Dialogue Management in the EMBASSI Realm

Using Description Logics to Reason about Ontology Concepts

Yves Forkl[*]*, Bernd Ludwig*[*]*, Kerstin Bücher*[†]

[*]*FORWISS and* [†]*AI chair of the CS Institute of the University of Erlangen-Nuremberg, Germany*

Keywords

Dialogue Management, Description Logics, Ontology, Domain Modeling.

Abstract

The integration of utterances in their contexts, by the help of reasoning about the concepts of the domain – organized in a formal ontology and laid down in Description Logics –, enables successful dialogue management for spoken language.

We outline our dialogue model and our approach of establishing a formal ontology within the EMBASSI scope.

1 Backdrop: the EMBASSI project

EMBASSI[1] wants to provide easy access for everybody to complex technical systems (A/V home theatre, car devices and public terminals), encouraging multi-modal user input. Besides a speech parser, our contribution consists, first, of a dialogue manager working with description logics, and second, a formal ontology which is modeling the application domain and serves as a basis for dialogue management.

2 Generic Dialogue Management in EMBASSI

2.1 Levels of Utterance Analysis

In a dialogue, some of the utterances cohere with the dialogue for application pragmatic reasons, while some utterances are coherent in terms of dialogue pragmatics. In each case, a particular level of the utterance explains how and why it is coherent. In a computational dialogue model we use the different levels as follows: Given a new utterance, its satisfaction state for each level is computed on the basis of the information available and the background knowledge about the dialogue and the application. Under the assumption that all knowledge is partial, three different possible satisfaction states (unique, ambiguous, none) for each level suffice to explain how the new utterance coheres with the current dialogue. By processing a dialogue it is attempted to obtain a unique satisfaction state for each level in order to define and execute a unique task and to update the information state.

2.2 Syntactic Level

The syntactic level involves parsing a lattice of word hypotheses, using a two-step model for syntactic derivation. In a first step, edges in the lattice are grouped in syntactic chunks [1].[2] In a second step, these fragments are combined into bigger units obeying semantic and pragmatic constraints, the applicability of which is checked by valency and case frames of the lexical units.

2.3 Semantic Level

A phrase such as *the film on ZDF* is assumed to be syntactically unique. However, in a more complex phrase such as *tape the film on ZDF* the attachment of the prepositional phrase is possible in more than one way, i.e. the phrase is ambiguous on the syntactic level. Such ambiguities induce different semantic readings. Analogously, lexical entries are ambiguous if they belong to more than one category or to different semantic types. Different readings show up in the Discourse Representation Structures (DRS)[3].

The semantics of an utterance may be unique, ambiguous, or impossible to construct. We have to distinguish between the meaning to be assigned to

[1] "Elektronische Multimediale Bedien- und Service-Assistenz", sponsored by German Fed. Ministry of Research

[2] This step is performed by a chart parser with a chunk grammar, working primarily with a head-driven bottom-up strategy.

[3] See [2] for an introduction to Discourse Representation Theory (DRT)

an utterance in terms of a terminology defined for an application (*intension*) and the enumeration (i.e. *extension*) of all objects satisfying the intension. We represent intensions with Description Logics (DL)[4]. Intensions may be ambiguous. If such ambiguities are detected, clarification dialogues are necessary to obtain a unique reading.

On the application pragmatic level, it must be validated whether the extension (i.e. the interpretation of the DRS) constructed for the utterance is consistent with what is known about the state of the application scenario. There are three possible outcomes of this analysis: **empty set**, **singleton set**, and a **set of several elements**. For each case, the dialogue has to be continued in a different way.

