A Speech-Driven Second Screen Application for TV Program Discovery

Peter Z. Yeh[1], **Ben Douglas**[1], **William Jarrold**[1], **Adwait Ratnaparkhi**[1], **Deepak Ramachandran**[1], **Peter F. Patel-Schneider**[1], **Stephen Laverty**[2], **Nirvana Tikku**[2], **Sean Brown**[2], **Jeremy Mendel**[3]

1. AI & NL Research, Nuance Communications, Sunnyvale, CA
2. Mobile Product Strategy, Nuance Communications, Cambridge, MA
3. Creative Development, Nuance Communications, Sunnyvale, CA

Abstract

In this paper, we present a speech-driven second screen application for TV program discovery. We give an overview of the application and its architecture. We also present a user study along with a failure analysis. The results from the study are encouraging, and demonstrate our application's effectiveness in the target domain. We conclude with a discussion of follow-on efforts to further enhance our application.

Introduction

The recent explosion of content (e.g. movies, TV shows, sports, etc.) available on television coupled with an increase use in mobile devices (i.e. smart phones and tablets) has created significant interest in *second screen* applications from both end-users and content providers. These applications enrich the television viewing experience in numerous ways. One of which is helping end-users effectively find and control content on television via spoken natural language (i.e. speech-driven TV program discovery).

Speech-driven TV program discovery applications have recently become available in the marketplace from select cable/satellite providers. However, these applications are limited. They support a pre-defined set of utterance types (e.g. *switch to <channel>*, *find a <genre>movie*, and *find a movie with <actor >*). Hence, end-users must conform to these types, and cannot combine them in an ad-hoc manner (e.g. search by genre, actor, and TV station).

More advanced research prototypes (Liu et al. 2012) do not have these limitations. However, these prototypes focus on a piece of the overall problem (e.g. entity recognition), and do not support the full range of features required of an end-to-end system. For example, these prototypes do not:

- Support question answering (e.g. *who is the french actress in the movie the dark knight*)
- Handle expressive utterances involving conjunction, disjunction, and negation (e.g. *find a movie without tom cruise and nicole kidman*)
- Handle the complexities of searching and controlling "live" television

In this paper, we report on an active R&D effort at Nuance Communications to develop an end-to-end speech-driven second screen application for television program discovery that addresses these limitations. Our solution integrates the following Artificial Intelligence (AI) and Natural Language (NL) technologies:

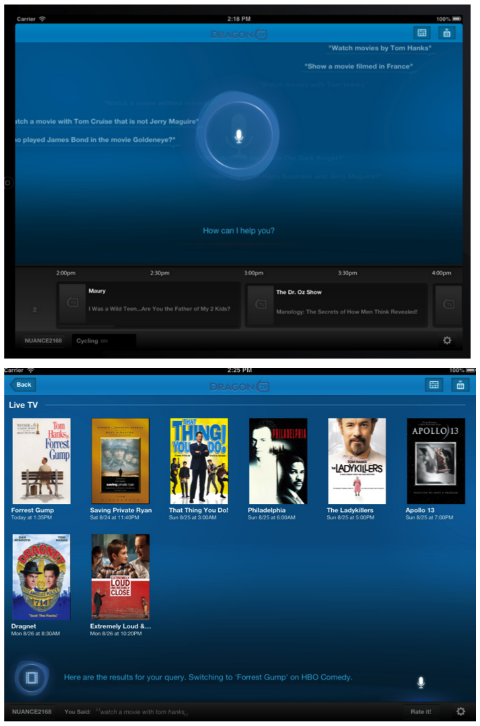
- Statistical and linguistic-based natural language understanding technologies (Ratnaparkhi 1996; Maxwell and Kaplan 1993) to construct a rich semantic representation of the end-user's utterance.
- A large-scale common sense knowledge-base (Cycorp 2013) that serves as the target output of linguistic processing and supports SQL query generation.
- Techniques from Natural Language Interface to Databases (NLIDB) (Popescu, Etzioni, and Kautz 2003) to transform the output of linguistic processing into a SQL query to execute against a commercial Electronic Program Guide (EPG) database, which is updated on a daily basis.
- NL generation technologies (Gatt and Reiter 2009) to summarize and confirm the outcome of acting on the end-user's utterance.

