A Robust Spoken Language Architecture to Control a 2D Game

Andrea Corradini [1], Thomas Hanneforth[1], and Adrian Bak[2]

1. Computational Linguistics Department
University of Potsdam
14415 Potsdam, Germany
{andrea,tom}@ling.uni-potsdam.de

2. TV2 | Interaktiv
Rugaardsvej 25
5100 Odense, Denmark
adba@tv2.dk

Abstract

Speech has been increasingly used as input and/or output modality in commercial systems and academic prototypes. Its capability to complement and enhance applications makes it possible to consider spoken language as a new means to support human interaction with computers in interactive domains: computer games being one of them.

This paper presents an architecture that we developed to play a 2D graphical version of a board game using spoken natural language. In our system, the sensed player's speech is syntactically parsed with a robust weighted finite state transducer algorithm that creates a simplified representation of the input sentence. In a pipelined process, a semantic parser is then responsible of splitting the input representation generated during syntactical analysis into a series of data structures each representing the semantics of the underlying text chunks within the original sentence. These data structures are then passed on to the game logic module which generates game commands, resolves possible ambiguities or incompleteness within the data structures, and checks preconditions related to the validity of the commands. If the preconditions are met, the instructions are carried out and the game state is updated as direct effect of the commands issued.

In a series of preliminary tests to assess the robustness of our approach, we obtained a correct classification of input sentences for up to 95.8% of the instructions issued by the user. Interaction with our systems occurs in real-time.

1. Introduction

In many emerging applications, the traditional input devices such as mouse and keyboard represent a bottleneck to the interaction. They make it impossible for people to exploit the full potential of their computers and prevent people communicating effectively. This occurs particularly within the context of interactive applications where designers and developers are faced with a series of decisions regarding the proper choice of input and output modalities, the breadth of the envisioned system, the context of its use and the skills of its final users.

As one of the oldest and most natural modes of communication among humans, speech recently gathered much attention from the industry and the academia. By exploiting transfer among human-machines interfaces from the human-human domain, speech represents a prime candidate for a new interaction modality. Even without resorting to such a powerful analogy, the benefit of using speech is apparent: it resides in its capability to complement and enhance interactive domains when traditional forms of communication reach their limits.

Driven by rapid advances in speech technology, spoken language interfaces have started to make an impact on computer use. Voice enabled interfaces have rapidly expanded and used to support new means of interaction with computers. Several enterprises already provide customer support through call centers manned by humans or through interactive applications in which the integration of a speech interface adds value to the existing application [18]. Speech has been integrated into a vast range of small-scale and embedded computing devices in the attempt to reduce their size and augment their usability. It has also been used to enhance traditional GUIs in office, CAD and drawing applications. Now, it has even become possible to play computer games by using speech.

In this paper, we report on the spoken language architecture that we have developed to play a 2D graphical version of a board game. In our system, the sensed player's speech is syntactically parsed with a finite state transducer algorithm that robustly creates a simplified representation of the input sentence. In a pipelined process, a semantic parser is then responsible of splitting the input representation generated during syntactical analysis into a series of data structures each representing the semantics underlying the text chunks of the original sentence as captured by the speech recognizer. These data structures are further passed on to the game logic module that generates game commands, resolves possible ambiguities or incompleteness within the data structures, and checks preconditions related to the validity of the commands on the basis of world knowledge. If the preconditions are met, the instructions are carried out and the game state is updated as direct effect of the commands issued.

We organized the rest of the paper as follows. We discuss related approaches to create interactive games operated by speech in Section 2. We present a detailed overview of the modules that make up our current system in Section 3. We provide a discussion about preliminary system tests and results in Section 4 and eventually conclude in Section 5.

2. Speech-enabled Games

The video games and entertainment industry have increasingly concentrated on speech technology for both

text to speech synthesis as well as speech recognition. While text to speech synthesis makes it possible to endow game's character and entities with their own voice, the addition of speech recognition makes real communication between player and game a reality. Altogether, speech brings a higher interactive experience into and adds real value to digital games.

