SEMANTIC MODELING FOR DIALOG SYSTEMS IN A PATTERN RECOGNITION FRAMEWORK

Kuansan Wang

Microsoft Research, One Microsoft Way
Redmond, Washington 98052, USA

ABSTRACT

In this paper, we describe a multimodal dialog system based on the pattern recognition framework that has been successfully applied to automatic speech recognition. We treat the dialog problem as to recognize the optimal action based on the user's input and context. In analogous to the acoustic, pronunciation, and language models for speech recognition, the dialog system in this framework has language, semantic, and behavior models to take into account when it searches for the best result. The paper focuses on our approaches in semantic modeling, describing how semantic lexicon and domain knowledge are derived and integrated. We show that, once semantic abstraction is introduced, multimodal integration can be achieved using the reference resolution algorithm developed for natural language understanding. Several applications developed to test various aspects of the proposed framework are also described.

1. INTRODUCTION

Pattern recognition is a problem of determining how to associate a given signal with an element from a collection of candidates so that such association satisfies certain optimal criterion. Many problems have been treated as pattern recognition problems with great success, among them, automatic speech recognition. Here the signal x is the acoustic utterance, and the candidate set consists of all possible word sequences w generated from a given lexicon. Assuming uniform cost, the optimal criterion leads to the well-known maximum *a posteriori* (MAP) decision rule where the best choice is the one that maximizes

$$P(w \mid x) = P(x \mid w)P(w) / \sum_{W} P(x \mid w)P(w) . \qquad (1)$$

The probabilistic measures $P(x \mid w)$ and $P(w)$ are called acoustic and language models, respectively.

We have been investigating how the framework can be extended to speech understanding for multimodal dialog systems [1]. The idea is based on the observation that the mission of speech understanding is to reach an optimal semantic representation that best describes the user's utterance given the dialog history. The dialog system then synthesizes proper actions based on its understanding of user's intent. Let S_n denote the semantic representation of the n-th dialog turn. We view the speech understanding problem is to arrive at a semantic representation that maximizes

$$P(S_n \mid S_{n-1}, x) = \sum_{w} P(S_n \mid w, S_{n-1})P(w \mid x, S_{n-1}) . \qquad (2)$$

Note that, other than having to account for dialog history S_{n-1}, evaluating the conditional probability $P(w \mid x, S_{n-1})$ in (2) is a central task in recognition. Assuming the dialog context does not affect the acoustic model, it is straightforward to derive from (1) that dialog context can be incorporated into the recognition process if a dialog state dependent language model $P(w \mid S_{n-1})$ is used. This often means the probabilities of certain words or phrases need adjustments based on the context of the dialog. Commercially available systems (e.g., [2]) often use probabilistic context free grammar (PCFG) for language modeling. Dialog states can be incorporated dynamically by adjusting the weights on the relevant PCFG production rules for each dialog turn. Recently, studies in normalizing the probabilities of phrases and word lists within N-gram are also conducted [3]. It is shown that the perplexity of the language model is better controlled than the unadjusted N-gram, and hence the recognition performance benefits considerably.

As in the case of speech recognition, the understanding system implements an optimization algorithm to search for the best semantic interpretation using the scores from the language model as well as the *semantic model*, denoted as $P(S_n \mid w, S_{n-1})$ in (2). Although (2) treats the recognition outcome as a hidden process, and hence the semantic evaluation theoretically should sum over all possible outcomes, we have found it beneficial to introduce N-best or Viterbi approximation commonly used in recognition to understanding as well. Specifically, one can approximate (2) as

$$P(S_n \mid S_{n-1}, x) \approx \max_{w} P(S_n \mid w, S_{n-1})P(w \mid x, S_{n-1}) .$$

In this paper, we focus our discussion on two aspects of semantic modeling. First, the inference of the current semantics S_n must take into account the past history S_{n-1} as well as the domain knowledge. Semantic model therefore must include *domain experts* that can interface with the knowledge source to assess the likelihood of an inference. We address the issue of knowledge representation in Sec. 2. Also, a pattern recognition framework must specify how patterns are specified. As words are basic building blocks for patterns in speech recognition, we call the building blocks for understanding *semantic objects*. Semantic model must address how semantic objects are defined, and how they can be fused into representing complex semantics. Such *semantic schema* can be viewed as a "lexicon" for understating, and is discussed in Sec. 3.

