FLEXIBLE COMMAND INTERPRETATION ON AN INTERACTIVE DOMESTIC SERVICE ROBOT

Stefan Schiffer, Niklas Hoppe and Gerhard Lakemeyer
Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany

Keywords: Natural language processing, Decision-theoretic planning, Interpretation, Domestic service robotics.

Abstract: In this paper, we propose a system for robust and flexible command interpretation on a mobile robot in domestic service robotics applications. Existing language processing for instructing a mobile robot often make use of a simple, restricted grammar where precisely pre-defined utterances are directly mapped to system calls. This does not take into account fallibility of human users and only allows for binary processing; either a command is part of the grammar and hence understood correctly, or it is not part of the grammar and gets rejected. We model the language processing as an interpretation process where the utterance needs to be mapped to a robot's capabilities. We do so by casting the processing as a (decision-theoretic) planning problem on interpretatory actions. This allows for a flexible system that can resolve ambiguities and which is also capable of initiating steps to achieve clarification.

1 INTRODUCTION

In this paper we present a system for flexible command interpretation to facilitate natural human-robot interaction in a domestic service robotics domain. We particularly target the *General Purpose Service Robot* test from the RoboCup@Home competition (Wisspeintner et al., 2009), where a robot is confronted with ambiguous and/or faulty user inputs in form of natural spoken language. The main goal of our approach is to provide a system capable of resolving these ambiguities and of interactively achieving user satisfaction in the form of doing the right thing, even in the face of incomplete, ill-formed, or faulty commands.

We model the processing of natural spoken language input as an interpretation process. More precisely, we first analyse the given utterance syntactically by using a grammar. Then, we cast the interpretation as a planning problem where the single actions available to the planner are to interpret syntactical elements of the utterance. If, in the course of interpreting, ambiguities are detected, the system uses decision-theory to weigh up different alternatives. The system is also able to initiate clarification to resolve ambiguities and to handle errors as to arrive at a successful command interpretation eventually. Since our current high-level control already knows about the robot's capabilities (the actions and the parameters that these actions need), we want to tightly connect the interpretation with it.

2 FOUNDATIONS AND RELATED WORK

In this section, we introduce the foundations, namely the situation calculus and GOLOG, that our approach builds upon. We then briefly review related work.

2.1 Foundations

The high-level control of our domestic service robot uses a logical programming and plan language called READYLOG. It is a dialect of GOLOG which itself is based on the situation calculus.

2.1.1 The Situation Calculus and GOLOG

The situation calculus (McCarthy, 1963) is a sorted second order logical language with equality that allows for reasoning about actions and their effects. The situation calculus distinguishes three different sorts: *actions*, *situations*, and domain dependent *objects*. The state the world is in is characterised by functions and relations with a situation as their last argument. They are called *functional* and *relational fluents*, re-

Schiffer S., Hoppe N. and Lakemeyer G..
FLEXIBLE COMMAND INTERPRETATION ON AN INTERACTIVE DOMESTIC SERVICE ROBOT.
DOI: 10.5220/0003707300260035
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (ICAART-2012), pages 26-35
ISBN: 978-989-8425-95-9

spectively. The world evolves from an initial situation S_0 only due to primitive actions, e.g., $s' = do(a,s)$ means that the world is in situation s' after performing action a in situation s. Possible world histories are represented as sequences of actions. For each action one has to specify a *precondition axiom* stating under which conditions it is possible to perform the respective action and *effect axioms* formulating how the action changes the world in terms of the specified fluents. An action precondition axiom states when an action can be executed. The effects that actions have on the fluents are described by so-called successor state axioms (Reiter, 2001).

GOLOG (Levesque et al., 1997) is a logic-based robot programming and plan language based on the situation calculus. It allows for Algol-like programming but it also offers some non-deterministic constructs. A *Basic Action Theory* (BAT), which is a set of axioms describing properties of the world, axioms for actions and their preconditions and effects as described above, and some foundational axioms, then allows for reasoning about a course of action.

There exist various extensions and dialects to the original GOLOG interpreter, one of which is READYLOG (Ferrein and Lakemeyer, 2008). It integrates several extensions like interleaved concurrency, sensing, exogenous events, and on-line decision-theoretic planning (following (Boutilier et al., 2000)) into one framework. In READYLOG only partially specified programs are needed which leave certain decisions open, which then are taken by the controller based on an optimisation theory. This is done using a Markov Decision Process (MDP) (Puterman, 1994); decision-theoretic planning is initiated with $solve(p,h)$, where p is a GOLOG program and h is the MDP's solution horizon). Two important constructs used in this regard are the non-deterministic choice of actions $(a|b)$ and arguments $(pickBest(v,l,p))$, where v is a variable, l is a list of values to choose from, and p is a GOLOG program. Then each occurrence of v is replaced with the value chosen. For details we refer to (Ferrein and Lakemeyer, 2008).