A number of speech enabled games have already hit the market. Some have been total failures some others enjoyed a partial success. The first results seem to show that this new form of human-machine interface suits especially social or strategy games such as Microsoft's Age of Empire [1] and Life Line [3] that were among the first successful games to provide speech in both input and output. These products feature voice-activated commands to guide the game's main character but effectiveness of speech input is arguable since it only replaces the menu selection task provided by the game GUI with a large set of predefined utterances. Instead of facilitating the interaction, this increases the cognitive load of the user who has to memorize a long list of arbitrary commands. Browsing the Internet, one can find several simple games for the blind. They mainly allow spoken input in terms of a single word. [15] is an interesting attempt at creating a conversational dialogue within an educational game context and goes beyond a speech-enabled game. In [7] an avatar within an emulated environment is controlled through speech. The system uses a grammar to interpret and generate game responses from spoken command. The work that displays more similarity to our system is [20] where a speech interface is used to play the popular Mahjong game over the network is presented. They however employ a much more limited grammar than ours and their commands consist mainly of single words.

3. Overall System Architecture

3.1 The Game

Pentomino is the popular board game of choice for our application. This puzzle game is named after the well-known Domino from which it differs only because its pieces are formed by joining five, instead of just two, equal sized squares together along complete edges. Excluding isomorphism (i.e. rotations, flipping and combinations of them), there are only twelve possible pieces. In Pentomino the player has to use up the set of pieces and land them into a predefined grid-shaped game board (see Figure 1). The user can choose to play in a slightly more challenging way by using the game timer and by attempting at producing a complete solution of the puzzle within the shortest time possible. From a mathematical perspective, Pentomino is a particular type of Polyominos [9] and as such it is considered an exact mathematical science.

A sketch of the architecture we developed to play Pentomino is displayed in Figure 2. It consists of a set of agents that communicate with each other by means of the OAA agent architecture [8].

Figure 1: The GUI showing an instance of the Pentomino puzzle game.

3.2 Speech Recognizer

Players are shown a GUI version of Pentomino (see Figure 1) that they can play using either mouse or keyboard or speech or a combination of two or more of these modalities. In order to allow users play with the game GUI using speech, we use the Sphinx-4 [19] open source speech recognizer. Sphinx-4 is a flexible state-of-the-art decoder capable of performing many different types of recognition tasks. It supports all types of HMM-based acoustic models, all standard types of language models, and several search strategies. It can be used along with a JSGF [1] grammar or with an N-gram language model in which case syntactic parsing is not performed by Sphinx-4. In Figure 2, the dashed line inside the speech processing module marks this distinction. We can use Sphinx-4 in either mode but the next section describes the syntactic parser employed in case where either Sphinx is run using an N-gram or the decoder is bypassed and input is typed-in. It is worth noticing however, that the result of step is processed in the very same way as if Sphinx were used with a JSGF grammar.

3.3 Robust Syntax Parsing

For the syntactic parsing of user's utterances we decided to employ an augmented context free style grammar (CFG). On the one hand, we augment a traditional CFG grammar with the addition of regular operators like disjunction (|), optionality (?), Kleene star (*) and plus closure (+) in the left side of non-terminals grammar rule. One the other hand, we allow for rules and/or rules' subconstituents to be added attributes that pin point locations within the grammar that have a relevant semantic content. In the realm of JSFG this is equivalent to having tags in the grammar. The following is an excerpt from our grammar:

PLAY	→	ACTION (and? ACTION)*
ACTION	→	SELECT PIECE_DESC
ACTION	→	ROTATE (PIECE_DESC|REF)
SELECT	→	(take | get | select) {"*fhead:action:select*"}
ROTATE	→	rotate {"*fhead:action:rotate*"}
REF	→	(it | that | this) {"*id:anaphora*"}

In the above representation, non terminals are capitalized, terminals underlined, while the semantic annotation is in italics within curly brackets. Possible text strings like "*I want to select the blue piece and rotate it*" or "*please get the blue cross shaped figure next to the red one and then show the ruler*" are all parseable by our grammar. Partially based on the analysis of data collected during human-human interaction with subjects playing Pentomino in a collaborative way, we noticed that our application domain is rather limited in terms of commands to the game. This simplifies the design of our grammar and makes the definition of center embedded rules along with other specific context free constructions not necessary within our framework. Thus, despite being a CFG, our grammar maps well onto a regular language. This property turns out to be very important for our approach when it comes to deal with parsing robustness.

