Natural Language Understanding in Façade: Surface-Text Processing

Michael Mateas*[1] and Andrew Stern*[2]

* co-authors listed alphabetically
[1] College of Computing and LCC, Georgia Tech,
michaelm@cc.gatech.edu
[2] InteractiveStory.net
andrew@interactivestory.net

1 Introduction

Façade is a real-time, first-person dramatic world in which the player, visiting the married couple Grace and Trip at their apartment, quickly becomes entangled in the high-conflict dissolution of their marriage. The *Façade* interactive drama integrates real-time, autonomous believable agents, drama management for coordinating plot-level interactivity, and broad, shallow support for natural language understanding and discourse management. In previous papers, we have described the motivation for *Façade's* interaction design and architecture [13, 14], described ABL, our believable agent language [9, 12], and presented overviews of the entire architecture [10, 11]. In this paper we focus on *Façade's* natural language processing (NLP) system, specifically the understanding (NLU) portion that extracts discourse acts from player-typed surface text.

The *Façade* NLP system accepts surface text utterances from the player and decides what reaction(s) the characters should have to the utterance. For example, if the player types "Grace isn't telling the truth", the NLP system is responsible for determining that this is a form of criticism, and deciding what reaction Grace and Trip should have to Grace being criticized in the current context. General natural language understanding is of course a notoriously difficult problem. Building a system that could understand open-ended natural language utterances would require common sense reasoning, the huge open-ended mass of sensory-motor competencies, knowledge and reasoning skills which human beings make use of in their everyday dealings with the world. While *Façade* is a micro-domain, a dramatically-heightened representation of a specific situation, not the whole world, there are still no general theories, techniques or systems which can handle the syntactic, semantic and pragmatic breadth of the language use which occurs in *Façade*. Instead, *Façade* makes use of broad, shallow, author-intensive techniques to understand natural language typed by the player.

Our approach in *Façade* it to view the natural language understanding problem as a dialog management problem, focusing on the pragmatic effects of language (what a language utterance does to the world) rather than on the syntax (the form of surface text) or semantics (meaning) of language. In dialog management systems (e.g. Collagen [19, 7], Trains [1, 4]), language use situations are seen as consisting of a discourse context within which conversants exchange speech acts. A conversant's sur

S. Göbel et al. (Eds.): TIDSE 2004, LNCS 3105, pp. 3-13, 2004.

face utterance is interpreted *as* a speech act in a manner dependent on the current discourse context. In *Façade*, discourse acts have been chosen that are appropriate for the conflict-filled interaction with Grace and Trip. This approach of choosing discourse acts that align with player narrative actions has similarities to the approach described in [2]. The discourse context is determined by dramatic beats, the smallest unit of dramatic action, and the "atoms" out of which *Façade* stories are constructed [10].

NLP is divided into two phases: phase 1 maps surface text into discourse acts representing the pragmatic effects of an utterance, while phase 2 maps discourse acts into one or more (joint) character responses. Where phase 1 determines, in a context specific way, *which discourse act* was produced by the player, phase 2 determines *what effect* this act will have, including, potentially, how it changes the discourse context. It is within Phase 2 that a stack of discourse contexts is actively maintained and used to determine a context-appropriate response to a given discourse act. Phase 1 only makes use of discourse contexts insofar that different collections of phase 1 rules may be turned on and off in different contexts. The rest of this paper focuses on the phase 1 (NLU) pipeline.

2 Introduction to Surface Text Rules

Forward chaining rules map surface text to discourse acts. Some rules map specific patterns of surface text directly to intermediate meaning representations, while other rules combine intermediate meanings to form more complex meanings. Eventually the process of mapping islands of surface text into intermediate representations, and combining these representations, produces the final meaning, consisting of one or more discourse acts.

```
"hello" → iGreet
"grace" → iCharacter(Grace)
iGreet AND iCharacter(?x) → DAGreet(?x)
```

Fig. 1. Pseudo-code for simple greeting rules

For example, imagine that the player types "Hello Grace". The first rule in Figure 1 matches on the appearance of the word "hello" anywhere in the text and asserts an `iGreet` intermediate fact. The second rule matches on the appearance of the word "grace" anywhere in the text and asserts an `(iCharacter Grace)` fact. The third rule matches on the occurrence of the two intermediate facts and asserts a `(DAGreet Grace)` discourse act fact, indicating that a greeting was directed at Grace. The rule makes use of a variable; `?x` binds to the argument of `iCharacter` on the left hand side and brings this value over to the assertion of `DAGreet` on the right hand side of the rule. In order to capture more ways of saying hello (e.g. "how're you doing?", "how's it going?", "what's up", etc.), additional `iGreet` rules can be written without changing the `iCharacter` or `DAGreet` rules.

