AI Magazine Volume 25 Number 3 (2004) (© AAAI)

Building Agents to Serve Customers

Mihai Barbuceanu, Mark S. Fox, Lei Hong, Yannick Lallement, and Zhongdong Zhang

■ AI agents combining natural language interaction, task planning, and business ontologies can help companies provide better-quality and more cost-effective customer service. Our customer-service agents use natural language to interact with customers, enabling customers to state their intentions directly instead of searching for the places on the Web site that may address their concern. We use planning methods to search systematically for the solution to the customer's problem, ensuring that a resolution satisfactory for both the customer and the company is found, if one exists. Our agents converse with customers, guaranteeing that needed information is acquired from customers and that relevant information is provided to them in order for both parties to make the right decision. The net effect is a more frictionless interaction process that improves the customer experience and makes businesses more competitive on the service front.

As companies optimize their production and supply-chain processes, more people use the quality of customer service to differentiate between alternative vendors or service providers. Customer service is currently a manual process supported by costly call-center infrastructures. Its lack of flexibility in adapting to fluctuations in demand and product change, together with the staffing and training difficulties caused by massive personnel turnovers, often results in long telephone queues and frustrated customers. This is a major cause for concern, as it generally costs five times more to acquire a new customer than to keep an existing one.

How can AI help in addressing this problem? For several years we have built a domain-independent AI platform for creating conversational customer-service agents that use a variety of natural language understanding and reasoning methods to interact with customers and resolve their problems. We have applied this platform to customer-service applications such as technical diagnosis of wireless-service delivery problems, product recommendation, order management, quality complaint management, and sales recovery, among others. The resulting solutions and the lessons learned in the process are the subject of this article.

Understanding, Interaction, and Resolution

Compared to the "newspaper page" model of Web sites, in which people have to navigate the site, find the information they need, and make the ensuing decisions on their own, natural language interaction combined with AI-based reasoning can change the interactive experience profoundly (Allen et al. 2001). The high bandwidth of natural language allows users to state their intentions directly, instead of searching for a place in the site that seems to address their problem. Having a reasoning system that plans (in an AI sense) for achieving the user goals increases the certainty that a solution that can satisfy both the user and the company will be found. The ability to converse guarantees that the relevant information is acquired from the user and provided to the user if and when needed in order for both parties to make the right decisions.

Ideally, a conversational customer-interaction agent should be able to understand language, converse and resolve problems with high accuracy within its main area of competence, and degrade gracefully as we depart from this area. Human users react with more displeasure when the agent exhibits complete failure to understand than when it shows partial understanding or even an effort to understand. They also assume that an agent should know about issues that are commonsensically related to its main competence area. Therefore, graceful degradation must be provided, supported by a critical minimum of commonsense knowledge around the main competence area.

The Internet currently reaches a variety of touch points through which the agent must be available: Web browsers on high-resolution desktops and small-screen devices such as cellular telephones and PDAs, e-mail, and the public telephony network accessible by telephone. Consequently, a user must be able to achieve the same goals with the same results through any of these channels. State changes performed on one channel (for example, changing an order) must be reflected on the other channels. As some interactions are easier on some channels than on others, the agent is responsible for planning and carrying out conversations such that it uses the advantages and avoids the pitfalls of each channel in part.

To create customer agents of this type, a large variety of types of knowledge need to be represented and applied, including linguistics knowledge, conversation-planning knowledge, and industry- and company-specific business knowledge. Last but not least, an engineering challenge is to represent all these types of knowledge uniformly in a complete ontology that is supported by visual editing tools and that facilitates componentization and reuse.

Overall Architecture

How do we achieve these objectives? An interactive customer agent built on our platform has the architecture shown in figure 1. The example illustrated is about games that are subscribed to online and downloaded through the air for use on Java-enabled cellular telephones.

The user input (from no matter what channel) is first converted into an internal message format. From this, the Natural Language Understanding module produces a set of semantic frames that represent the user's intention and contain relevant information extracted from the input. These frames are passed to the Conversation Management module. This module identifies whether the issue can be answered immediately or whether a longer conversation needs to be initiated. In the latter case, a workspace is created for the Interaction Planner, consisting of a goal stack and a database. The goal stack is initiated with the starting goal. The planner uses a hierarchical task-planning method to decompose goals into subgoals and perform a backtracking search to satisfy them. If the user complains about not being able to use a specific game, the planner creates subgoals for applying a diagnosis method to determine the cause. Normally the investigation would start with obtaining user account information from the back end and determining the conditions under which the incident occurred. The Business Reasoning module performs domain inferences, such as determining that the user's device must be J2ME enabled, and applies business policies, such as verifying content-rating restrictions on user accounts (in the process, accessing account information from the back end). In figure 1, the incident location was not provided in the input, and thus has to be asked for. The Ask-incident-location method used for the Establish-incident-location goal determines the incident location by requesting the Natural Language Generation module to create the question for the user. At this point, the user can answer the question or diverge and create another conversation topic by asking a different question. Dialogue management policies in the agent allow such diversions to be handled in various ways.

Understanding Natural Language

The need for both breadth and depth of interaction has led us to integrate two natural language-understanding methods. The first, conferring high accuracy in limited domain segments, is analytic. It relies on a staged approach involving tokenizing, limited syntax analysis, semantic analysis, merging, and logical interpretation. The second is a similarity-based method that computes the degree of similarity between a question and a set of possible answers, returning the answers that have the highest similarity. This is used to provide general answers from an arbitrary set of preexisting documents. The two methods can be used in combination based on a unique domain ontology.

Domain Ontology

An *ontology* is a conceptualization of an application domain shared by a community of interest, in our case including at least the vendors and the customers of the products the agent is servicing.

We use description logics (Borgida et al. 1989) to represent ontologies. Description logics represent knowledge by means of structured concepts organized in lattices. A *lattice* is an acyclic-directed graph with a *top* and a *bottom* node. See figure 2 for an example. An edge in the graph represents a subsumption relation between two concepts. A concept *a* subsumes a

