Finding Ways to Get the Job Done: An Affordance-Based Approach

Iman Awaad and **Gerhard K. Kraetzschmar**
Bonn-Rhein-Sieg University
and B-IT Center
Grantham-Allee 20
53757 Sankt Augustin, Germany

Joachim Hertzberg
Osnabrück University
and DFKI RIC Osnabrück Branch
Albrechtstrasse 28
49076 Osnabrück, Germany

Abstract

Adapting plans to changes in the environment by finding alternatives and taking advantage of opportunities is a common human behavior. The need for such behavior is often rooted in the uncertainty produced by our incomplete knowledge of the environment. While several existing planning approaches deal with such issues, artificial agents still lack the robustness that humans display in accomplishing their tasks. In this work, we address this brittleness by combining Hierarchical Task Network planning, Description Logics, and the notions of affordances and conceptual similarity. The approach allows a domestic service robot to find ways to get a job done by making substitutions. We show how knowledge is modeled, how the reasoning process is used to create a constrained planning problem, and how the system handles cases where plan generation fails due to missing/unavailable objects. The results of the evaluation for two tasks in a domestic service domain show the viability of the approach in finding and making the appropriate goal transformations.

Introduction

Humans are able to find fixes and adjust their plans, almost effortlessly. In fact, humans *expect* each other to find ways to get the job done, and are often less concerned with *how* others accomplish the task. This requires the ability to apply fixes to failures both during planning and execution, such as by substituting objects. For example, we drink water in a mug instead of a glass, or water the plants with a tea kettle instead of a watering can. Humans find shortcuts and take advantage of opportunities. Enabling such behavior in artificial agents is highly desirable and is the focus of this work.

To address the challenge of finding appropriate, socially-acceptable substitutes, we argue that the functional affordances (Hartson 2003) of objects, what objects are meant to be used for, should play a major role. Affordances are "opportunities for action" (Gibson 1979). We adopt Norman's definition of *perceived affordances* which states that *opportunities for action* are also " *based on the actors's goals, plans, values, beliefs and past experience*" (Norman 2002).

Similarity also plays a role in choosing object substitutes. However, common similarity measures used in robotics may not yield the desirable results, e.g. instead of the color of an object the presence of a handle may be more relevant. For measuring the conceptual similarity between original objects and possible substitutes, we use *Conceptual Spaces* (CS) (Gärdenfors 2004). CS provide a multi-dimensional feature space where each axis represents a quality dimension, for example brightness, intensity, and hue. Points in a conceptual space represent objects, while regions represent concepts. CS can also combine quality dimensions to represent shapes, such as a 'handle'. The importance of particular dimensions for given tasks would provide us with a more robust measure of the suitability of a substitute.

Procedural knowledge is encoded in Hierarchical Task Network (HTN) (Erol, Hendler, and Nau 1994) methods and operators. We model our domain, including the functional affordances of objects, and of parts of objects, in Description Logics (DL). The methods and operators are transformed into OWL-DL as proposed in (Hartanto 2011). This allows us to use more domain knowledge to assert a focused planning problem. A slightly modified version of the JSHOP2 (Ilghami and Nau 2003) planner allows us to extract the reasons why the plan generation process failed (failed preconditions for example, bindings and the task network up to the point of failure). In cases where planning fails due to a missing object, the algorithm reasons about possible substitutes and expands the domain to include them.

Modeling the domain

Two tasks are used to demonstrate how the domain is modeled. The first task involves the robot making a cup of tea and the second involves it watering a plant. The knowledge base (KB) holds all models for the domains. The methods and operators that decompose the tasks are transformed into the OWL 2 DL format (cf. (Hartanto 2011)) and are included in the TBox. The conceptual knowledge of the world is also saved there. The ABox holds the knowledge specifying the state of the world as it changes as well as the constrained planning domain. When a new task is assigned, this information is stored in the KB. When new objects are perceived, or when actions change the state of the world, this too is reflected in the KB.

The KB provides a common knowledge source, or vocabulary, for agents and their components.The concepts of objects and locations which are used by the motion and grasp

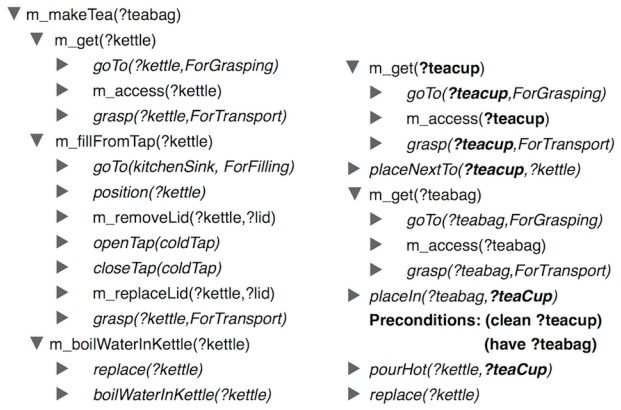

Figure 1: Sample decomposition of the tea-making task

planners originate within this KB, although these planners also use their own KBs (e.g. OCL in the case of the grasp planner). Providing task-relevant information to the motion and grasp planners is also part of the domain model and is covered in the coming sections.

In addition to the KB, a blackboard is used to communicate lower-level information, in particular, it is where affordance cues (Fritz et al. 2006), in the form of CS quality dimensions, are posted as the agent moves through its environment while executing a plan. These CS might be of varying complexity (from simple color hues which would cost very little in terms of perceptual processing to more complex concepts such as shape which might have been picked up as part of the plan's execution) and would be kept in the system for a given duration. Upon plan failure, the cues which are in close proximity can be used to identify viable candidates for substitutions. The same cues allow the agent to take advantage of opportunities as it carries out tasks during execution, what (Levihn et al. 2013) call "serendipity" in a navigation domain, although this is currently future work. This is another motivation for a distinct affordance-based approach. In situations where little is truly within the agent's control, it makes sense to consider what opportunities for action the environment affords, rather than considering those which it may or may not provide. For example, a cupboard full of glasses would guide the agent to grasp any of them.