3 Discourse Acts

Our current set of discourse acts appears in table 1 below. All player utterances map into one or more of these discourse acts. Phase 1 processing is a strong many-to-few mapping – the huge, rich range of all possible strings a player could type is mapped onto this small set of discourse acts. Besides ignoring any nuance in the tone of the player's text, the system also focuses almost exclusively on the pragmatics of the utterance, ignoring most of the semantics (denotative meaning).

Table 1. *Façade* discourse acts

Representation of Discourse Acts	***Pragmatic Meaning of Discourse Acts***
`(DAAgree ?char)`	Agree with a character. (e.g. "certainly", no doubt", "I would love to")
`(DADisagree ?char)`	Disagree with a character. (e.g. "No way", "Fat chance", "Get real", "Not by a long shot")
`(DAPositiveExcl ?char)`	A positive exclamation, potentially directed at a character. ("Yeah", "Wow", "Breath of fresh air")
`(DANegExcl ?char)`	A negative exclamation, potentially directed at a character. (e.g. "Damn", "That really sucks", "How awful", "I can't stomach that", "D'oh!")
`(DAExpress ?char ?type)`	Express an emotion, potentially directed at a character. The emotion types are *happy* ("I'm thrilled", ":-)"), *sad* ("That bums me out", ":-("), *laughter* ("ha ha"), and angry ("It really pisses me off", "grrrr").
`(DAMaybeUnsure ?char)`	Unsure or indecisive, potentially directed at a character. This discourse act is usually a response to a question. (e.g. "I don't know", "maybe", "I guess so", "You've lost me")
`(DAThank ?char)`	Thank a character (e.g. "Thanks a lot")
`(DAGreet ?char)`	Greet a character. (e.g. "Hello", "What's up")
`(DAAlly ?char)`	Ally with a character. (e.g. "I like you", "You are my friend", "I'm here for you")
`(DAOppose ?char)`	Oppose a character. (e.g. "Kiss off", "You're the worst, "I hate you", "Get out of my life")
`(DADontUnderstand ?char)`	Don't understand a character utterance, or the current situation (e.g. "I'm confused", "I don't get it", "What are you talking about")
`(DAApologize ?char)`	Apologize to a character. (e.g. "I'm sorry", "My bad", "How can I make this up to you")
`(DAPraise ?char)`	Praise a character. (e.g. "You're a genius", "What a sweetheart you are", "You've got good ideas")
`(DACriticize ?char ?level)`	Criticize a character. There are two levels of criticism, *light* ("You're weird", "Don't be so up tight"), and *harsh* ("Idiot", "What a dipshit")
`(DAFlirt ?char)`	Flirt with a character. (e.g. "You look gorgeous", "Kiss me", "Let's get together alone sometime")
`(DAPacify ?char)`	Pacify a character. (e.g. "Calm down", "Relax, guys" "Keep your shirt on", "Take it easy")

Representation of Discourse Acts	*Pragmatic Meaning of Discourse Acts*
`(DAExplain ?param ?char)`	Give an explanation about the social situation to a character (e.g. “Grace is lying”, “Trip is having an affair”, “You’re not happy together”)
`(DAAdvice ?param ?char)`	Give advice to a character. (e.g. “You should get a divorce”, “Try to work it out”)
`(DAReferTo ?char ?obj)`	Refer to an object, potentially directed at a character. There are a number of different objects in the room, including the *couch* (“I like the couch”), the *wedding picture* (“Your wedding picture looks nice”), and *paintings* (“Where did you get these paintings”).
`(DAIntimate ?char)`	Ask a character to share their thoughts or feelings with you. (e.g. “What’s wrong”, “Let it all out”, “Talk to me”)
`(DAGoodbye ?char)`	Say goodbye to a character. (e.g. “Catch you later”, “So long”, “I’m out of here”)
`(DAInappropriate ?char)`	Utterances containing vulgar or socially inappropriate words or phrases. (e.g. “blow job”, “slut”)
`(DAMisc ?char ?type)`	Miscellaneous specialized discourse acts, often specific to certain beats or contexts. The *type* encodes the specialized act, e.g. *ask for drink* (“I’d like a drink”), and *should I leave* (“Is this a bad time”, “Should I leave”).
`(DASystemCannotUnderstand)`	Catch-all for all utterances which trigger no other discourse acts.