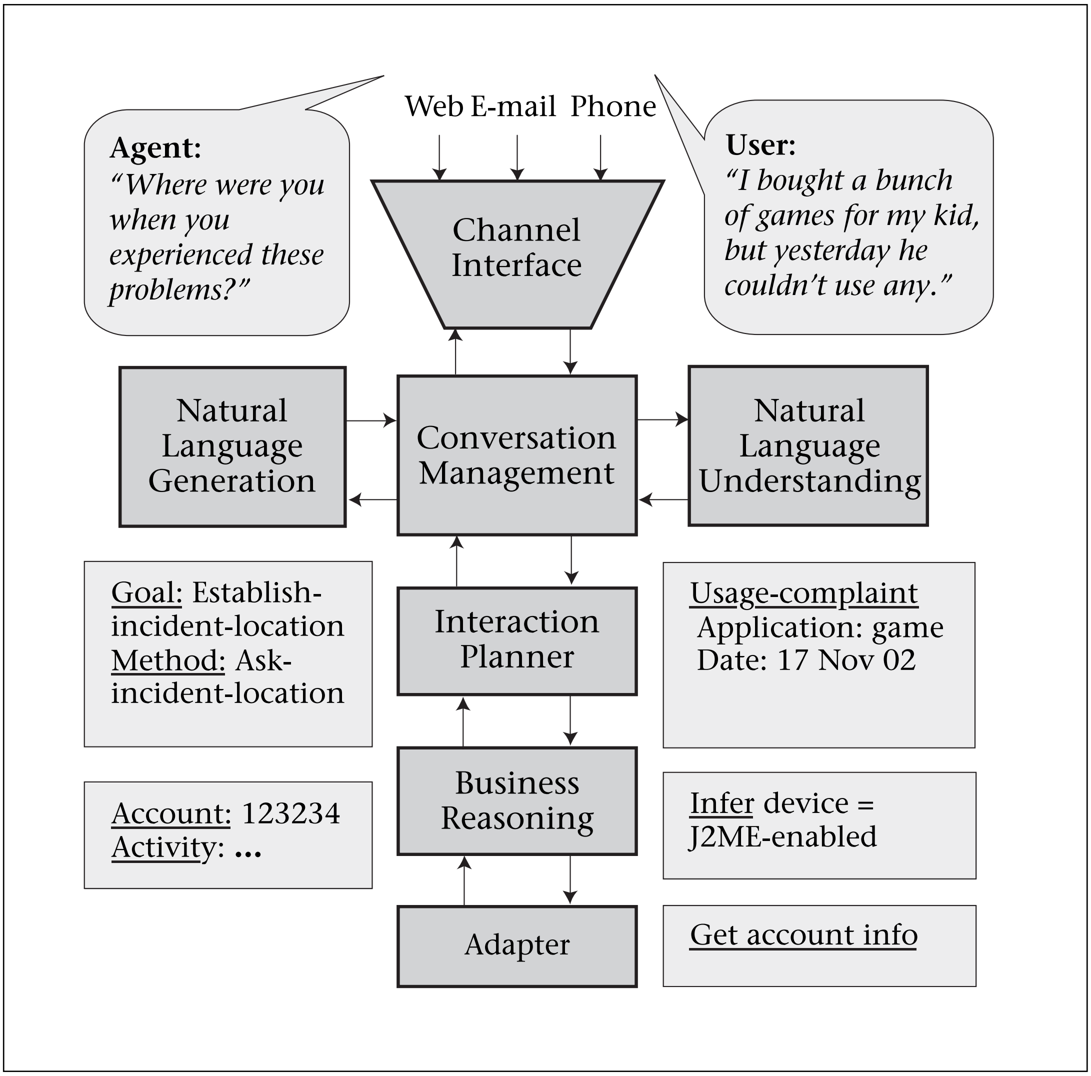

Web E-mail Phone
Agent:
"Where were you when you experienced these problems?"
Channel Interface
User:
"I bought a bunch of games for my kid, but yesterday he couldn't use any."
Natural Language Generation
Conversation Management
Natural Language Understanding
Goal: Establish-incident-location
Method: Ask-incident-location
Interaction Planner
Usage-complaint
Application: game
Date: 17 Nov 02
Account: 123234
Activity: ...
Business Reasoning
Infer device = J2ME-enabled
Adapter
Get account info

Figure 1. Overall Agent Architecture.

concept *b*, *subsumes(a, b),* if and only if any instance of *b* is also an instance of *a*. Description logics provide automated concept classification (see figure 3, "Classification in Description Logics"). Figure 2 shows a very small, classified product lattice. It also illustrates the ability of the formalism to handle disjoint subconcepts. These are subconcepts that do not have common instances, such as those marked with an "x" in the figure (Vegetable and Meat). Classification is valuable for ontology construction, as it enforces and clarifies the semantics of concepts by making explicit the logical consequences of their definitions. (See figure 3.)

Knowledge-Based Language Understanding

We use linguistic and domain knowledge to understand natural language expressions by applying a sequence of steps to the linguistic input: tokenization, syntax analysis, semantic analysis, merging, and logical interpretation. Here's what each step does.

Tokenization. Words are first tagged with part of speech information. Main tags indicate nouns, verbs, adjectives, adverbs, prepositions, determiners, as well as the tense and the singular or plural form. Words not found in the dictionary are spelling-checked and corrected. Remaining words are tagged as general alpha-digit or digit strings. Alpha-digit and digit strings are then recognized as URLs, e-mails, or telephone numbers. Token sequences are recognized as dates, addresses, person and institution names, and a few others. Domain-specific token grammars defined as regular expressions are applied if defined.

Syntax Analysis. Syntax analysis is limited to

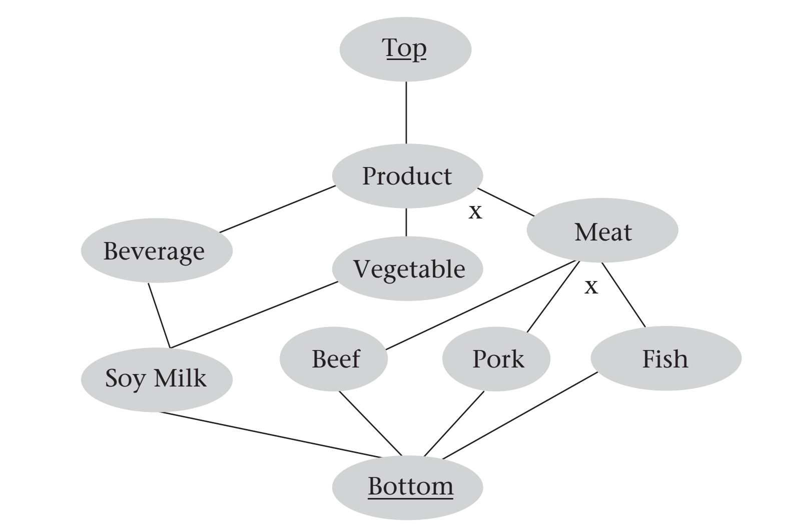


Figure 2. Classified Concept Lattice.

Given a set of nodes N and the subsumption relations amongst them as a subset of N*N, classification generates a lattice containing the nodes N such that any node *a* in N is connected to its most specific subsumers (MSS) and most general subsumees (MGS). The set MSS contains subsumers of *a* such that no node exists that subsumes *a* and is subsumed by a node from MSS. The set MGS contains subsumees of *a* such that no node exists that is subsumed by a and subsumes a node in MGS.

Figure 3. Classification in Description Logics.

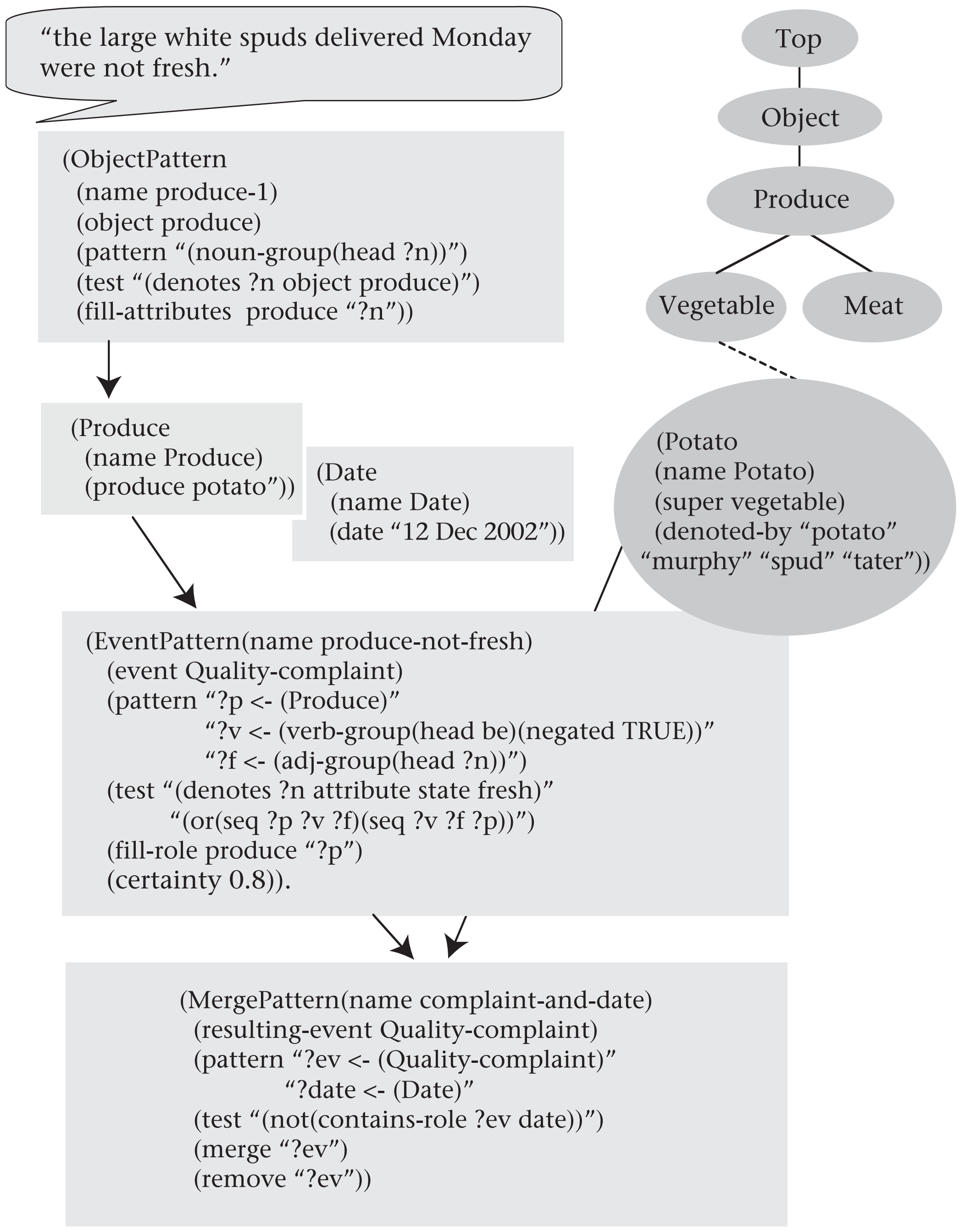
the recognition of structures that can be reliably recognized. Based on the Fastus (Appelt et al. 1993) approach, we recognize only noun groups and verb groups. The noun group consists of the head noun of a noun phrase together with its determiners and modifiers, for example, "the new game" or "two pounds of brown-spotted Bonita bananas." The verb group consists of a verb together with its auxiliaries and adverbs, for example, "were delivered late yesterday evening" in "two pounds of brown-spotted bananas were delivered late yesterday evening." Verb groups are tagged as active, passive, infinitive, gerund, and negated. Noun and verb groups are scored in a manner that increases with the percentage of the input they account for. This biases the system toward preferring longer subsuming phrases to shorter ones. However, no phrase is discarded.

