Building Agents for the Customer Service Front

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

Novator Systems Ltd.
364 Richmond Street West, Suite 300
Toronto, M5V 1X6, Canada
{mihai, msf, lhong, yannick, zhang}@novator.com

Abstract

AI has the potential to play an important role in the customer service field. By leveraging the high bandwidth of natural language customers will be able to state their intentions directly instead of searching for the places on the Web site that may address their concern. By using a planning system to find the solution to the problem we increase the confidence that a resolution satisfactory for both the customer and the company will be found, if one exists. Being able to converse guarantees that needed information will be acquired from customers and relevant information will be provided to them in order for both parties to make the right decision. The net effect is a more frictionless interaction process that will improve customer experience and make businesses more competitive on the service front.

Introduction

As companies optimize their production and supply chain processes, customer service emerges as the new competitive battleground. 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 result in long telephone queues and frustrated customers. As it generally costs 5 times more to acquire a new customer than to keep an existing one, frustrating a customer is hardly an option for anybody.

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 or are applying this platform to customer service applications like 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 paper.

Understanding, Interaction and Resolution

Compared to the "newspaper page" model of web sites where 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 the user 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 will be found that can satisfy both the user and the company. Being able to converse guarantees that the relevant information will be 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 common-sensically related to its main competence area. Therefore, graceful degradation must be provided, supported by a critical minimum of common sense knowledge around the main competence area.

The Internet currently reaches a variety of touch points through which the agent must be available, with the web, email and phone being the most important. 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 (e.g. 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.

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 on-line and downloaded through the air for use on Java enabled cell phones.

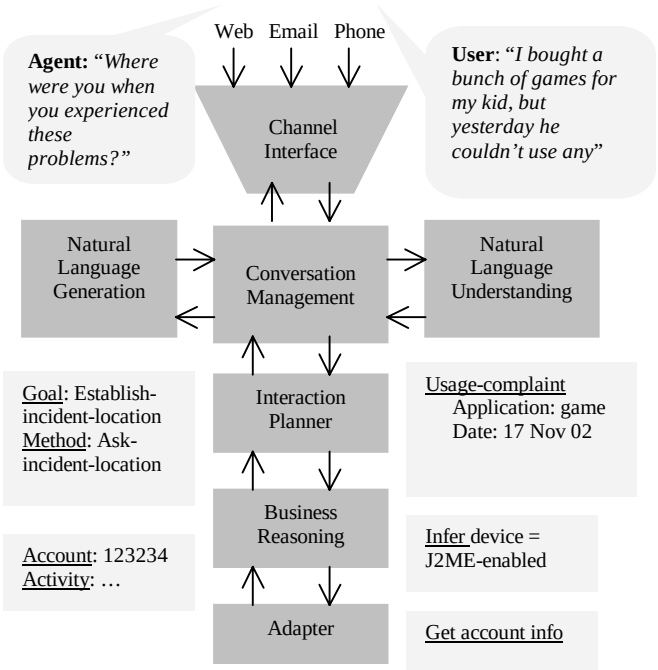


Figure 1. Overall Agent Architecture

The user input (from no matter what channel) is first converted into an internal message format. From this, Natural Language Understanding 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 identifies whether the issue can be answered immediately, or 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 sub-goals and perform backtracking search to satisfy them. If the user complains about not being able to use such a game, the planner will create sub-goals for applying a diagnosis method to determine the cause. Normally the investigation would start with obtaining user account information from the backend and determining the conditions under which the incident occurred. The Business Reasoning module performs domain inferences like determining that the user's device must be J2ME enabled and applies business policies like verifying content rating restrictions on user accounts (in the process accessing account information from the backend). In the figure, the incident location was not provided in the input, thus has to be asked for. The `Ask-incident-location` method used for the `Establish-incident-location` goal achieves this 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 integrating 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 highest similarity. This is used to provide general answers from an arbitrary set of pre-existing 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.