2. KNOWLEDGE REPRESENTATION

Many knowledge representation methods have been attempted, including the database model we used in our system. The

database model describes domain knowledge in terms of tables and the links of the columns of the tables. A row of a table represents a specific data item, whereas the columns describe the attributes of the data. In the following, we call a row in the database an *entity*. Since a table is basically a collection of data of the same type, we use the term *entity type* to refer to a table.

Consider a simplified example of modeling a stock trading firm using relational database. This domain may have a table listing all the assets the firm is currently holding. Columns on this table may consist of stock symbol, owner of the stock, transaction date and price. Another table records the information of all the customers of the firm, including names, address, phone numbers, balance, and privileges. The knowledge of the amount of stocks a customer currently owns is modeled by a query into the asset table with the search key on the owner column. The knowledge of whether a customer can sell a particular stock is represented by a more elaborated query that first checks the asset table to see if the customer currently owns the sufficient amount of the stock and, if not, checks the account table to see if the customer has the privilege to short sell. In general, the database model is most straightforward for domains where changes in the domain state can be simply modeled as database updates, and a question of whether something is possible can be answered by a query into the database. Not all combinations of database updates and queries make sense. A major task of knowledge engineering is to define a set of database operations that are useful and permissible on the domain. These operations are called *functions* of the domain. Not all the functions should be exposed at NL semantic level. For instance, it makes sense to have stock quote query and trade functions, but not adding or deleting an account through verbal commands.

In addition to entities explicitly defined in the backend, we have found the concept of negation a powerful means in representing knowledge. To this end, we introduce a notion of *anti-entity* into our system. Anti-entities are entities known to be impossible for a particular context. For example, after the user answers negatively to an explicitly confirm question, the entities in the question are anti-entities and their linguistic realization can be weighted down in the language model.

In practice, however, even though the domain knowledge can be conceptually modeled by a relational database, the access to the domain knowledge is not necessarily implemented as a database. In many cases where objected oriented programming style is practiced, the domain usually manifests itself as an object model where entity types are modeled by objects. The properties of an object describe the attributes (columns) of the entity, and the relations among entity types are often encapsulated as the methods of the object. The most common way of describing an object model is through Interface Definition Language (IDL), an ISO 14750 standard. Recently, the object access mechanism has been ported to Web and standardized in XML. A Web-based object implementation, in which objects can be instantiated and method calls executed by requests in XML Protocol, is called a Web service.

To reduce the complexity of interfacing with these types of knowledge sources (direct database tap, conventional object model, and Web services), we introduce an abstract layer called *domain expert* in our system to shield the variations of the back end from the rest of the system. The mission of a domain expert is to translate the semantic interpretation into database queries, object model or web service function calls as appropriate, and convert the results back into the semantic representation.

3. SEMANTIC SCHEMA

There are two notions of semantics in our system. *Surface semantics* refers to a representation of user's intention from the raw signal that is independent of the syntactical structure and the input modalities being used to convey the intention. *Discourse semantics*, on the other hand, refers to the system's belief of user's intention after domain knowledge is applied. In this work, both surface and discourse semantics are represented in tree structures. The nodes of the surface semantic tree are called *semantic objects*, while the nodes of the discourse semantic tree can be either semantic objects or domain entities.