4 The Template Language

The rules which map surface text to discourse acts are written in a custom rule language that compiles to Jess [6], a java implementation of the CLIPS rule language [15]. The custom rule language is a superset of Jess, adding an embedded template description language that allows compact descriptions of surface text patterns to appear on the left hand side of rules. An initial implementation of the template compiler was done by Mehmet Fidanboylu, an undergraduate working under Michael’s direction.

```
;; A template rule for agreement
(defrule global-agree-rule1
  (template (toc (love (to | it | that))))
 =>
(assert (iAgree)))
```

Fig. 2. Rule example using the template sublanguage

The rule in Figure 2 contains a template test on the LHS – a specification of a pattern that will be tested over the input string. If the input string matches this pattern, the RHS asserts an `iAgree` fact (an intermediate fact) that will eventually be combined with an intermediate fact indicating the character addressed (potentially the null character `none`) to produce the final `DAAgree` fact.

4.1 The Pattern Language

The sub-language for specifying template patterns consists of a combination of regular expressions and occurrence expressions. Regular expressions are sensitive to the positions of terms in a pattern – for the regular expression `(love it)` to match a string, "it" must appear right after "love" in the string. Occurrence expressions don't care about position, only term occurrence – for the expression `(tand love it)` to match a string, "love" and "it" must both appear in the string, but do not have to be contiguous and can appear in either order. Note that when an occurrence expression is embedded in a regular expression, the occurrence expression only tests across the substring delimited by the regular expression. Examples of all the template pattern language expressions appear in Table 2.

Table 2. Template pattern language expressions

Example Expressions	*Meaning*
`(X Y)`	An *and* regular expression. Matches an input string if it consists of X immediately followed by Y.
`(X | Y)`	An *or* regular expression. Matches an input string if it consists of X or Y.
`([X])`	An *optional* regular expression. Matches an input string if it consists of X or nothing.
`*`	A *match all* wildcard. Matches any number of words in an input string.
`?`	A *match one* wildcard. Matches any one word in an input string.
`(tand X Y)`	An *and* occurs expression (tand is short for template-and). Matches an input string if it contains X and Y in any order.
`(tor X Y)`	An *or* occurs expression (tor is short for template-or). Matches a input string if it contains X or Y.
`(toc X)`	An *occurrence* occurs expression (toc is short for template-occurs). Matches an input string if it contains X.
`(tnot x)`	A *not* occurs expression (tnot is short for template-not). Matches an input string if X does not occur in it.

The `tor`, `tand` and `toc` expressions are syntactic sugar for classes of regular expressions: `(tor X Y)` can be rewritten as `((* X *) | (* Y *))` (`toc` is a special case of `tor`), and `(tand X Y)` can be rewritten as `((* X * Y *) | (* Y * X *))`.

Of course the various expressions can be recursively nested to produce compound patterns such as `(tand (X (tnot Y)) Z)` or `(X | ([(tor Y Z)] W))`.

Individual words are not the only atomic terms in the pattern language: besides matching an individual word, one can also match a positional fact, match an entire collection of words determined by WordNet [3] expansion, and match stemmed forms of words, using a WordNet stemming package.

4.2 Matching Positional Facts

Facts asserted by template rules can be *positional facts*. A positional fact includes information about the substring range matched by the rule which asserted the fact, and, potentially, author-determined information. Positional facts can then be matched as atomic terms within other templates. An example appears in Figure 3.