We give an overview of the main features of our application followed by a description of its architecture. We then present results from a user study along with a thorough analysis of the failure cases. Finally, we describe follow-on efforts to further enhance our application with the eventual goal of making it available to a large user population.

Application Overview

When a user starts the application for the first time, it will prompt the user for his/her zipcode and cable/satellite provider. The application uses this information to limit all results to the user's provider and viewing area. The user is then taken to a screen with a speech icon that he/she can tap on to begin speaking to the application (see Figure 1 top).

If the spoken utterance is a search request (e.g. *watch an action movie tonight* or *find a movie with tom hanks*), then the application will display all relevant results (ordered by start time) along with a confirmation of these results in the prompt box (see Figure 1 bottom). The user can scroll through these result, and tap on any one to view additional


Carrier 2:18 PM 100%

"Watch movies by Tom Hanks"

"Show a movie filmed in France"

atch a movie with Tom Cruise that is not Jerry Maguire"

o played James Bond in the movie Goldeneye?"

How can I help you?

2:00pm	2:30pm	3:00pm	3:30pm	4:00pm
Maury		The Dr. Oz Show		
I Was a Wild Teen...Are You the Father of My 2 Kids?		Manology: The Secrets of How Men Think Revealed!		

NUANCE2168 Cycling on

Carrier 2:25 PM 100%

Back

Live TV

Forrest Gump	Today at 1:35PM
Saving Private Ryan	Sat 8/24 at 11:40PM
That Thing You Do!	Sun 8/25 at 3:00AM
Philadelphia	Sun 8/25 at 6:00AM
The Ladykillers	Sun 8/25 at 5:00PM
Apollo 13	Sun 8/25 at 7:00PM
Dragnet	Mon 8/26 at 8:30AM
Extremely Loud &...	Mon 8/26 at 10:20PM

Here are the results for your query. Switching to 'Forrest Gump' on HBO Comedy.

NUANCE2168 You Said: "watch a movie with tom hanks" Rate it!

Figure 1: **Top:** Start screen. User can tap on speech icon to begin speaking to the application. **Bottom:** Results screen. A text confirmation is also shown at the bottom of the screen.

details such as the program synopsis, cast, ratings, etc. The user can also tap on the speech icon to issue additional utterances.

If the utterance is a question (e.g. *where was tom cruise born*), then the application will display the answer (i.e. Syracuse, NY) in the prompt box. It will also display any programs relevant to the question. For example, any Tom Cruise movies or TV shows that are playing.

If the utterance is a command (e.g. change channel, increase volume, etc.), then the application will execute the command. For channel change commands, the application will also display the programs that are currently showing on the new channel.

The application will prompt the user accordingly for utterances that it does not understand. Table 1 shows a sample of utterance types supported by the application.

Architecture Overview

Our application implements a client-server architecture (see Fig. 2). The client is responsible for calling Nuance's ASR service to convert the speech input to text,[1] displaying the results, and controlling the TV. The server is responsible for the natural language interpretation, retrieval of results, and response generation. We focus on the server in this paper, which is implemented as a hub-and-spoke architecture. Each spoke performs a specific task, and the hub invokes them in the proper order. Hence, the resulting system is highly modular, allowing future spokes to be added with minimal impact to the rest of the system.

[1] A detailed description of Nuance's Automatic Speech Recognition (ASR) is outside the scope of this paper. Information on a publicly available, commercial version of the ASR service that we used is available at: http://nuancemobiledeveloper.com.