Semantic Analysis. The next step is to recognize instances of ontology concepts. At the topmost level (immediately under the *top* node), the ontology is divided into Objects, Events, Attributes, Roles, and Values. Objects have single or multiply valued attributes, while events have single or multiply valued roles. Instances of these concepts are recognized by semantic patterns. Semantic patterns rely on the presence of denoting words as well as on verifying constraints among entities. The denoting words are sets of words or expressions that, if encountered in the linguistic input, imply the possible presence of that concept. The denoting words and expressions are inherited by subconcepts. An object or event can be implied directly, by finding any of its denoting words, or by finding denoting words that imply any of its subconcepts in the classified ontology.

Figure 4 illustrates a semantic pattern for the Produce concept and the way the fragment "the large white spuds" is recognized as denoting an instance of produce. The particular ObjectPattern shown simply requires a noun group whose head is a word that denotes (through the inheritance-based denoting-words mechanism) the concept Produce. Fragments like "the fatty Atlantic salmon" or "a box of President's Choice soy milk" would be recognized as denoting the Produce concept in exactly the same way by this pattern.

Objects and events recognized by some patterns can be bound in other patterns to recognize further objects and events. For example, having recognized "the large white spuds" as an instance of Produce makes it possible to apply the produce-not-fresh EventPattern in figure 4 to the full input sentence "the large white spuds delivered Monday were not fresh." This results in recognizing the input as a complaint of the type "<produce> is <not fresh>." Note that a sentence of the type "is <not fresh> <produce>" would also have been recognized by the same event pattern, because the sequencing constraints in the pattern (the seq predicates in the test clause) allow both of these orders.

Merging. After applying the patterns in the semantics stage, the merging stage combines objects and events into higher-order aggregates. In figure 4, a Quality-complaint event (not containing the role date) and a Date object are merged into a new Quality-complaint event. The new event contains the union of the roles and attributes of the merged objects and



“the large white spuds delivered Monday were not fresh.”

```
(ObjectPattern
  (name produce-1)
  (object produce)
  (pattern “(noun-group(head ?n))”)
  (test “(denotes ?n object produce)”)
  (fill-attributes  produce “?n”))
```

```
(Produce
  (name Produce)
  (produce potato”))
```

```
(Date
  (name Date)
  (date “12 Dec 2002”))
```

```
(Potato
(name Potato)
(super vegetable)
(denoted-by “potato”
“murphy” “spud” “tater”))
```

```
(EventPattern(name produce-not-fresh)
  (event Quality-complaint)
  (pattern “?p <- (Produce)”
           “?v <- (verb-group(head be)(negated TRUE))”
           “?f <- (adj-group(head ?n))”)
  (test “(denotes ?n attribute state fresh)”
        “(or(seq ?p ?v ?f)(seq ?v ?f ?p))”)
  (fill-role produce “?p”)
  (certainty 0.8)).
```

```
(MergePattern(name complaint-and-date)
  (resulting-event Quality-complaint)
  (pattern “?ev <- (Quality-complaint)”
           “?date <- (Date)”
  (test “(not(contains-role ?ev date))”)
  (merge “?ev”)
  (remove “?ev”))
```

Figure 4. Semantic Analysis and Merging.

events. Merge patterns solve most of the pronominal references to entities occurring in the message, as, for example, binding "it" to "an order" in "I placed an order yesterday on the site. Now I need to change it." To solve references to entities introduced in the dialogue by the system, these entities are appended to the user message by the Natural Language Generation moduleand then merged as usual.

Logical Interpretation. The application domain may logically constrain the meaning of a user message by making certain interpretations inconsistent with each other. For example, it is not possible for someone at the same time to change the date of an order and to cancel it. We define a consistent interpretation of a message as a set of mutually consistent recognized objects and events. Assuming we have an order with unique ID 735, {(change-order 735), (cancel-order 735)} is not a consistent interpretation, but both {(change-order 735)} and {(cancel-order 735)} are. The score of an interpretation is the sum of the scores of its final components (objects and events). A component is said to be final if it is not contained in other components (as values of their roles or attributes). The logical-interpretation stage has the goal of finding a consistent interpretation of the input that has a maximal score (Fox and Mostow 1977). The problem can be solved exactly by discrete optimization methods with branch and bound or other forms of backtracking search. We have not yet encountered situations in which this would be needed. Instead, we provide incompatibility patterns that detect inconsistent interpretations and locally resolve them by heuristically choosing what components to eliminate.

Similarity-Based Language Understanding

The analytic approach can provide high accuracy but is knowledge intensive. To obtain wide domain coverage and graceful degradation, we have integrated a similarity-based method into our system. This approach does not provide a first-principles interpretation of a sentence. Rather, it computes the degree of lexical similarity between a message and a set of lexical (multiword) entities, returning the entities with highest similarity. The set of lexical entities we match against can be arbitrary documents or more specialized data such as product catalogues, geographical locations, institution names, and so on.

We define a lexical entity *e* as a tuple: *e* = <Q, R, C, O>. Here, Q is a reference question, R is the response to it, C is a set (corpus) of questions, C = {q_i}, such that each q_i has R as its response or is semantically a synonym of R. Finally, O is a list of ontology concepts established when this entity is recognized.