2.2 Related Work

We want to build upon the theory of *speech acts* as introduced by Austin (Austin, 1975) and Searle (Searle, 1969). Based on these works, Cohen and Levesque (Cohen and Levesque, 1985) already investigated a formal theory of rational interaction. We, restrict ourselves to command interpretation and do not aim for a full-fledged dialogue system. Nevertheless, we follow their formal theory of interpretation and we carry out our work in the context of the situation calculus.

The use of definite clause grammars for parsing and interpreting natural language has already been shown in (Beetz et al., 2001). Despite being relatively ad hoc and the fact that the small grammar only covered a constrained subset of English, their system provided a wide spectrum of communication behaviours. However, in contrast to their approach we want to account for incomplete and unclear utterances both by using a larger grammar as well as adding interpretation mechanisms to the system.

(Fong et al., 2003) developed a system on a robot platform that manages dialogues between human and robot. Similar to our approach, input to the system is processed by task planning. However, queries are limited to questions that can either be answered with yes or no or a decimal value. A more advanced system combining natural language processing and flexible dialogue management is reported on in (Clodic et al., 2007). User utterances are interpreted as communicative acts having a certain number of parameters. The approach is missing a proper conceptual foundation of objects and actions, though. This makes it hard to adapt it to different platforms or changing sets of robot capabilities.

(Görz and Ludwig, 2005), on the other hand, built a dialogue management system well-founded by making use of a concept hierarchy formalised in Description Logics (DL). Both, the linguistic knowledge as well as the dialogue management are formalised in DL. This is a very generic method for linking lexical semantics with domain pragmatics. However, this comes with the computational burden of integrating description logics and appropriate reasoning mechanisms. We want to stay within our current representational framework, that is, the situation calculus and Golog, and we opt to exploit the capabilities to reduce computational complexity with combining programming and planning.

3 METHOD & APPROACH

As mentioned before, we cast the language processing of spoken commands on a domestic service robot as an interpretation process. We decompose this process into the following steps. First, the acoustic utterance of the user is being transformed into text via a speech recognition component which is not part of this paper's contribution. The transcribed utterance is then passed on for syntactic analysis by a grammar. After that, the interpretation starts, possibly resolving ambiguities and generating intermediate responses. If the utterance could be interpreted successfully, it is executed, otherwise it is being rejected. We will now present the individual steps in more detail.

3.1 Syntactical Language Processing

Given the textual form of the user utterance, the first thing we do is a syntactical analysis. This syntactic operation uses a grammar. Since the entirety of the English language is not context-free as revealed by (Shieber, 1985) and the targeted application domain allows for a reasonable restriction, we confine ourselves to directives. Directives are utterances that express some kind of request. Following Ervin-Tripp (Ervin-Tripp, 1976) there are six types of directives:

1. Need statements,
   e.g., "I need the blue cup."
2. Imperatives,
   e.g., "Bring me the blue cup!"
3. Imbedded imperatives,
   e.g., "Could you bring me the blue cup?"
4. Permission directives,
   e.g., "May I please have the blue cup?"
5. Question directives,
   e.g., "Have you got some chewing gum?"
6. Hints, e.g., "I have run out of chewing gum."

Ervin-Tripp characterises question directives and hints as being hard to be identified as directives even for humans. Moreover, permission directives are mostly used only when the asker is taking a subordinate role, which will not be the case of a human instructing a robot. That is why we restrict ourselves to a system that can handle need statements, imperatives and imbedded imperatives only.

3.1.1 A Grammar for English Directives

For any of these directives what we need to make the robot understand the user's command is to distill the *essence* of the utterance. To eventually arrive at this, we first perform a pure syntactic processing of the utterance. An analysis of several syntax trees of such utterances revealed structural similarities that we intend to capture with a grammar. An example for a syntax tree is given in Figure 1.