Utterance Type	**Example**
Search: Multi-Slot	Action movies with Tom Cruise playing tonight.
Search: Hi-Precision	Find a French movie with a British actor.
Search: Logical Expression	Watch action movies without Tom Cruise or Bruce Willis.
WH-Question	Who directed the Dark Knight? Where was Terminator filmed?
Command	Switch to HBO.

Table 1: Sample of supported utterances.

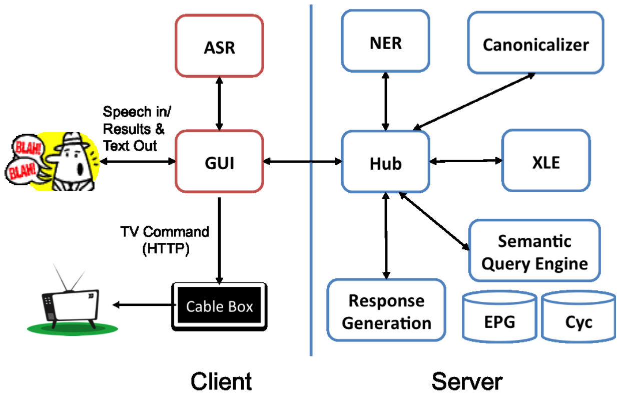


Figure 2: Architecture overview. The Hub invokes the spokes in a clock-wise manner starting with NER.

Ontology and Data Source

Our hub-and-spoke architecture requires a common representation across all the spokes. Moreover, this representation should support challenges that may occur during NL interpretation and SQL query formulation. We believe these requirements can be served by a large multi-purpose ontology, and chose ResearchCyc (Cycorp 2013) for this purpose. For example, the NER (see below) may have difficulty distinguishing between TV and movie titles. Cyc's rich subsumption hierarchy can provide one concept that subsumes both and can be the target for NER. In particular, the Cyc term `VideoConceptualWork` includes the desired categories of 'movie' and 'tv show', and excludes undesirable but related ones such as books or music. Similarly, XLE (see below also) can produce rich logical forms containing semantic relations grounded in Cyc between the entities detected by NER. Cyc's rich domain and range constraints on these relations can be used during SQL query formulation to further constrain the query.

Input	a tv show with jerry seinfeld playing this weekend
Output	a [TVShow-CW] tv show [/] with [Person] jerry seinfeld [/] playing [CalendarDay] this weekend [/]

Table 2: Example of NER input and output.

Our application also requires a continuously up-to-date database of programs playing on TV. We use a 3rd party commercial Electronic Program Guide (EPG) as our target database. This EPG is a relational database and contains schedule information for all cable and satellite providers in the U.S. and Canada for the upcoming two week period. It also contains additional information for each program such as the cast, filming location, birth dates of cast members, etc. Moreover, the EPG vendor also provides a daily update, which our system downloads and applies on a nightly basis.

Named Entity Recognition

The Named Entity Recognizer (NER) takes the ASR output from the client and detects proper nouns like movie titles and people names. It also detects other phrases that are not proper nouns but have significance in the TV domain, e.g. genres and time phrases. Table 2 shows an example of NER input and output where the tag for each detected entity is grounded in our target ontology.

Our NER is a *BIO*-style tagger where each word is tagged with `bX`, `iX`, or `o`, indicating the start of entity `X`, the continuation of entity `X`, or that the word is outside any entity, respectively. The NER is a machine-learned approach and uses the maximum entropy framework to predict *BIO* tags from annotated data, similar to (Borthwick et al. 1998). The model features and search algorithm are borrowed from the part-of-speech tagging approach of (Ratnaparkhi 1996), but the original contextual features have been changed to include:

- All consecutive word bi-grams in a window of ± 2 words from the current word
- The previous tag, and previous 2 tags conjoined with the current word

Our NER also uses list match features to flag phrases in the utterance that match those in an externally provided dictionary. We construct this dictionary by extracting ˜160K entries (i.e. movie and TV show titles, actor names, and role names) along with their type (i.e. movie, actor, etc.) from our 3rd party commercial EPG (see subsection above). Each word in a phrase is assigned a feature if the phrase has an *exact match* in the dictionary. The features are of the form `bY`, `iY`, `eY`, and denote the beginning, middle, and end of a phrase of type `Y`. A word can receive multiple list match features if it participates in multiple matches.

