A DOMAIN MODEL CENTERED APPROACH TO SPOKEN LANGUAGE DIALOG SYSTEMS

Dale Russell
Motorola Human Interface Laboratory

ABSTRACT

This paper presents a view of Spoken Language Dialog Systems in which a Domain Model is the unifying feature, providing a common representation of knowledge about the domain of application that is shared by many components of the system. The information in the Domain Model is used in different ways by the grammar and parser, the speech recognizer, the dialog manager, and the back end interface. Sharing a common knowledge base allows these components to operate together seamlessly, while maintaining modularity.

1. THE MOTIVATION

A Spoken Language Dialog System (SLDS) typically consists of a number of interrelated modules, including speech recognition, natural language understanding, dialog management, real-world reasoning, an interface to a back end, language generation and speech synthesis. As these components involve many diverse technologies, it is desirable to develop them independently and then to integrate them into a complete system, with the ability to remove one version of a given component and substitute a different version, in order to achieve the best overall performance.

However, while these components are in some ways independent, they must also interact within an SLDS. Therefore, it is useful to have a common representation of the entities in the domain of application, shared by all components which utilize knowledge about that domain. This representation must be flexible enough to accommodate the requirements of different components of the system, and broad enough to encode all of the relevant information. In addition, it must be formal enough to allow for computational manipulation, and provide the basis for the development of components of the SLDS which make use of domain knowledge.

Such a formal representation of domain knowledge is referred to as a Domain Model (DM). A Domain Model should capture, in an intuitive form, an expert’s conceptualization of the entities in the domain and their characteristics. These are referred to as Domain Objects and Domain Attributes, respectively.

2. THE STRUCTURE OF A DOMAIN MODEL

Domain Models have been implemented with a variety of structures[3][4], though the major elements are fairly consistent. The structure of Domain Models described in this paper is very simple. A Domain Model consists of a name, specifying the domain, and an unordered collection of Domain Objects. All information about relationships among objects or actions performed on objects must be contained within one or more of the Domain Objects themselves. A Domain Object consists of the name of the object and an unordered collection of Domain Attributes. A Domain Attribute consists of the name of the attribute and either the type of its value, the value itself, or both.

There is no preset range of possible value types, but all value types are either atomic, complex or lists. Atomic value types include (but are not limited to) integers, real numbers and strings. A complex value is a Domain Object, often (but not necessarily) one which also exists as a top-level object within the given domain. The value of an attribute may also be a list, where the elements of the list are of any well-defined value type.

With such a structure, a Domain Model can be implemented straightforwardly in an Attribute-Value Matrix (AVM) structure. The top-level node of the AVM corresponds to the Domain Model itself, with each of the Domain Objects constituting a node immediately under the top-level node. The node for each Domain Object contains a Feature-Value pair for each of its attributes. The feature names the attribute, and the value can be any of the atomic, complex or list values allowed for that feature. Where a feature value is complex, it is represented as an embedded AVM structure.

We distinguish several related types of Domain Models, each of which is appropriate for different components of a Spoken Language Dialog System. First, a Complete Instantiated Domain Model encodes the system’s knowledge about the domain, and is used in the interface of the SLDS with the world knowledge being handled by the computer. Second, an Abstract Domain Model encodes information about what constitutes a possible object in the domain, and is used in the construction of grammars for language understanding and generation. Finally, a Partially Instantiated Domain Model represents a piece of the world reflected in the user’s query and the context of the exchange, and is used by the dialog manager. Each of these three types of Domain Models will be discussed in turn.

3. THE COMPLETE INSTANTIATED DOMAIN MODEL

A Complete Instantiated Domain Model (CIDM) encodes the knowledge that the system has about the elements of the given domain. It is “complete” in the sense that it includes all the knowledge that the system possesses about the objects, attributes and values of the domain at a given point of time. This does not, however, mean that every attribute must be included with every object, or that all possible attributes or values actually occur in the CIDM. The CIDM represents the sum total of what the system knows about the world. However, not everything about the world may be known, and even if it is, the current state of the world may not encompass all possible domain attributes and values.
The term “instantiated” in a Complete Instantiated Domain Model means that every attribute of every object has a value specified. If the value of an attribute is unknown, that attribute is not included with the object in the Complete Instantiated Domain Model. Therefore, not every attribute that is defined for an object must be included with it in a CIDM.

