CHAPTER 6

Concept-level error handling in Higgins

In Part II, it was shown that, in a conversational dialogue setting, it is indeed possible to detect errors and extract meaning from recognition hypotheses containing a lot of errors – at least for humans. In this part, we will investigate how this and other issues may be handled in a complete spoken dialogue system.

As stated previously, speech recognition output in conversational dialogue systems is often only partially correct. Therefore, error handling in such systems should be done on the concept level, not on the utterance-level. In this chapter, we will present a model for how the *grounding status* of individual concepts may be tracked. Instead of modelling how this grounding status gets updated by a special set of "grounding acts", we will show how all utterances, even those that are mainly task-related, may contribute to the grounding process by updating the grounding status. The grounding status includes the history of when and how the concept is grounded by the participants, and the system's confidence in this. Since the grounding status is modelled on the concept level, the choice of surface realisation will affect the system's model of what has been grounded.

As part of the work for this thesis, the HIGGINS[7] spoken dialogue system has been developed and evaluated (Edlund et al., 2004; Skantze et al., 2006). The system has served as a testbed for developing and evaluating methods and models for concept-level error handling, such as robust interpretation, modelling grounding status in the discourse, displaying understanding, posing clarification requests, and late error detection. This chapter will describe the domain, semantics and components of this system, and how concept-level error handling is done in all parts of the system. In the next chapter, an evaluation of the system will be presented.

[7] The system, as well as its components, bear Pygmalion-related names. This includes the Greek myth *Pygmalion & Galatea*, the Bernard Shaw play *Pygmalion*, and the musical *My Fair Lady*.

6.1 The Higgins navigation domain

The initial domain chosen for HIGGINS is the same as the one used in Chapter 4: pedestrian city navigation and guiding. As noted previously, this domain is similar to the now classic Map Task domain (Anderson et al., 1991), as well as to a number of guide systems, such as REAL (Baus et al., 2002). In HIGGINS, the user tells the system where she wants to go and the system guides the user by giving verbal instructions. The system does not have access to the user's position by any positioning device. Instead, it has to figure out the position by engaging in a dialogue in which the user describes the surroundings. Since the user is moving, the system continually has to update its model of the user's position and provide new, possibly amended instructions until the destination is reached. Unlike the experimental setting in Chapter 4, a 3D model of a virtual city is used for simulation, see Figure 6.1.

Figure 6.1: The 3D simulation that is used for user tests.

It should be noted that this application would perhaps not be directly useful (as is) for pedestrian navigation in a real city, partly because it would be infeasible to model the city completely and partly because a real application would utilise a GPS. However, the domain is challenging from an error handling perspective; it is a conversational dialogue setting where complex semantic relations and referring expressions need to be expressed.

To illustrate how error handling is done in the HIGGINS system, a made-up example scenario will be used, which is shown in Table 6.1. In this scenario, the ASR results are shown as well, revealing a fair amount of speech recognition errors. The HIGGINS system is currently built for Swedish, but most examples in this thesis have been translated into English.

Table 6.1: A made-up example scenario. The rightmost column shows ASR results in which word confidence scores are represented with different shades of grey, where dark means high confidence.

Turn	Transcription	ASR result
S.1	*Where do you want to go?*	
U.2	To an ATM.	**TO AN ATM**
S.3	*Ok, an ATM, can you describe where you are now?*	
U.4	I have eeh a large concrete building on my left and a bus stop in front of me.	**I HAVE TREE A LARGE CONCRETE BUILDING ON MY LEFT AND BUS STOP ON MY RIGHT**
S.5	*What colour is the concrete building?*	
U.6	Red.	**RED**
S.7	*Ok, can you see a wooden building?*	
U.8	I can see a brown building.	**I CAN SEE A BLUE BUILDING**
S.9	*Blue?*	
U.10	No, brown.	**NO BROWN**
S.11	*Do you really have a bus stop on your right?*	
U.12	No, in front of me.	**NO IN FRONT OF SEE**
S.13	*Ok, I think I know where you are. Walk a little bit forward and take left after the red building.*	

The system starts by requesting the user's goal (S.1) and the user replies (U.2). Based on the confidence score of the stated goal, the system decides to generate a *separate display utterance* ("an ATM") as part of the next turn, and continues with a general request about the user's position (S.3), which the user answers to by describing surrounding landmarks (U.4). This utterance is poorly recognised by the ASR, which results in a misunderstanding: the system now believes that the user has a bus stop on his right (which is really in front of him). However, the system stores all confidence scores and information about what has been grounded, so that such errors may be identified later on. To constrain the user's position, the system asks a question about the concrete building (S.5). When doing this, it uses a definite description to refer to the building, which is a way of simultaneously *displaying its understanding*, which the user does not object to in the next utterance (U.6). This way, the system's uncertainty about the concrete building has been reduced. To further constrain the user's position, the system now asks a more specific question (S.7). The user does not directly answer the question, but provides a description that nevertheless helps to constrain the position (U.8). The colour of the building that the user describes is erroneously recognised, but since it gets a low confidence score, the system makes a *fragmentary clarification request* (S.9), and the user corrects the system (U.10). Due to the misunderstanding in U.4, the system now finds out that there is no

place that the user can be. To solve this error, the system employs *late error detection* and searches the discourse history and finds out that there is one concept with a relatively low confidence score that has not been grounded: the belief that the user has a bus stop on his right. The system makes a *misunderstanding repair* – it checks the belief with the user (S.11) and the user corrects the system (U.12). The system has now constrained the user's position and may start to give route directions (S.13).

Before describing the details of these error handling mechanisms, we will introduce the semantic representations and architecture used in the HIGGINS spoken dialogue system.

6.2 Semantic representations

The surroundings the user and system talk about contain complex landmarks and relations that are challenging to interpret and represent semantically. For such semantic representations, deep semantic structures are needed – not just simple feature-value lists. Semantic descriptions are consistently represented as rooted unordered trees of semantic concepts. Nodes in the tree may represent objects, relations and properties. Such structures are very flexible and can be used to represent deep semantic structures, such as nested feature structures, as well as simple forms, depending on the requirements of the domain. By using tree matching, similar to Kilpeläinen (1992), a pattern tree can be used to search for instances in a given target tree. Thus, larger semantic structures can form databases which may be searched. It is also possible to include variables in a pattern tree for specifying constraints and extracting matching nodes, as well as using special pattern nodes for negation, etc.

The semantic tree structures in HIGGINS, including the database, are represented with XML, using a schema that is specific for the domain. Figure 6.2 shows an example: an abstract representation of a wooden building. Figure 6.3 shows how the same structure can be visualised graphically as a tree structure using XSLT and XHTML. The database in the HIGGINS navigation domain is a large XML structure (about 60 000 elements) containing all landmarks and their properties, as well as possible user positions and how they relate to the landmarks. All objects in the database have id's. The XML in Figure 6.2 could be used as a pattern to search the database. Values starting with a dollar sign – `id4` in the example – are interpreted as variables. The result of this search would be a list of all possible bindings of variable `id4`, that is, a list of the id's of all the wooden buildings in the database.

The semantic representations may be enhanced with "meta-information", for example about confidence scores, communicative acts, and if information is new or given. Figure 6.4 shows the representation of the utterance "the building is made of wood". The structure tells us that this is a communicative act (CA) of the type ASSERT, that the object is singular (SING), and that the object and type are GIVEN information but the material NEW. This meta-information is needed for representing the structure of utterances, but is not contained in the database. By removing meta-information, the structure can be transformed to a database search pattern, like the one in Figure 6.3, in order to find possible referents to the object. The meta-information is easily removed if placed in a special namespace.

```
<object id="$id4">
  <properties>
    <type>
      <value>building</value>
    </type>
    <material>
      <value>wood</value>
    </material>
  </properties>
</object>
```

Figure 6.2: An abstract semantic representation of a wooden building in XML.

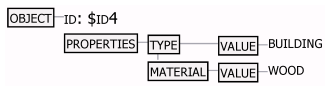


Figure 6.3: The same structure as in Figure 6.2, visualised graphically.

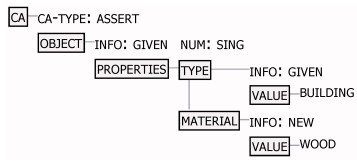


Figure 6.4: The semantic representation of the utterance “the building is made of wood”.

Table 6.2: An example of unification using a template.