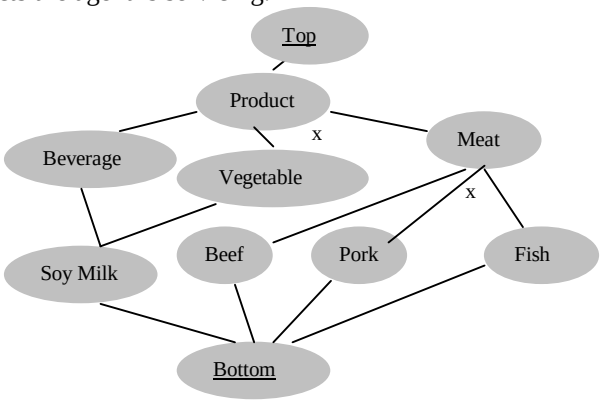


Figure 2. Classified concept lattice.

We use description logics (Borgida et al. 1989) to represent ontologies. Description logics provide automated concept classification. Figure 3 shows a very small, classified product lattice, containing disjoint sub-concepts (that do not have common instances, marked with an "x" in the figure, e.g. `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.

Knowledge Based Language Understanding

By this we mean a first principles method that uses linguistic and domain knowledge to understand natural language expressions. Our method processes the linguistic input by applying a sequence of steps as follows: 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 (noun, verb, adjective, adverb, preposition, determiner, as well as the tense and the singular or plural form). Regular grammars are applied to recognize URL-s, emails, phone number, dates, addresses, person and institution names, and a few others.

Syntax Analysis. Syntax analysis is limited to the recognition of structures that can be reliably recognized. Based on the Fastus approach (Applet et al. 1993), we only recognize 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 "2 pounds of brown spotted Bonita bananas". The verb group consists of a verb together with its auxiliaries and adverbs. For example, the group for the verb "deliver" in "2 pounds of brown spotted bananas were delivered late yesterday evening". Verb groups are tagged as active, passive, infinitive, gerund or negated (e.g. "were not delivered"). Noun and verb groups are scored in a manner that increases with the percentage of the input they account for. This biases the system towards preferring longer subsuming phrases to the shorter ones. However, no phrase is discarded.

Semantic Analysis. The next step is to recognize instances of ontology concepts. At the topmost level, 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 sub-concepts. 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 sub-concepts in the classified ontology. Figure 3 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 produce-s in exactly the same way by this pattern.

Recognized objects and events can be used in patterns to recognize further objects and events. For example, the EventPattern in fig. 3 is applicable to recognize complaints of the type "<produce> is <not fresh>" or "is <not fresh> <produce>" (because the seq predicate allows both of these orders). The certainty of an object or event recognized is computed by a function monotonic in the certainty of the components and the certainty coefficient of the used pattern.