The Complete Instantiated Domain Model may be static or dynamic. A static CIDM is one that is not expected to change during the operation of the system. A static CIDM might represent the knowledge contained in a relational database taken from a reference work, where the SLDS is designed only to retrieve information from the database, not to update it. A dynamic CIDM is one whose attribute values, and even the attributes themselves, can change over time. A dynamic CIDM could be created to reflect the information contained in a web page that is constantly being updated, or a database whose information can be modified by the SLDS.

The following is one small piece of a Complete Instantiated Domain Model for an application dealing with restaurants in the Chicago area, with the Domain Model name “Chicago_Dining.” Two instances of the Restaurant Domain Object are shown, followed by one each of the Shopping_Center and Credit_Card Objects.

```
( Chicago_Dining
  ( Restaurant
    ( Restaurant_Name
      ( value “Blind Faith Café” )
    )
    ( Address
      ( value “525 Dempster Ave.” )
    )
    ( City
      ( value “Evanston” )
    )
    ( Reservations
      ( value Accepted )
    )
    ( Credit_Card
      ( value "Visa", "Discover" )
    )
  )
  ( Restaurant
    ( Restaurant_Name
      ( value “Let Them Eat Cake” )
    )
    ( Address
      ( value “179 W. Washington Ave” )
    )
    ( City
      ( value “Chicago” )
    )
    ( Located_In_Shopping_Center
      ( value “Lakeview Plaza” )
    )
    ( Accepted_At
      ( value "Blind Faith Café" )
    )
  )
  ( Restaurant
    ( Restaurant_Name
      ( value “Blind Faith Café” )
    )
    ( Address
      ( value “179 W. Washington Ave” )
    )
    ( City
      ( value “Chicago” )
    )
    ( Located_In_Shopping_Center
      ( value “Lakeview Plaza” )
    )
    ( Accepted_At
      ( value "Blind Faith Café" )
    )
  )
)
```

Figure 1: Portion of a Complete Instantiated Domain Model for a Chicago restaurant application.

In this figure, complex attribute values which are themselves other objects have been abbreviated by simply writing the name of the object. In the actual internal representation of the Domain Model, this value would be an entire instantiated object, or a co-reference to one[6]. In this sample from a CIDM, it should be noted that the two instances of the Restaurant Object do not have the same set of attributes. This is because only those attributes with a specified value are included. Blind Faith Café is not known to be located in any shopping center, and no information is available as to whether “Let Them Eat Cake” accepts reservations, so these attributes are omitted.

The attribute values in a CIDM do not necessarily reflect the full range of possible values for that attribute, but only the values that happen to be instantiated. For example, in the Chicago restaurant CIDM, a Restaurant object has an attribute Restaurant_Name. The values of this attribute as they occur in the Chicago_Dining Domain Objects represent the complete set of names of actual restaurants included in the database. However, the CIDM itself does not include any information on what constitutes a possible restaurant name, such as the set of legal characters.

4. THE ABSTRACT DOMAIN MODEL

An Abstract Domain Model (ADM) specifies the parameters of a possible object in the given domain. It encodes a type hierarchy which represents the complete set of objects in the domain, attributes for each object, and values for each attribute. For each attribute, the ADM also specifies whether it is obligatory or optional, and gives the type of the value(s) it takes. For example, the ADM contains the information that a given attribute must have a value whose type is string, but does not include any particular string values. This is in contrast to the CIDM, which includes all current values for each attribute, but does not specify the limits on what those values can be.