Template	**Unification**	
T =	*S1* =	*S2* =
`<template>` `  <object>` `  <properties count="1">` `  <type count="1">` `  <value count="1"/>` `  </type>` `  <colour count="*">` `  <value count="1"/>` `  </colour>` `  <size count="*">` `  <value count="1"/>` `  </size>` `  </properties>` `  </object>` `</template>`	`<object>` `  <properties>` `  <type>` `  <value>` `  building` `  </value>` `  </type>` `  </properties>` `</object>` *Unify(S1, S2, T)* = `<object>` `  <properties>` `  <type>` `  <value>building</value>` `  </type>` `  <colour>` `  <value>red</value>` `  </colour>` `  </properties>` `</object>`	`<colour>` `  <value>` `  red` `  </value>` `</colour>`

Tree structures may also be unified, as shown in Table 6.2. A semantic template is used to specify how the nodes may be structured, to guide the unification. As can be seen, the nodes in the template may be marked with how many times it may occur at that location, by using the attribute `count`. The template also makes it possible to unify structures starting at different levels in the tree, as is the case for *S1* and *S2*.

The use of a template for unification makes it possible to easily represent the semantics of fragments (such as verbs, relations, properties, etc.) and combine them into full propositions. Such fragments may be ambiguous, that is, they may fit into different parts of the template. When the fragments are unified and they start at different levels, the unification algorithm tries to combine them with the shortest distance possible. In other words, they get disambiguated. For example, if the semantic concept PRICE (as part of the question "what does it cost") gets unified with the semantic structure VALUE:100 (a representation of the answer "100"), this may result in the structure PRICE:VALUE:100, provided that the template allows such a structure. In itself, VALUE:100 is ambiguous, and may fit into different structures.

6.3 Architecture

The HIGGINS spoken dialogue system is a distributed architecture with modules communicating over sockets. Each module has well defined interfaces, and can be implemented in any language, running on any platform. The interfaces are described using XML schema. Figure 6.5 shows the most important modules and messages in HIGGINS, when run in the navigation domain.

From the ASR, the top hypothesis with word confidence scores (2) is sent to a natural language understanding module, called PICKERING. PICKERING makes a robust interpretation of this hypothesis and creates context-independent semantic representations of communicative acts (CA's). In HIGGINS, dialogue management is not implemented as a single module. Instead, this processing is divided into a discourse modeller (called GALATEA) and a set of action managers. GALATEA may be regarded as a further interpretation step, which takes the context into account. Based on incoming CA's, GALATEA builds a discourse model. This discourse model is then consulted by a set of action managers, which initiate systems actions. The purpose of this separation between discourse modelling and action selection is to make the discourse modelling more generic, while the action selection may be highly domain specific. This separation is similar to the approaches taken in Allen et al. (2001b) and Pfleger et al. (2003).

CA's from the user (3) are sent from PICKERING to GALATEA, which adds them to a discourse model. The discourse model (4) is then sent to a *grounding action manager* (GAM) which initiates grounding actions (such as making clarification requests). If the turn is not yielded to the user, the discourse model (5) is passed on to the *navigation action manager* (NAM), which initiates navigation actions (such as requesting the user's position or giving route directions). To do this, the NAM has access to the *domain database*. The NAM may also make modifications to the discourse model, for example if an error is detected, and send it back (6) to GALATEA.

System initiated communicative acts from the action managers (7,8) are sent to a natural language generator (OVIDIUS) which generates a surface string (with some prosodic markup). This string (9) is sent to a TTS which synthesises the spoken output (10). But the communicative acts from the system are also sent back to GALATEA (11), which treats them in the same way as the communicative acts sent from PICKERING (3). Thus, GALATEA models communicative acts both from the user and the system; ellipsis, anaphora and grounding status is handled and modelled in the same way for all communicative acts.

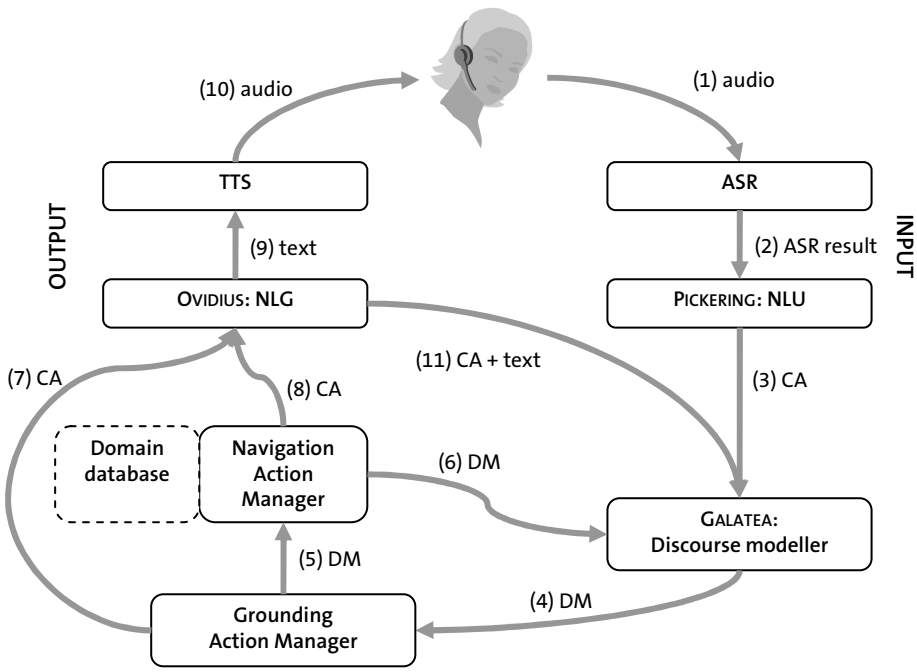


Figure 6.5: The most important modules and messages in the HIGGINS navigation domain. CA stands for communicative act. DM stands for discourse model.

All modules in HIGGINS are fairly generic – the resources that are needed for the specific application are all encoded in XML. The NAM, on the other hand, is written specifically for the domain. However, much of the work that a typical dialogue manager has to do is already handled by the GAM and GALATEA.

All modules operate asynchronously, which means that, for example, the ASR may be recognising a user utterance or the TTS rendering a system utterance, while an action manager is generating a new action.

6.4 PICKERING: Natural language understanding

PICKERING is a robust interpreter, designed to work with continuous incremental input from a speech recogniser with n-gram language models in a conversational dialogue system. The grammar used to parse recognition results is based on context-free grammars (CFG), with some modifications. To add robustness, PICKERING may automatically allow deviations from these grammars, such as allowing partial results, insertions and non-agreement.

Although the combination of features included in PICKERING is (to our best knowledge) unique, much work in the literature has been focussed on achieving robustness in parsing and semantic interpretation beyond keyword spotting. Examples include Mellish (1989), which deals with insertions in chart parsing, and Kasper et al. (1999), in which partial results are combined.

6.4.1 Grammar

The PICKERING grammar rules are enhanced with semantic rules for generating the kind of semantic trees described in 6.2 above. The CFG consists of a rule-set, a collection of lexical entries, and an optional morphology, all of which may carry semantics. Figure 6.6 shows a simple grammar which covers the Swedish phrase "den röda byggnaden" ("the red building").

In this example, there are three lexical entries and one rule. Both entries and rules have an associated list of features (in the f-namespace), a `<match>` part that specifies what they match, and a `<sem>` part which specifies the resulting semantics (in the s-namespace). Entries may match words, and rules may match words, entries or other rules. In Swedish, words in noun phrases must be congruent: they have to agree on gender, number and definiteness. Features that should agree are specified in the `<agreement>` element. The attribute `propagate="true"` also tells that the agreeing features should be propagated to the matching rule.

Grammar rules also contain instructions for combining semantics from matching entries and rules. A common instruction, as seen in the grammar example, is `<unify>`, which is used to unify semantics. `<ref>` is used to refer to the semantics of the matching parts. `<add>` states that the resulting features should be copied into the semantics, according to the template.

The parse result of the phrase "den röda byggnaden", using the grammar in Figure 6.6, is shown in Figure 6.7.

```
<grammar>
  <lexicon>
    <entry f:name="det" f:info="given" f:gen="utr" f:num="sing">
      <match>den</match>
      <sem>
        <s:object/>
      </sem>
    </entry>
    <entry f:name="attr" f:info="given">
      <match>röda</match>
      <sem>
        <s:colour><s:value>red</s:value></s:colour>
      </sem>
    </entry>
    <entry f:name="nom" f:info="given" f:gen="utr" f:num="sing">
      <match>byggnaden</match>
      <sem>
        <s:object>
          <s:type><s:value>building</s:value></s:type>
        </s:object>
      </sem>
    </entry>
  </lexicon>
  <rules>
    <rule f:name="object" top="true">
      <agreement features="f:info f:gen f:num" propagate="true"/>
      <match>
        <entry f:name="det"/>
        <entry f:name="attr" link="attrlink"/>
        <entry f:name="nom" link="nomlink"/>
      </match>
      <sem>
        <b:unify>
          <b:ref link="nomlink">
            <b:add f:info="$info" f:num="$num" to="template"/>
          </b:ref>
          <b:ref link="attrlink">
            <b:add f:info="$info" to="template"/>
          </b:ref>
        </b:unify>
      </sem>
    </rule>
  </rules>
</grammar>
```

Figure 6.6: A PICKERING grammar fragment.