Using common linguistic concepts, the main structure elements are verb (V), auxiliary verb (AUX), verb phrase (VP), noun phrase (NP), conjunction (CON), preposition (PREP), and prepositional phrase (PP). We further introduce a structure element *object phrase* which is a noun phrase, a prepositional phrase, or concatenations of the two. Multiple verb phrases can be connected with a conjunction. What

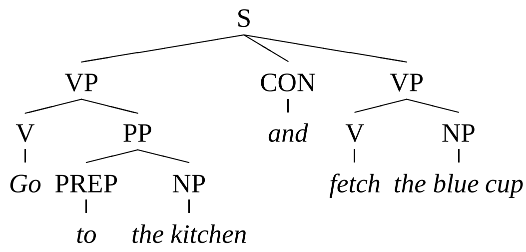

Figure 1: Syntax tree for the utterance "Go to the kitchen and fetch the blue cup".

is more, commands to the robots may be prefixed with a salutation. Also, for reasons of politeness, the user can express courtesy by saying "please". Putting all this together, we arrive at a base grammar that can be expressed in Extended Backus-Naur Form (EBNF) (Scowen, 1993) as shown in Figure 2.

```
s —> salutation utterance
s —> utterance
%
          utterance —> needstatement |
                       imperative    |
                       imbedded_imperative
%
      needstatement —> np vp |
                       needphrase vp
         imperative —> vp
imbedded_imperative —> aux np vp
         needphrase —> "i" prompt "you to"
% verb phrase
                 vp —> vp'
                 vp —> vp' conjunction vp
                vp' —> verb
                vp' —> verb obp
                vp' —> courtesy vp'
% object phrase
                obp —> np | pp
                obp —> np obp | pp obp
% noun phrase
                 np —> noun | pronoun
                 np —> determiner noun
% propositional phrase
                 pp —> prep np
```

Figure 2: Base grammar in EBNF.

In addition to the base grammar we need a base lexicon that provides us with the vocabulary for elements such as prepositions, auxiliary verbs, courtesies, conjunctions, determiners, and pronouns. To generate a system that is functional in a specific setting, we further need a lexicon containing all verbs for the capabilities of the robot as well as all the objects referring to known entities in the world. This depends on the particular application, though. That is why we couple this to the domain specification discussed later. The base grammar, the base lexicon, and the domain specific lexicon then yield the final grammar that is used for syntactical processing.

Since we are only interested in the core information, the most relevant parts of the utterance are verbs, objects, prepositions, and determiners. We can drop auxiliary verbs, filler words, courtesies, and alike without losing any relevant information. Doing so, we finally arrive at an internal representation of the utterance in a prefix notation depicted below, that we use for further processing.

```
[and, [[Verb, [objects, [[Preposition,
          [Determiner,Object]],...]] ]], ...]
```

The list notation contains the keyword `and` to concatenate multiple verb phrases and it uses the keyword `objects` to group the object phrase. If an utterance is missing information we fill this with `nil` as a placeholder.

3.2 Planning Interpretations

After syntactic pre-processing of an utterance into the internal representation, the system uses decision-theoretic planning to arrive at the most likely interpretation of the utterance, given the robot's capabilities. The interpretation is supposed to match the request with one of the abilities of the robot (called a skill) and to correctly allocate the parameters that this skill requires.

In order to do that, we need to identify the skill that is being addressed first. We are going about this from the verb which has been extracted in the syntactical processing, possibly leaving ambiguities on which skill is referred to by the verb. Secondly, the objects mentioned in the utterance need to be mapped to entities in the world that the robot knows about. Lastly, a skill typically has parameters, and the verb extracted from the utterance has (multiple) objects associated to it. Hence, we need to decide which object should be assigned to which parameter. To make things worse, it might very well be the case that we have either too many or too few objects in the utterance for a certain skill.

We cast understanding the command as a process where the single steps are interpretation actions, that is, interpreting the single elements of the utterance. At this point READYLOG and its ability to perform decision-theoretic planning comes into play. The overall interpretation can be modelled as a planning problem. The system can choose different actions (or actions with different parameters) at each stage. Since we want to achieve an optimal interpretation, we make use of decision-theoretic planning. That is to say, given an optimisation theory, we try to find a plan, i.e. a sequence of actions, which maximises the expected reward.

3.2.1 Domain Specification

During the interpretation process we need to access the robot's background knowledge. We organise this knowledge to capture generic properties and to make individual parts available to (only) those components which need them. Three types of information are distinguished: *linguistic*, *interpretation*, and *system*. The linguistic information contains everything that has to do with natural language while interpretation information is used during the interpretation process and system information features things like the specific system calls for a certain skill. The combination of these three types is then what makes the connection from natural language to robot abilities. We use ideas from (Gu and Soutchanski, 2008) to structure our knowledge within our situation calculus-based representation.