The first rule, `positional_Is`, asserts a positional fact `(iIs ?startpos ?endpos)` when an "is" word is recognized, such as "is" "seems", or "looks". The appearance of the special variables `?startpos` and `?endpos` is what makes this a positional fact – these variables are bound to the starting and ending position of the substring matched by the template.

```
;; Rule for recognizing positional "is" fact
(defrule positional_Is
  (template (tor am are is seem seems sound sounds look
                 looks))
=>
   (assert (iIs ?startpos ?endpos)))
;; Rule for recognizing positional "positive
;; description" fact
(defrule positional_PersonPosDesc
  (template (tor buddy comrade confidant friend genius
                 go-getter pal sweetheart))
=>
  (assert (iPersonPosDesc ?startpos ?endpos)))
;; Rule for recognizing praise
(defrule Praise_you_are_PersonPos
  (template ({iPersonPosDesc} | (you [{iIs}] [a | my]
            {iPersonPosDesc} *)))
=>
  (assert (iPraise)))
```

Fig. 3. Example rule for recognizing *praise* discourse act using positional facts

The second rule, `positional_PersonPosDesc`, asserts a positional fact when a word or phrase that is a positive description of a person appears.

The last rule, `Praise_you_are_PersonPos`, one of the rules for recognizing *praise* discourse acts, recognizes praises that consist of only a positive description

(e.g. “friend”), or of a sentence of the form “You are <positive description>” (e.g. “You are a friend”). When an atomic term appear in curly braces, this tells the template compiler to treat the term as the name for a positional fact, and to use the positional information (start position and end position) associated with the fact to determine if the pattern matches.

Positional facts make the template language as powerful as a context free grammar – when template patterns are combined with arbitrary Jess matches on the left-hand-side, the language becomes Turing complete. However, the *spirit* of the template language is to recognize simple text patterns and map them to discourse acts, not to build grammars that accept all and only the set of grammatically correct strings. Positional rules such as those in Figure 3 can be viewed as noise-tolerant parsing rules.

4.3 Term Retraction

When a template matches terms (words and patterns) within a string, it can remove terms to prevent other templates from matching on the same terms. This is particularly useful when matching idiomatic phrases (e.g., “fat chance”) that could potentially trigger other templates that match on the same words according to their more typical meanings. The retraction operator “–” can be placed in front of any term in a LHS template pattern; if the rule matches, the marked words are retracted from the internal representation of the string, and any facts that were asserted directly or indirectly from those words are also retracted (compiled template rules enforce proper truth maintenance). The retraction operator can be thought of as creating an implicit retraction action on the right hand side of the rule.

4.4 Compilation Strategy

Input strings are represented as a collection of facts, one fact for each word. Each unique word is represented as a unique word occurrence fact; each of these unique facts includes the start position and end position of the word, which are always the same[1]. For example, for performing template matching on the input string “I like it”, the string is represented as a collection of three facts: `(wo-i 1 1) (wo-like 2 2) (wo-it 3 3)`. The fact names for each word in a string are constructed by appending the word onto “`wo-`” (for *word occurrence*).

The template compiler compiles template patterns into collections of rules whose LHSs ultimately test word occurrence facts. Thus word occurrence (wo) facts are chained together by generated rules to produce intermediate phrase occurrence (po) facts. The intermediate phrase occurrence facts are ultimately tested on the LHS of the authored rules, which assert author defined facts (e.g. discourse acts, positional facts). By representing the individual words of an input string as word occurrence facts, and compiling templates into networks of rules which test these facts, the expensive string tests which would potentially have to be run for every template rule

[1] Even though words always have the same start and end position, explicitly representing the start and end position makes it easier for compiler generated rules to combine word occurrence facts with phrase facts, which *do* have different start and end positions.

LHS are eliminated. Further matching efficiencies are gained through Jess' use of the Rete [5] matching algorithm.

5 Idioms for Template Rules

The general (non-beat specific) template rules are organized in the following way. First, a collection of high salience template rules recognize generically useful patterns and synonyms. An example is the "is" rule in Figure 3; others include sets of synonyms for positive and negative words, greets, flirts, insults, curses, and so on.

Salience declarations can be used to declare that some rules should be preferred over others. We use salience to create tiers of template rules, with higher salience rules recognizing lower-level features than lower-salience rules.

After general low-level patterns, the next (lower-salience) tier of rules recognizes idiomatic expressions. Each idiomatic rule uses the retraction operator to retract the idiomatic expression once it is found – this prevents other rules from incorrectly matching on individual words in the idiomatic expression. There are generally a large number of idiomatic expression rules for each discourse act (e.g. agree) or sub-discourse act "meaning" (e.g. idioms for "person positive description", used by the praise rule in Figure 3). To compile an initial list of such idioms, we examined the phrase resources [8, 16, 17, 18] to compile a large number of expressions (~ 9000 phrases); ~1000 of these were applicable to our domain and were categorized in terms of our discourse acts.