If *e* is a frequently asked question (FAQ), an example is Q(*e*) = "How do I contact you?" R(*e*) = "Call 1-800-555-1212," C(*e*) = {"Do you guys have a phone number?" "Where do I call you?"} and O(*e*) = {FAQ}

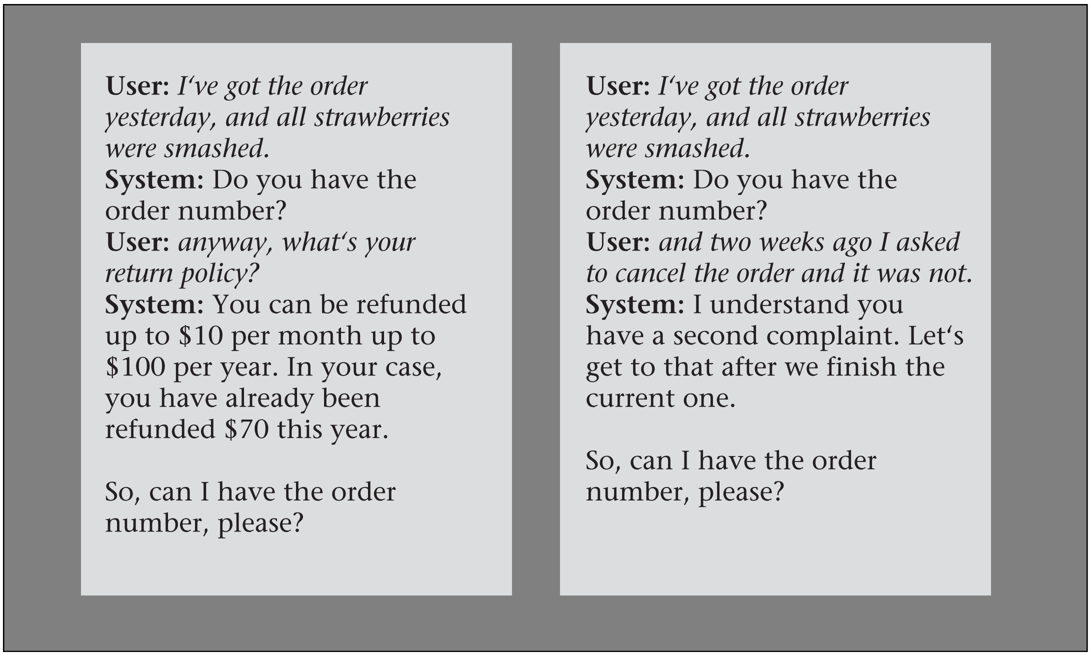
With *e* a geographical location, an example is Q(*e*) = "," R(*e*) = "Quebec, Canada," C(*e*) = {"Quebec," "La Belle Province"}, O(*e*) = {Location}.

Finally, with *e* a product name, an example is Q(*e*) = "," R(*e*) = "Red Rabbit" and C(*e*) = {"Red Rabbit by Tom Clancy," "Clancy's latest book"}, O(*e*) = {Product}.

Searching and Matching. The standard solution to recognizing the presence of such entities in a message is key word search, where key words consist of words from the Q, R, and C components after stop-word elimination, stemming, and synonym equivalence. Found words are scored based on measures such as inverse corpus frequency or fixed scores based on the part of speech (nouns and verbs score more than adjectives and adverbs). The final similarity score is computed in several ways depending on the application. These include the standard information retrieval *tf*idf* method (Salton and McGill 1983) (for FAQs) or more-specific methods that consider word order as well (for geographical location and product names). The implementation is based on the Jakarta Lucene public-domain search engine (jakarta.apache.org/).

Reinforcement Learning. The similarity approach relies on having a rather large and up-to-date corpus C for each entity. We use reinforcement learning (Watkins and Dayan 1992) to automate the construction and maintenance of this corpus. Given a set of entities E where each *e* = <Q, R, C, O>, reinforcement learning continuously updates a relevance measure of the value of each element in C(*e*) every time the entity *e* is being asked about. The low-valued elements are eventually forgotten, while new useful ones are being learned, guaranteeing that the corpus C(*e*) is always up to date.

Assume the user has input the question *q* = "Can I give a few bucks to the driver?" First, the best *n* matches to this are found. The reference questions are displayed and the user clicks the one conveying the intended meaning, for example, "Is tipping necessary?" Based on this information, we update the relevance scores for the questions in the corpus for this selected entity. Let q_m be the corpus question that actually did match. Its score is increased by *rele-*



User: *I've got the order yesterday, and all strawberries were smashed.*
System: Do you have the order number?
User: *anyway, what's your return policy?*
System: You can be refunded up to $10 per month up to $100 per year. In your case, you have already been refunded $70 this year.

So, can I have the order number, please?

User: *I've got the order yesterday, and all strawberries were smashed.*
System: Do you have the order number?
User: *and two weeks ago I asked to cancel the order and it was not.*
System: I understand you have a second complaint. Let's get to that after we finish the current one.

So, can I have the order number, please?

Figure 5. Switching between Deliberative and Reactive Behaviors.

$vance(q_m) <- d*relevance(q_m) + (1 - d)*1$. The other questions q_i were not useful, so their score is reduced by $relevance(q_i) <- d*relevance(q_i) + (1 - d) * (-1)$. In both cases, $0 <= d <= 1$ is a constant that determines how quickly we devalue past experiences. Finally, *q* is added to the corpus of the selected entity. If the size of the corpus for the selected entity exceeds a set limit, the lowest-scored question is removed.

Similarity-based language understanding has several uses in our system. It is the fallback approach when the analytic method fails. Often FAQ sets are used as the repository in this case, ensuring that the user will get some relevant answers. And it is also used in situations where first-principles methods are simply inapplicable. For example, to recognize product names from product catalogues, important in retail applications, or to recognize references to geographical locations in applications dealing with the delivery of goods or services.

Managing the Dialogue

The results of the language-understanding stage are expressed as frame structures that have slots for the information extracted from the input. An input like "I got the order delivered yesterday, and I noticed that the strawberries were all smashed" is expressed as a Quality-complaint frame, with strawberries as the product, physically damaged as the quality issue, and "26 Nov 2002" (say) as the date. This frame implies that the user's goal is to make a complaint and obtain a resolution. User goals are treated in two ways. Complex goals, like the quality complaint, are treated in a deliberative manner. The agent starts planning a conversation that will decide what information to elicit and what business policies to apply to find a mutually satisfactory resolution. Simple goals, usually requests for information ("Do you charge a fee for bounced checks?") are treated reactively, with the system providing the response and considering the issue terminated. The two types of goals are treated differently during the interaction. In the planned mode, the user can always ignore the current system question and ask a reactive question—this is immediately answered, and the planned mode is resumed (figure 5, left). If the request can be addressed only in planned mode, the currently executing planned interaction may or may not be interrupted, depending on the priorities of the respective goals (figure 5, right).

For each deliberative interaction, the Conversation Management module creates a workspace, which contains a local database and a goal stack initialized with the inferred user goal, for the Interaction Planner. When the workspace is active, goals are refined into subgoals, questions are asked, inferences are made, and business policies are applied. Several workspaces can be active in the same time, corresponding to various conversation topics. Mechanisms are provided for a workspace to inherit values from other workspaces, creating an interaction context that extends across the entire session. Reactive goals can generate responses sensitive to the current deliberative goal as context, as illustrated in figure 5.

The Interaction Planner

In a business environment, a planning approach to interaction is necessary to guarantee that the solution search space is systematically explored and that a solution is found if one exists. This is a strong argument for AI agents—they never tire in the search for the best solution and either find it or tell you if none exists.

We use a hierarchical task network (HTN) (Wilkins 1988) type of planning for its clarity and naturalness. In designing the interactive planning language, the issue has not been the complexity of the task networks or the size of the search space—in many business environments these have to be simple enough for people to manage. The issue has been conducting the dialogue such that the logical flow is clear, backtracking in the middle of the dialogue can be made comprehensible, and the agent keeps communicating what it is doing and why.

Hierarchical task networks consist of goal refinements into subgoals. Figure 6 illustrates a task network example from the wireless-service diagnosis application, together with an example customer-service conversation. Refinements marked with OR are disjunctive (at least one must be carried out), while the others are conjunctive and ordered (all must be carried out in the shown left-to-right order).