We apply the above feature patterns to the training data to create the actual feature set used by the model training algorithm. We use a combination of real and synthetic utterances for training (i.e. 19K vs. 166K utterances). The synthetic utterances are necessary because the real ones do not cover all the anticipated linguistic phenomena, and are generated using a combination of manually authored natural language patterns and dictionary derived from our 3rd party EPG.

Canonicalizer

The canonicalizer takes relevant entities detected by NER and maps them to the corresponding database element based on the surface form in the utterance. This mapping is necessary because of the mismatch between how a user may refer to an entity of interest (e.g. movie, actor, etc.) and how the entity is encoded in our target EPG. For example, a user may refer to the second terminator movie as *terminator two*, but the EPG may encode it as "Terminator 2: Judgment Day" (the official title).

We implement our canonicalizer using the open source search engine Solr because it provides a wide array of fuzzy match options (which is absent from most relational database systems), allowing us to fine-tune the match strategy. Hence, for each relevant entity (e.g. TV show, movie, actor, etc.) the canonicalizer performs a fuzzy match lookup of the entity's surface form in the corresponding Solr index, i.e. the index built over the EPG table and attribute corresponding to the entity's type. Each match result is a 3-tuple of the form $< T, A, I >$ where T is the table corresponding to the entity's type, A is the attribute in T containing the unique identifier for the entity, and I is the unique identifier. If there are multiple matches (e.g. "Avatar" referring to both the movie and animated TV show), then the top N, based on popularity, are returned.

These results are associated with their respective entity for use by downstream spokes to further constrain the SQL query during query formulation. Moreover, downstream spokes need only include the identifier (and not the surface form) in the resulting SQL query, which speeds up query execution.

XLE

Our system uses the Xerox Language Environment (XLE) (Maxwell and Kaplan 1993), which incorporates a Lexical Functional Grammar (LFG) parser and an integrated rule system, to parse input utterances and rewrite them into Logical Forms (LFs) grounded in our target ontology.

The LFG parser produces not just a single parse, but a packed representation (Maxwell and Kaplan 1993) that compactly encodes all the viable alternative parses of the utterance, e.g. encoding multiple prepositional phrase attachments. Moreover, entities detected by NER are used to control the parsing. For example, in "watch tom cruise" if NER tagged "tom cruise" as a Person type, then the parser will observe this tag, and not generate alternative parses for the phrase such as Tom being the subject of a cruise event.

The XFR rule system (Crouch and King 2006) rewrites the parse output into alternative LFs using three sets of rewrite rules. First, XFR rewrites the parse structure by adding WordNet (Miller 1995) word senses for each concept term (including NER entities) in the parse.

XFR then rewrites the resulting structure into alternative Abstract Knowledge Representation (AKR) formulae (Bobrow et al. 2005), which encode the space of possible thematic roles between the concept terms based on the alternative parses from the LFG parser. The AKR formulae also use logical contexts to capture various linguistic notions such as utterance type (e.g. question, command, etc.), disjunction, negation, etc. We note that the AKR representation serves as an intermediate representation that allows different ontologies to be supported, hence increasing the modularity of our system.

Finally, XFR rewrites the AKR formulae into alternative LFs in our target ontology. WordNet senses for each concept term are mapped to appropriate terms in the ontology. Thematic roles are mapped to predicates (i.e. semantic relations), and type-checking rules are applied to ensure terms are compatible with the arguments of the predicates, removing alternatives that are ill-typed. For example, the AKR representation of "play terminator two" has multiple WordNet word senses for "play", including one for playing a musical instrument and one for playing a movie. The former can be removed because "terminator two" is detected as a movie by the NER, and choosing it triggers a type violation. Additional structural rewrites may be performed to better align a LF alternative with the ontology (e.g. rewriting a set of binary thematic roles and their arguments into a ternary predicate).