3.1 Semantic Objects

Since we only model semantics in the context of domain specific goal-oriented dialog, semantic objects are linguistic embodiments of either domain entities, domain functions, or common sense entities such as numbers, currencies, and basic date/time specifications. The task of semantic evaluation is simply to uncover what each semantic object is supposed to represent. Our system uses a simple algorithm to traverse the surface semantic tree in the depth-first manner [1]. Sequentially, the domain expert corresponding to the domain for the semantic object is invoked. If the semantic object refers to a domain entity, the domain expert returns a representation of that entity. If the node on the tree refers to a domain function, the operations defined by the function are executed and the domain expert reports back the execution results. In either case, the returned instance then replaces the semantic object on the tree. The process continues until the root of the tree is encountered, or the domain expert reports an error. To facilitate reference resolution, the semantic engine also maintains an entity memory [1].

In our implementations, semantic objects are declared in an XML document called semantic schema. Each semantic object is declared as an XML element and its constituents as sub-elements. One of the constituents of a semantic object is the URL of its domain expert. To insure domains can work together without conflicts, each domain defines its own XML namespace.

When a user utterance encompasses multiple domains, the semantic objects are combined together dynamically based on their type compatibility. Take the PIM task in MiPad (Sec. 4) for example. The "send mail" function of the "email" domain declares one of its constituents, recipient, is of the entity type "person". The "calendar management" domain declares a semantic object "organizer" of entity type "person" that corresponds to the individual initiating a particular appointment. A user utterance "send mail to the organizer of the review meeting" will lead to a surface semantic where the "organizer" semantic object is embedded into the "send mail" object as a recipient because their types are compatible, even though these semantic objects are from different domains. During semantic evaluation, the domain expert of "calendar management" will be invoked first to resolve "organizer of the review meeting" and, if no error occurs, followed by the "email" domain expert. Note

that, since semantic objects are combined only after a user's utterance, new domains can immediately enjoy interoperability with others as soon as their semantic schemas are published. Say the user signs on an on-line music management system that declares a semantic object of type "person" that represents the authors of a song. As soon as the semantic schema is published to the system, the user can seamlessly include this new domain to the existing ones by saying "send mail to the creator of this song." The understanding of this utterance is distributed through the respective domain experts across the Web, and the developers of the "email" domain do not have to anticipate the existence of the music domain in order for their systems to work together. Semantic objects and domain entities can be generated via graphical user interface (GUI) as well. For example, a user can click on a photo on the screen while saying "send mail to this person." The click here creates a domain entity as a cross-modality referent. Semantic objects and entities from all modalities are stored in the entity memory, and the system merges them into the semantic tree using the reference resolution algorithm [1]. In other words, once objects are created by the raw input device, the rest of the semantic evaluation can operate in a modality transparent manner.

In addition to modality transparency, our design also supports a concept we call *turn insensitive principle.* The idea is semantic representations should be insensitive to exact number of dialog turns through which they are obtained. The concept is inspired by the observation that a naïve user usually takes more turns to covey the same message an experienced user would do in a single turn. As long as the semantic contents are identical, we argue that the semantic representation, discounting the anti-entities, should be the same as well. In other words, turn insensitive principle aims at abstracting the semantics from the details of user interaction. All our implementation examples suggest the principle is quite enforceable.

3.2 Dynamic Schema Principle

Our semantic schema is designed to support a concept known as *dynamic schema principle*. The idea originates from the design of XML. Since XML allows everyone to invent new markups on demand, there must be a mechanism to insure the new markups can be recognized by others. At minimum, there must be certain contract for each XML document specifying how the document is organized through these markups. This contract is called an XML schema. Any XML source may publish an XML schema to instruct its user how to make sense of its outputs. Statically published schemas, however, require the XML producers and consumers to be constantly in sync. For example, banks that request XML credit reports from a credit bureau should be aware of the XML schema of the reports. If the credit bureau adds functionality and changes the schema, all the banks need to update their programs processing the XML reports from the bureau. Depending upon how often and how much the schema is changed, the update efforts can be straightforward or demanding.