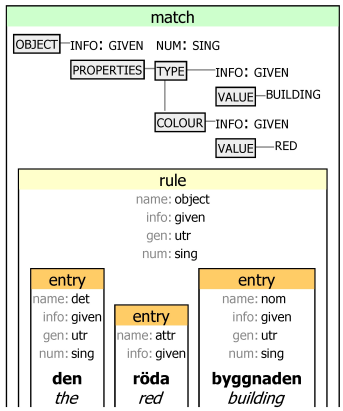


Figure 6.7: Parsing of the phrase “den röda byggnaden” (the red building) using the grammar in Figure 6.6.

6.4.2 Robust interpretation

To add robustness, the interpreter applies a number of additional techniques to the standard CFG parsing algorithm. To illustrate these techniques, an example interpretation of U.4 in Table 6.1 is shown in Figure 6.8. In the figure, the corresponding Swedish phrase is also shown in order to highlight non-agreement in the noun phrase “a large concrete building”, as explained below.

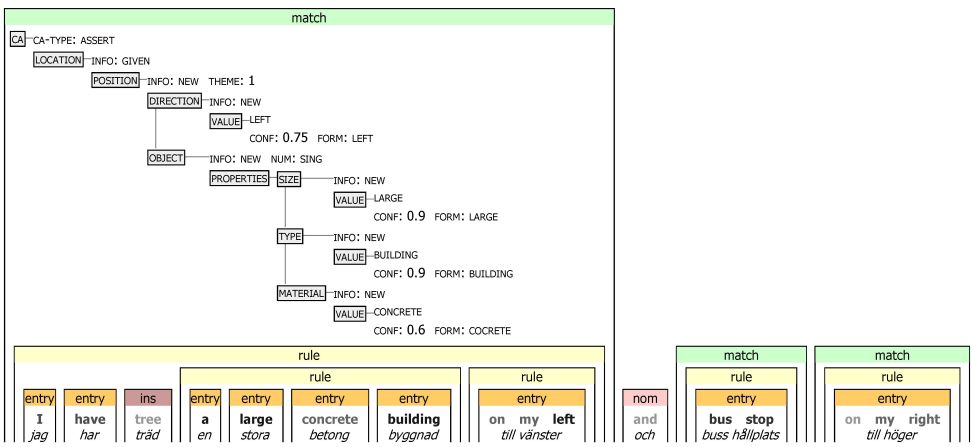


Figure 6.8: An example interpretation of the erroneously recognised utterance U.4 in Table 6.1. A corresponding Swedish translation is shown below containing an additional morphological error. The semantic results of the last two phrases are not shown.

6.4.2.1 Insertions

Disfluencies not modelled by the n-gram language models may easily give rise to unexpected words in the middle of phrases. An example of this is the third word "tree" in Figure 6.8. PICKERING allows insertions of unexpected words anywhere inside a phrase. A parameter can be set that constrains the number of subsequent insertions that are allowed. A simple keyword-spotter would probably have included this content-word in the semantic result, but thanks to the grammatical analysis, PICKERING can treat this as an error and ignore it.

6.4.2.2 Non-agreement

In a complex domain such as pedestrian navigation, morphological distinctions are meaningful in order to distinguish both number and definiteness, which may signal whether objects and properties are given or new. Such morphological distinctions may be more important in some languages than others (which is the case for Swedish compared to English). However, speech recognisers with n-gram language models may often fail to produce the right morphological inflections. Moreover, speakers may also make morphological mistakes in conversational language. This may give rise to non-agreement among the constituents of a phrase. PICKERING deals with this by allowing non-agreement when features are combined according to the `<agreement>` element. If the features do not agree, the majority class is selected (a random choice is used if there is a tie). An example of this is the phrase "a large concrete building" in Figure 6.8. The corresponding Swedish translation shown below contains a morphological error: it is erroneously recognised as "en stora betong byggnad". In Swedish, the correct morphological inflection would be "den stora betong byggnaden" (INFO:GIVEN) or "en stor betong byggnad" (INFO:NEW). Since the latter interpretation is more consistent with the input, it is selected by PICKERING in the robust interpretation. While non-agreement is allowed, it is considered when different solutions are ranked.

6.4.2.3 Fragment spotting

PICKERING does not have to find a rule that covers the complete input string. Instead, it tries to choose the smallest number of matching phrases which covers the largest number of words. Between these phrases, non-matching words are allowed. In Figure 6.8, the best fit is three phrases with a non-matching word ("and") in-between. Incomplete phrases may then be combined by the discourse modeller GALATEA, which will be described in 6.5.2.

6.4.2.4 Concept confidence scores

The semantic template used for unification can be marked with slots for confidence scores. The confidence scores for the words that are involved in creating a node with such a slot are then averaged to compute a confidence score for the node (similar to Gabsdil & Bos, 2003). These confidence scores are then transferred to the semantic result according to the template. Figure 6.8 shows how the concepts are marked with such scores (by the attribute `conf`). Insertions, such as "tree" in the example, should ideally lower the confidence score for the concepts involved in the phrase, but they are not considered in the current implementation. These

concept-level confidence scores may then be used for concept-level error handling, which will be described later on.

6.4.2.5 Surface form

Within a certain domain, a given semantic concept may have several surface forms. For example the forms "building" and "house" both correspond to the concept TYPE:BUILDING in the HIGGINS navigation domain. To keep track of the form used, the semantic template may be marked with slots for surface form. These slots are filled with the forms of the lexical entries that were involved in the production of the semantic concepts. Examples of this are shown in the semantic result in Figure 6.8, by the attribute `form`. These forms may later be used to for example pose fragmentary clarification requests in a correct way, as described later on.

6.4.3 Implementation

PICKERING is a modified chart parser (Jurafsky & Martin, 2000) implemented in Oz[8]. There are some general challenges with robustness in an interpreter. First, there is a risk that the interpreter will find too many interpretations covering different parts of the input without being semantically distinct. PICKERING utilises the semantic results to filter out solutions that are semantically equivalent or are a subset of another solution. Another potential problem is that the interpreter may find erroneous interpretations based on errors in the input. This is a serious problem for keyword-spotters, since virtually every erroneous content word will result in errors in the interpretation. For a very strict parser this is not much of a problem – errors are likely to make parsing of the input impossible – but correct partial solutions are lost. PICKERING deals with this problem by searching for the best set of partial solutions. Finally, robustness can be inefficient if every interpretation is to be considered. The algorithm used in PICKERING is a kind of generate-and-filter technique that ensures that all interpretations are found. This can be costly, but the cost is balanced by the incremental processing – utterances are processed while the user is still speaking. For a more detailed description of the implementation of PICKERING, see Skantze & Edlund (2004).

6.5 GALATEA: Discourse modelling

The discourse modeller in HIGGINS is called GALATEA and is also implemented in Oz. It is designed to be generic – the required configurations and resources are all encoded in XML. As seen in Figure 6.5, the task of GALATEA is to take the communicative acts from both the user (as identified by PICKERING) and the system (as produced by an action manager), and build a discourse model – a model of what has been said during the discourse and which entities are referred to. The discourse model (encoded in XML) consists of two lists:

[8] A multi-paradigm programming language supporting open distributed computing, constraints and logical inference and concurrent object-orientation. See http://www.mozart-oz.org/.

- *CA-list*: A list of past communicative acts in chronological order, with the most recent act first.
- *Entity list*: A list of entities mentioned in the discourse, with the most recently mentioned entity first.

As a new CA is added to GALATEA, the following things are done:

1. SPEAKER and CAID attributes are added to the CA. These attributes contain information about which speaker made the contribution and an id for the CA (an automatically incremented number).
2. Grounding information is added to concepts in the CA, i.e., information about who added the concept to the model, in which turn, and how confident the system is in the concept.
3. Transformations of the CA are made, based on past CA's in the CA-list and a set of transformation rules. This way, ellipses may be resolved.
4. Discourse entities are identified in the CA and are assigned entity id's.
5. The identified entities are extracted from the CA and integrated into the entity list. If an anaphora is identified, the entities are unified.
6. The resulting CA is added to the CA-list.