2.4 Discourse Pragmatic Level

Our approach to speech act recognition is based on two main ideas: First, speech acts are actions in a discourse. Therefore, basically, the recognition of (possible) speech acts depends on whether the preconditions for a speech act(ion) are satisfied in the current discourse situation. Speech acts are defined independently from applications, and their number is kept small in order to enable robust analysis. Second, preconditions are formalised by using the notion of expectation and obligation introduced into a discourse by previous utterances, by describing how the propositional content of an utterance contributes to the recognition of a speech act. Finally, assumptions about the knowledge of the dialogue participants are taken into account. Again, there are three possible outcomes of the analysis: First, **no speech act can be identified**, second, **several possible speech acts are detected**, third, **a unique speech act** is recognised.

2.5 Satisfaction of an Utterance

If a unique speech act has been recognised, a unique dialogue goal consisting of the speech act and the content of the utterance can be established. To continue the dialogue coherently, it is necessary to know whether the goal can be achieved. For this purpose, we introduce the notion of plans as a means to abstract from application-specific algorithmic solutions. The dialogue goal can be achieved **uniquely** if a single plan satisfies it. The goal **is unsatisfiable**, if no plan could be developed, or **ambiguously satisfiable**, if several plans have been found. In this case, they should all be presented to the user.

[4]For an introduction to DL see e.g. [3]

2.6 Result

In the case of a unique satisfaction of the dialogue goal, a problem solver executes the corresponding action and reports the status of the execution back to the dialogue manager (DM). If the action has been executed **successfully**, the result is communicated. In the **case of an error**, the user should be informed about the possible cause. Otherwise, the result consists of a number of **different alternatives** which are presented to the user.

3 A Formal Ontology for EMBASSI

3.1 Necessity of a Formal EMBASSI Ontology

In our approach, which features a declarative modeling of the system's evolving *information state* rather than following a simple FSA-based procedural strategy, a fine-grained and well-structured ontological hierarchy of semantic concepts is extremely important to permit useful logical inferences on these concepts using a theoreme prover.

The novel architecture of the EMBASSI project brings up two problems to be mastered in the *domain model* (or ontology): First, the processing of user input is separated from the execution of system operations, introducing an interface between the dialogue manager and the applications which are controlled by so-called "assistants". Second, there is a great variety of application components and different manufacturers.

Both, bidirectional communication – of user goals that are sent to the "core" and of messages that are received from the system's executive components –, as well as a multitude of functions available in various applications, depend on a clear and unambiguous language spoken between the dialogue manager and the applications/assistants.

The EMBASSI ontology furnishes this language, which functions as a standardized system-wide terminology encompassing the whole world of functions and objects referred to by all applications and assistants for audio/video control at home and in the car. Each of these functions and objects must be both uniquely named and given a precise semantic definition as a *concept*. For any concept, it is crucial that it be integrated at the appropriate place in the hierarchical semantic network established by the ontology. This also holds for the con-

cept's logical relations to other concepts, i.e. the *role* its instances may play in the definition of another concept, so a role compares to a feature or a slot.

One key feature of the EMBASSI system is that it allows poly-modal user input, e.g. speech, gestures, and pointing. The PMI (poly-modal input) fusion component is responsible for recognizing the same user intention expressed equivalently in any one mode: hence, possible user goals need to be represented as mode-independent unique semantic representations. References between objects originating from different modes have to be resolved by the PMI in order to interpret their meaning: the device the user points to has to be linked to the pronoun in the simultaneous utterance "turn it on". To this end, the ontology provides for the representation of co-reference and of descriptions that allow to analyse reference relations logically.

3.2 Organization of the Domain Model

The formalism used to represent the ontology is *Description Logics*, in the form of CICLOP[5], a java implementation of a prover and a system for managing DL theories (i.e., terminologies plus assertions about individuals).

The ontology (or domain model) is organized in three domain areas, reflecting a fundamental partitioning of the dialogue manager's "world": First, the **Linguistic Domain**, comprising the speech parser's linguistic knowledge, especially from lexical semantics. Second, the **Discourse Domain**, which contains internal concepts (independent of a specific input mode, but supporting poly-modal processing) of the dialogue manager. They reflect its actions and the kinds of objects processed in the context of interacting with the user and the core system. Third and most important, the **Application Domain**: this is the area where knowledge about the functionality of all applications, assistants and devices is held. It equals the "domain model" in the classical sense, i.e. a semantic description of the functions and objects pertaining to the domain of application. In contrast to the preceding two areas, the terminology defined here is shared among the dialogue manager and the assistants and applications (the core system), for they need this language to exchange commands and requests.

