Utilizing online content as domain knowledge in a multi-domain dynamic dialogue system

Craig Wootton, Michael McTear, Terry Anderson

School of Computing and Mathematics, University of Ulster, Belfast, Northern Ireland
{wootton-sl, mf.mctear, tj.anderson}@ulster.ac.uk

Abstract

Recent research in dialogue systems has investigated the feasibility of relying on information extracted from the Internet as a source of content and domain knowledge. However, this information needs to be processed and prepared into a form understandable by the dialogue manager. The number of domains and web sites are often restricted to a finite number, with prior knowledge of the site structure itself usually required by the dialogue manager. We present an architecture which demonstrates that multi-domain dialogue, relying on information extracted from online sources, is possible without the need for human intervention or knowledge of the site structure itself.

Index Terms: Spoken dialogue system, dynamic dialogue system, RSS, API, online information retrieval.

1. Introduction

Typically, spoken dialogue systems are developed and delivered within a well defined and closed domain. Possible paths through the dialogue are usually fixed, prompts and recognition grammars are often hand crafted, and both dialogue and domain knowledge are generally tightly coupled into the same system. More recently, advanced dynamic dialogue systems have been developed in which paths and utterances are not fixed or pre-defined. The domain knowledge is usually required to be separated from the dialogue manager, often available in some accessible and structured way, such as a database or ontology.

Content stored online, however, is somewhat different from this well structured and defined domain knowledge. Recently, research has overcome this by preparing in some way the online content extracted from the Internet, as defined by the designer during development. This limits the operation of the dialogue system to a finite set of domains and Internet sites for which it has been specifically developed.

Currently, no system is capable of truly non-restricted dialogue, based on information stored online in an unstructured manner, separate from the dialogue manager. We have developed a system that can utilize content from any domain, and is not required to be prepared in any way, allowing for opportunistic dialogue based upon current information held online.

The remainder of the paper is structured as follows. Section 2 introduces the system and its architecture, with section 3 introducing the current focus of research, the content manager. A brief evaluation of some preliminary work is presented in section 4, followed by related and future research presented in sections 5 and 6.

2. System description

A dialogue system has been developed that can use information held online as its basis for dialogues. Users ask questions, and the system then encourages dialogue with the user based upon the answer. It is assumed that queries expected from the user are those which are common, everyday tasks that occur during normal browsing, such as reading the news, or booking a flight ticket. Example questions might be, “What was that on the news about Tony Blair?” or “Did England win their football match last night?” or “I would like to book a flight to London.” The system must then decide from which online source to extract the content.

Within the system, information is classified as one of two types – task-based, for example flight booking or purchasing an item from an online shop, or information-based, such as requesting a news story or sports report.

2.1. System architecture

The system utilizes a modular architecture that supports the separation of domain knowledge from dialogue knowledge, and is also based on a client-server paradigm, where the client represents the device for interaction, and the server represents the main system and its components. This will support the independence of the device from the dialogue system, allowing devices of different types and capabilities to interact with the system and vice versa.

The architecture is presented in Figure 1, showing the main components of the system, the dialogue manager and, the content manager.

2.1.1. Dialogue manager

The dialogue manager has the role of encouraging dialogue between system and user. Functionalities include maintaining the language model, applying understanding to the user’s input, constructing a query for content understandable to the content manager, interacting with the user model, and generating the outputs to be sent back to the clients. Of particular importance for a dynamic dialogue system is the creation and maintenance of the language model. This is achieved in our system by utilizing the functionality of the content manager (see section 3). By using the techniques included in the content manager to extract information from the online sources, the dialogue manager currently uses this information to create an XML based grammar for use with the VoiceXML dialogue specification. For initial dialogues, the grammar will consist of all the words from the document space (N), explained in 3.3.

2.1.2. Content manager

The novel component of the architecture is the content manager, which includes the content spotter and the current feeds available in the environment (see 3.3). The content manager is key to the overall operation of the system, being responsible for managing these feeds, accepting the key words of the user’s query from the dialogue manager, extracting the information from online sources and delivering this content back to the dialogue manager.
3. Content manager

3.1. Role of the content manager

The inclusion and functionality of the content manager makes our dialogue system different from typical dialogue systems that are developed within a finite set of domains, using well structured domain knowledge.

Previously, online information not structured in a standardized way has been prepared into a structure which is meaningful for the dialogue manager (section 5). Although possible in a limited domain dialogue system, where only a finite set of sites need to be investigated, this would not be possible with a multi-domain system, as it would be impossible to predict the structure of sites that are initially not known to the dialogue manager.

The main role of the content manager is to extract the content from the Internet. Functionality is not limited by domain, content type or location of content, only by the number of RSS and API feeds available to the environment. The operation of the content manager is reliant on a number of key tasks, such as having a standard method of interacting with the different feeds, a method for choosing which feed, or document, best matches the current query, and a method for retrieving this information from its online source. Additionally, where task-based dialogues are concerned, such as booking a flight, the content manager must have an understanding of the required parameters for the particular API, and derive the values of these from the user in a suitable way. For an API from Expedia, for example, this could be departure airport and date.