After the discourse model has been updated, it may be consulted by an action manager that decides what to do next.

6.5.1 Grounding status

The grounding status that is added to concepts in the CA contains information about who added the concept to the model, in which turn, and how confident the system is in the concept. The grounding status is represented as a list, where each item represents an occurrence of the concept in the discourse. Each item in the list contains the following information:

- Who contributed the concept (SPEAKER)
- When was the contribution made (CAID)
- How confident is the system that the contribution was made (if not contributed by the system)? (CONF)
- How was the contribution realised (if not contributed by the system)? (FORM)

The CONF and FORM attributes are taken from the PICKERING results (concept confidence scores and surface form) and placed under a GROUNDING element, together with SPEAKER and CAID attributes, which have been assigned to the CA. In the semantic template used for unification, places where grounding information should be added are marked. Figure 6.9 shows how grounding status has been added to the CA from the parse result in Figure 6.8.

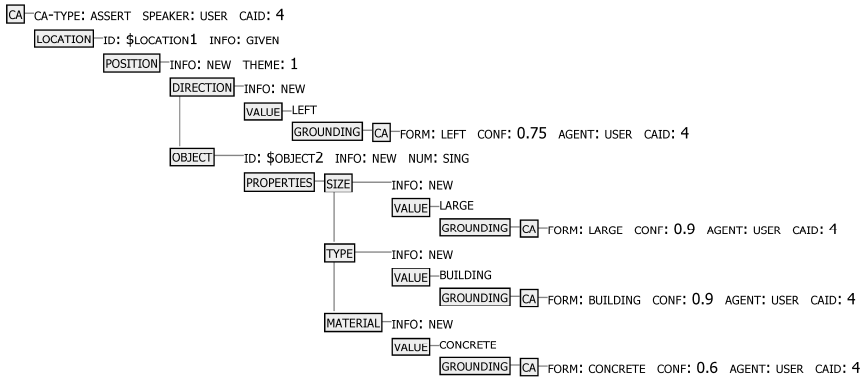


Figure 6.9: The semantic interpretation of the first CA in Figure 6.8, after grounding status has been added and entities have been identified.

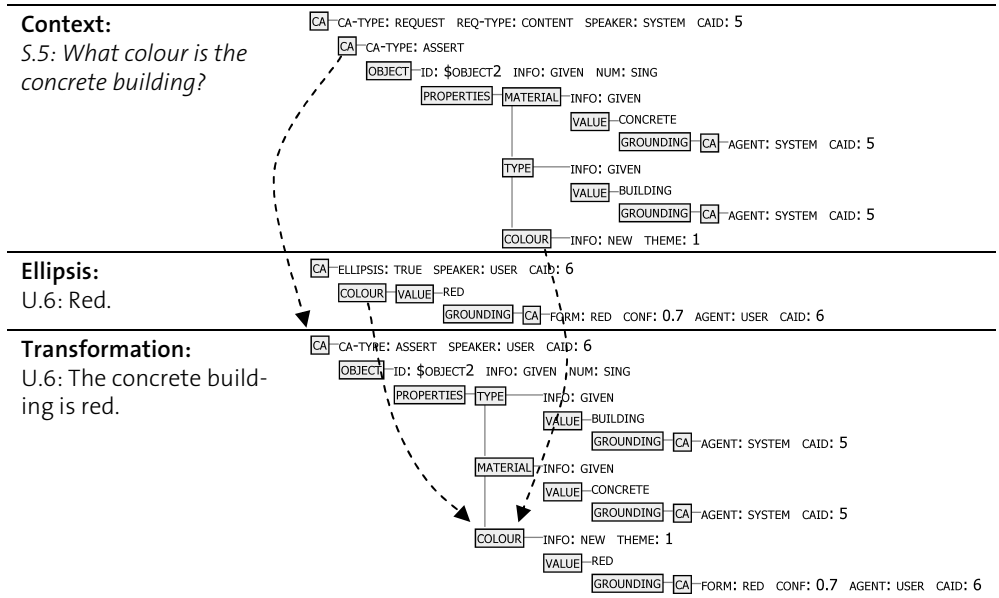
The grounding status can be compared with the "contextual functions" used in Heisterkamp & McGlashan (1996), and the "discourse pegs" used in McTear et al. (2005), that are used to model the grounding status (as discussed in 3.3.3.2).

6.5.2 Ellipsis resolution

GALATEA resolves ellipses by transforming them into full propositions. To do this, domain dependent transformation rules are used that transform communicative acts based on previous acts, similar to Carbonell (1983). Each rule has semantic preconditions for the current elliptical CA and the previous CA's, and a transformation description. The preconditions are formulated as semantic pattern trees that are matched against the target CA's. Each rule is applied in order; if the matching and transformation is successful, the algorithm restarts with the transformed CA until no more transformations can be done. Thus, a cascade of rules may be applied. The rules are written in XML, but will not be explained in more detail here.

Table 6.3 exemplifies a transformation based on a rule that handles all answers to wh-requests (which are called content-requests here). The preconditions for this rule are that the new CA is an ellipsis, and that there is a content-request in the CA-list with a requested node marked with THEME:1. The transformation description states that the ellipsis should be replaced by a new CA of type ASSERT, where the top node in the request is copied and the theme node is unified with the first node that can be unified in the ellipsis – in this case the COLOUR node. If the unification fails, the rule is not applied. The example also shows that the grounding status is added before resolving ellipses. This ensures that only concepts that were part of the original utterance are grounded, not those that are added in the ellipsis resolution.

Table 6.3: Example transformation of an ellipsis into full proposition.



Transformation rules may also be used for robustness to interpret utterances where PICKERING may have identified some fragments. An example of this was shown in Figure 6.8. The second and third phrases are identified as elliptical by PICKERING. In the context of the second phrase ("bus stop"), GALATEA will transform the third phrase ("on my right") into "I have a bus stop on my right". It is, of course, also possible to transform non-elliptical CA's that are dependent on the context for their interpretation. Each rule has a fairly generic purpose. Currently, about 10 different transformation rules are used for the navigation domain.

6.5.3 Anaphora resolution

GALATEA has no access to the domain database. Thus, it does not map discourse entities to "real" objects in the database. Instead, it keeps a list of entities that are mentioned (e. g., "a large building") in the discourse and assigns variable id's to them. The action manager may then use the entities in the discourse model as patterns and make a database search to find possible referents, that is, bindings to the entity id variables. What counts as an entity in a specific application must be specified so that GALATEA can recognise entities, and it is up to the dialogue system designer to define this. In the HIGGINS domain, entities that are modelled are landmarks, user locations and user goals. Table 6.4 shows a list of the entities modelled during the discourse in Table 6.1, and the variable id's that are assigned to the entities.

Table 6.4: The entities modelled during the discourse in Table 6.1.

Entity	Occurs in turn	Variable id
user goal	S.1, U.2	$goal1
ATM	U.2	$object1
user location	S.3, U.4, S.7, U.8, S.9, U.10, S.11, U.12	$location1
large red concrete building	U.4, S.5, U.6, S.13	$object2
bus stop	U.4, S.11, U.12	$object3
brown building	U.8, S.9, U.10	$object4

As shown previously, when semantic structures are created in PICKERING, they are marked with given/new status, based on definiteness and sentence structure. Some parts may be given and some new, for example when asserting information about a given object (see Figure 6.4 for an example). After a CA has been transformed, entities are identified and assigned entity id's according the following principles:

- If the entity has somehow already been assigned an id (for example by the action manager generating it), it is not affected.
- If the entity is marked as new, a new id is generated.
- If the entity is marked as given, the entity list is searched from top to bottom for an antecedent. The nodes marked as given in the entity to be added are used as a search pattern and the potential antecedents as targets, and a pattern match is performed.
    - If an antecedent is found, its id is used for the entity to be added.
    - Otherwise, a new id is generated.

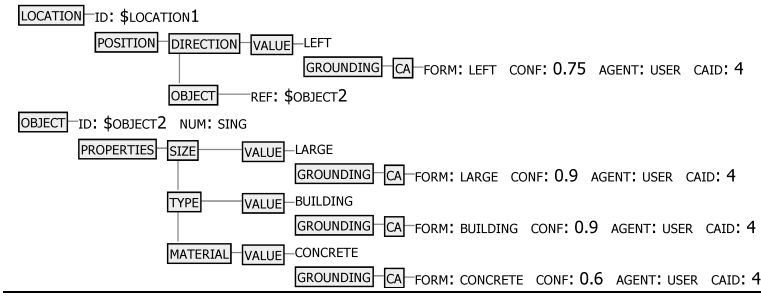
The identified entities are then added to the entity list according the following principles:

- If the id of the entity to be added is the same as for an entity in the entity list, these entities are unified and moved to the first position in the list.
- Otherwise, the entity is simply placed first on the entity list.

