Discourse and Application Modeling for Dialogue Systems

Kerstin Bücher, Yves Forkl, Günther Görz, Martin Klarner, and Bernd Ludwig
Computer Science Institute and FORWISS, Erlangen, Germany
eMail: goerz@informatik.uni-erlangen.de

Abstract

Spoken – and even written – language dialogue systems become of increasing importance for various information gathering and device controlling tasks. With an example taken form the EMBASSI joint project, after a brief discussion of the linguistic processing part, we describe how description logics are used in modeling the application domain as well as the linguistic domain. Linking lexical semantics with application specific concepts is a nontrivial problem, in particular in cases where there is no direct correspondence. So, the paper ends presenting our approach to solve this linking problem.

1 Generic Dialogue Management in EMBASSI

1.1 Application Background: The EMBASSI project

To a large extent, our research and development work in the field of dialogue systems is done within the German joint project EMBASSI[1] which aims to provide easy access for everybody to complex technical systems (A/V home theatre, car devices and public terminals), encouraging multi-modal user input, e.g. speech, gestures, and pointing. Besides a speech parser, our contribution consists of, first, a dialogue manager, second, a formal ontology and third, a generation component to communicate system utterances to the user.

In our approach to dialogue management, which features 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

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

hierarchy of semantic concepts is extremely important to enable logical inferences on these concepts. We formalize them using Description Logics (DL)[2] since they suit our needs quite well.

The novel architecture of the EMBASSI project brings up two problems to be mastered in the *domain model* (based on the formal ontology): First, the processing of user input is separated from the execution of system operations by introducing an interface between the dialogue manager and the applications which are controlled by so-called "assistant modules". These operations have to be derived from the user's utterances by constructing and representing their semantics and linking them to application concepts. To accomplish this, messages have to be exchanged between the dialogue manager and the assistants. We chose A-Box statements in DL (embedded in KQML statements) as a formalism for these messages. Second, there is a great variety of application components by different manufacturers. The EMBASSI ontology functions as a standardized system-wide terminology encompassing the whole world of functions and objects referred to by all applications and assistants controlling several audio/video devices in the home environment. Each of their functions and objects must be both uniquely named and given a precise semantic definition as a *concept*.

1.2 System Architecture and Knowledge Representation

EMBASSI is implemented as a distributed system with several autonomous modules communicating among each other in order to exchange information about the current state of the interfaced applications. Modules make their internal state partially available to the dialogue system as far as it is necessary in order to continue a dialogue. The current state of the application is represented by an A-Box containing assertions about instances which refer to objects of the application.

For example, `AvEvents`[3] may be retrieved from a data base and assertions about them are eventually added to the A-Box if they become relevant for the dialogue.

In addition, the A-Box also contains instances for all actions which have been executed up to now by some module of the (multi agent) system. All actions which are primitive with respect to the application domain model are executed procedurally.

If a module executes an action, a transition from one state to another is achieved as adding the effects of the action changes the content of the A-Box and the facts which can be inferred.

For this purpose, application modules exchange information with the dialogue system via KQML messages containing assertions about application ob-

[2]For an introduction to DL see e.g. [Don96]

[3]instances of the concept which describes all kinds of audio and video transmissions.

jects and actions. In particular, messages about actions to be executed and the computed effects are communicated between the modules. In order to update its knowledge about the application state, the dialogue system adds the content of incoming messages to the appropriate A-Box.

In complete analogy to the description of the current application state, the current state of the dialogue is represented as an A-Box whose assertions are about instances from the discourse and linguistic domain. In this case, objects are so called discourse referents representing instances of linguistic concepts as introduced in utterances. Again, instances can assert facts about actions as well as objects. Primitive actions in this domain are performatives like *I want, I ask, I state, I command.* They are executed by the dialogue system and modify the state a dialogue is currently in. From this point of view, the dialogue system behaves like an application module using a different A-Box to represent the current state of affairs.

The task of the system as a whole is to compute whether utterances from the user are satisfiable in the given state of the application. When passed to the dialogue system by the parser, an utterance only affects the A-Box for the discourse domain as its semantic representation is added there. So, what is the relationship between the two A-Boxes? The idea is that assertions in the discourse A-Box have to be satisfiable with respect to the application A-Box (i.e. if the state of the application allows the satisfiability directly or after a sequence of appropriate actions in the application domain). For example, the utterance *Record the thriller this evening!"* requires that the application A-Box contains an instance of `AvEvent` satisfying the imposed constraints and that the preconditions for `Record` are satisfiable.

In the case of a unique satisfaction of the utterance, an application assistant 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. The content of the system responses is represented in terms of A-Box statements.