The next tier of rules uses retraction to cancel out adjacent negative words. For example, for the surface text "you are not bad", the words "not" and "bad" get retracted and replaced with `iAgree`, as if the original surface text had been "you are good".

The final tier(s) of rules consists of keyword and combination rules. Keyword rules watch for individual words or short, non-idiomatic phrases that are indicative of a discourse act or sub-discourse act meanings. The `positional_PersonPos Description` rule in Figure 3 is an example of a sub-discourse act keyword rule. Discourse act keyword rules similarly match on words or short phrases, but directly assert a discourse act. In an attempt to reduce the number of false positives, discourse act keyword rules tend to impose a limit on the total number of words that can appear in the surface utterance. Unlike sub-discourse act keyword rules, which can depend on combination rules further along the chain to impose additional constraints, discourse act keyword rules are more likely to be fooled by a longer utterance because something else in the utterance contradicts the keyword assumption.

Combination rules, such as the `Praise_you_are_PersonPos` rule in Figure 3, combine intermediate positional facts, and impose constraints on the relative positions of these facts, to recognize discourse acts.

Anaphora resolution occurs by simply looking up a stored referent for each anaphoric reference. In the *Façade* architecture, the autonomous characters are responsible for updating referents as they deliver different dialog lines, particularly the referent for "it".

6 Templates and Ungrammatical Inputs

Template rules tend to be promiscuous, mapping a large number of ungrammatical inputs to discourse acts. Keyword rules, in particular, tend to produce false positives. False positives can be reduced by writing ever more elaborate combination rules (in essence, moving towards full parsing), but at the expense of increasing false negatives (player utterances that should be recognized as a discourse act but aren't).

Given this tradeoff, the template rules for *Façade* err on the side of being overly permissive. This is based on the design approach that it is more interesting for the characters to eke some meaning out of a broad set of utterances, and thus have some interesting response for this broad set, than to only have interesting responses for a narrow set, and respond with some form of "huh?" to the rest. While players will sometimes try to break the NLU by seeing what kinds of ludicrous sentences they can get the characters to respond to, the templates are designed not to robustly support this meta-activity, but rather to extract meaning from a broad a collection of "natural" utterances likely to arise during the course of playing *Façade*.

7 Relationship to Chatterbots

Our approach for mapping surface text to discourse acts, while bearing some similarities to chatterbot approaches such as AIML [20], significantly extends the capabilities of these approaches. As AIML is representative of contemporary chatterbot language processing, here we briefly summarize AIML's features. In AIML, the fundamental unit is a pattern-template pair, where patterns match player input and templates produce output text in response to the player input matching the corresponding pattern (note that in our NLU approach, we use the word "template" for what AIML calls a "pattern"). AIML pattern syntax is a subset of regular expression syntax, excluding regexp-or and optional subexpressions. Templates can recursively invoke pattern matching, potentially introducing new words into the recursively matched expression. Templates can get and set (side-effecting) the value of unary predicates; unary predicates *cannot* be accessed in patterns. AIML's support for anaphora resolution is similar to ours, using a collection of unary predicates to keep track of the current referents for he, she and it, placing the burden of maintaining the current referents on the author. Pattern matching can depend on the bot's previous utterance, introducing limited support for discourse context.

There are a number of differences between our approach and AIML.

1. Our NLU template language doesn't map surface text directly to a reaction, but rather to a discourse act; phase II of our NLP processing is responsible for selecting a reaction to the discourse act. This separation supports reasoning that can take into account more sophisticated discourse context than just the last agent utterance. When given ambiguous input, our NLU system produces all possible interpretations, letting the next layer of the NLP decide which discourse act(s) to respond to. AIML uses implementation-dependent, non-author-accessible heuristics to decide which single response to give.
2. Positional facts are the mechanism through which we introduce recursion into the NLU matching process. The support for the inclusion of author-determined

information in positional facts, plus the ability to match facts and chain variables on the left-hand-side of our NLU rules, makes our rule language more expressive than AIML. Our framework supports rule strategies ranging from simple pattern matching, through (noise tolerant) context-free and context-sensitive parsing, to arbitrary computation, all of which can co-exist within a single rule set.
3. Author-declared rule salience allows authors to specify their own tiers of rule processing. In contrast, the matching order for AIML patterns is fixed by the AIML interpreter.
4. Retraction supports more robust handling of idioms and double negatives.
5. Wordnet expansions and stemming supports the matching of a wider variety of player inputs.