At the top level of the illustrated task network, troubleshooting consists of first acquiring a problem description followed by establishing the cause of the problem. There can be two classes of problems in the example, which we discuss briefly. Download problems are identified by a recorded attempt to download (through the air). The attempt to download can fail because the device has insufficient memory or because the device loses the connection in the process. In the former case, users are advised as to how to increase the memory on their telephone. The latter is treated as a report of network failure, and the service database is updated with the location where the failure was reported. If the download was successful, then the problem is local to the device, and the user is advised on how to deal with it.

Nondownload problems fall into several subclasses, and more than one can occur at the same time. The device may be incompatible with the service, there may be account restrictions (such as content rating), subscriptions may have run out, or the network may be down. Any of the established problems are explained to the user. This process is in fact a form of heuristic classification (Clancey 1985), a well-known problem-solving method applicable to diagnosis tasks.

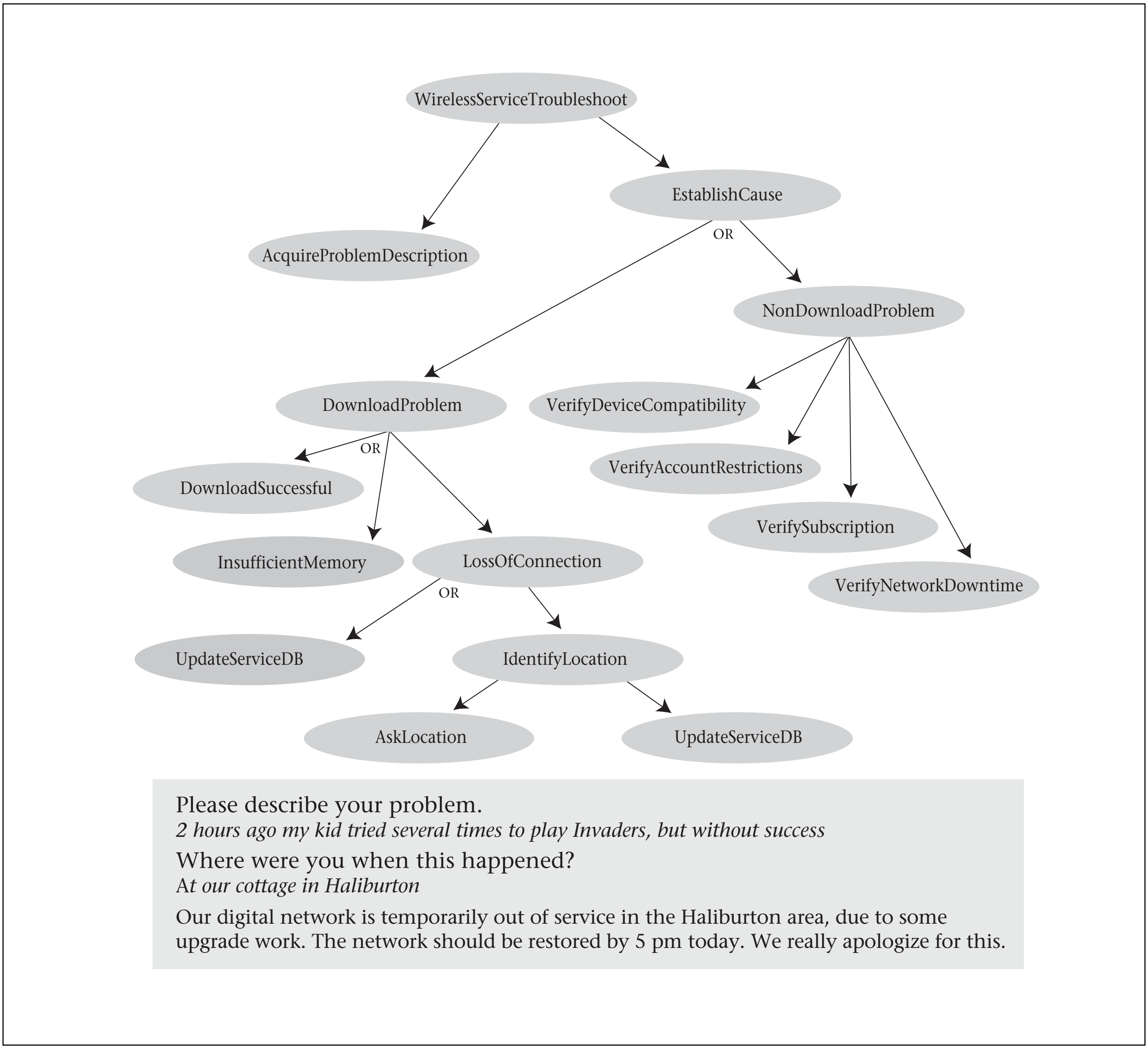
To achieve a goal, three types of operators are available, Ask, Infer, and Expand. The interactive operator (Ask) generates a question for the user (via the natural language generation module), obtains the interpretations of the response from Natural Language Understanding, and uses them to update the values of the planning variables associated with the current workspace. The AskLocation goal in figure 6 can be achieved by an Ask operator that generates the following example dialogue fragment:

> *Agent:* Where were you when you experienced the loss of connection?
> *User:* On Highway 400, south of Barrie.

In this case, "Highway 400, south of Barrie" is extracted as the location of the incident and stored in the current workspace, where it can be accessed by other operators. The main components of the Ask operator are: (1) a precondition in terms of the workspace variables and the current and past semantic frames; (2) a language-generation request for question formulation; (3) the function or method to interpret the response; and (4) the number of times the question can be reasked if the response is not understood. If this is exceeded without the answer being understood, the operator fails and backtracking is initiated.

The noninteractive operator (Infer) is used to perform deductions based on the existing values in the workspace. It is similar to Ask, but does not generate any questions. It may, however, inform the user about inferences drawn to further clarify the interaction.

If the goal cannot be achieved directly, it can be expanded into an ordered sequence of subgoals using the Expand operator. Figure 6 is composed of possible refinements as specified by this operator. This operator is similarly instrumented to clarify the dialogue flow. Inform-type messages can be specified for use when a goal is expanded into certain subgoals (for example, "I'm going to ask you now a few questions about your account"). Other Inform messages are for use when an expansion failed and backtracking is about to begin (for example, "I could not find your order based on the provided information") or when an expansion succeeded. The Expand operator also specifies the constraints that must be satisfied in order for the expansion to be considered successful. The constraints are of two kinds. Computed constraints are predicates among workspace variables (for example, that a subscription to a



Please describe your problem.

2 hours ago my kid tried several times to play Invaders, but without success

Where were you when this happened?

At our cottage in Haliburton

Our digital network is temporarily out of service in the Haliburton area, due to some upgrade work. The network should be restored by 5 pm today. We really apologize for this.

Figure 6. Task Network for Service Diagnosis and Example Conversation.

service by a user is still active). Interactive constraints replace the predicate by an accept-reject type of question addressed to the user, for example "Service upgrade during weekends is subject to an extra $5.00 per month. Would you like to upgrade?" Computed constraints are similar to Infer operators, while interactive ones are similar to Ask operators, except that they do not update workspace data.

This description applies to the deliberative behavior. The reactive behavior allows users to deviate for a short time from the planned interaction, ask their own questions, get the response, and continue with the planned interaction. To make the agent reactive, we use the React operator. This looks for unexpected semantic frames (coming from unexpected input) and answers them if the preconditions are satisfied. Reactions can be supported everywhere or in specified contexts, for example, when a certain (deliberative) goal is being pursued or during certain Ask executions. A reaction can specify the response messages to be generated and the side-effect actions to execute. In many cases, we use reactions to answer FAQs in context, as illustrated in figure 5.