In an ontology, for every *Skill* we store a *Name* as an internal identifier that is being assigned to a particular skill during the interpretation. A skill further has a *Command* which is the denotation of the corresponding system call of that skill. *Synonyms* is a list of possible verbs in natural language that may refer to that skill. *Parameters* is a list of objects that refer to the arguments of the skill, where *Name* again is a reference used in the interpretation process, *Attributes* is a list of properties such as whether the parameter is numerical of string data. *Significance* indicates whether the parameter is optional or required, and *Preposition* is a (possibly empty) list of prepositions that go with the parameter. For the information on entities in the world (e.g. locations and objects) we use a structure *Object* which again has a *Name* as an internal identifier used during the interpretation. *Attributes* is a list of properties such as whether the object "is a location" or if it "is portable". *Synonyms* is a list of possible nouns that may refer to the object and *ID* is a system related identifier that uniquely refers to a particular object.

3.2.2 Basic Action Theory

Now that we have put down the domain knowledge on skills and objects, we still need to formalise the basic action theory for our interpretation system. We therefore define three actions, namely *interpret_action*, *interpret_object*, and *assign_argument*. For all three we need to state precondition axioms and successor state axioms. We further need several fluents, that describe the properties of the interpretation domain we operate in. Let's take a look at those fluents first. We use the fluents $spoken\_verb(s)$ and $spoken\_objects(s)$ to store the verb and the list of objects extracted in the syntactic processing. Further, we use the fluents *assumed_action*(s) and *assumed_objects*(s) to store the skill and the list of objects that we assume to be addressed by the user, respectively. Both these fluents are *nil* in the initial situation S_0 since no interpretation has taken place so far. The fluent *assumed_arguments*(s) contains a list of pairings between parameters and entities. Finally, $finished(s)$ indicated whether the interpretation process is finished.

Let us now turn to the three interpretation actions. The precondition axiom for *interpret_action* states that *interpret_action*(k) is only possible if we are not done with interpreting yet and the word k actually is a synonym of the verb spoken. Similarly, *interpret_object*(e) is possible for an entity e only if we are not finished and the object (from *spoken_object*(s)) is a synonym appearing for e. Finally, the precondition axiom for *assign_argument* for an entity e and parameter p checks whether the interpretation process is not finished and there is no entity assigned to the parameter yet. Further, p needs to be a parameter of the assumed skill and we either have no preposition for the object or the preposition we have matches the preposition associated with the parameter. Lastly, the attributes associated to parameter p need to be a subset of the attributes for the entity. To allow for aborting the interpretation process we additionally introduce an action *reject* which is always possible. We omit the formal definitions here for space reasons.

After detailing the preconditions of actions, we now lay out how these actions change the fluents introduced above. The fluents *spoken_verb* and *spoken_objects* contain the essence of the utterance to be interpreted. The effect of the *interpret_action*(k) action is to reset the fluent *spoken_verb* to *nil* and to set the fluent *assumed_action* to the assumed skill k. The action *interpret_object*(e) iteratively removes the first object (in a list of multiple objects) from the fluent *spoken_objects* and adds it to the fluent *assumed_objects* along with its preposition (if available). The action *assign_argument*(p) removes the object from the fluent *assumed_objects* and it adds the pair (p,e) for parameter p and entity e to the fluent *assumed_arguments*. Finally, the fluent *finished* is set to *true* if either the action was *interpret_action* and there are no more objects to process (i.e. *spoken_objects* is empty) or the action was *assign_argument* and there are no more objects to assign (i.e. *assumed_objects* is empty). It is also set to *true* by the action *reject*.

3.2.3 Programs

Using the basic action theory described above, the overall interpretation process can now be realised with READYLOG programs as follows. In case of multiple verb phrases we process each separately. For each verb phrase, we first interpret the verb. Then, we interpret the objects before we assign them to the parameters of the skill determined in the first step. The procedures to do so are

```
proc interpret_verbphrase
    solve( {
        ( pickBest( var, AllActions,
                interpret_action(var) )
        | reject )
        while ¬finished do
            interpret_objectphrase endwhile
    }, horizon, reward_function )
endproc
```

with

```
proc interpret_objectphrase
    (pickBest( var, AllEntities,
                    interpret_object(var) )
    | reject)
    if finished then nil
    else
        (pickBest( var, AllParams,
                    assign_argument(var) )
        | reject)
    endif
endproc
```