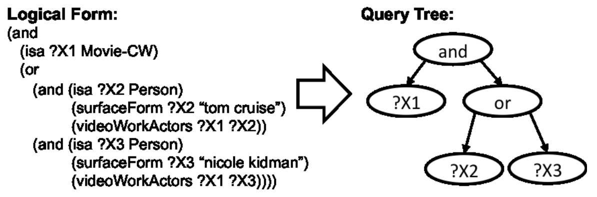
The resulting alternative LFs are scored using a set of simple heuristics that prefer the most common (i.e. frequently occurring) interpretation for the TV domain. For example, in "watch a movie with tom cruise on tv" it is unlikely that "tom cruise" will be sitting on the TV, so this alternative is scored lowly (and removed). Should multiple LFs (and hence unresolved ambiguity) remain, then one is randomly selected as the final result.

Semantic Query Engine

The Semantic Query Engine (SQE) takes the output of the NER and XLE spokes, and maps it to a SQL query. There are two approaches to this problem: 1) learn the mappings from an utterance to a target query (Zelle and Mooney 1996; Kate, Wong, and Mooney 2005); or 2) compose a query from manually defined mappings between linguistic and database elements (Popescu, Etzioni, and Kautz 2003). We adopt the latter approach because it does not require training examples, which is difficult to acquire at scale for this task.

SQE first tries to specialize each NER entity's type based on semantic relations between them produced by XLE. This step compensates for fine-grained types that may be difficult for NER to detect. For example, given the utterance *movies with tom cruise*, NER tags *tom cruise* as a *Person* type, and XLE relates *tom cruise* to *movies* via a *videoWorkActor* relation. Hence, SQE can retrieve the domain (and range) constraints of *videoWorkActor* from the underlying ontology. If this type constraint (i.e. Actor) is a subclass of the original type (i.e. Person), then SQE can specialize it.

Second, SQE adds structure to the bag of NER entities by traversing the XLE output (in a depth-first manner) to construct a query tree. Each logical connector (i.e. and, not, or) traversed is converted into an internal node. Each entity is converted to a leaf node, and attached to the most recent internal node traversed (see Figure 3). For compactness, an



Logical Form:

```
(and
  (isa ?X1 Movie-CW)
  (or
    (and (isa ?X2 Person)
         (surfaceForm ?X2 "tom cruise")
         (videoWorkActors ?X1 ?X2))
    (and (isa ?X3 Person)
         (surfaceForm ?X3 "nicole kidman")
         (videoWorkActors ?X1 ?X3))))
```

Figure 3: **Left:** Logical Form (LF) produced by XLE based on NER output for the utterance: *"movies with tom cruise or nicole kidman"*. isa clauses denote NER entities, and $?Xn$ are entity IDs. **Right:** Query tree constructed from LF.

and or *or* node with one child is removed, and its child is attached to its parent node. SQE uses this tree in the next step to generate nested-queries and to connect them.

SQE then maps each entity type into a SQL fragment: a 3-tuple of the form $< T, A, C >$ where T is the database table to include in the *from* clause of the query, A are relevant attributes from T to include in the *select* clause, and C is a set of constraints to include in the *where* clause. Each constraint is a 3-tuple of the form (A', V, Op) where A' is the constraint attribute from T, V is the constraint value on A', and Op is the constraint operator (e.g. equality, membership, etc.). We manually define these mappings based on our target EPG database. Canonicalizer results (see above) associated with the entity are also added to C. For example, the tuple for *tom cruise* (an *Actor* type) and associated canonical is:

$$< credit, name, \{(type, `Actor', =), (id, 30914, =)\} >$$

Based on these mappings, SQE finds the shortest join path between the tables in each fragment pair via a breadth-first search over possible joins in the database. SQE also observes the structure of the query tree, and greedily merges fragments with overlapping database elements (i.e. tables and attributes).