2 Organisation of the Domain Model

The formal ontology structuring the domain model is organised 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. This domain, although of crucial importance for dialogue management, will not be described here because of lacking space. 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.

2.1 Modeling the Application Domain

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. Concepts for actions are distinguished from those for objects – in analogy to the distinction between objects and their methods in object-oriented programming languages.

The application situation is represented in terms of A-Box assertions; so, the scenario is an extension of the domain model.

In order for the dialogue manager 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.

As an example, we consider a HAVi[4]-programmable VCR. Commands available in the HAVi class VCR may be uttered in natural language to use the VCR. HAVi ist defined in terms of a Java class hierarchy for object and function types which are translated automatically into a hierarchy of DL concepts. Mandatory derivation of all classes had to start from a set of predefined ontology base classes representing the *Generic Base Model* which comprises the top-level concepts for a broad range of application domains.

Currently we translate OO-subclass relations into DL-subconcept relations. Field members of classes are translated into forall constraints in DL, as shown below. The signature of OO-methods is represented by embedding each method as a subconcept of a concept of a generic base model to be chosen manually. Parameters of of methods are represented in the DL translation as forall constraints. As we use CICLOP as inference tool, we translate enumeration types and constants into primitive concepts. As a consequence, we do not need high

[4]HAVi stands for *Home Audio Video Interoperability* and represents a common, openly-licensable specification for networking digital home entertainment products.

expressivity for the DL language. We are currently investigating the limits of this approach. For feature-value formalisms it seems to be sufficient.

As an example for the derivation of a fragment of the DL domain model consider the following class which is part of the of the HAVi VCR class:

```
class VCRObj{
   int id;
   play();
   record();
   variableForward(forwardSpeedLevel speed);
}
```

with `enum forwardSpeedLevel = {fastForward5, ...}` .

From the viewpoint of DL, the definition of `VCRObj` introduces a subconcept of `object` (class) and three subconcepts of `action` (methods):

$$
\begin{array}{lcll}
\texttt{VCRObj} & := & \texttt{object} & \sqcap\ \forall\texttt{has-id.int} \\
\texttt{VCRObj-play} & := & \texttt{action} & \\
\texttt{VCRObj-record} & := & \texttt{action} & \\
\texttt{VCRObj-variableForward} & := & \texttt{action} & \sqcap\ \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}
$$

For `VCRObj-variableForward`, this means that it is an `action`, and its only parameter has-speed has to be a `forwardSpeedLevel`. This is essentially the semantics of the method definition above, but `VCRObj-variableForward` is permitted only for objects of class `VCRObj`. We can account for this by refining our definition of in the following way:

$$
\begin{array}{lcll}
\texttt{VCRObj-variableForward} & := & \texttt{action} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
 & & & \sqcap\ \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}
$$

Using these concept definitions, we are now able to infer the specific operation from user utterances like “schnell vorspulen!”, the meaning of which is represented in the A-Box by a domain-instantiated DRS. So, there is a `VCRObj` l_1 in our scenario asserting $\texttt{VCRObj}(l_1)$.

The invocation of the method l_1.`variableForward(fastForward5)` is asserted by

$$\texttt{action}(\alpha) \wedge \texttt{has-subject}(\alpha, l_1) \wedge \texttt{has-speed}(\alpha, \texttt{variableForward5})$$

from which $\texttt{VCRObj-variableForward}(\alpha)$ is derivable.

At this point, we have to refine the domain model again. Up to now, it differentiates between several concepts for objects (e.g. `VCRObj`, `forwardSpeedLevel`), but `action` is the only primitive concept for actions. Therefore, all verbs in the lexicon would have the same meaning, namely `action`. Consequently, whatever verbs were used, always e.g. `VCRObj-play` would be inferred. To cope with this difficulty, `action` has to be refined by introducing primitive subconcepts of it that are distinct from each other in order to separate different verb meanings. E.g. we introduce $\texttt{play} \subseteq \texttt{action}$, $\texttt{record} \subseteq \texttt{action}$ and $\texttt{wind} \subseteq \texttt{action}$. These (distinct) subconcepts of `action` would allow us create the following entries:

abspielen	`play`		
aufnehmen	`record`	*aufzeichnen*	`record`
spulen	`wind`		

These lexical entries suffice to distinguish which methods are meant by a certain verb, if we refine the definitions for the methods of the class `VCRObj`:

$$
\begin{array}{lcll}
\texttt{VCRObj-play} & := & \texttt{play} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
\texttt{VCRObj-record} & := & \texttt{record} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
\texttt{VCRObj-variableForward} & := & \texttt{wind} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
 & & & \sqcap\ \forall\texttt{has-direction.direction} \\
 & & & \sqcap\ \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}
$$

Generalizing the example above, we have outlined how formal DL domain model parts may be derived from underlying class definitions for objects and methods used in the implementation of the application assistant.