[5] see [4]

3.3 Joint Elaboration of the Ontology

The modeling of the application domain demands a joint effort of the ontology creators on the one side and the application and assistant developers on the other side, for only the latter have precise knowledge of the funcionality they offer in their component. Any component's function and its parameter types must be integrated correctly in the ontology.

As an example, turning on a device would be done using the concept **TurnOn** together with its role **device** which serves to indicate the concerned device. To restrict possible role fillers to instances of devices, the fillers' concept type is required to be **Device**. Each concept is derived from a higher-level ontology concept: **TurnOn** is a subconcept of **DeviceCommand**, **Device** is subsumed by the more general concept **Component**.

We asked application and assistant developers to deliver descriptions of the functionalities of their components which we transformed into a formal ontology. To facilitate both tasks, we proposed a standardized specification scheme, exploiting the similarities between our goal format – CICLOP-style description-logic definitions – and the object-oriented programming languages commonly used by the developers: both techniques are centered around hierarchical and inheritance structures that are very much akin.

Consequently, our scheme prescribed Java definitions of class hierarchies for object and function types which we would turn into a hierarchy of DL concepts. Mandatory derivation of all classes had to start from a set of predefined ontology base classes representing the *Upper Model*, i.e., the top-level branches of the application domain ontology. The type of each of a method's parameters has to be declared and must equally derive from one of the predefined base classes for data types.

Each object or function class definition had to be side-stepped by a verbose comment explaining its semantics, i.e. detailing the purpose or content of the defined entity.

In order for the dialogue manager to understand, i.e., be able to reason with errors, any error statement must confine to notions that have precise semantics and are part of the ontology. Thus, the application and assistant developers must give a semantic definition for any error type that their component may signal to the dialogue manager. For instance, the VCR *vcr1* refusing to execute a re-

quest submitted using the **Record** concept due to a lacking tape should be able to "explain" this to the dialogue manager by sending back an instance of the concept **NoTape** (probably in conjunction with an indication of *vcr1* as a filler of that concept's role **tapeDevice**.)

The various Java descriptions of applications and assistants had to be corrected and merged, trying to minimize the number of concepts and share as many as possible of them among several partners in the EMBASSI project. The important concept **AvEvent** (audio/video event), e.g., is used by a great number of applications from different partners. Shared use implies the necessity of structuring the distributed definitions in a scope hierarchy.

The conversion of the Java class definitions into description logics concept and role definitions (CICLOP format) was done automatically, using an enhanced Java parser. However, since derivation of methods is unknown in Java, all application functions (or actions) defined as Java methods needed manual placement in the ontological hierarchy (i.e., derivation from higher concepts) to ensure precise semantic integration. Further manual work was required in several other cases.

3.4 Sharing the Ontology in the EMBASSI Project

XML being the standard for inter-component communication in EMBASSI, the ontology is exchanged in form of an XML DTD specifying concepts and their roles as nested elements.

References

[1] Abney, Steven. *Parsing By Chunks.* In: Berwick, Robert; Abney, Steven; Tenny, Carol (eds.) *Principle-based Parsing.* Kluwer, 1991

[2] Kamp, H. and Reyle, U. (1993), *From Discourse to Logic*, Dordrecht: Kluwer.

[3] Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A. (1996). Reasoning in Description Logics. In: Brewka, G. (ed.), *Foundations of Knowledge Representation*, pp. 191-236. Stanford: CSLI Publications.

[4] LIIA-ENSAIS, University of Strasbourg (1999), CICLOP version 1.b3 User Manual. `http://massenet.u-strasbg.fr/LIIA/ciclop/ciclop.htm`