topic development in domain terminals
8. Develop standard topic pattern lists
9. Cauterize/uncauterize as necessary

CLAIMS

1.

2.

3.

4.

5.

ABSTRACT 57