3.2. Using RSS and API feeds

The use of RSS and API feeds from various content providers provides a method of accessing structured content of various online domains. More specifically, the content manager makes RSS and API feeds available to the content spotter, as shown in Figure 1, represented by $X_1$, $X_2$, … $X_N$, where N is the total number of feeds within the environment.

For RSS feeds, the source of the feed can simply be made known to the content manager, and are used to drive informative dialogues. API feeds, however, are more challenging, requiring more effort due to their non-standard format, and are used to drive the task based dialogues. An API from Amazon will be entirely different from one from Expedia, for example – not only in terms of functionality offered and required parameters for operation, but also in how it is represented in terms of mark-up and specification.

To operate generically, APIs should be transparent to the dialogue manager, and this has been catered for in the functionality of the content manager, which, to the best of our knowledge, is the first example of a system which can handle varying types of API specification generically. The content manager includes a module to perform API requests and responses. This module enables it to handle any type of API request and response generically irrespective of the API specification, from flight bookings to purchasing a book on eBay. When an API specification is added to the environment, the developer is required to declare any required and optional parameters using basic XML syntax.

Once the information has been retrieved from the relevant RSS or API feed, a grammar understandable by VoiceXML is created dynamically and the content can also be inserted into a VoiceXML <prompt>, and also a visual XHTML page, depending on whether VoiceXML or X+V is currently in use.

3.3. Content spotter

Within the content manager, the content spotter chooses the most appropriate source of content. Comparable to the domain spotter of the Queen’s communicator [1] and also evaluators of the JASPIS architecture [2], the content spotter is a mechanism available to the content manager for making decisions with respect to the online sources and choosing the most relevant content based upon the user’s query.

To understand how the content spotter handles a query, such as “What was the sports news?” one must first understand the structure of an RSS feed. Each feed is specific to a particular topic, for example Sports News, containing many <item> elements, each of which is a different story of that topic. A typical RSS feed consists of between 20-30 different <item>-elements.

The process of the content spotter can be split into two different tasks, that of preparing the input query (Q), and that of preparing the document space (N), illustrated in figure 2.
of the same format, a cosine similarity function can then be applied to match the most appropriate document from \( (N) \) to \( (Q) \), where

\[
\cos \theta = \frac{v_1 \cdot v_2}{|v_1||v_2|} \tag{1}
\]

\( v_1 \) and \( v_2 \) are the term-frequency vectors of \( (Q) \) and \( (N) \) respectively. The cosine similarity function produces a result \( [0,1] \) where a value of 0 indicates no similarity and 1 indicates identical documents. Given a query \( (Q) \), the module can match documents \( (N) \) from a range of feeds to \( (Q) \), and return the best matched result. In order to do this, a cosine similarity function is performed on the bag-of-words (BOW) of the query and document vectors. The formula for the similarity function is:

\[
V_d = [W_{1,d}, W_{2,d}, ..., W_{N,d}]^T \tag{2}
\]

\[
W_{i,d} = tf_i \cdot log \frac{|d|}{|e_i|} \tag{3}
\]

In (3), \( tf_i \) is the term frequency of \( t \) in a document \( d \), and \( D \) is the total number of documents in the document space. \( log \frac{|d|}{|e_i|} \) is the logarithm of the inverse document frequency. The cosine similarity function polls all the available documents \( (N) \), matching those documents \( (n) \) which are in the term space, calculating the term weights \( (tw) \) of each document, computing the vector magnitudes and finally normalizing this vector space, producing the most similar document to \( (Q) \).

Currently, the best matched result is returned to the user. This can be output as a VoiceXML or multimodal X+V document. The returned document also acts as the foundation for the next VoiceXML grammar, on the assumption that the user’s next query often relates to the previous response.

If the returned result is an API to handle a task-oriented dialogue, XPATH and XSLT can be used to query the XML specification and generate the required <form> and <fields> in either VoiceXML or X+V to collect the values. The module then performs the actual API request from the vendor. Once the response is sent back, the module can use XPATH and XSLT to generate the VoiceXML form or X+V multimodal form to be output to the user.

### 3.4. Implementation technologies

The dialogue manager is entirely represented as VoiceXML, and XML is used within the system for communications between components. Further compliance of standards is also observed by relying on RSS and API technologies, both of which are implemented as XML based technologies, as are XSLT and XPATH, both of which are used for querying documents and generating required output. ASP.NET is used as the scripting language.

## 4. Evaluation

### 4.1. Experimental setup

Preliminary experiments have been conducted to test the effectiveness of the content spotter, as this is the key module of the architecture. Table 1 gives an overview of the experimental setup which was used. During the experiment, text input was used to avoid recognition errors, leading to results that were a reflection of the operation of the functionality of the content spotter. Inputs to the system were natural language queries randomly produced to test the document space of the included content.