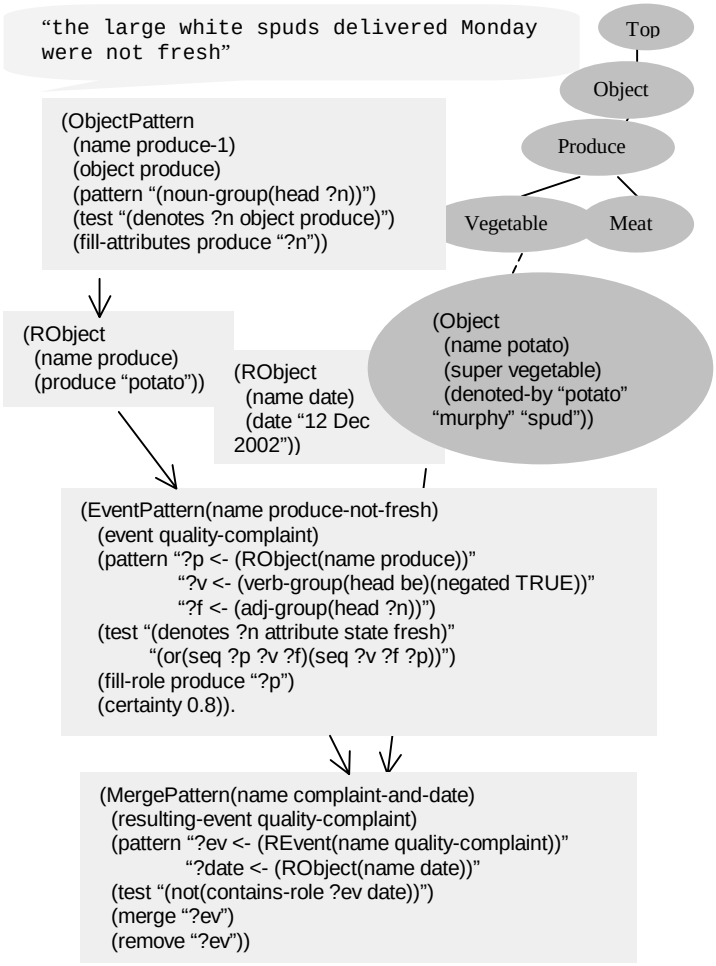


Figure 3: Semantic analysis and merging.

Merging. After applying the patterns in the semantics stage, the merging stage combines together objects and events into higher order aggregates. In figure 3, a quality-complaint event (not containing the role date) and a date object are merged into a new quality-complaint event. The new quality-complaint event contains the union of the roles and attributes of the merged objects and events. Merge patterns solve most of the pronominal references to entities occurring in the message, as e.g. 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 language generation module, and 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 in 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 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 where this would be needed. Instead, we provide incompatibility patterns that can be specified to 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 (multi-word) entities, returning the entities with highest similarity. The set of lexical entities we match against can be arbitrary documents or more specialized data like product catalogues, geographical locations, institution names, etc.

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 an 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 Provence"}, 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 like inverse corpus frequency or fixed scores based on the part of speech (nouns and verbs scored more than adjectives and adverbs). The final similarity score is computed in several ways depending on the application. These include the vector space method (Salton and McGill 1983) (for FAQ-s) or more specific methods that consider word order as well (for geographical location and product names).

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 point and clicks on the one conveying the intended meaning, e.g. "*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 $relevance(q_m) <- d*relevance(q_m) + (1-d)*1$. 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 exceeded 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've got the order delivered yesterday and I've noticed that the strawberries were all smashed*" is expressed as a QualityComplaint 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 NSF 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 4, left). If the request can only be addressed in planned mode, the currently executing planned interaction may or may not be interrupted, depending on the priorities of the respective goals (figure 4, right).

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?

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?

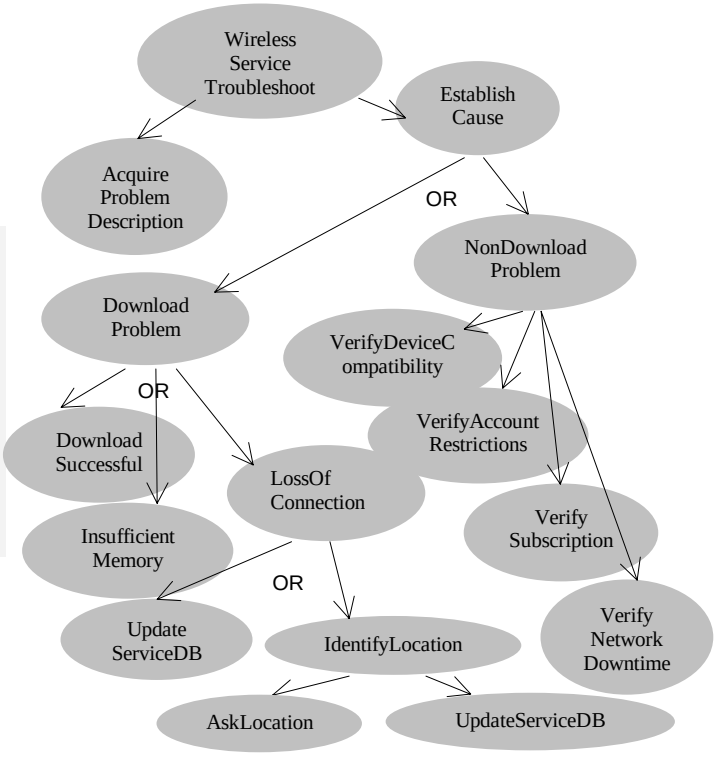
Figure 4. Switching between deliberative and reactive behaviors.