As a first step, we compile offline our grammar into a *weighted finite state acceptor* (WFSA). A WFSA [16] is a 7-tuple $A = <\Sigma,Q,q_0,F,E,\lambda,\rho>$ over a weight structure $\mathbb{W}$ with the following components:

1. Σ the finite input alphabet
2. Q the finite set of states
3. $q_0 \in Q$ the start state
4. $F \subseteq Q$ the set of final states
5. $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{W} \times Q$ the set of transitions
6. $\lambda \in \mathbb{W}$ the *initial weight*
7. ρ: $F \mapsto \mathbb{W}$ the *final weight function* mapping final states to elements in $\mathbb{W}$

As the weight set $\mathbb{W}$ we choose the set of strings used for the semantic annotation of the grammar. Weights can occur at the transitions and the final states of the WFSA. For the actual grammar compilation we resort to the method proposed in [17].

For the correct interpretation of the string weights in the algebra of WFSAs (which is for example extensively used during grammar compilation) we impose on the weight set the algebraic structure of a *semiring* [4]. Formally a semiring is a 5-tuple $<\mathbb{W},\oplus,\otimes,\mathbf{0},\mathbf{1}>$ such that

1. $<\mathbb{W},\oplus,\mathbf{0}>$ is a commutative monoid with **0** as the identity element for $\oplus$.
2. $<\mathbb{W},\otimes,\mathbf{1}>$ is a monoid with **1** as the identity for $\otimes$
3. $\otimes$ distributes over $\oplus$: $\forall a,b,c \quad a \otimes (b \oplus c) = (b \oplus c) \otimes a = (a \otimes b) \oplus (a \otimes c)$
4. **0** is an annihilator for $\otimes$: $\forall w \in \mathbb{W}$, $w \otimes \mathbf{0} = \mathbf{0} \otimes w = \mathbf{0}$.

In the case of string weights we use a system $<\Sigma^*, lcp, \cdot, s_\infty, \varepsilon>$, a so called *string semiring*: Σ is our game vocabulary (including a special tag for unknown tokens), the abstract addition operation $\oplus$ is mapped to the operation of taking the *longest common prefix* of two strings (with the infinite string s_∞ as the identity element)[1], the abstract multiplication $\otimes$ is string concatenation, and the identity element is the empty string ε.

The weight $\Omega(x)$ of a string x accepted by starting at the start state q_0 and following some path π to a final state is defined by the following generic formula[2]:

$$\Omega(x) = \bigoplus_{\pi \in \Pi(\{q_0\},x,F)} \lambda \otimes w(\pi) \otimes \rho(n(\pi))$$

For instance, the WFSA compiled from our grammar maps the input sequence "*I want to select the blue piece and rotate it*" to a string that consists of a sequence of semantic annotations as encountered in the grammar during sentence parsing. In this particular example this amount to "*fhead:action:select id:1 fhead:action:rotate id:anaphora*" since the string weights carrying semantic annotation are abstractly multiplied, that is, concatenated along an accepting path.