### 4.2. Results

<table>
<thead>
<tr>
<th>EXPERIMENTAL RESULTS</th>
<th>Overall</th>
<th>Excluding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of queries</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Relevant results returned</td>
<td>77%</td>
<td>87%</td>
</tr>
<tr>
<td>Irrelevant results returned</td>
<td>12%</td>
<td>13%</td>
</tr>
<tr>
<td>‘Out of domain’ returned</td>
<td>11%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 2: Experimental results

Table 2 highlights the main findings of the evaluation. Overall, 26 queries were performed, each one illustrative of a single user utterance in a dialogue. 77% of results returned by the content spotter were considered relevant, 12% not relevant and 11% recognized as out of domain. A result was classified as relevant if an appropriate answer was given, and out of domain represents a query that has been asked to which the answer is not currently available to the content manager.

If we assume that someone using the system would already have some knowledge of the topic, and exclude the 11% that were out of domain, then the ratio of relevant results increased to 87%.

### 4.3. Discussion

The results show that it is possible for a dialogue system to use RSS and API feeds for extracting content from different domains as 87% of relevant results was obtained. Although these results are preliminary, the next stage will be an evaluation on a larger scale once the architecture has been further developed.

It should be noted that, although the core mechanisms of the architecture were operating through the evaluation, it is the performance of the cosine similarity function that determines the relevant or irrelevant retrieval of documents. This is a tried and tested mechanism that has been used in other dialogue systems, such as call routing [3] and spoken document retrieval [4].

Furthermore, the current implementation of the cosine similarity function was performed using a bag-of-words (BOW) approach. Once a larger evaluation has been conducted, it is hoped that these results can be used as a benchmark for a further study, where the BOW can be compared to an enhancement of the similarity function, in the hope of improving the number of relevant results returned. It is thought the cosine similarity function can be improved by applying some weighting function to key words or terms, or by using WordNet to solve sense relations between words. As the BOW approach of the cosine similarity function simply matches literal words, similar contexts and terms went unmatched during the evaluation, such as “carry on” and “go

---

ahead”, leading to the relevant result in the document space not being returned to the user.

5. Related research

5.1. Dynamic dialogue

The area of dynamic dialogue can be categorized into two different areas. The first area concerns those systems in which the entire dialogue system is generated from scratch dynamically, motivated by the minimum amount of human effort possible. An example of such a system is GEMINI [5].

Closer to our research is the second area of dynamic dialogues relying on the abstraction of domain knowledge to generate prompts and individual dialogues during run-time. The domain knowledge can be represented in the system in a variety of ways, usually in a database or structured XML, as in the system presented by Montoro et al. [6]. Here, an ontology represented in XML is used to generate the dialogues to allow the control of devices in the home.

Other systems have investigated the use of online content to dynamically create dialogue. GENESIS [7] and a system presented by Pargellis et al. [8] both re-use information held online as a means for producing prompts and dialogues. Unlike our system, however, either a domain structure is required to be created that is well defined, or the information can only be extracted from a limited set of online sources, where the site structure and markup have been investigated and are known by the dialogue manager.

5.2. Related Architectures

Various advanced dialogue architectures have been created to solve particular problems, such as better adaptivity or utilization of mobile devices. The JASPIS architecture was developed with both objectives in mind [2]. Based on the concepts of agents, managers and evaluators, it introduces the polling and selection of agents capable of each handling a specific task. The Queen’s Communicator, a different architecture, takes a similar approach, based on experts and a domain spotter [1]. Each domain that the task-based dialogue system can handle is represented as an expert object, and the spotter must choose the most appropriate expert to handle the current request. The CONVERSE system has a similar mechanism based upon an auctioneering approach, where each action module bids for their chance to handle the interaction based upon its belief of how well suited it is to do so [9].

6. Future work

The next step in the research is to include a more advanced natural language understanding module. Statistical methods to language understanding are currently being investigated to achieve more flexible dialogue, leading to a comparison of a VoiceXML approach to one containing a Sphinx recognizer, N-gram language model and a our own dialogue manager. A further comparison study will evaluate the BOW approach against an enhanced version of the similarity function found in the content spotter.

Another area for future development is the user manager component which could incorporate technologies such as collaborative filtering and the use of recommender algorithms when used in a spoken dialogue system as opposed to their traditional use in graphical systems.

Research is also ongoing investigating the management of different devices and their capabilities within the environment, so that the best output device could be determined and suggested by the system, depending on what type of content and media is to be output to the user.

7. Conclusions

In this paper a novel approach using RSS and API feeds has been discussed for extracting content and domain knowledge from online sources for use in a dialogue system. We believe that the architecture and the mechanisms used overcome the limitations of preparing domain knowledge for use in a dynamic dialogue system, enabling the system to construct an opportunistic dialogue with a user using content delivered from online sources that have not been prepared or structured for this particular system in any specific way.

8. References