8 Experiences with the Template Language

In the course of authoring *Façade* we've written ~800 template rules, which compile to ~6800 Jess rules. On a 2GHz machine, with the rest of the *Façade* AI running, as well as the animation engine, the rules fire to completion (generally proposing several discourse acts) in 300 milliseconds or less, giving us adequate real-time performance.

As we continue play testing, we use session traces to find NLU failures and modify the rules. However, preliminary play testing has found our initial set of rules to be surprisingly robust. The context of the *Façade* dramatic situation does, as we'd hoped, guide the player to use language appropriate to the situation.

References

1. Allen, J., Byron, D., Dzikovska, M, Ferguson, G., Galescu, L., and Stent, A. 1998. An Architecture for a Generic Dialog Shell. *Natural Language Engineering* 6 (3).
2. Cavazza, M., Martin, O., Charles, F., Mead, S. and Marichal, X. Users Acting in Mixed Reality Storytelling, *Second International Conference on Virtual Storytelling (ICVS 2003)*, Toulouse, France, pp. 189-197.
3. Fellbaum, C. (Ed.). 1998. Wordnet:An Electronic Lexical Database. MIT Press.
4. Ferguson, G., Allen, J. F., Miller, B. W., and Ringger, E. K. 1996. *The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant*. TRAINS Technical Note 96-5, Department of Computer Science, University of Rochester, Rochester, NY.
5. Forgy, C. L. 1982. Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem. *Artificial Intelligence* 19, 17-37.
6. Friedman-Hill, E. 1995-2003. Jess, the Rule Engine for Java. Sandia National Labs. http://herzberg.ca.sandia.gov/jess/.
7. Lesh, N., Rich, C., and Sidner, C. 1999 Using Plan Recognition in Human-Computer Collaboration. In *Proceedings of the Seventh International Conference on User Modeling*. Banff, Canada.
8. Magnuson, W. 1995-2002. English Idioms, Sayings, and Slang. http://home.t-online.de/home/toni.goeller/idiom_wm/.

9. Mateas, M. and Stern, A. 2004. A Behavior Language: Joint Action and Behavior Idioms, in Prendinger, Helmut and Ishizuka, Mitsuru (Eds.), *Life-Like Characters: Tools, Affective Functions, and Applications*, Springer-Verlag, 2004.
10. Mateas, M and Stern, A. 2003a. Integrating plot, character and natural language processing in the interactive drama *Façade*, *1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE '03)*, Darmstadt, Germany , March 24 – 26, 2003 .
11. Mateas, M and Stern, A. 2003b. Façade, an experiment in building a fully-realized interactive drama, *Game Developers Conference (GDC '03),* San Jose, CA, USA, March 4 – 8, 2003.
12. Mateas, M and Stern, A. 2002. A behavior language for story-based believable agents, *IEEE Intelligent Systems,* July/August 2002, 17 (4), 39-47.
13. Mateas, M., and Stern, A. 2001. Façade. *Digital Arts and Culture* 2001. Brown University, Providence RI. 2001.
14. Mateas, M. and Stern, A. 2000. Towards Integrating Plot and Character for Interactive Drama. In *Working notes of the Social Intelligent Agents: The Human in the Loop Symposium*. AAAI Fall Symposium Series. Menlo Park, CA: AAAI Press. 2000.
15. NASA. 1985-2002. C Language Integrated Production Systems (CLIPS). Originally developed at NASA's Johnson Space Center. http://www.ghg.net/clips/WhatIsCLIPS.html.
16. Oliver, D. 1995-2002. The ESL Idiom Page. http://www.eslcafe.com/idioms/id-mngs.html.
17. PhraseFinder. 2002. Phrases, Sayings, Quotes and Cliches at The Phrase Finder. http://phrases.shu.ac.uk/index.html.
18. Phrase Thesaurus. 2002. http://www.phrasefinder.co.uk/index.html.
19. Rich, C., and Sidner, C. 1998. COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*. Vol. 8, No. 3/4, pp. 315-350.
20. Wallace, R. The Anatomy of ALICE. http://www.alicebot.org/anatomy.html.