After grammar compilation, the WFSA is not yet robust because it accepts only the patterns defined in the grammar and fails on other input sequences. We say a WFSA is *robust* if there is a path through the automaton, not necessarily reaching a final state, for every input sequence $w \in \Sigma^*$ for a given alphabet Σ. As a special case, a deterministic WFSA is robust if it is *complete*, that is, every state has an outgoing transition for every input symbol $a \in \Sigma$. Our next step is thus to make the original WFSA A robust by applying a *regular replace operator* to it [13]. Basically, this operator adds states and transitions to A which account for input sequences originally not in A and is defined in the following way[3]: let *not-A* be the regular language $\neg(\Sigma^* \cdot A' \cdot \Sigma^*)$. A' is derived from the grammar automaton A by replacing all (string) weights in A by ε. The FSA *not-A* accepts all strings which do not contain an instance of a sequence originally accepted by A. We are now ready to define a function *robust*(A) as:

$$robust(A) =_{\text{def}} (not\text{-}A \cdot A)^* \cdot not\text{-}A$$

If we apply *robust*(A) to some input sequence we do in effect compute the weighted intersection of *robust*(A) with a linear FSA representing that sentence. As long as it does not contain a string accepted by our grammar automaton A

[1] For example $lcp(abc,abd) = ab$ and $lcp(abc, s_\infty) = abc$.

[2] If the path π consists of a sequence of transitions $t_1 t_2 \ldots t_k$ then $w(\pi)$ is the abstract multiplication of all the transition weights: $w(\pi) = w(t_1) \otimes w(t_2) \otimes \ldots \otimes w(t_k)$.
$n(\pi)$ is the final state reached after following t_k.

[3] In the following we use regular expressions (like Σ^*) and FSAs (like A) interchangeably due to the known equivalences holding between both.

we simply skip it with *not-A*. If we then find a substring *s* in the sentence to parse that is accepted by *A* we compute the output string weight assigned to *s* by *A*. The star closure operation in the definition of *robust*(*A*) accounts for repeated occurrences of substrings accepted by *A* interleaved with substrings not in *A*.

The function *robust*(*A*) is regular (so the result is representable by a WFSA) because we use only operations which preserve regularity. The key for robust behaviour is here the negation/complementation operation. Unweighted FSAs are closed under complementation, that is, we can construct for every unweighted FSA *M* a FSA *M*' which accepts all strings which are not in *M* [10]. This closure under complementation is only true for unweighted FSA, so we define *A*' by a function that removes all string weights in *A* by replacing them by ε.

In accordance to our concept of robustness, whenever the grammar WFSA *A* fails to process an input string *s* there is a path for *s* in *not-A*. Of course, the WFSA constructed using the robust operator is not deterministic in the general case because the *A* submachine does not "know" in advance whether it will accept a string. Furthermore, *robust*(*A*) does not in general admit an equivalent deterministic WFSA since not every WFSA can not be made deterministic [16].

The ultimate goal of syntactical parsing is of course constructing the mapping of the natural language input to sequences of semantic annotations strings. For that purpose we need to target an additional issue. Given the ill-formed input "*select the green eating*", there is no accepting path in the sub FSA *A* in *robust*(*A*) and the sequence is accepted in *not-A* with weight ε (since *not-A* is unweighted). On the other hand, that ill-formed input has some meaningful parts, such as "*select the green*" which can contribute to the extraction of its semantics. To get some possibly incomplete semantic information even in the case of ill-formed inputs we changed our grammar in two ways: a) we marked all terminal symbols not contributing to the semantics as optional and b) we take all composite rules and add to the grammar the rules that allow for the parsing of their component. For instance, given the rule ACTION → SELECT PIECE_DESC we add two more rules, notably, ACTION → SELECT and ACTION → PIECE_DESC. Due to these changes, the grammar WFSA is also able to accept incomplete utterances and thus successfully parse also chunks of the input sentence.

3.4 Semantic Parsing

To mentally represent in a simple way our process of syntactic parsing, one can think of it as the task of building parsing trees and collecting the traversed attributes present in the original grammar. These attributes are further connected together in the order they were encountered and sent out to the semantic parser. Each attribute consists of either a *name:value* pair or a *head:action:value* triple. While the fields *name* and *head* are always instantiated, the values in the other fields may be not.