where *AllActions*, *AllEntities*, and *AllParams* are sets of all skills of the robot, all entities known to the robot, and all parameters of a skill in the robot's domain specification, respectively. We consider more intelligent selection methods than taking all items available in the evaluation. The *solve*-statement initiates decision-theoretic planning, where **pickBest**$(var, VarSet, prog)$ is a non-deterministic construct that evaluates the program *prog* with every possibility for *var* in *VarSet* using the underlying optimisation theory given mainly by the reward function, which rates the quality of resulting situations. To design an appropriate reward function situations that represent better interpretations need to be given a higher reward than those with not so good interpretation. A possible reward function could be to give a reward of 10 if the assumed action is not *nil* and one could further add the difference between the number of assigned arguments and the total number of parameters required by the selected skill. Doing so results in situations with proper parameter assignment being given higher reward than those with fewer matches. If to possible interpretation have the same reward, one can either enquire with the user on which action to take or simply pick one of them at random.

3.2.4 Example

Consider the exemplary utterance "Move to the kitchen." After syntactical processing we have the internal representation `[and, [[move, [objects,` `[[to, [the,kitchen]] ]]] ] ]`. Using the program given above and a small basic action theory as introduced before, one of the skills available to the robot that has *go* as a synonym may be *goto* which is stored in *assumed_action* by the action *interpret_action*. Then, *interpret_object*(*kitchen*) will assume *kitchen* as the object (along with the preposition *to*). However, it could also interpret "move" as bringing some object somewhere which leads to a lower reward, because a parameter slot remains unassigned. Trying to assign arguments for the skill *goto* may succeed since *kitchen* is an entity that has the *Location* attribute as would naturally be required for the target location parameter of a *goto* skill. Comparing the rewards for the different courses of interpretation the system will pick the interpretation with the highest reward, which is executing the *goto*(*kitchen*) skill.

3.3 Clarification and Response

Things might not always go as smooth as in our example above. To provide a system that has capabilities beyond a pure interface to translate utterances to system calls we therefore include means for clarification if the utterance is missing information.

If the verb is missing, our grammar from the syntactical processing will already fail to capture to utterance. Hence, we only consider missing objects for clarification in the following. We propose to model clarification as an iterative process where the user is enquired for each missing object. To generate the appropriate questions to the user we make use of the information that has been extracted from the utterance already and of the information stored in the ontology. Assuming that we know about the skill that is being addressed we can look up the parameters required. Using a template that repeats the user's request as far as it has been interpreted we can then pose an accurate question and offer possible entities for the missing objects.

Consider that the user said "*Go!*" missing the required target location. So the target location is what we want to enquire about. This can be achieved with using a generic template as follows:

$$\text{“}\textit{you want me to } [\textit{assumed\_action}]\ [\textit{assumed\_arguments}].$$
$$[\textit{preposition}]\ \textit{which}\ [\textit{attribute}]\ ?\ \ [\textit{list of entities}]\text{”}$$

where $[preposition]$ is the preposition associated to the parameter in question and $[attribute]$ is one of the attributes associated to the parameter. Only including one of the parameter's attributes seems incomplete, but suits the application, since it still leads to linguistically flawless responses. Including [*assumed_arguments*] in the response indicates what the system has already managed to interpret and additionally reminds the user of his original request. The system would respond to the utterance "*Go!*" from above with "*You want me to go. To which location? kitchen or bath?*", which is exactly what we want.

To avoid annoying the user we put a limit on the number of entities to propose to the user. If the number of available entities exceeds, say, three we omit it from the question. Moreover, to improve on the response we add what we call "unspecific placeholders" to the domain ontology. So for locations we might add "*somewhere*" and for portable thing we might add "*something*" which are then used in the response at the position of a missing object.

There might be cases where information is not missing but instead is either wrong or the skills available to the robot do not allow for execution. Our system should provide information on rejecting faulty or not executable requests. Depending on the type of error, we propose the following templates for explanation.

1. "*I cannot [spoken_verb].*" if the verb could not be matched with any skill, i.e. *spoken_verb* $\neq$ *nil*.
2. "*I do not know what [next spoken_object] is.*" if the object could not be matched with any entity known to the robot, i.e. *spoken_objects* $\neq$ *nil*.
3. "*I cannot [assumed_action] [preposition] [next assumed_object].*" if the object could not be assigned to a parameter of the skill that is being addressed, i.e. *assumed_objects* $\neq$ *nil*.

Note that *[next some_list]* retrieves the next element from *some_list*. Also note that the fluent values we mentioned above are sound given our basic action theory since the action *reject* sets the fluent *finished* to true and leaves the other fluents' values as they were when the utterance was rejected.