If an attribute can take any arbitrary string as its value, this information is stored in the type information for this attribute in the ADM. However, if the attribute can only take a certain set or range of strings as its value, this information must be included as well. For example, while the set of strings specifying a restaurant name is unbounded, only a finite number of strings can encode the name of a state of the U.S. This set of strings is specified in the ADM. There may also be restrictions on possible attribute values. For example, a string representing a person’s surname can never contain numerical characters, while other types of strings, such as those for restaurant names and street addresses, can. This restriction can be enforced by defining a new value type, which is a subtype of string, and limiting the characters of this new type to a subset of the full set of characters.
While a CIDM and an ADM are not identical, it is possible to make use of a CIDM in the construction of an ADM, with input from a human developer who knows something about the domain. If the Complete Instantiated Domain Model contains at least one instance of each domain object and attribute, then the objects and attributes of the Abstract Domain Model may be derived straightforwardly from the Complete Instantiated Domain Model. In such a case, the ADM contains exactly one instance of each object and one instance of each attribute for each object. The type of each attribute value is inferred from the values specified in the CIDM, along information concerning restrictions on value types. This type information is stored in the ADM, but not the instantiated values themselves.

In cases where the information in the CIDM does not exhaust the possible range of domain objects and attributes, the ontological and typing information of the Abstract Domain Model will need to come from some other source. In such cases, this information may be elicited from a domain expert or extracted from a database specification. In most instances, the CIDM, whether derived from a database, web page, or some other source, will be less than completely exhaustive. If the CIDM is reasonably representative of the domain of application, it should at least contain an instance of every object in the domain. If the Complete Instantiated Domain Model contains at least one instance of each object in the domain, and probably instances of most of the domain attributes. However, in most applications, the CIDM will not include instances of every possible attribute value, particularly for value types such as real number and string. Value type information and value restrictions must therefore be inferred or elicited for inclusion in the ADM.

The following is an ADM corresponding to the CIDM given in Figure 1 above:

```
( Chicago_Dining
  ( Restaurant
    ( Restaurant_Name
      ( obligatory TRUE )
      ( type string )
    )
    ( Address
      ( obligatory TRUE )
      ( type string )
    )
    ( City
      ( obligatory TRUE )
      ( type string )
    )
    ( Reservations
      ( obligatory FALSE )
      ( type ReservationType )
    )
    ( Located_In_Shopping_Center
      ( obligatory FALSE )
      ( type Shopping_Center )
    )
    ( Credit_Card
      ( obligatory FALSE )
      ( type list<CreditCardType> )
    )
  )
  ( Shopping_Center
    ( Shopping_Center_Name
      ( obligatory TRUE )
      ( type string )
    )
    ( Contains_Restaurants
      ( obligatory FALSE )
      ( type list<Restaurant> )
    )
    ( Credit_Card
      ( Credit_Card_Name
    )
  )
)
```

Figure 2: Portion of an Abstract Domain Model for a Chicago restaurant application, corresponding to the Complete Instantiated Domain Model of Figure 1.

The Abstract Domain Model provides the representation of domain knowledge which is used by the semantic component of the grammar.[2] As user utterances about the domain are parsed, their output is a semantic structure which expresses statements or queries about the domain of application. This semantic structure is created from the elements of the ADM, and thus provides a representation of user utterances which is compatible with other components of the SLDS.

5. THE PARTIALLY INSTANTIATED DOMAIN MODEL

During the course of spoken language interactions with the user, the SLDS will create Partially Instantiated Objects (PIOs), and from these, Partially Instantiated Domain Models (PIDMs). These are formal representations of concepts referenced by the user, or jointly constructed by the user and the system, within the domain of application.

A Partially Instantiated Object is formally identical to an object of a Complete Instantiated Domain Model. However, while a CIDM represents the system’s knowledge about actual entities, a PIO is typically used to represent descriptions of hypothetical entities formed by the user, which may or may not correspond to actual entities. While a CIDM represents the sum total of the system’s knowledge about the domain of application, the PIDM, made up of PIOs, represents a history of entities which have been referenced during the course of an exchange, generally a small fraction of the total number of entities in the domain.

5.1. Partially Instantiated Objects in User Utterances

In interacting with a SLDS, the user produces utterances involving the entities of the domain and their attributes. In a database query system, for instance, the user may pose a query about the existence of an object with certain specified attribute values. The interpretation of this query, the semantic representation which results from parsing it, includes a Partially Instantiated Object (PIO) representing this hypothetical object. The system then determines whether any actual objects correspond to the Partially Instantiated Object.