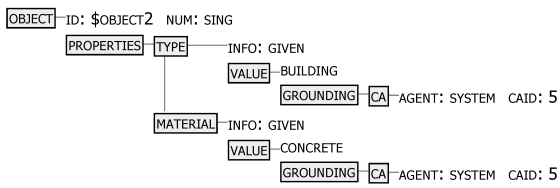
The entity list represents unified asserted information about entities. Therefore, information concerning the structure of the utterances they were extracted from is removed. This includes THEME and INFO attributes, as well as concepts that are only requested, such as COLOUR:RED in the request "is the building red?" Some early error detection is also done on the concept level – concepts with low confidence are filtered out. These concepts may be added later on if they are clarified, which is described in 6.7.2 below. This means that the entity list will contain unified information about entities in which the system has relatively high confidence. Thus, the entity list could also be viewed as the system's model of the common ground. Some examples of extracted entities are shown in Table 6.5.

Table 6.5: Examples of entities extracted from CA's.

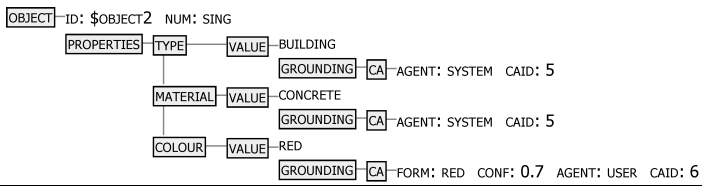
U.4: I have a large concrete building on my left ...



S.5: What colour is the concrete building?



U.6: Red



As the entities are unified in the entity list, the grounding status gets updated. Figure 6.10 shows an example of how the instances of $object2 extracted during U.4-U.6 in Table 6.5 are unified into one entity.

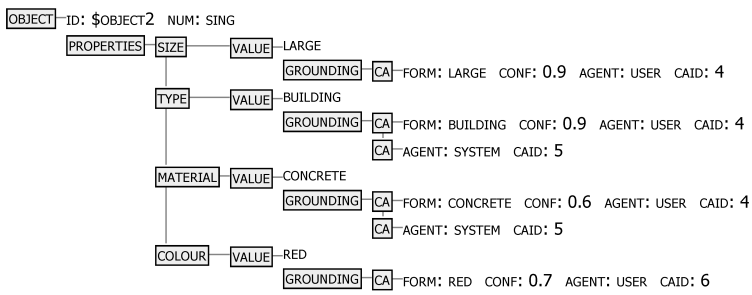


Figure 6.10: How the grounding status for entity $object2 in the entity list has been updated after U.6.

Since assertions about entities are unified in the entity list, it is possible to refer to an entity using a description that have not been used before to refer to that entity. For example, there is a reference in utterance S.13, in Table 6.1, to "the red building". There is no entity directly referred to in this way before, but the entity list will contain one after U.6.

The entity list may also be used by the action manager to select an appropriate referring expression for an entity, such as S.5 and S.11 in Table 6.1. If the entity is on top of the list, a simple pronoun may be used (unless the entity needs more grounding, which is described in 6.7.4 below). If there are other entities above it, the system may use a more elaborate definite noun phrase.

6.6 NAM: Navigation action manager

While GALATEA keeps the state of the discourse, the action manager(s) may keep the state of the system's intensions and its model of how the discourse entities map to objects in the database. This approach is different from the one taken in for example TrindiKit and the *information state approach* (Larsson & Traum, 2000), where all contextual information is kept in the same store. The purpose of this modularisation is to make the discourse modeller reusable, while the action manager may be highly domain dependent, implemented in any programming language, and limited in its tasks.

In the HIGGINS navigation domain, the navigation action manager (NAM) is the only module that has access to the map database and it is this module that performs the task-related decisions concerning the system's behaviour. Each time the discourse model gets updated, the NAM uses the entity list as a search pattern to find possible referents in the database, as described in 6.2. Table 6.6 below shows an example during turn U.4-U.6. This example shows how the value of one entity variable ($location1) may be constrained as more information is added about another entity ($object2), since the discourse model keeps information about the relations between these.

Table 6.6: How the possible bindings of the variable id's are constrained as more information is added.

		$location1	**$object2**
U.4	I have a large concrete building on my left [...]	loc734, loc82, loc293, loc83, loc94	obj25, obj04, obj73, obj94
S.5	*What colour is the concrete building?*		
U.6	Red	loc734, loc82	obj4, obj73

The action manager makes decisions based on a fairly simple decision algorithm, similar to a decision tree, which is traversed each time the discourse model gets updated. The decision algorithm for the NAM is shown in Table 6.7.

Table 6.7: The decision algorithm for the navigation action manager (NAM).

	Decision	Yes	No
1	Is the latest user utterance a request about the route?	Answer the request. End the turn.	Continue with 2.
2	Has the user stated the goal?	Continue with 3.	Request the goal. End the turn.
3	Is there any place that matches the goal description?	Continue with 4.	Tell the user that there is no such place. End the turn.
4	Has the user given any description of his location?	Continue with 5.	Request the user's position. End the turn.
5	Is there any location the user can be?	Continue with 6.	Perform late error detection and repair (described in 6.7.6).
6	Is the user's location exactly determined?	Continue with 11.	Continue with 7.
7	Is the user's location roughly determined?	Tell the user to position himself between two known objects in the vicinity. End the turn.	Continue with 8.
8	Are there a large number of possible user locations?	Ask the user to describe something more. End the turn.	Continue with 9.
9	Is there any entity in the entity list that may have several instances in the database and lacks description of properties?	Request more information about properties. End the turn.	Continue with 10.
10	Is it useful to ask a y/n-question about a specific object in a specific direction?	Ask the most optimal question. End the turn.	Ask the user to describe something else. End the turn.
11	Is the user at the goal?	Tell the user that he has arrived at the goal. End the turn.	Calculate the shortest path to the goal. Give a route direction to the next waypoint. End the turn.

For example, after U.2 in Table 6.1, the grounding action manager will first decide to display understanding of "an ATM" (as explained in 6.7.1 below). The NAM will then check the discourse model for the user's goal, and find that it is known. The next item on the check list is the user's position, and since there is no information about that in the discourse model, the NAM poses an open request on the user's position (S.3).

The notion of "issues" is central in the "issue-based approach" to dialogue management proposed by Larsson (2002). In this approach, the system keeps track of which issues are

raised and when they are resolved or rejected. In the domain considered here, we could say that an issue has been raised for example when the system requests the user's position. However, we have not found the explicit representation of such issues necessary for managing this domain using the approach presented in this chapter. Actually, it would be quite problematic to model issues in this domain, since it may often be hard to determine when issues are resolved or rejected. Consider turn S.7-U.8 from Table 6.1, where the system needs more information about the user's position:

(45) S.7: *Ok, can you see a wooden building?*
U.8: I can see a brown building.

In this example, the user does not directly answer the question. However, using the decision algorithm presented above, the system may now find out that it has enough information to continue with route directions. Whether the "issue" raised by the first question is resolved or not does not matter.

6.7 Error handling actions

By using the grounding status in the discourse model, the action manager(s) may perform various error handling actions, as described in this section.

6.7.1 GAM: Grounding action manager

As seen in the system architecture in Figure 6.5, the grounding action manager (GAM) is located before the navigation action manager (NAM) in the pipeline. The task of the GAM is to produce actions that are not dependent on the domain database. The GAM may do one of the following:

- Produce turn-yielding actions (such as clarification requests) and end the turn.
- Produce turn-keeping actions (such as acknowledgements) and pass the discourse model to the navigation action manager for more actions.
- Do nothing and simply pass the discourse model to the NAM to take actions.

This separation of action selection between the two action managers serves two proposes. First, since the GAM does not have to consult the database, it can typically act faster so that the system may be more responsive. Since the modules operate asynchronously, it may quickly produce actions (such as acknowledgements) that are performed while the NAM is processing. Second, since the GAM only reacts to the content in the discourse model (and does not consult any external knowledge sources), it is fairly generic. It is simply configured with a set of transformation rules written in XML, similar to the ones used in GALATEA for resolving ellipses. Whereas the transformation rules in GALATEA reinterpret new CA's based on past CA's, the

transformation rules in the GAM produce new system CA's based on past CA's. The GAM decision algorithm presently used in the HIGGINS navigation domain is presented in Table 6.8.

Table 6.8: The decision algorithm for the grounding action manager (GAM).