4 EXPERIMENTAL EVALUATION

To investigate the performance of our system we evaluate it along two dimensions, namely understanding and responsiveness.

4.1 Understanding

The aim of our approach was to provide a system that is able to react to as many commands for a domestic service robot given in natural language as possible. With the generic grammar for English directives our approach is able to handle more utterances than previous approaches based on finite state grammars such as (Doostdar et al., 2008). To evaluate how far off we are from an ideal natural language interface we conducted a user survey. The survey was carried out online with a small group of (about 15) predominantly tech-savvy students. A short description of the robot's capabilities was given and participants were asked to provide us with sample requests for our system. Participants took the survey without any assistance, except the task description.

We received a total of 132 submissions. Firstly, we are interested in the general structure of the answers to see whether our grammar is appropriate. Therefore, Table 1 shows the submissions itemised by sentence type.

Table 1: Survey results by sentence type.

type	absolute frequency	relative frequency
imperatives	114	87%
imbedded imperatives	6	5%
need-statements	2	2%
hints	4	3%
wh-questions	3	2%
others	3	2%

Syntactically speaking, the grammar can cover imperatives, imbedded imperatives and need-statements, which make for 92.37% of the survey results. However, some of these utterances do not possess the verb-object-structure we assumed in our system. For example, "Make me a coffee the way I like it" contained an adverbial ("the way I like it") which we did not account for neither in the grammar nor in the interpretation process. It is technically possible to treat adverbials as entities and thus incorporate such utterances. A better founded approach, however, would be to introduce the concept of adverbials to our system as a special case of objects that modify the mode of a skill. We leave this for future work, though. Still, 77.01% of the survey entries provide the assumed modular verb-object-structure and can therefore be processed by our system successfully.

4.2 Responsiveness

To evaluate the performance of our system in terms of speed, we evaluated the system using the following domain. The example agent has four different skills: getting lost (no parameter), going somewhere (1 parameter), moving an object to some location (2 parameters) and moving an object from some location to some location (3 parameters). Additionally, our domain contains different entities with appropriate attributes: a kitchen (location), a bath (location), a coffee cup (portable object) and a football trophy (decoration). Some of the synonyms for skills and entities are ambiguous, namely (1) "go" may refer to "get lost" as well as to "go somewhere", (2) "move" may refer to "get lost", "go somewhere", "move something somewhere" or "move something from somewhere to somewhere", and (3) "cup" may refer to the coffee cup as well as to the football trophy.

We tested four different versions of the system with different requests involving various degrees of complexity using the following utterances:

(i) "scram"
(ii) "go to the kitchen"
(iii) "could you please move the cup to the kitchen"
(iv) "go to the kitchen and move the cup to the bath room"
(v) "i need you to move the cup from the bath room to the kitchen"

Utterance (i) is a very simple request. It addresses a skill with no parameters and the used synonym "scram" is unambiguous. The skill addressed in utterance (ii) involves one parameter and the used synonym "go" is ambiguous. Utterance (iii) involves a skill with two parameters and the synonym "move" is also ambiguous. Utterance (iv) is the combination of utterances (ii) and (iii) linked with an "and". The skill requested in utterance (v) has three parameters and the synonym "move" is again ambiguous.

The depth of the search tree spanned in the planning process depends on the number of objects. For example, the depth of the search tree for utterance (i) is exactly 1 while the depth of the search tree for utterance (v) is 7. Note that utterance (iv) involves two distinct search trees, since it contains two independent verb phrases which are interpreted separately.

The five utterances were tested with the following versions of the system. First, we used the base system as described in Section 3, it does not include any explicit performance improvements speed-wise. The first row of Table 2 shows the performance of the base system.

4.2.1 Improvements

Second, we considered systems incorporating different pre-selection methods. For each interpretation step (interpreting action, entity and parameter), we can pre-select the candidates that may be considered by the appropriate interpretation action. This can lead to considerably lower branching factors.