Dynamic schema, on the other hand, lets the consumers dynamically specify the return format they want. In the above example, some banks would like to have the unique identification number to be attached to each person as an XML attribute, some might prefer it to be a sub-element. A bureau can accommodate these different needs by letting the banks to specify the format for each request in a standardized schema definition language. Since the bureau always formats its result based on the request, it does not have to synchronize with all its clients when features or functionality are added to its offering. On the other hand, the banks have more freedom in deciding whether and when to take advantage of the new offerings by including them into the dynamic schema. The banks can also act freely to experiment alternative formats because doing so has minimum impact on their content providers.

As one of our design goals is to facilitate dynamic understanding across multiple domains, we embrace the concept of dynamic schema in semantic schema so that declarations of semantic objects can be changed on demand. Rather than statically defining the semantic representation the discourse manager will send to the domain experts, our system allows each domain to register its semantic schema dynamically with the discourse manager. The semantic schema then acts as the data model for both surface and discourse semantics, namely, the discourse manager will format the semantic representation based on the schema before sending it to the respective domain expert, and expect the domain expert to follow the same rules in return so that the discourse manager can interpret the results. We define a schema definition language in XML, called *semantic definition language* (SDL), for specifying semantic schema. SDL was described in [1]. Through this language, each domain can dynamically change the semantic lexicon as needs arise.

4. IMPLEMENTATION EXAMPLES

We implemented several applications to test the framework described above, the most extensive one being the project called MiPad [4]. The goal of MiPad is to speech enable the Personal Information Management (PIM) tasks which consist of domains such as accessing to email, calendar, contact list, notes, etc. The physical device is a mobile PDA that has sound capturing capability. A distinctive feature of MiPad is a general purpose "Command" field to which a user can issue naturally spoken commands such as "Schedule a meeting with John tomorrow at two o'clock." The system will recognize and parse the utterance and, in response, pop up a "meeting arrangement" page with proper fields filled based on user's utterance. The multiple PIM domains are combined together through the distributed understanding mechanism described in Sec. 3.1. MiPad uses Microsoft Outlook object model as the backend knowledge source.

"Tap-and-talk" is a key feature of MiPad's user interface design. The user is asked to use the stylus as the pointing device to indicate the field he would like to issue command to. A speech command can be uttered after the user taps on a field on the screen. The user lifts the stylus after the command is fully uttered. This design helps a spoken language system in the following two aspects. First, tap-and-talk enlists the user's effort to end-point the speech, thereby reduces the end-pointing errors that can be made by the recognizer. Second, tap-and-talk also functions as a user-initiative dialog-state specification. The dialog focus that leads to the language model used for recognition is entirely determined by the field tapped by the user. A user can navigate freely using the stylus. In contrast to most

speech-only applications, there is no need for MiPad to include any special mechanism to handle spoken dialog focus and digression. The graphical display also simplifies dramatically the dialog design. For instance, all the inferred user intentions are confirmed implicitly on the screen. Whenever an error occurs, the user can correct it on using soft keyboard or speech. The display also allows MiPad to confirm and ask the user many questions in a single turn.

MiPad's contact list domain was later extended as a directory assistant application based on Microsoft human resource database [1]. While MiPad has a fixed UI based on PDA with wireless network access, the goal of directory assistant is to explore the system's adaptability to multiple access modalities, specifically, a speech-only and a speech-enabled GUI browser. Both the speech-only and the GUI versions share the same knowledge and semantic understanding system. The only part that differs is the user interface. We use a template based response generation algorithm authored in XSLT to transform the discourse semantic XML into HTML for GUI and text-to-speech markup language for the speech-only system [1]. Without a visual display, the speech-only templates are slightly more complicated as dialog functions like explicit confirmation and error recovery must be handled in speech, an error-prone input method. These functions are handled at the end user device and, owing to the turn insensitive principle (Sec. 3.1), the semantic representations received at the discourse manager are identical for speech-only and GUI cases. We note that, even though spoken dialog appears to be very different from GUI interactions, the underlying dialog infrastructure and programming model can be the same.