For each deliberative interaction, the Conversation Management module creates a workspace for the Interaction Planner, containing a local database and a goal stack initialized with the inferred user goal. 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 form 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 4.

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 a solution will be found if one exists. This is a strong argument for AI agents – they will never tire in the search for the best solution and will 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 environment 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.



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 5. Task network for service diagnosis and example conversation.

Hierarchical task networks consist of goal refinements into subgoals. Figure 5 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, troubleshooting consists of first acquiring a problem description followed by establishing the cause of the problem. There can be two classes of problems.

Download problems are identified by a recorded attempt to download (through the air). This can fail because of insufficient memory on the device or because of loss of connection in the process. In the former case the user is advised as to how to increase memory on their phone. 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. Non-download problems fall in several sub-classes and more than one can occur in the same time.

The device may be incompatible with the service, there may be account restrictions (e.g. content rating), subscriptions may have run out or the network might have been down. Any of the established ones 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 5 can be achieved by an Ask operator that will generate 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, "*Hwy 400 south of Barrie*" is extracted as the location of the incident and stored in the current workspace, where can be accessed by other operators. The main components of the Ask operator are:

- A precondition in terms of the workspace variables and the current and past semantic frames.
- A language generation request for question formulation.
- The function or method to interpret the response.
- The number of times the question can be re-asked if the response is not understood. If this is exceeded without the answer being understood, the operator fails and backtracking is initiated.

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

If the goal can not be achieved directly, it can be expanded into an ordered sequence of subgoals using the Expand operator. Figure 5 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 (e.g. "*I'm going to ask you now a few questions about your account*"). Other informs are for use when an expansion failed and backtracking is about to begin (e.g. "*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 amongst workspace variables (e.g. that a subscription to a service by an 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 week-ends 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 the user 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: when a certain (deliberative) goal is being pursued or during certain Ask executions. A reaction can specify response messages to be generated and/or side effect actions to be executed. In many applications we use reactions to answer FAQ-s in context, as illustrated in figure 4.

The interaction planning system is aware of the interaction channel (web, email or voice) on which the system functions. The differences between these channels require that we use the advantages and avoid the drawbacks of each. For example, plan to fill a form in a browser, parse information from emails, or prompt and check pieces of information individually on the phone. Channel specific behaviors are created by specifying alternative goal expansions conditioned by the interaction channel.

To complete the presentation, two more modules need to be discussed. The purpose of the Business Reasoning module is to allow the specification and application of business policies. In the wireless service diagnosis example, a business policy is that accounts can not contain applications violating defined content ratings. Business policies for retail applications may specify lead times and cutoff times for order modification, various constraints on refunds and returns etc. Business reasoning models are specified by classes for business objects, business policies, business rules, queries and actions.

The introduction of this module clarifies the distinction between the strategic level at which all interactions are managed by Conversation Management, the level of planning 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 second module is the language generation component tasked with creating 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 generated and collected into several buckets. Each component message is created by instantiating a template. Templates are annotated with pragmatic attributes of the text they generate, for example *concise*, *verbose*, *polite*, etc. The choice of the actual template used depends on a score that measures how many of the requested attributes are provided. In the second stage, generation policies assemble and edit these messages in the final form to be seen by the user, by sorting the components, filtering out redundant ones, inserting punctuation, formatting the text, etc.

On interaction channels that use text (web, email), the system generates either simple text or HTML forms. On the voice channel it generates VoiceXML (W3C 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 ASP-s like Voxeo (www.voxeo.com). Interestingly, adding VoiceXML scripts to the generation module was the only platform modification required to have our conversational approach function on the telephony network.