The pre-selection process for *interpret_action* involves two criteria: synonym and parameter count. This means that candidates are eliminated from the

Table 2: Response times in different test scenarios.

	i	ii	iii	iv	v
base	0.08 s	0.28 s	2.37 s	2.67 s	9.06 s
action pre-select	0.08 s	0.24 s	2.10 s	2.29 s	7.15 s
entity pre-select	0.06 s	0.19 s	2.01 s	2.16 s	7.41 s
parameter pre-select.	0.09 s	0.19 s	1.06 s	1.20 s	4.05 s
action + entity	0.05 s	0.16 s	1.70 s	1.85 s	6.07 s
entity + parameter	0.05 s	0.13 s	0.99 s	1.10 s	3.75 s
action + parameter	0.09 s	0.13 s	0.71 s	0.83 s	2.52 s
full combination	0.07 s	0.10 s	0.68 s	0.76 s	2.35 s

list if the spoken verb is not one of the candidates' synonyms or if the number of parameters the candidate provides is lower than the number of spoken objects. This is due to the fact that we want every spoken object to be assigned to a parameter slot, so we only have to consider skills that provide a sufficient amount of parameter slots. If we would also consider skills with fewer parameters, we would have to drop parts of the user's utterance. One could argue that reducing the set of available skills is a restriction from a theoretical point of view. However, ignoring elements that where uttered could easily frustrate the user. Hence, the restriction only has little practical relevance. The second row of Table 2 illustrates the performance of the base system plus *action pre-selection*.

Entities are pre-selected just by checking whether the spoken object is one of the entity's synonyms. The third row of Table 2 shows the response times including the base system plus *entity pre-selection*.

Pre-selecting parameters involves checking the attributes and the preposition of the corresponding candidate. Hence, the attributes of the parameter slot have to be a subset of the entities attributes, and if a preposition was provided along with the spoken object or entity, respectively, then it has to match the preposition required by the parameter. The fourth row of Table 2 lists response times of the base system plus *parameter pre-selection*. Rows five, six and seven illustrate the performance of different pairs of the three pre-selection methods. The last row shows the performance of the system including all three enhancements. As we can see, the full combination yields an improvement except for utterance i where the difference is negligible. The relative improvement of the enhancements increases with the complexity of the utterances. That is to say, the more complex the utterance, the more the speed-ups pay off.

Altogether, the complexity of the search tree is affected by the different branching factors at each level, and the depth which depends on the number of spoken objects. The branching factor at the first level depends on the number of actions that have the spoken verb as a synonym. The branching factor at the second level depends on the number of entities that have the spoken object as a synonym. At the third level the branching factor depends on the number of parameters of the respective skill. We further evaluated our optimised system by varying the two complexity factors independently.

Along the rows of Table 3 we varied the number of spoken objects. Along the columns we varied the number of actions that have the spoken verb as a synonym and the number of entities that have the spoken object as a synonym. The number of parameters of the appropriate skill are not varied, since this number already depends on the amount of spoken objects. In this test scenario the parameters of a skill became distinguishable for the system by providing distinct prepositions for each parameter. Different entities became distinguishable through their attributes and the skills were distinguishable by the number of parameters. So we had five skills with 1, 2, 3, 4 and 5 parameters, respectively.

Table 3: Response times (in seconds) depending on the two types of difficulty.

# of obj.	tree depth	#actions/#entities 1/1	1/5	5/1	5/5
1	3	0.15	0.32	0.48	1.27
2	5	0.47	0.96	1.61	3.50
3	7	2.54	4.83	7.40	13.92
4	9	18.77	34.00	39.72	68.19
5	11	153.40	267.55	154.97	276.20

Table 3 shows that the number of spoken objects has a greater influence on the computation time than has ambiguity. This is indicated by the last two rows which only contain measurements greater than 10 seconds. That is unacceptable for fluent human-robot interaction. We can also observe that action pre-selection performs very well in this test scenario. All tests in the last row address a skill with five parameters. In this test scenario there was no other skill involving five or more parameters. As a consequence, the action pre-selection can rule out the other four skill candidates which implies nothing less than reducing the branching factor of the top node from 5 to 1 and thus reducing the computation time by a factor of approximately 5. This also results in comparable computation times for the combinations 1/1 (153.40 sec) and 5/1 (154.97 sec) as well as 1/5 (267.55 sec) and 5/5 (276.20 sec).

Finally, we analysed whether the lexicon size poses a computational problem. Therefore, we simply added 50,000 nouns to the lexicon and used the full combination test setup from Table 2. Now, Table 4 indicates that the additional computational effort to process the utterances with a large lexicon plays no significant role.

Table 4: Response times with different lexicons.

	small lexicon	large lexicon
utt. i	0.07 sec	0.08 sec
utt. ii	0.10 sec	0.14 sec
utt. iii	0.68 sec	0.90 sec
utt. iv	0.76 sec	1.15 sec
utt. v	2.35 sec	2.51 sec

4.3 Discussion

An important point towards successful human-robot interaction with respect to the user's patience is the system's reaction time. The average human attention span (for focused attention, i.e. the short-term response to a stimulus) is considered to be approximately eight seconds (Cornish and Dukette, 2009). Therefore, the time we require to process the utterance of a user and react in some way must not exceed 8 seconds. Suitable reactions are the execution of a request, rejection, or to start a clarification process.