The interaction planning system is aware of the interaction channel (Web, e-mail, or voice)

```
(BusinessRule
  (name fastest-delivery)
  (patter  "(shortest-lead-time-query (from ?from) (to ?to))"
             "?crt <– (current-shortest ?from ?to ?time ?mode ?cost)"
             "(delivery-lead-time-policy
              (from ?from)(to ?to)(lead-time ?lead)(price-per-kg ?pr))"
  (test "(< ?lead ?time)")
  (do "(retract ?crt)"
        "(assert (current-shortest ?from ?to ?lead ?pr))")).
```

Figure 7. Sample Business Rule.

on which the system functions. The differences among these channels require that we use the advantages and avoid the drawbacks of each. For example, plan to fill in a form in a browser, parse information from e-mails, or prompt and check pieces of information individually on the telephone. Channel-specific behaviors are created by specifying alternative goal expansions conditioned by the interaction channel. Voice interactions based on commercial voice-recognition engines are particularly limited by the size of the vocabulary that can be reliably recognized and are therefore handled in our applications by specific goal expansions.

Business Reasoning

Our architecture makes a clear distinction among the strategic level at which all interactions are managed by Conversation Management, the level of planning of goal-directed interactions by the Interaction Planner, and the domain reasoning level supported by the Business Reasoning module. This is consistent with previous work on knowledge acquisition and problem-solving methods (McDermott 1988) that has emphasized the recognition of the types and roles of knowledge as the basis of systematic development of AI applications.

The main purpose of the Business Reasoning module is to provide the framework for specifying and applying business-specific policies. In the wireless-service diagnosis example, a business policy is that accounts can not contain applications that violate defined content ratings. Business policies for e-commerce applications may specify lead times and cutoff times for order modification, various constraints on refunds and returns, and so on. The second purpose is to map the data model used by back-end systems into a format understood by our platform and to update the back end in a transactional manner.

The provided conceptualization for describing business reasoning consists of classes for business objects, business policies, business rules, queries, and actions. The system is used through an Ask and Tell interface. The Ask operator is used to pose a query that will be answered, while Tell is used to assert facts, which the system may act on at its discretion. For example, (ask shortest-lead-time Toronto Vancouver) asks for the shortest lead-time for a delivery from Toronto to Vancouver. To answer it, the system applies business rules that match queries with policies to produce answers. For an example, see figure 7.

Language Generation

The language-generation component creates the linguistic form for the system's questions and messages. We use a two-stage generation process. First, the individual messages requested by various planning operators are collected into several buckets. Second, these messages are assembled and edited in the final form to be seen by the user. The component messages can contain any number of Inform speech acts and a single Ask speech act. Each component message is created by instantiating a GenerationTemplate. The templates themselves are annotated with pragmatic attributes of the text they generate, for example *concise*, *verbose*, *polite*, and so on. There can be multiple templates generating the same information content in different forms. The choice of the actual template used depends on a score that reflects how well its pragmatic attributes match the attributes in the initial generation request (the latter are set by the planning operators and are based on the context and nature of the interaction). Randomization is used when scores are equal. The assembly and editing of the individual messages into the final form are performed by language-generation operators named GenerationPolicies. These can sort the components, filter out redundant ones, insert punctuation, format the text, and so on.

On interaction channels that use text (Web, e-mail), the system generates either simple text or HTML forms. On the voice channel, it generates VoiceXML (Boyer et al. 2000) scripts that play vocal prompts, define input grammars, and evaluate the voice response using a voice-recognition engine. The Natural Language Understanding module is bypassed for voice interactions, as we use the comparatively limited language-understanding ability of the voice-recognition engine. We currently use commercial voice-recognition engines provided by voice ASPs like Voxeo (www.voxeo.com), which also provide all the telephony interfaces. The voice ASP handles the voice exchange according to the VoiceXML script re-

	Gifts (Production)	Web Grocery (Beta)	Flowers (Beta)	Department Store (Client Evaluation)	Books and Music (Client Evaluation)
Correctly Solved	**66%**	**65%**	**52%**	**55%**	**64%**
Order Status	33%	10%	3.3%	–	20%
Order Change	2%	3%	4.6%	–	12%
Quality Issue	–	17%	–	–	–
Missing Items	–	16%	–	–	4%
Account Issues	15%	9%	4.9%	12%	6%
Product Availability	5%	10%	7.2%	–	1%
Delivery Inquiries	21%	2%	2.8%	–	4%
Business Policy Questions	3%	11%	56.4%	67%	11%

Table 1. Applications Based on the Retail Repository.

ceived from our server. Interestingly, adding VoiceXML scripts to the generation module was the only platform modification required to have our conversational approach function on the telephony network.

Ontology-Driven Development

Managing the application development process for this platform is challenging due to the diversity and specialization of the knowledge involved. We are addressing this problem by adopting an ontology-driven development approach. In essence, we develop an application by creating an ontology that contains frames for all the types of knowledge we need. This is possible because, as illustrated, we have defined frame structures for every type of knowledge in the system-linguistic components (such as object and event patterns, merge patterns), conversation and planning components (such as the Ask, Infer, Expand, and React operators), and language-generation and business-reasoning components.

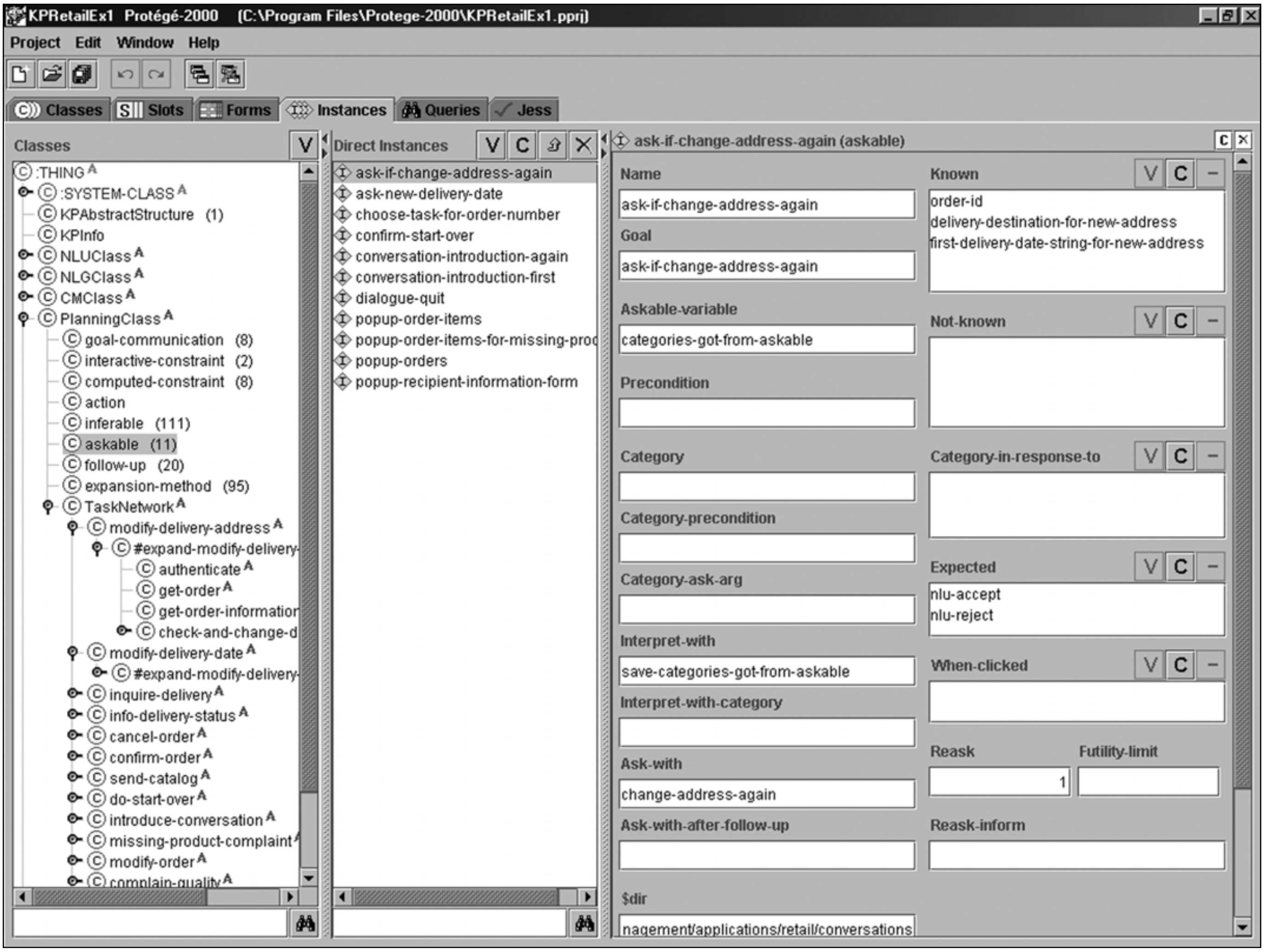
We use Protégé (Noy, Fergerson, and Musen 2000) as our visual ontology editor (figure 8). The frames manipulated in the development process are translated into the platform's native Jess (Friedman-Hill 2003) rule language objects via the Jess Tab included in Protégé. Once translated into Jess objects, the normal processing takes place. Some objects are interpreted by Jess rules to provide the desired functionality, for example, the planning and the language-generation operators. Others are compiled into Jess rules using Jess's ability to generate rules dynamically. This is how the natural-language and business-reasoning components are treated.