Given a sequence of attributes, the role of the semantic parsing is twofold. First, it has to break down the sequence into chunks in an attempt to reflect the number of commands in the original input sentence. For instance, let us consider the following sequence of semantic tokens:

"*mood:polite fhead:action:select color:blu shape:cross to: next color:red id:anaphora fhead:meta:and fhead:action:set what:ruler*"

The sequence could have been generated by the input sentence "*please get the blue cross shaped figure next to the red one and then show the ruler*" or any rephrasing of it. In terms of game semantic, such a compound sentence is made up of two valid elementary instructions i.e. "*get the blue cross shaped figure next to the red one*" and "*show the ruler*", respectively. The goal of the semantic parser is therefore to split the sequence of attributes into two substrings that capture the semantic underlying the two original elementary instructions. Splitting occurs by employing a set of hierarchical rules that correlate syntactic parser outcomes and corresponding input sentences. The rules are based on heuristics and look, among others, for the occurrence or the lack of certain attributes, for the value of specific fields, at the spatial positions of the single pairs, for the occurrence of un-instantiated values within pairs, and for the spatial relationships among pairs. Occurrences of triples among the token are exploited to mark the start of a new command. Basically, we apply rewriting rules to reduce the number of pairs and triples in the input sequence without lost of any semantic information. This is also accomplished by the introduction of new semantic pairs and triples that replace sequences of tokens generated by the syntactic parser.

Once the sequence of semantic attributes cannot be further reduced, we employ a unification algorithm which attempts at matching that new semantic representation with one of a set of predefined template frames that we designed during development. We currently have about 25 frames, each representing a different class of elementary game instructions. Unification of a sequence of tokens with a frame occurs by unifying each slot in the frame with a token in the attribute sequence that has the same *name* field in case of pairs and *head* field in case of triple. Since it is possible for a sequence to unify with more than one frame, we give precedence to unifications that involves frames with the largest number of slots. The idea is here equivalent to give preference to longer input chunks in partial parsing [5]. Our frame structures can be extended to become typed feature structures [11] to integrate additional information from other input modalities such as mouse clicks or keyboard key activity.

Elementary game instructions can be divided into three classes: those to operate directly on the Pentomino pieces, those to change the setting of the game GUI, and those to indicate a dialogue with the uses. At this stage of development, the latter class is the smallest one as we hard

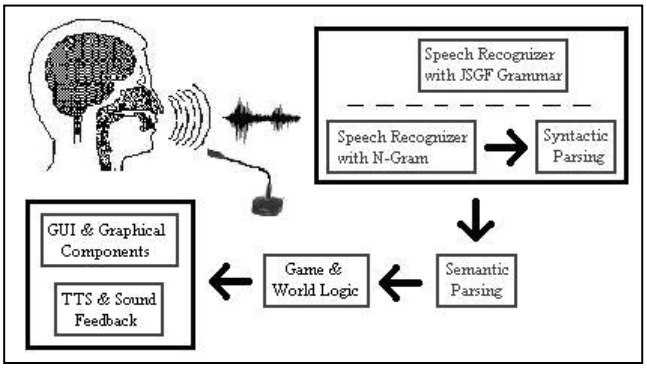

Figure 2: Sketch of the Complete Systems

coded into the system a mechanism to simulate short dialogue in few domains such as game authors, version etc.

3.5 The Game and World Logic

The Game and World logic agent is responsible for turning the rich semantic information stored in the frames into valid game instructions.

To reach this goal, this module starts off by mapping un-instantiated and underspecified frame fields onto entities (such as Pentomino pieces, board, or board locations) within the world. Referential expressions are resolved as well. Their treatment is simplified by our fairly restricted game setting coupled with knowledge about the game history and the current game state. When multiple commands are issued at once and joint together by an "*or"* conjunction, they are also heuristically resolved. For instance, the command "*drop the cross-shape pentomino in the upper left corner or rotate it*" is decomposed into its two single component commands "*drop the cross-shape pentomino in the upper left corner*" and "*rotate it*". The anaphora in the latter one is then resolved. Further, the "*or*" operation is taken care of by checking if the single commands can be carried out singularly or if they are conflicting. A small set of rules based on game pragmatics and semantics determines if and what of the commands is eventually carried out. Reasoning about temporal vicinity is additionally employed to rule out ambiguities in anaphoric expressions.