In contrast to MiPad where the backend knowledge source is an object model, directory assistance utilizes a backend that is a relational database. Through the abstract layer of domain expert (Sec. 2), it is demonstrated that various types of knowledge sources can be seamlessly knitted together in spoken language. This idea is further stretched and tested for a stock trading application, in which the knowledge sources consist of a database with two tables as described in Sec. 2, and a stock quote Web service from MSN Investor. As in the case of directory assistant, the stock trading application can be accessed through a speech-only interface or a GUI browser on a PDA with tap-and-talk. The stock trading domain can be integrated to the email domain in MiPad directly through the distributed understanding mechanism described in Sec. 3.1. An example command "send mail to Microsoft share holders" will prompt the discourse manager to invoke domain expert from stock trading app to extract the email addresses for the email domain. Since MiPad was developed long before stock trading app, this shows applications that work with each other do not have to be designed specifically for each other.

For more elaborated multimodal integration, we implemented a geographical application that allows a user to query points of interest and plan travel routes on a map. We used Microsoft MapPoint as our knowledge source and added spoken dialog capability through MapPoint's object model. This domain is designed to understand utterances like "Where is the nearest gas station", "show me driving directions to get there", "add this <click> to the route." We develop a GUI "parser" that translates a point-and-click event into a location, and plan to enhance the parser to recognize an area specification through a dragging event. As mentioned in Sec. 3, we view the multimodal integration a reference resolution problem. The above three sentences contain examples of ellipsis ("… gas station [*from the current location*]"), anaphora ("… to get *there*"), and cross-modality deictic reference (mouse click). We have found the reference resolution algorithms designed for natural language processing can be extended for multimodal integration once the raw signals from difference modalities are converted into a unified semantic representation. In addition to multimodal integration for a single turn, the memory mechanism also allows cross-turn cross-modality reference resolution. For example, an apparently ambiguous reference "add *the restaurant* to the route" might not indeed ambiguous because the user had clearly indicated his choice through GUI in the previous turn. In this particular application, all the clicks on the map are "parsed" into an entity of type location and stored in the turn memory (Sec. 3.2). Although our underlying speech platform [2] provides a mechanism called recognition bookmark that allow us to denote the timing of each GUI event to the acoustic stream, we found it useful to preserve only the sequential order for these events. The detailed time stamps are not robust to user's behavior as speech may be uttered before, after, or amidst a GUI action, all appear natural to the user.

5. CONCLUSION

Our experiments seem to indicate it is viable to treat multimodal dialog as a pattern recognition problem. As object-oriented design has been proven very useful for GUI, we have found doing the same to speech simplifies the process of extending GUI applications to be speech-aware. Throughout our experiments, we notice that semantic modeling requires less intensive efforts than language modeling. In order for speech to become a mainstream UI, the process of developing language model needs to be simplified. Recently, we have started exploring how semantic schema can help developing PCFG based language model [5], and whether structure N-gram [6] can be used to learn the structure of surface semantics.

Acknowledgement

The MapPoint application was developed by Mr. Issam Bazzi while he visited Microsoft as a summer intern.

REFERENCES

[1] Wang K., "A plan-based dialog system with probabilistic inference measures", *Proc. ICSLP-2000*, Beijing, China, October 2000.

[2] Microsoft Speech Application Program Interface (SAPI), http://www.microsoft.com/speech.

[3] Wang Y., Mahajan M., Huang X., "Unified Language Model", *Proc. ICASSP-2000*, Istanbul Turkey, May 2000.

[4] Huang X. *et al*, "MiPad: A next generation PDA prototype", *Proc. ICASSP-2001*, Salt Lake City, May 2001.

[5] Wang Y., "Grammar learning for spoken language understanding." Submitted to ASRU-2001.

[6] Chelba C., Jenelik F., "Structured language modeling", Computer Speech and Language, 14(4), pp.283-332, 2000.