The advantage of this approach is the ability to represent a large knowledge typology in a uniform frame format for which a sophisticated visual knowledge editor exists. The editor makes it possible to develop industry-specific repositories of components, such as the retail and telecommunication repositories from which we used several examples throughout the article.

Evaluation

The retail repository was used to create several applications, currently in different stages of deployment (table 1). Each column in table 1 shows an application, with each row representing a service issue dealt with by the application. Client Evaluation means the application has been evaluated by the client company prior to beta testing on site. For each issue (row), we show, as a percentage, how often it occurs in interactions in the application. (As we show only the main issues, they sum to less than 100 percent for each application.) The Correctly Solved row shows what total percentage (across all issues in an application) was correctly solved by the system. An 80/20 rule was apparent: a few issues dominate the majority of interactions. This is good news because it helps focus natural language and problem solving on those issues.

To cope with the remaining issues, we have



```
C :THING A
  C :SYSTEM-CLASS A
  C KPAbstractStructure (1)
  C KPInfo
  C NLUClass A
  C NLGClass A
  C CMClass A
  C PlanningClass A
    C goal-communication (8)
    C interactive-constraint (2)
    C computed-constraint (8)
    C action
    C inferable (111)
    C askable (11)
    C follow-up (20)
    C expansion-method (95)
    C TaskNetwork A
      C modify-delivery-address A
        C #expand-modify-delivery-
          C authenticate A
          C get-order A
          C get-order-informatior
          C check-and-change-d
      C modify-delivery-date A
        C #expand-modify-delivery-
      C inquire-delivery A
      C info-delivery-status A
      C cancel-order A
      C confirm-order A
      C send-catalog A
      C do-start-over A
      C introduce-conversation A
      C missing-product-complaint A
      C modify-order A
      C complain-quality A
```

```
ask-if-change-address-again
ask-new-delivery-date
choose-task-for-order-number
confirm-start-over
conversation-introduction-again
conversation-introduction-first
dialogue-quit
popup-order-items
popup-order-items-for-missing-proc
popup-orders
popup-recipient-information-form
```

ask-if-change-address-again (askable)

Name	ask-if-change-address-again
Goal	ask-if-change-address-again
Askable-variable	categories-got-from-askable
Precondition	
Category	
Category-precondition	
Category-ask-arg	
Interpret-with	save-categories-got-from-askable
Interpret-with-category	
Ask-with	change-address-again
Ask-with-after-follow-up	
$dir	nagement/applications/retail/conversations
Known	order-id delivery-destination-for-new-address first-delivery-date-string-for-new-address
Not-known	
Category-in-response-to	
Expected	nlu-accept nlu-reject
When-clicked	
Reask	1
Futility-limit	
Reask-inform	

Figure 8. Ontology Editor Screen (Protégé).

	Telecommunications Solutions Vendor (Client Evaluation)	Consulting Company (Client Evaluation)
Correctly Solved	**74%**	**85%**
Diagnosis and explanation of wireless service issues	100%	–
Wireless billing questions	–	100%

Table 2. Applications Based on the Telecommunications Repository.

included in each application a base of FAQs recognized through similarity matching.

Table 2 shows similar results for the telecommunications repository. Here we have two different applications. The diagnosis one uses heuristic classification (Clancey 1985) as the problem-solving method and leads to longer conversations. The wireless billing application has a broad but shallow one question–one answer nature and was done by similarity methods.

The critical aspect in all applications is the ability to understand language. We monitor and assess this constantly on the basis of graphs like the one shown in figure 9 (a typical week from the production Gifts application). We distinguish in-scope questions (those that the knowledge base is prepared to deal with) from out-of-scope ones, but we try to extend the scope by similarity methods, currently answering about 30–40 percent of questions overall.

These results are generally satisfactory for a low-cost "self-serve" customer service solution, as they show the ability of the automated system to reduce call-center costs and give customers more access to information.

There is also considerable room for improvement, which we do in several ways. The industry-specific repository approach based on ontology modeling languages allows us to improve our language-understanding ability by creating *reusable* concepts and semantic patterns. The lower-accuracy but wide-coverage similarity-matching method provides a simple, nontechnical way to extend the scope of the system dynamically. Finally, we are extending the similarity-matching approach with statistical text-classification methods (Dumais et al. 1998), such as Support Vector Machines, that can learn to perform more-complex mappings between text and abstract categories than possible with the standard information-retrieval model.

Lessons Learned and Conclusions

Paradoxically, designers of conversational customer-interaction systems do not control the scope of their system. Users do. People will ask about anything they need to know or do, without caring about the "scope" of the system as defined by its designers. If the e-commerce site being serviced happens to be slow for whatever reason, the customer service agent will get complaints about it. If any information is missing or unclear on the site, the agent will get questions about it. The scope of the system is always "anything commonsensically related to the business." Achieving such breadth through detailed semantic language analysis is very hard, if at all possible. For this reason, we are always aiming for a "bell curve" relation between the breadth and depth of understanding. Business subareas that require deep understanding (for example, quality complaints in retail) are supported by elaborated linguistic patterns. Other subareas are supported by the less-accurate but wide-coverage similarity method that taps into the existing document base of any business. This graceful degradation property ensures that people get an answer in most cases, even if an approximate one. Experience has shown that users are more forgiving if they get an approximate or partially relevant response than no response at all. Finally, if none of these works, clear provisions must be made to connect customers to a live representative before they run out of patience. A good rule of thumb is to be risk adverse: every time the answer has been determined by similarity search, insert information for live-person connection.

The second lesson has to do with how responses are written. In the absence of nonverbal communication, gestures that play such an important role in interhuman interaction, the connotations contained in the textual responses must be weighed carefully. Freshness, variety, and being cute when you don't understand will always help.