The division of the domain model in a generic upper part of primitive concepts not specific for the application and a lower part of application relevant concepts allows to reuse the generic part ([Ram97]). So, the knowledge representation task for the design of a specific dialogue system is considerably simplified.

Under quantitative aspects, our DL model for the A/V home theatre domain currently contains 88 primitive and 24 complex concepts, whereas the car entertainment domain has 12 primitive and 47 complex concepts. Both domains taken together will soon dispose of about 500 complex concepts, due to the integration of increased application functionality and much more detailed linguistic modeling.

2.2 Modeling the Linguistic Domain

Our ultimate aim in interpreting an user utterance is to construct one or more hypotheses of its meaning in terms of the concept definitions from the domain model. In order to achieve this, we have to devise a method for mapping natural language utterances onto terms describing method invocations.

We use a lexicon for the vocabulary V specific to the domain in which each word w refers to one or more *lexical concepts* which serve several purposes. First of all, synonymous words will receive the same lexical concept. Second, it is possible to account for polysemous words by having multiple entries, each with a different associated lexical concept. Third, the linguistic modeling of relations expressed in utterances can dispense of application-specific characteristics. Last but not least: In the domain model, these lexical concepts form a hierarchy of their own which is defined independently of the hierarchy structuring the domain model in the narrower sense, i.e. that of the application domain. This helps avoid those very common problems arising from the mingling of cognitive-lexical and factual relations.

In order to model the lexical semantics in a way that is maximally independent from any particular application, we employ a hierarchy of semantic types which is implemented as a DL terminology. Types are represented as disjoint atomic concepts and assigned to elements in the lexicon. For example, we have the following assignment of surface word forms (lower case) to atomic lexical concepts (upper case):

```
sehen SEHEN, anschauen ANSCHAUEN, gucken GUCKEN,
aufnehmen AUFNEHMEN, aufzeichnen AUFZEICHNEN
Krimi SPARTE
```

To capture the inter-chunk semantics of an utterance, i.e. the meaning of the combination of chunks, we introduce complex concepts obtained from interpreting case frames which describe the syntactic and semantic relations between chunks as concept definitions. The idea is as follows: Given an excerpt of the case frame for "aufnehmen" (with the thematic role `patiens` and the `acc`usative case marker)

```
aufnehmen patiens SENDER | SENDUNG | SPARTE | TITEL
          acc
```

we allow four different concepts that my fill the thematic role *patiens* of the lexical concept of `AUFNEHMEN` representing the semantic type of `aufnehmen`. This informal concept definition is captured precisely in terms of the following $\mathcal{ALC}$ expression:

$$C \doteq \mathtt{AUFNEHMEN} \wedge \forall \mathtt{patiens}.(\mathtt{Sender} \vee \mathtt{Sendung} \vee \mathtt{Sparte} \vee \mathtt{Titel})$$

Such a concept definition can be used to determine whether two chunks can be combined consistently with the available lexical semantics:

`[den Krimi] [aufnehmen]`

Analyzing the syntactic features (accusative) of the two chunks given, one finds that `[den Krimi]` may potentially serve as *patiens* for `[aufnehmen]`. This fact is validated by computing that the concept resulting from the combination of the two chunks is consistent with the above definition:

$$\mathtt{AUFNEHMEN} \wedge \forall \mathtt{patiens}.\mathtt{Sparte} \subseteq C$$

2.3 Linking Lexical Semantics with Domain Pragmatics

For linking lexical semantics with an application specific domain model we have to master the following tasks: *First*, relating application concepts with lexical concepts. For linking notions of the application with semantic types, we first have to cope with the usage of synonyms in natural language for a uniquely determined application concept. And *second*, relating roles with thematic roles, which means reasoning about individuals, i.e. in the A-Box.

To continue the example of `[den krimi] [aufnehmen]`, we have a look at the DRS constructed by the parser that makes use of the case frame shown above:

$$\left[\begin{array}{l} a\ k\ \text{krimi} \\ \hline \texttt{AUFNEHMEN}(a) \\ \texttt{patiens}(a,k) \\ \texttt{SPARTE}(k) \\ \texttt{wert}(k,\text{krimi}) \\ \texttt{SPARTENWERT}(\text{krimi}) \end{array}\right]$$

As discussed above, the task to be carried out now is to link lexical semantics and the application domain model in order to construct a representation for the application specific meaning of this DRS (A-Box).