Evaluation

The retail repository was used to create several applications, currently in different stages of deployment, as shown in table 1 (where Client Evaluation means evaluation by the client company, prior to beta testing on site). For each application, we show the major customer service issues addressed. For each issue, we show how often it occurs in interactions in every application. The **Solved** row shows what total percentage (across all issues) was successfully 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 rest of issues, we have included each time a base of FAQ-s 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 (McDermott 1988) 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 table 3 (a typical week form the production Gifts application). We distinguish in scope questions (those which the knowledge base is prepared to deal with) from out of scope ones, but we try to extend the scope by similarity and analytic methods.

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 two ways. The industry specific repository approach (based on ontologies) allows us to improve our language understanding ability by creating *reusable* concepts and semantic patterns. And the lower accuracy but wide coverage similarity matching method provides a simple, non technical way to extend the scope of the system dynamically.

	Gifts (Production)	Web Grocery (Beta)	Flowers (Beta)	Dept. Store (Client Eval.)	Books & Music (Client Eval.)
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.

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

Table 2. Applications based on the telecommunications repository.

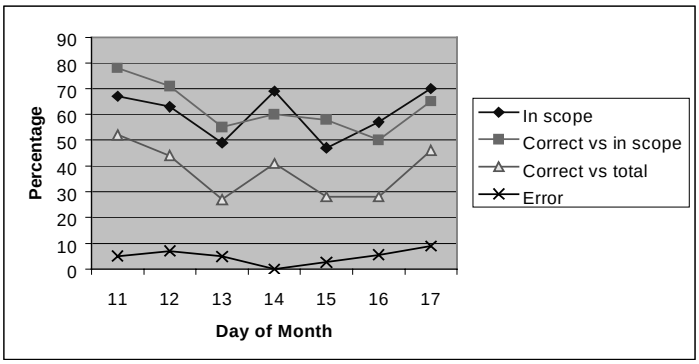


Table 3. Natural Language Understanding performance.

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 happened 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 common-sensically related to the business". Achieving such breadth through detailed semantic 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 sub-areas that require deep understanding (e.g. quality complaints in retail) are supported by elaborated linguistic patterns. Other sub-areas 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 less precise. 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 the customer to a live representative, before his or her tolerance is exhausted.

The second lesson has to do with how responses are written. In the absence of non-verbal communication gestures that play such an important role in inter-human 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.

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. We believe that this happened because the initial goals of the system were diluted in the process. Our to date experience has been the opposite and our current system contains a quite wide assortment of integrated AI solutions.

One reason why 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 (Friedman-Hill 2003), 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 gave us unconstrained access to all Internet and other programming resources ever needed.

The platform has been used to create customer interaction agents in industry verticals like 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 only happen 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. Galescu, L. 2001. Toward Conversational Human Computer Interaction. AI Magazine, Winter 2001, 27-37.

Applet, D.E. Hobbs, J.R. Bear, J. Israel, D. Tyson, M. 1993. FASTUS: A Finite-State Processor for Information Extraction from Real-World Text. Proceedings of IJCAI-93, Chamberey, France, August 1993.

Borgida, A. Brachman, R.J. McGuiness, D. Resnick, L.A. 1989. CLASSIC: A Structural Data Model for Objects. Proc. 1989 ACM SIGMOD International Conference on Management of Data, June 1988, 59-67.

Clancey, W.J. 1985. Heuristic Classification. Artificial Intelligence 27, 289-350.

Friedman-Hill, E. 2003. "Jess in Action", Manning Press, 2003.

McDermott, J. 1988. A taxonomy of Problem Solving Methods. In S. Marcus (ed): Automating Knowledge Acquisition, Kluwer Academic Press, 1988, 225-226.

Fox, M.S. and Mostow, J. 1977. Maximal Consistent Interpretations of Errorful Data in Hierarchically Modeled Domains. Proceedings of IJCAI-77, Morgan Kaufmann, 1977, 165-171.

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

Watkins, C.J.C.H. and Dayan, P 1992. Q-learning. Machine Learning 8(3), 279-292.

Wilkins, D. 1988. Practical Planning. Morgan Kaufmann San Mateo, CA.

W3C 2000. Voice Extensible Markup Language (VoiceXML) W3C Note 5 May 2000, http://www.w3.org/TR/voicexml.