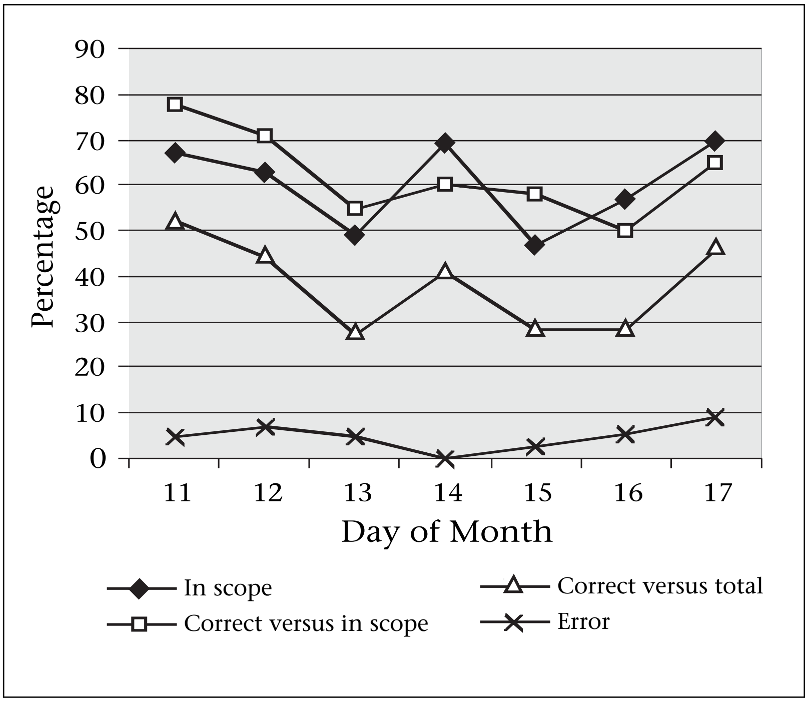


Figure 9. Natural Language Understanding Performance.

The third lesson is about the role of AI. Many previous applied AI systems have reported the "disappearance of AI" phenomenon— fewer and fewer AI technologies were used as the system was being deployed commercially. Our to-date experience has been that this need not be the case—our current system contains a good assortment of integrated AI solutions, each applied to significant parts of the overall task.

One reason this was possible is the fourth lesson learned, that we couldn't have done it without using an AI programming language. Most of the system is written in Jess, a Java-embedded descendant of OPS5 and Clips. Properly understood and used, rule-based programming and AI-style "second-order programming" based on constructs like *eval*, *apply,* and the Jess *build* are invaluable for writing and integrating complex algorithms in a concise and timely manner. And their integration with Java gives us unconstrained access to all the Internet and other programming resources ever needed.

The platform has been used to create customer interaction agents in industry verticals such as wireless-service provisioning and several retail domains. These have undergone extensive beta testing; some of them have been in production, and more are to come. The jury is still out with respect to whether we have managed to build the knowledgeable and infinitely patient agents able to provide the frictionless customer-service experience envisioned. However, we strongly believe that this will happen only at the end of a serious integration effort aimed at putting together a wide range of AI, Internet, and conventional technologies.

References

Allen, J. F.; Byron, D. K.; Dzikovska, M.; Ferguson, G.; and Galescu, L. 2001. Toward Conversational Human Computer Interaction. *AI Magazine,* 22(4)(Winter): 27–37.

Appelt, D. E.; Hobbs, J. R.; Bear, J.; Israel, D.; and Tyson, M. 1993. FASTUS: A Finite-State Processor for Information Extraction from Real-World Text. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. San Francisco: Morgan Kaufmann Publishers.

Borgida, A.; Brachman, R. J.; McGuiness, D.; and Resnick, L. A. 1988. CLASSIC: A Structural Data Model for Objects. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 59-67. New York: Association for Computing Machinery Special Interest Group on Management of Data.

Boyer, Linda; Danielsen, Peter; Ferrans, Jim; Karam, Gerald; Ladd, David; Lucas, Bruce, and Rehor, Kenneth. 2000. Voice Extensible Markup Language (VoiceXML) W3C Note 5 May. (www.w3.org/TR/voicexml). Piscataway, N.J.: VoiceXML Forum

Clancey, W. J. 1985. Heuristic Classification. *Artificial Intelligence* 27(3): 289-350.

Friedman-Hill, E. 2003. *Jess in Action*. Greenwich, Conn.: Manning Publications Company.

McDermott, J. 1988. A Taxonomy of Problem Solving Methods. In *Automating Knowledge Acquisition,* 225–226, ed S. Marcus. Dortrecht, Holland: Kluwer Academic Press.

Dumais, S. T.; Platt, J.; Heckerman, D.; and Sahami, M. 1998. Inductive Learning Algorithms and Representation for Text Categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management,* 148–155. New York: ACM Press.

Fox, M. S.; and Mostow, J. 1977. Maximal Consistent Interpretations of Errorful Data in Hierarchically Modeled Domains. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* 165–171. Los Altos, Calif.: William Kaufmann, Inc.

Noy, N. F.; Fergerson, R. W.; and Musen, M. A. 2000. The Knowledge Model of Protégé 2000: Combining Interoperability and Flexibility. In *Proceedings of the Twelfth European Workshop on Knowledge Acquisition, Modeling and Management,* 17–32. Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Salton, G.; and McGill, M. 1983. *Introduction to Modern Information Retrieval*. New York: McGraw Hill.

Watkins, C. J. C. H.; and P. Dayan, P. 1989. Q-learning. *Machine Learning* 8(3): 279-292.

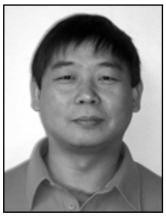
Wilkins, D. 1988. *Practical Planning*. San Mateo, Calif.: Morgan Kaufmann Publishers.

Mihai Barbuceanu's research interests are in creating theories and models of interaction and coordination between human and artificial agents. He has created one of the first agent-coordination languages based on a conversational model; modeled joint work in organizations using deontic relationships; and applied decision theory, game theory, and constraint optimization to develop a negotiation method that achieves optimal allocations of goods and services. Recently, his work has been focused on human computer interaction using written and spoken natural language. He is the principal architect of a commercial framework for conversational customer-interaction systems that integrate knowledge-based and statistical methods for language understanding, information retrieval, commonsense and specialized domain ontologies, multitopic dialogue management, and hierarchical task-planning methods. Barbuceanu received his Ph.D. from the Technical University of Bucharest. He can be reached at mihai at novator.com.

Mark S. Fox cofounded and is chairman and chief executive officer of Novator Systems Ltd., a provider of e-commerce services and software since 1994. In October 2003, Fox launched ChocolatePlanet.com, which will provide same-day delivery of premium chocolates across North America. Fox is also a professor of industrial engineering at the University of Toronto, where his research focuses on enterprise integration and artificial intelligence. Fox is known for his pioneering work in constraint-directed scheduling, which underlies current logisitics, supply-chain management, and scheduling solutions, and for his work in ontologies for enterprise modeling. In 1984 Fox cofounded Carnegie Group Inc. (CGIX on Nasdaq until 1998), a software company that specialized in intelligent systems for solving engineering, manufacturing, and telecommunications problems. In 1975 Fox received his B.Sc. in computer science from the University of Toronto, and in 1983 he received his Ph.D. in computer science from Carnegie Mellon University, where he was an associate professor of computer science and robotics until his return to Toronto in 1991. He is a Fellow of the American Association for Artificial Intelligence. His e-mail address is msf at eil.utoronto.ca.

Lei Hong received her B.Sc. degree from Harbin University of Technology, China, in 1990, and a Master's degree from University of Toronto, Canada, in 2000. In 1990 she joined China Aerospace Industry Inc., where she spent most of her time on real-time control-system development for various industrial applications. From 2000 to 2003 she worked for Novator Systems Inc., where she actively worked on e-commerce applications. Hong's main interests include software development for intelligent systems.

Yannick Lallement received his Ph.D. in computer science from the University of Nancy I, France, in 1996. He later joined the Human Computer Interaction Institute at Carnegie Mellon University, developing models of human performance in Soar. He is now with Novator Systems in Toronto, working on automated customer-care solutions. His interests focus on artificial intelligence, cognitive modeling, and human-computer interaction. His e-mail address is yannick at novator.com.



Zhongdong Zhang received his Ph.D. from the Department of Computer and Information Science, University of Constance, Germany, in 1998. He is currently working on e-commerce and automated customer-care solutions. His research interests include Web-based information systems, text categorization/information extraction, and Web analytics. His e-mail address is zhang at novator.com.