	Decision	Yes	No
1	Was the latest user CA a request for repetition?	Repeat the last system CA. End the turn.	Continue with 2.
2	Was the latest user CA a request to wait?	Acknowledge. End the turn.	Continue with 3.
3	Did the user's last CA contain a value (or values) with a low grounding status?	Request clarification on the value or a whole object. End the turn.	Continue with 4.
4	Was the last user CA an assertion?	Acknowledge. Continue with 5.	Continue with 5.
5	Did the user's last CA contain a value with a medium grounding status?	Display understanding. Continue with 6.	Continue with 6.
6	Was the last user CA an expression of greeting or thanks?	Express greeting or thanks. Continue with 7.	Continue with 7.
7	Was the last user CA a fragmentary direction?	Ask the user what he can see in the direction. End the turn.	Continue with 8.
8	Was the last user CA a fragmentary object?	Ask the user if he can see the object. End the turn.	Send the discourse model to the NAM.

Currently, a very simple distinction is made between different levels of grounding status. If the grounding status contains a mention of the concept from the system, it is considered to be high. If the concept is only mentioned by the user, the highest confidence score is compared against a set of pre-defined thresholds. This simplistic model is refined later on in this thesis.

6.7.2 Fragmentary clarification

The following turns from Table 6.1 exemplify the use of fragmentary clarification:

(46) U.8: **I CAN SEE A BLUE BUILDING**
S.9: *Blue?*
U.10a: **NO**
U.10b: **BROWN**

This use of fragmentary clarification requests in spoken dialogue systems have not been studied to a great extent. As discussed in 3.3.2.5, if correctly handled, such requests may increase

both the naturalness and efficiency of the dialogue. If the hypothesis is incorrect, the user should be able to efficiently correct the system, as in the example. To handle the turns in the example correctly, a system should be able to do the following things:

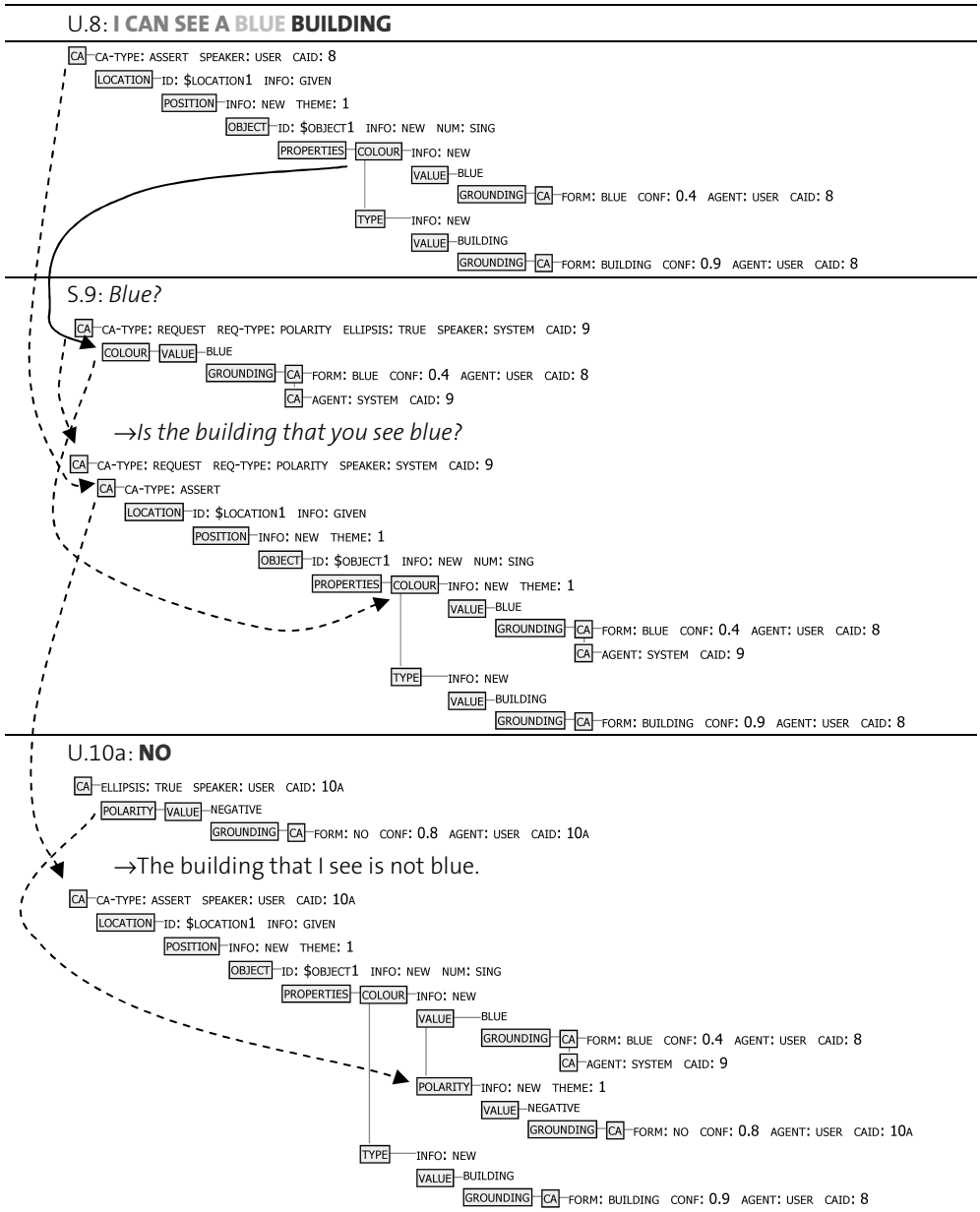
- Identify the problematic concept(s) (in U.8).
- Produce the request (S.9) accurately.
- Interpret the negation (U.10a) correctly. Notice that the user simply negates the proposed colour of the building – the fact that the user can see a building is still accepted.
- Interpret the correction (U.10b) correctly; to understand that the user can see a brown building.
- Understand that only the COLOUR concept has been grounded, not the entire contribution U.8.

We will now show how these requirements are handled in HIGGINS. As seen in Table 6.8, if a concept or a tree of concepts with low grounding status is detected in decision 3, the GAM may pose a fragmentary clarification request and end the turn. In HIGGINS, such utterances are not treated as a special kind of grounding or feedback utterance. Instead, they are resolved just like other ellipses into a full proposition. However, since grounding is modelled for all utterances, the clarification request will help to boost the weak grounding status. The clarification request is very simple to produce – the GAM simply has to embed the concepts in a CA of type REQUEST and send it to OVIDIUS (the natural language generator).

OVIDIUS will make a surface realisation of a fragmentary clarification request (with prosodic markup) and send it to the TTS (which is described in 6.7.7 below). When GALATEA receives this elliptical CA, it is transformed into a full yes/no request. This way, subsequent reactions to this request will be interpreted correctly, while only the concepts that are actually realised in the ellipsis will get an updated grounding status. An example of how this is done for U.8-U-10a is shown in Table 6.9. As can be seen in the example, negations are represented with POLARITY nodes that are attached to concepts. This makes it easy to represent and integrate "no" answers, as well as adverbial negations.

Figure 6.11 shows the resulting entity in the entity list after the dialogue. As can be seen, the negative answer is kept in the model. This is useful when constraining possible user locations, since the POLARITY nodes are taken into account when doing tree pattern matching. A concept may have several POLARITY nodes with different polarities and the POLARITY nodes may also have grounding status, as can be seen in the example. This makes it possible for one participant to confirm something while another participant negates it. Thus, POLARITY nodes can be used to model the "acceptance" level discussed in 3.1.1. This also means that POLARITY nodes themselves may have a low grounding status, for example if the "no" in Table 6.9 would get a low confidence score, and need further grounding.

Table 6.9: How a fragmentary clarification request is constructed and interpreted. Dotted lines are part of ellipsis resolution in GALATEA. Solid lines are part of action construction in the GAM.



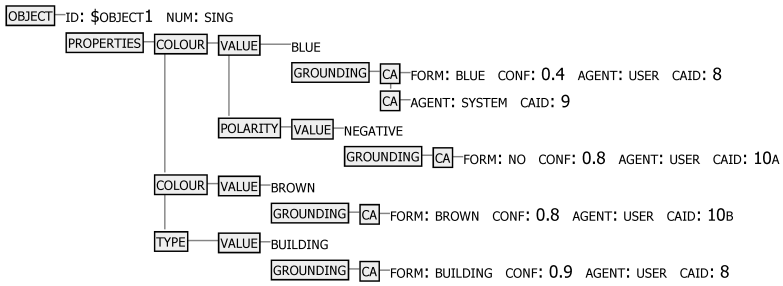


Figure 6.11: How the grounding status for entity $object1 in the entity list has been updated after U.3.