Even the player's knowledge about the game can be used to restrict the set of input interpretations and guide the resolution process. For instance, it is very unlikely for the user to issue a command that rotates the cross shaped piece since this produces no changes.

Once ambiguities are resolved, a sequence of instructions (each corresponding to one frame) that describe the actions the player wants to take, is generated based on the frames' content. At this point, the module checks whether these instructions meet a set of preconditions given the current game world and, if so, accordingly updates the game GUI. Feedback messages are simultaneously played back by the TTS either to describe the action performed or to inform about problems in carrying out the detected commands.

4. Discussion and Preliminary Results

We run several preliminary tests to assess the accuracy of our system on PC platforms with Pentium 4 CPU, 2.40GHz, and 1GB of RAM under the Windows XP operating system. Since game-player interaction revolves about performing actions on pieces in the game world and displaying the effects that these manipulations give rise to, we were interested in the percentage of input that maps onto equivalent correct game actions.

Our preliminary study involved 9 subjects, all computer science literate, including 2 English native speakers. When we restrict our analysis to input utterances that are properly recognized by the speech decoder, our system displays a correct language interpretation in up to 95.8% of the cases. This figure however, quickly degrades to 63.7%, on average, when all input utterances, i.e. including those misrecognized by the speech recognizer, are considered over 10 to 15 minute interactive sessions. The response time to user inputs is variable. It ranges from a minimum of 250msecs to a maximum value of around 1300msecs. These values largely depends on the speech recognizer settings and particularly on the settings concerning the type of purging algorithm and the total number of grammar states (the grammar we used compiles into a search space made up of something less than 5000 nodes). Barge in is not supported. If the user utters a sentence while the speech recognizer is still processing an audio stream, the system is likely to react with an unexpected long delay before presenting the current recognition results.

We collected also data about the usability and ease of use of our architecture. Based on subjective answers from the participants to a post interaction questionnaire, it seems that our speech interface enhances game play interaction. At the same time, there is a clear indication that when time plays a relevant role for the interaction, speech is not the preferable modality anymore. This occurred when subjects were asked to complete the puzzle game with the goal to minimize the time to complete it.

We hypothesize that this is due to the fact that speech processing takes longer than a mouse click to process and is also more error prone. Under time pressure the users seem not to be willing to have to wait because of longer processing time. They seem less tolerant to system interpretation errors for they cost additional precious time to repair or make up for. In case of recognition errors, users usually utter more commands into the mic regardless of whether the speech recognizer is ready to process a new audio stream. This turns out to be a serious additional source of problems. Speech makes a valid, useful and pleasant input modality in support of existing ones but is not the preferred one when the system latency, response time, and network communication is of paramount importance. Network communication issues should also be considered in the case the game is played in multiplayer mode over a network. In this case, local players can connect to a central server where a single instance of the game is running. This requires less work on the part of the players' computers that have to process speech and

language on the local machines. Only game logic is done at server's end which is an advantage because more server resources can be used to keep everyone in synch.

5. Conclusion

By its nature, voice user interfaces have a number of advantages over other user interfaces. Speech technology is a challenge for digital signal processing and computer systems. It requires minimal intrusive equipment to be embedded in interactive applications but still needs to be integrated with other modes for effective communication.

In the context of an interactive spoken system, the need of a robust parsing is essential for the success of the application. Poor understanding results in a disrupted interaction and drastically degrades the user's entertaining experience. By its nature, speech displays a high rate of extra grammatical phenomena that easily make most syntactic parser fail. Most spoken dialogue systems have tried to deal with that issue by limiting the linguistic analysis to keywords spotting [6]. In our approach, the natural language understanding problem focuses on the pragmatic effects of language i.e. what changes an utterance causes changes in the game world rather than on a syntactic and/or semantic analysis of language only. A similar approach, yet with more weight on pragmatic information, was followed in [14]. We adopted a pipelined parsing formalism based on an extended domain of locality that is capable of achieving a fairly high degree of integration of syntactic and semantic information.