Finally, SQE checks the type of the utterance produced by XLE. If the type is a WH-question, then SQE includes the table and attribute associated with the question type in the *from* and *select* clause of the query respectively, and extracts the return value as the answer. This strategy is sufficient because many WH-questions can be answered by applying the appropriate facet over the set of results satisfying the question constraints. The resulting SQL query is executed against the EPG.

Natural Language Response Generation

The Natural Language Generation (NLG) component generates responses across three categories:

1. **Confirmation Prompts**: A restatement of the constraints requested by the user. With noisy ASR and NER, confirmations let the user know whether his/her request was understood correctly. In cases where no results are found, the system will also indicate this.
2. **Answers**: Presentation of possible answers found for WH-questions posed by the user. Additional processing,

such as converting the time represented in the EPG to local time, are performed based on the question type.

3. **Exception Responses**: Responses to inform the user of exception conditions, e.g. NER did not detect any entities, no answers were found for a question, etc.

This component uses templates, the SimpleNLG package of (Gatt and Reiter 2009), and transformation heuristics to generate concise prompts. SimpleNLG allows us to more easily enforce common grammatical constraints such as number, noun-verb, and article-noun agreement. We predefine a set of SimpleNLG syntax tree templates, and our system selects the appropriate one based on the slots and values that need to be expressed in the prompt. The selected template is instantiated appropriately, and relevant transformations (e.g. suppressing portions of the template) are applied based on the context (e.g. number of results, result type, etc).

For example, if the NLG component is asked to generate a prompt for the slot-value tuple (`genre` = ”romantic comedy”, `type` = ”movie or tv show”), it will suppress the `type` slot (if the result includes both movies and TV shows) to generate the concise response “romantic comedies”. A pure template-based approach will generate the more verbose response “romantic comedy movies or tv shows” . This strategy allows our system to better handle variation, brevity, and fluency of natural English.

Evaluation

We present a joint user study with Nuance’s Usability & Interaction Design group to assess the performance of our application.

Experiment Design

We used the following experiment design to answer three key questions:

1. How satisfied is the user with the application?
2. How effective is the application at finding results matching the user’s intent?
3. What is the response time of the application?

We sourced 16 subjects from a third party staffing agency for this study. These subjects represent target users of our application: users between the ages of 18 and 55, mix of males and females, and half have technical backgrounds (the other half do not).

For each subject, a moderator first gave the subject a high-level overview of the application and the experiment environment – i.e. a simulated living room with a TV that can be controlled by our application (see Figure 4).

The moderator then gave the subject instructions for a practice trial. The subject was informed of a stack of magazines in the living room, and asked to relax as if he/she is at home. While the subject is relaxing, he/she was told to flip through these magazines for inspiration on what to watch on TV. The subject was then told to tell the application what they wanted to watch. Based on the results returned, the subject was asked to:

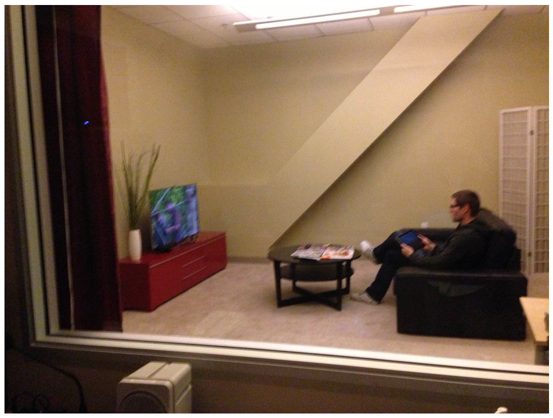


Figure 4: Picture of the simulated living room taken through a one-way mirror from the adjacent moderator room.