Like all requests, clarification requests do not need to be answered. The concepts which are to be clarified are not transferred to the entity list, since they have low grounding status. Thus, if the clarification request would not have been answered, there would be no information about the concept BLUE for this entity. This is also true if the user would have answered just "brown", in which case the entity would have the concept BROWN, but no information on the concept BLUE. If the user reactions have low confidence scores, this will trigger new clarification requests. Of course, reactions in the form of full propositions are also possible, such as "I can see a brown building".

The fragmentary clarification requests discussed above express request for confirmation for concepts that the system lacks confidence in. However, as discussed in 3.1.4, clarification requests may also be caused by (partial) lack of hypotheses and express request for repetition. The following example (taken from a real dialogue presented in the next chapter) illustrates such a request for partial repetition:

(47) S: *Can you see a brick building on your left?*
U: **NOW ON MY RIGHT** (No, on my right.)
S: *What do you see on your right?*

In this example, the system misrecognises the first part of the user correction ("no") and GALATEA thereby fails to interpret the elliptical utterance "on my right". However, this is recognised as an unresolved fragment containing a direction, and Decision 7 in Table 6.8 will lead the GAM to pose a clarification request for the missing object.

6.7.3 Separate display utterances

As seen in Table 6.8, decision 5 may lead the GAM to produce a display utterance, presumably after it has triggered on an assertion and produced an acknowledgement (decision 4). The NAM may then continue and produce the next task-related utterance. This is exemplified in Table 6.10.

Table 6.10: How a display utterance is selected and interpreted by GALATEA.

Turn	Decision	Before ellipsis resolution	After ellipsis resolution
S.1	NAM: 2-no	*Where do you want to go?*	*Where do you want to go?*
U.2		**TO AN ATM**	I want to go to an ATM.
S.3a	GAM: 4-yes	*Ok*	*Ok*
S.3b	GAM: 5-yes	*an ATM*	*you want to go to an ATM.*
S.3c	NAM: 4-no	*Can you describe where you are now?*	*Can you describe where you are now?*

The utterances "Ok" and "an ATM" are synthesised and played back while the NAM is asynchronously deciding on the next act. In this example, it takes a very short time to produce S.3c, and the utterance is simply queued up in the TTS. However, if the NAM had needed more time for database searches, this would have helped the system to act more responsively.

Display utterances are handled in a way very similar to fragmentary clarification requests. However, while these ellipses are resolved as requests, display utterances are resolved as assertions ("you want to go to an ATM"), which the user may object to. Since the concepts that are displayed have a higher confidence score, they are directly transferred to the entity list. Therefore, the user does not have to (but may, if he wish) confirm the displayed concept. If the user objects, a negative POLARITY node is attached.

It is also possible that the user might object to a displayed misunderstanding by other means, such as those listed in example (39) on page 57. Such objections could be handled by either representing them as a special type of negation (which are only treated as negations by GALATEA after a display of understanding), or let one of the action managers remove the erroneous concepts in the discourse model if such an objection is detected (see late error detection below).

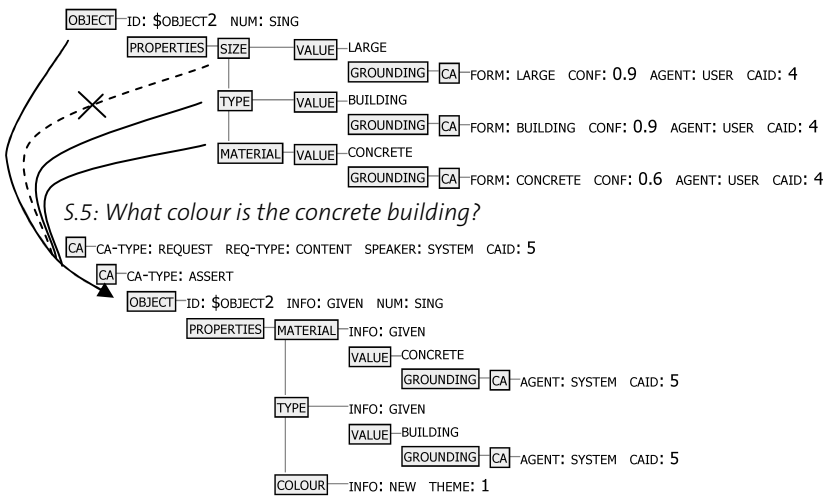
6.7.4 Integrated display of understanding

Speakers do not only display their understanding using separate display utterances. They also do this to various extents while performing task-related CA's (i.e., integrated display of understanding). As example (35) on page 54 shows, one way of doing this is to choose how to refer to entities. In HIGGINS, the NAM makes such decisions. Every time the NAM refers to a given entity, appropriate integrated display of understanding is automatically done. To construct a referring expression to an entity that is in the entity list, the NAM simply makes a copy of the entity and removes all concepts with high grounding status. This ensures that the concepts with low grounding status will get a high grounding status. An example is shown in Table 6.11. When the system needs to ask a question on the colour of the building, it copies

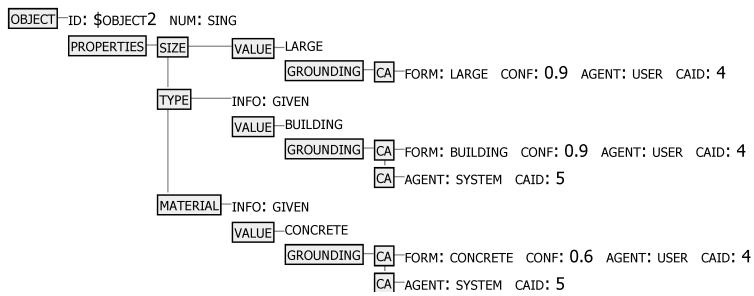
the entity and removes the concept LARGE, since it has a high grounding status, based on the confidence score. The TYPE concept (BUILDING) is not removed, since it is often needed for a valid referring expression – otherwise it would say "what colour is the concrete". Since GALATEA also models the system's actions, those concepts will then get a high grounding status.

Table 6.11: How the system creates a referring expression to $object2 and how this affects the grounding status of $object2 in the entity list.

Entity list: $object2 before S.5



Entity list: $object2 after S.5



6.7.5 Non-understanding recovery

As discussed in 3.3.2.4 and as the results of the experiment in Chapter 4 suggest, it may not be optimal to signal non-understanding when the system does not understand what the user is saying, even if there is a complete non-understanding. Instead, humans tend to ask new task-

related questions. This behaviour is implemented in HIGGINS in the following way. Every time the NAM selects an action according to Table 6.7, it stores this as the last utterance. If the next user utterance is not understood by the system, the decision algorithm interpreter is programmed to not produce the same action again. For many decisions, there are several possible outcomes. For example, 10-yes may result in different questions. The most optimal question is selected first, but after a non-understanding, the system will be hindered to ask this question again, and will thus select the second most optimal question. Here is an example:

(48) S: *Can you see a tree in front of you?*
U: [non-understanding]
S: *Can you see a bus stop on your left?*
U: yes

6.7.6 Late error detection and repair

Due to the misrecognition in U.4 (in Table 6.1), the discourse model will contain an error. However, after turn U.10, the system discovers that there is no place where the user can be. The decision 5-no in Table 6.7 tells the NAM that it should now do late error detection and repair. To do this, it looks through the discourse model to find concepts with low grounding status. The only concept with a relatively low grounding status is shown in Figure 6.12.

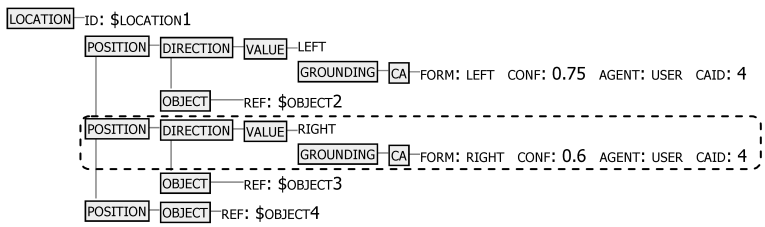


Figure 6.12: A potential error is detected in the user location.

One way of repairing the potentially erroneous concept is to remove it from the discourse model and search for possible user locations again. Note that this only would be done on the concept level; the system would still keep the information that the user can see a tree somewhere.

In this example, the system instead chooses to make a late clarification request. Table 6.12 shows how this clarification is interpreted. Late clarification requests are realised as full propositions and not as fragments, since this would generally not be suitable (the context needed for their resolution is too distant). Figure 6.13 shows the resulting user location entity after this clarification.

Table 6.12: The interpretation of the late clarification request.