In our system, language processing is facilitated by the Pentomino game domain which naturally restricts the set of utterances that the player produces. Nevertheless, we can easily extend our methodology to comply with a deeper syntactic formalism that gives more importance e.g. to sub categorizations that accounts for stronger syntactic dependencies within an input sentence (such as e.g. [12]). We are currently investigating such research area and are attempting to extend the current implementation into a prototype architecture capable of supporting dialogue.

Acknowledgement

This work has been carried out within the framework of the DEAWU project supported by the Marie Curie Transfer of Knowledge Programme, grant EU #FP6-2002-Mobility. Out thanks go to Bojan Popadic for programming support as well as to Sigfried Wrobel for troubleshooting networks and software problems on UNIX platforms.

References

[1] http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/

[2] http://www.microsoft.com/games/empires/multimedia.htm

[3] http://www.konami.com/lifeline/

[4] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974

[5] Abney, S., Partial Parsing via Finite-State Cascades In *Proceedings of the ESSLLI Robust Parsing Workshop,* pp. 8-15, 1996

[6] Bratt, H., Dowding, D., and Hunicke-Smith, K., The SRI Telephone-based ATIS System, In *Proceedings of the Spoken Language Systems Technology Workshop*, 1995

[7] Cavazza, M., Bandi, S., and Palmer, I., Situated AI in Video Games: Integrating NLP, Path Planning and 3D Animation, In *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games*, 1999

[8] Cheyer, A., and Martin, D., The Open Agent Architecture. In *Autonomous Agents and Multi-Agent Systems*, 4(1-2), pp. 143-148, March/June 2001

[9] Golomb, S. W., *Polyominoes*. Scribner's, 1965

[10] Hopcroft, J. E., and Ullman, J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979

[11] Johnston, M., Cohen, P.R., McGee, D., Oviatt, S.L., Pittman, J.A., and Smith, I., Unification-based Multimodal Integration In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 281-288, 1997

[12] Joshi, A.K., and Schabes, Y., Tree-Adjoining Grammars, , In Rozenberg, G., and Salomaa, A. (Ed.), *Handbook of Formal Languages*, Vol. 3, Springer Verlag, pp. 69-124, 1997

[13] Karttunen, L., The Replace Operator. In Roche, E., and Schabes, Y. (Ed.), *Finite-State Language Processing*, MIT Press, 1997

[14] Mateas, M., and Stern, A., Natural Language Undertanding in Façade: Surface-text Processing, Lecture notes in computer science, Vol. 3105, Springer Verlag, pp. 3-13, 2004

[15] Mehta, M., and Corradini, A., Understanding Spoken Language of Children Interacting with an Embodied Conversational Character. In *Proceedings of the Combined Workshop on Language-Enabled Educational Technology and Development and Evaluation of Robust Spoken Dialogue Systems at the ECAI'06*, pp. 51-58, 2006

[16] Mohri, M., Finite-State Transducers in Language and Speech Processing. In *Computational Linguistics*, 23(2), pp. 269-311, 1997

[17] Mohri, M., and Pereira, F.C.N., Dynamic Compilation of Weighted Context-free Grammars. In *COLING-ACL*, pp. 891-897, 1998

[18] Rosenfeld, R., Olsen, D., and Rudnicky, A., Universal Speech Interfaces, In *Interactions*, 8(6), pp. 34 – 44, 2001

[19] Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J., Sphinx-4: A Flexible Open Source Framework for Speech Recognition, Sun Microsystems, TR-2004-139, 2004

[20] Zhang, J., Zhao, J,, Bai, S., and Huang, Z., Applying Speech Interface to Mahjong Game. In *Proceedings of the 10th International Multimedia Modeling Conference*, pp. 86-92, 2004