- Rate their overall satisfaction on a 7-point Likert scale.
- Assess the effectiveness of the application by scoring the trial as a success or failure. A trial is successful if: 1) at least one of the results on the first page matched the subject’s intent or 2) the application correctly gave no results when no program matching the subject’s intent is showing on TV.

The application also logged the time spent to process the request.

After the practice trial, the moderator instructed the subject to perform ten additional trials following the same instructions as above. During these trials, the moderator observed the subject in an adjacent room through a one-way mirror, and only interacted with the subject if he/she had any questions (or experienced any technical issues).

This design is unintrusive (putting the subject at ease), and limits the introduction of biases. Subjects were not exposed to example utterances, which may bias their requests. They came up with requests entirely on their own.

Results

A total of 160 trials were completed (10 per subject). Two raters reviewed each trial to determine those that are out-of-scope – e.g. requests in adjacent domains such as music or unsupported requests such as showing trailers of upcoming movies. A total of 39 trials – where both raters agreed as out-of-scope – were removed. 5 additional trials were removed because the moderator had to intervene due to technical issues, and 13 trials where the subject incorrectly recorded his/her ratings were removed as well. The remaining 103 trials were used to compute the results.

Figure 5 shows the user satisfaction ratings. The average rating is 4.81 on a 7 point scale with a standard deviation of 1.69. This result is encouraging given the in-the-wild nature of the trials – i.e. subjects were allowed to pose any request that came to mind to the application. Moreover, this result is statistically significant compared with a random baseline that assumes an uniform expected frequency over the ratings ($p < 0.01$ for the chi-square goodness-of-fit test, $df = 6$).

Failure Type	Description	# Trials
Incorrect NER	NER spoke incorrectly detected an entity (i.e. wrong phrase boundary or type).	17
Missed NER	NER spoke missed an entity critical to processing an utterance.	11
Incorrect ASR	ASR component produced incorrect text output for speech input.	6
Failed DB Mapping	SQE spoke failed to map an entity required to formulate correct SQL query.	4
Incorrect LF	XLE spoke produced a LF that prevented the generation of correct a SQL query.	3

Table 3: Top five failure types, and number of affected trials. A trial can have multiple failures.

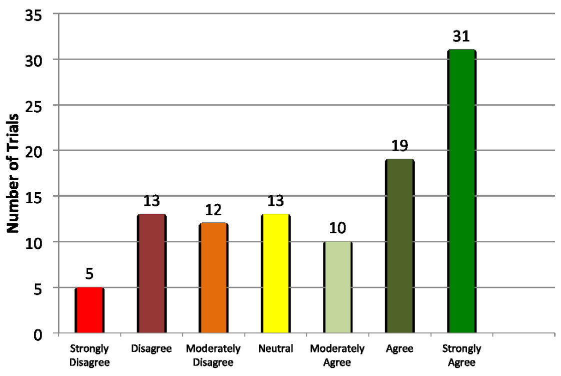


Figure 5: User rating distribution over a 7 point scale in response to the question: *"The system was able to provide the information I was looking for."*

# Successful	# Failed	Accuracy (%)
64	39	62.14%

Table 4: Number of successful vs. failed trials, and overall task accuracy. We define task accuracy as the number of successful trials over the total number of trials.

Table 4 shows the number of successful vs. failed trials. Again, these results are encouraging given the in-the-wild nature (and hence difficulty) of the trials. We also found a strong positive correlation between the percentage of successful trials and the average satisfaction rating per subject ($p < 0.005$ for the Spearman rank correlation, $\rho = 0.8338, n = 16$). This positive correlation supports the validity of the satisfaction ratings.

Finally, the average response time of our application across all trials is 828.41ms ($sd = 1097.77ms$). None of the subjects expressed concerns over the response time during the evaluation, but this is an area that can be improved upon.