We have already mentioned that via $\forall$`lexconcept.AUFNEHMEN` and $\forall$`lexconcept.SPARTE` we find the application specific meaning of both lexical concepts by inferring their descendants in a given T-Box. The issue still to be addressed is how the roles are mapped onto the corresponding ones in the application domain. If `SENDUNG` was the only `patiens` for `AUFNEHMEN`, one possibility would be to apply a rule like following one in the A-Box as (variables x, y, u, v are all-quantified):

$$\texttt{patiens}(x,y) \wedge \texttt{lexconcept}(u,x) \wedge \texttt{SENDUNG}(y) \wedge \texttt{lexconcept}(v,y) \rightarrow \texttt{has-avevent}(x,y)$$

But the example shows that this is not sufficient, as

$$\texttt{patiens}(x,y) \wedge \texttt{lexconcept}(u,x) \wedge \texttt{SPARTE}(y) \wedge \texttt{lexconcept}(v,y) \rightarrow \texttt{has-avevent}(x,y)$$

would wrongly imply y to be an `AvEvent` instead of a `Genre`. Therefore, before letting such a rule fire one has to determine two things:

- What are concepts in the range of `has-avevent`, if we assume `Record` to be in its domain? As we find `AvEvent`, but not `Genre`, the case frame is admissible only, if `Genre` is a part of the definition for `AvEvent`. To state this more precisely:

- Is it possible – by using the transitivity of roles – to construct a role that has `AvEvent` in its domain and `Genre` in its range? Given our definition of `AvEvent` we find that $R := \texttt{has-bibdata} \circ \texttt{has-genre}$ is the desired solution.

Now we are allowed to introduce new individuals of the concepts "on the path from `AvEvent` to `Genre`". In the example, there is only one such concept, namely `BibData`. As a consequence, we get the following DRS that corresponds a DRS in the application domain model:

$$\left[\begin{array}{l} r\ e\ b\ \text{krimi} \\ \hline \texttt{Record}(r) \\ \texttt{has-avevent}(r, e) \\ \texttt{AvEvent}(e) \\ \texttt{has-bibdata}(e, b) \\ \texttt{BibData}(b) \\ \texttt{has-genre}(b, g) \\ \texttt{Genre}(g) \\ \texttt{value}(g, \text{krimi}) \\ \texttt{GenreValue}(krimi) \end{array}\right]$$

2.4 Conclusion

This papers presents an DL based approach on domain modelling for natural language dialogue systems. It devises a automatic approach to the translation of feature-value formalisms into DL. Most domains for automatic information services can be modelled in a very simple DL language. Therefore, the approach serves for automatic and efficient configuration of dialogue systems. We showed that another important point is the separation between application domain and dialogue domain which is obtained at the cost of manually linking lexical semantics with an application domain model. In this way it is much easier to reuse linguistic resources like lexicons and grammars for the purpose of configuration. Additionally, we can increase the robustness and naturalness of dialogue systems in comparison with other approaches basing on keyword spotting and finite state automata.

References

[Abn91] S. Abney, *Parsing By Chunks.* In: R. Berwick, S. Abney, C. Tenny (Eds.), *Principle-based Parsing*. Kluwer, 1991

[KaR93] H. Kamp, U. Reyle, *From Discourse to Logic*. Dordrecht: Kluwer, 1993

[CIC99] LIIA-ENSAIS, *CICLOP version 1.b3 User Manual*. Strasbourg, 1999. `http://massenet.u-strasbg.fr/LIIA/ciclop/ciclop.htm`

[Don96] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, *Reasoning in Description Logics*. In: G. Brewka (editor), *Foundations of Knowledge Representation*. CSLI-Publications, 1996, 191–236

[Abn95] S. Abney, *Chunks and Dependencies: Bringing Processing Evidence to Bear on Syntax*. In: *Computational Linguistics and the Foundations of Linguistic Theory*. CSLI-Publications, 1995

[Kus96] S. Kuschert, *Higher Order Dynamics: Relating Operational and Denotational Semantics for λ-DRT*. CLAUS-Report 84, Saarbrücken, 1996

[Lit99] D. Litman, M. Walker, M. Kearns, *Acquiring Knowledge of System Performance for Spoken Dialogue*. In: *Proceedings of the IJCAI 99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*. Stockholm, Sweden, 1999, 73–80

[Ram97] A. Ramsay, *Does It Make Any Sense? Update Semantics as Epistemic Reasoning*. In: *Proceedings of ACL 97*. Morgan Kaufman Publishers, 1997

[Sut98] S. Sutton et al., *Universal speech tools: The CSLU toolkit*. In: *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP'98)*. Volume 7, Sydney, Australia, 1998, 3221–24

[Tra94] D. Traum, *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. Thesis, Computer Science Dept., University of Rochester, 1994