A typical user utterance in a SLDS involves not only a Partially Instantiated Object, but also an action to be performed on that object. In a database query system, the user may direct the system to return a boolean indicating whether or not the posited PIO exists; to return the number of objects that are consistent with the partially instantiated object; or to return other attributes of objects that are consistent with the partially instantiated
object, if any exist. Examples of each of these types of queries follow.

Q1: Are there any Chinese restaurants in Evanston that accept American Express?
Q2: How many Thai restaurants are there on 55th Street in Chicago?
Q3: What is the address of “Let Them Eat Cake” in Chicago?

The semantic structure which results from parsing these queries is translated into a form which is compatible with the interface to the back-end, such as SQL. For example, Q3 above translates into a query to find an object in the CIDM which has the name “Let Them Eat Cake” and the city “Chicago.” The PIO created for this query is shown in Figure 3. As part of the information retrieval, this PIO is matched against the objects of the appropriate type in the CIDM, such as that listed in Figure 1. 4 For all such objects that match, the value of their Address attribute is returned.

![Figure 3: Partially Instantiated Object created for the query “What is the address of ‘Let Them Eat Cake’ in Chicago?”](image)

5.2. Partially Instantiated Domain Models for Dialog Management

One or more such partially instantiated objects constitute a Partially Instantiated Domain Model. PIDMs are used to keep track of the context of the dialog, by maintaining records of the objects, real or hypothesized, that have been referred to by the user in the course of the dialog. This is useful in anaphora resolution and in the interpretation of context-dependent utterances. For example, in processing Q3 above, the system creates the PIO of type Restaurant shown in Figure 3. This object has values for two attributes: “Let Them Eat Cake” for the Restaurant_Name attribute, and “Chicago” for the City attribute. This PIO is entered into the record of the context of the dialog, such as by placing it on a stack of most recently referred to objects. 5

Even if it should turn out that this PIO does not correspond to any actual object, the PIO is still available for reference further on in the dialog. For example, suppose that the system response to Q3 is “I’m sorry, there is no restaurant by the name of Let Them Eat Cake in Chicago. The user can then continue the dialog by saying, “How about in Evanston?” The phrase “how about” is a signal to repeat the previous query, with the most recently referred to objects, updated for any new attribute values specified. In this case, the only new information is “Evanston,” which is recognized as a city name. The most recently referred to PIO that is consistent with this attribute is retrieved from the context stack, updated for the new value of the City attribute, and used in the new query.

6. CONCLUSION

This paper has described three main types of Domain Models, each with its own use in SLDSs. The Complete Instantiated Domain Model encodes the system’s complete knowledge about the entities in the domain of application, and serves as the basis for the creation of the Abstract Domain Model. This in turn delineates the objects and attributes of the domain, showing not the objects that actually exist, but the parameters of objects, attributes and attribute values that can possibly exist. During the course of a dialog between a SLDS and a human user, Partially Instantiated Objects are created, and are used as the basis for carrying out commands in the system. A set of Partially Instantiated Objects constitutes a Partially Instantiated Domain Model, which is used in dialog management. These three types of Domain Models are related yet distinct. Together, they provide a representation for Domain Objects and Attributes which makes for ease of communication among the components of a Spoken Language Dialog System.

7. REFERENCES


---

4 Ensuring that “Let Them Eat Cake” is identified as a restaurant name and “Chicago” as a city name, and that these attributes and the Address attribute are all associated with a Restaurant object is the responsibility of the grammar writer.

5 This issue of how such matching is done is not addressed in this paper. Clearly, it cannot be done simply by unifying the two AVM structures, as an unspecified value (a missing attribute) in the objects of the CIDM would unify with any value in the query, incorrectly resulting in a match.

6 The specifics of how dialog management are implemented are not discussed here. Maintaining a context stack [1] for each type of object in the domain, is one possibility, but far from the only one.