Failure Analysis

We performed an analysis of the failed trials to better understand the cause. For each failed trial, we identified the spoke that caused the failure, and categorized the nature of the failure. Table 3 shows the top-five failure categories.

From this analysis, we identified the NER spoke as the top source of failure. Incorrect or missed NER accounted for 28 of 39 failed trials. The primary reason for these failures is that the combination of real and synthetically generated training examples did not fully cover the breadth of user requests and linguistic phenomena that occurred in practice, resulting in an under trained NER model.

Further analysis confirmed that 24 of these trials would have been successful had the NER performed correctly, increasing the number of successful trials from 64 (62.14%) to 88 (85.44%). Hence, we are actively investigating ways to improve the performance of this spoke.

Another interesting failure is Failed DB Mapping, which occurs when a subject refers to a database element at a different level of granularity. For example, a subject may want to watch a show in the "home decorating" genre, but the EPG only encodes the more generic genre of "home & garden". This granularity mismatch may be resolved using logical inferences (e.g. subsumption), which we are investigating.

Conclusion and Follow-On Efforts

In this paper, we presented a speech-driven second screen application for TV program discovery. Our application has several unique features such as the use of a common ontology across the different components, support for a wide range of expressive natural language utterances, and operating at scale (i.e. enabling TV program discovery over "live" EPG data that covers the entire U.S. and Canada). We also presented a user study along with a detailed failure analysis. The results were encouraging given the in-the-wild nature of the evaluation. Moreover, our failure analysis helped us identify key components of the application to target for improvement.

Further enhancements and investigation, however, are needed before our system can be made available to a large user population. First, we are actively addressing the top failure types from our failure analysis. For example, we can achieve a significant performance lift by improving the robustness of the NER. We are also developing additional enhancements to our application – such as user preference modeling, rich user conversation, etc. – to further improve the end-user experience. We plan to evaluate the practical utility of these enhancements, as they become available, through additional user studies. Finally, we are actively investigating possible ways in which our system could be embedded within existing mobile virtual assistant capabilities and solutions to support rich user requests (and interactions) in the TV domain. We plan to report on these efforts as progress is made.

Acknowledgment

The authors would like to thank Ronald M. Kaplan for his encouragement, guidance, and support on this project. The authors would also like to thank the anonymous reviewers for their helpful feedback and suggestions for improving the paper.

References

Bobrow, D.; Condoravdi, C.; Crouch, R.; Kaplan, R.; Karttunen, L.; King, T. H.; de Paiva, V.; and Zaenen, A. 2005. A basic logic for textual inference. In *Proceedings of the AAAI Workshop on Inference for Textual Question Answering*.

Borthwick, A.; Sterling, J.; Agichtein, E.; and Grishman, R. 1998. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 152–160.

Crouch, D., and King, T. H. 2006. Semantics via f-structure rewriting. In *Proceedings of the LFG06 Conference*, 145–165.

Cycorp. 2013. Research Cyc. http://www.cyc.com/platform/researchcyc.

Gatt, A., and Reiter, E. 2009. SimpleNLG: A realisation engine for practical applications. In *Proceedings of ENLG-2009*, 90–93.

Kate, R. J.; Wong, Y. W.; and Mooney, R. J. 2005. Learning to transform natural to formal languages. In *AAAI*, 1062–1068.

Liu, J.; Cyphers, S.; Pasupat, P.; McGraw, I.; and Glass, J. 2012. A conversational movie search system based on conditional random fields. In *INTERSPEECH*.

Maxwell, J., and Kaplan, R. 1993. The interface between phrasal and functional constraints. *Computational Lingusitics* 19(4):571–589.

Miller, G. A. 1995. WordNet: A lexical database for english. *Communications of the ACM* 38(11):39–41.

Popescu, A.-M.; Etzioni, O.; and Kautz, H. 2003. Towards a theory of natural language interfaces to databases. In *IUI*, 149–157.

Ratnaparkhi, A. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 133–142.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, 1050–1055.