Hence, the question whether computation times are reasonable is in fact the question whether the computation times exceed eight seconds. Nonetheless, the answer is not as easy as the question. The optimised system performs well in a realistic test scenario as shown by the last row of Table 2. In turn, complex test scenarios can lead to serious problems as Table 3 indicated. However, we saw that ambiguity is a smaller problem than the length of an utterance[1]. Skills that have more than three parameters are rare in the field of mobile service robots. In fact, the skills with four or five parameters we used in the tests of Table 3 needed to be created artificially in lack of realistic examples.

5 CONCLUSIONS & FUTURE WORK

We presented a system for interpreting commands issued to a domestic service robot using decision-theoretic planning. The proposed system allows for a flexible matching of utterances and robot capabilities and is able to handle faulty or incomplete commands by using clarification. It is also able to provide explanations in case the user's request cannot be executed and is rejected. The system covers a broader set of possible requests than existing systems with small and fixed grammars. Also, it performs fast enough to prevent annoying the user or loosing his or her attention.

Our next step is to deploy the system in a RoboCup@Home competition to test its applicability in a real setup. A possible extension of the approach could be to include a list of the *n* most probable interpretations and to verify with the user on which of these should be executed. Moreover, properly integrating the use of adverbials as qualifiers for nouns both in the grammar and the interpretation process would further improve the system's capabilities.

[1] By the length of an utterance, we mean the number of spoken objects.

REFERENCES

Austin, J. L. (1975). *How to Do Things with Words*. Harvard University Press, 2 edition.

Beetz, M., Arbuckle, T., Belker, T., Cremers, A. B., and Schulz, D. (2001). Integrated plan-based control of autonomous robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. of the 17th Nat'l Conf. on Artificial Intelligence (AAAI-00)*, pages 355–362. AAAI Press/The MIT Press.

Clodic, A., Alami, R., Montreuil, V., Li, S., Wrede, B., and Swadzba, A. (2007). A study of interaction between dialog and decision for human-robot collaborative task achievement. In *Proc. Int'l Symposium on Robot and Human interactive Communication (RO-MAN'07)*, pages 913–918. IEEE.

Cohen, P. R. and Levesque, H. J. (1985). Speech acts and rationality. In *Proc. of the 23rd Annual Meeting on Association for Computational Linguistics*, pages 49–60.

Cornish, D. and Dukette, D. (2009). *The Essential 20: Twenty Components of an Excellent Health Care Team*. RoseDog Books.

Doostdar, M., Schiffer, S., and Lakemeyer, G. (2008). Robust speech recognition for service robotics applications. In *Proc. of the Int'l RoboCup Symposium 2008 (RoboCup 2008)*, pages 1–12. Springer.

Ervin-Tripp, S. (1976). Is Sybil there? The structure of some American English directives. *Language in Society*, 5(01):25–66.

Ferrein, A. and Lakemeyer, G. (2008). Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*, 56(11):980–991. Special Issue on ”Semantic Knowledge in Robotics”.

Fong, T., Thorpe, C., and Baur, C. (2003). Collaboration, dialogue, human-robot interaction. In *Robotics Research*, volume 6 of *Springer Tracts in Advanced Robotics*, pages 255–266. Springer.

Görz, G. and Ludwig, B. (2005). Speech Dialogue Systems - A Pragmatics-Guided Approach to Rational Interaction. *KI–Künstliche Intelligenz*, 10(3):5–10.

Gu, Y. and Soutchanski, M. (2008). Reasoning about large taxonomies of actions. In *Proc. of the 23rd Nat'l Conf. on Artificial Intelligence*, pages 931–937. AAAI Press.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *J Logic Program*, 31(1-3):59–84.

McCarthy, J. (1963). Situations, Actions, and Causal Laws. Technical Report Memo 2, AI Lab, Stanford University, California, USA.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Reiter, R. (2001). *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Scowen, R. (1993). Extended bnf – generic base standards. In *Proc. of the Software Engineering Standards Symposium*, pages 25–34.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, London.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

Wisspeintner, T., van der Zant, T., Iocchi, L., and Schiffer, S. (2009). Robocup@home: Scientific Competition and Benchmarking for Domestic Service Robots. *Interaction Studies. Special Issue on Robots in the Wild*, 10(3):392–426.