Turn	CA	Resolved by GALATEA
S.11	*Do you really have a bus stop on your right?*	
U.12a	**NO**	I do not have a bus stop on my right.
U.12b	**IN FRONT**	I have a bus stop in front of me.

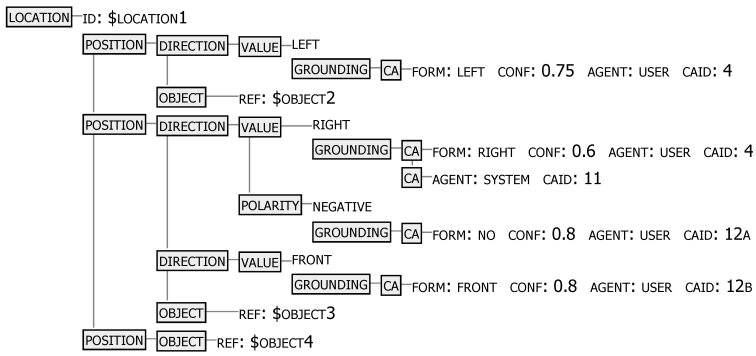


Figure 6.13: The user location after the late clarification request.

It is also possible to remove information that is associated with a specific turn, by looking at the CAID attribute in the grounding status, for example after the user objects to a display of understanding. Since the model also contains information about what the user has grounded, it is also possible to detect cases where the user misunderstands the system. For example, the user never displays any understanding of the concepts wood and building in U.8 in Table 6.1, and this can be detected in the discourse model.

When the system discovers that there is no place where the user can be, there will sometimes be no information with low grounding status in the discourse model that may be removed. In such cases, information about the user's location will simply be cleared in the discourse model and the system will say "sorry, I have lost you". Decision 4 in Table 6.7 will then lead the NAM to produce something like "can you describe where you are".

6.7.7 Utterance generation

As can be seen in the system architecture in Figure 6.5, the textual representation of system utterances are realised by a module called OVIDIUS, also implemented in Oz. OVIDIUS takes a system CA as input and generates a text that is to be synthesised by a TTS. The text may also have some prosodic markup.

OVIDIUS uses a set of template rules, working much like inverted PICKERING grammar rules – they match on semantic structures and produce text strings. The processing is similar

to XSLT (Clark, 1999): one template rule may call another template rule to generate sub-parts of the semantics (such as nominal phrases). Much of the vocabulary expected from the user is also used in the system output. To make the input and output vocabularies coherent and to reduce double work, OVIDIUS may use the same lexicon that is used in PICKERING to map semantic structures to surface strings.

As discussed in 2.3.4, humans engaging in dialogue tend to coordinate their linguistic behaviour with each other. Users of spoken dialogue systems tend to adjust the vocabulary and expressions to the system and may expect the system to do the same. When a speaker deliberately chooses another wording than their interlocutor, the other speaker may feel obliged to adapt to this. This phenomenon is especially important to understand when making clarification requests. Consider the following example:

(49) U.1: I have a violet house on my left.
S.2: *A purple building?*

In S.2, the system makes a surface realisation of the semantic representation of the fragment to be clarified. The template rules and lexicon allows OVIDIUS to generate different forms from the same semantic representation – the underlined phrases have exactly the same semantic representations (in this specific domain). Thus, it would make perfect sense to the system to produce S.2 in order to clarify the referring expression. However, this may be confusing for the user, who may get the impression that the system is trying to correct her or enforce a certain language use. In HIGGINS, this is avoided by tracking the surface form in the grounding status, as mentioned previously. As can be seen in Table 6.9, when a clarification ellipsis is generated in the GAM, the previous grounding status of the concepts to be clarified is not removed. Figure 6.14 shows what the message from the GAM to OVIDIUS looks like when utterance S.2 in example (49) above is to be generated. The FORM attribute in the grounding status helps OVIDIUS to select the same lexical entry that was used when parsing the original user utterance.

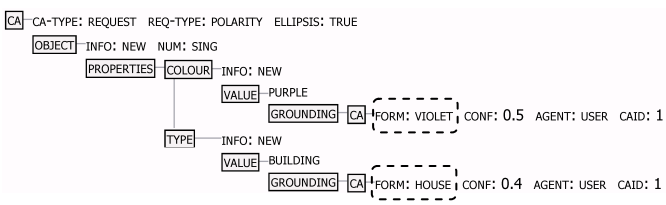


Figure 6.14: The semantic representation of S.2 in example (49), to be transformed into surface form by OVIDIUS.

Fragmentary clarification requests and display utterances have the same textual form. The interpretation of these utterances is therefore dependent on prosody to a large extent. For speech synthesis, the KTH rule-based speech synthesis system (Carlson et al., 1982) was used, to-

gether with a diphone Swedish male MBROLA voice (Dutoit et al., 1996). The only available parameters for this TTS were pitch range, speaking rate and pitch base. Since fragmentary grounding utterances are dynamically generated, it is not possible to hand-craft the pitch curve for each utterance. To approximate the commonly described tonal characteristic for questions – overall higher pitch (Hirst & Cristo, 1998) – the pitch range was initially simply increased for fragmentary clarification requests, in order to distinguish them from display utterances. In Chapter 9, we will explore the relationship between the prosodic realisation of fragmentary grounding utterances and the interpretation of them in more depth.

6.8 Discussion

This chapter has focussed on how to model the way all utterances may provide evidence of understanding, not just special "grounding acts". Thus, as the system gives the user a route direction (a mainly task-related act), it may simultaneously display its understanding by the way it refers to landmarks. Another implication is that for example clarification requests are treated in the same way as other requests and not as a special type of utterance. The tracking of groundings status ensures that the evidence that is simultaneously provided is accounted for. It has also been shown how the grounding status is tracked while performing ellipsis and anaphora resolution. This means that the choice of referring and elliptical expressions affects the system's model of what has been grounded.

The notion of high and low grounding status is very simplistic. A more advanced model, where the grounding status could be evaluated as a continuous (possibly probabilistic) score – similar to the "belief updating" approach in Bohus & Rudnicky (2005a) – would be interesting to explore. To improve such classification, the grounding status could be enriched with more information on how the concept was grounded, such as prosodic information. The confidence scores used for the grounding status are taken directly from the speech recogniser. An interesting improvement would be to include the methods for early error detection explored in Chapter 5.

Currently, the thresholds used for choosing error handling strategies have been manually tuned. Also, the choice of strategy is not dependent on the current task. For example, when the user asserts her goal, as in U.2 in Table 6.1, the system might navigate the user to the wrong place and have to start all over again. Thus, the system should have a relatively high threshold for accepting such hypotheses, since the cost of task failure is higher. When the user asserts her position, as in U.4, an error would probably be detected rather early, when the system finds out that there is no place the user can be, and the threshold should be lower. In Chapter 8, we will show how a decision-theoretic framework might be applied to make such choices.

In this chapter, we have only described how evidence may be given on the perception level (with the reading "did you say X?"). Since the domain encourages the use of ellipses and anaphora, it would be useful to also give evidence on the understanding level, as in the following example:

(50) U: Now I can see the building.
S: *The red building?*

In this example, the system does not make a clarification request due to low ASR confidence score, but rather because the reference resolution is ambiguous. To handle this, GALATEA should be endowed with the capability of detecting that a referring expression is ambiguous and, if this is the case, not resolve the expression, but to let the GAM pose a clarification request.

Neither does the system currently handle fragmentary clarification requests from the user, since this would require prosodic analysis. This might be a problem if the users adapt to the system and start to make such requests (which they are likely to do).

An important question is to what extent the methods and models described in this chapter may apply to other domains. The HIGGINS components have been used in Connector, a dialogue system acting as an automatic switchboard and secretary (Edlund & Hjalmarsson, 2005). Connector is part of the EU-funded CHIL-project, a project investigating automatic tracking and support of interactions in meeting rooms. The HIGGINS components have also been used in the conversational training game DEAL (Hjalmarsson et al., 2007). DEAL is a dialogue system for second language learners, where the user talks to an embodied conversational agent in a flea market domain, in order to train conversational skills.

6.9 Summary

In this chapter, we have described how speech recognition errors are handled in the HIGGINS spoken dialogue system. It has been shown how all utterances may operate on the domain level, while simultaneously providing evidence of understanding. The discourse modeller GALATEA keeps track of this by modelling the grounding status of concepts while resolving ellipses and anaphora. The grounding status includes information such as concept confidence scores and surface realisation, which are extracted by the robust interpreter PICKERING. The grounding status may be used by a set of action managers to perform concept-level error handling, such as display of understanding, clarification requests and misunderstanding repair.