The Tea-Making Task

A sample decomposition specified by methods and operators in the domain for the tea-making task is given in Figure 1. It calls for a clean teacup to be used. Substituting objects, in the case one is not found, is in essence allowing other object types to bind with the ?`teacup` variable.

The variable types used in the planning domain are also modeled as classes in the ontology. The ontology itself builds on the upper ontology of (Tenorth and Beetz 2009) and as such includes concepts such as `SpatialThing`, `TemporalThing`, `AgentGeneric` and so on. Another part of the ontology models those concepts necessary when converting between OWL and JSHOP syntaxes. The rest of the ontology is modeled based on the dictionary definition of objects. No *ad hoc* categories are created as is often the case.

Dictionary definitions tend to be concise and unambiguous, providing information that would help the perception components to 'search' for the relevant cues and allow them to trigger afforded behaviors when sensed. In addition, they also tend to provide the functional affordances of objects. For example, a teacup, is defined as "a cup from which tea is drunk" (McKean 2005), and a cup is "a small, bowl-shaped container for drinking from, typically having a handle" (McKean 2005). Therefore, dictionaries make ideal sources to guide the modeling of the ontology.

The models for a cup and a teacup, for example, are given here in Manchester OWL Syntax:

```
Class : Cup
SubClassOf : Container
AND (hasObjectToActOn some Liquid)
AND (hasShape some BowlShaped)
AND (isUsedFor some DrinkingFrom)
AND (hasSize only Small)
AND (canHavePart some ObjectPart)

Class : Teacup
EquivalentTo : Cup
AND (hasObjectToActOn some Tea)
AND (isUsedFor some DrinkingFrom)
```

Functional affordances are modeled by the `isUsedFor` property. As a *defined class*, any instance that is a member of the `Cup` class and that is used to drink tea from will be inferred by the reasoner to be a `Teacup`. These are necessary and sufficient conditions that enable the KB to be used for inference and not simply as a database. In addition to these properties, the `Teacup` concept also inherits various other properties from the superclass concepts (including additional `isUsedFor` properties). In this case, it inherits everything from the `Cup` and `Container` superclasses and so inherits everything described above for `Cup`, and two more properties inherited from `Container`: `isUsedFor some : Holding` and `isUsedFor : some : Transporting`.

`Holding`, `Transporting`, and `DrinkingFrom` are all examples of the `ActionOnObject` concept. This is the superclass of all the functional affordances of objects and parts of objects.

The `isAlsoUsedFor` property enables new functional affordances to be learned through experience, for example, by having been successfully substituted for a task. A bottle is defined as "a container, typically made of glass or plastic and with a narrow neck, used for storing drinks or other liquids" (McKean 2005). Should a user drink from the bottle, the property `isAlsoUsedFor some DrinkingFrom` would be added to the `Bottle` concept. This would also provide a quick way to look up which objects have previously been approved as substitutes and thus, allows the transfer and re-use of this knowledge.

We extend the functional view of objects to our actions by modeling the reason why the action is be-

ing performed so that the motion and grasp planners can, for example, provide suitable poses. The operator goTo(?kettle, ForGrasping) shown in Figure 1, for example, communicates the reason why the agent needs to go to the kettle. This information is used by the motion planner to take into consideration the necessary constraints in identifying the final pose to reach. These constraints may also be influenced by the perception components (it needs to be in a position that would allow it to localize the kettle in order to proceed with the grasping action). In addition, a constraint-based system for grasping (Schneider 2013) uses the information to verify that the action can indeed be performed using the specified object and with the specified hardware. For example, the action grasp(?cleanerBottle,ToSpray) would trigger the system to validate that such a task is possible. This is important as faults can occur at any time and it is therefore not sufficient to specify capabilities once. Moreover, these capabilities depend on the agent, the object and the intended use, hence the use of affordances to link them.

The Plant-Watering Domain

This scenario serves to highlight the limitation of using functional affordances to make substitutions. When objects have very specific uses, it is no longer possible to find other objects that share the same affordance in order to make viable substitutions. Surprisingly, the opposite is also true. When many objects share the same functional affordance, they would all be equally likely to be used as substitutes for each other, although there may be some which, in reality, would make better substitutes. For example, a glass, teacup, cup, mug, and bottle may all share the DrinkingFrom concept. However, the best object to use is very often that which is most similar to the original. In both these cases, a means to measure the conceptual similarity between objects would allow the better choice to be made. This example also serves to motivate that in some situations, the plans themselves may need to be adapted, and why goal transformation and replanning alone are insufficient.

The domain specification for the plant-watering task is straightforward. Figure 2 shows a sample task decomposition specifying that a watering can should be fetched, filled, and used to water a plant. For this scenario, let us assume that a watering can is not available and so, to avoid failing to accomplish the task, a substitution is attempted. Querying for another object which isUsedFor Watering Plants yields no instances, and neither does a query for isUsedFor Watering alone.

As mentioned above, the similarity measures which are often used may not yield the results we have in mind, hence our use of CS. Learning the relation between these quality dimensions and given tasks would allow the agent to choose the most appropriate object within the KB. For example, for watering plants, the capacity to hold water is perhaps the most important concept, followed by the presence of a handle and a spout, and in this case, the relevant query may return the tea kettle. Such relations could then be used as weighting factors to determine how well an object would substitute for another in achieving a given task (similarity is measured as the weighted Euclidean distance). The modeling of the objects in CS and learning these weights is actively being investigated.

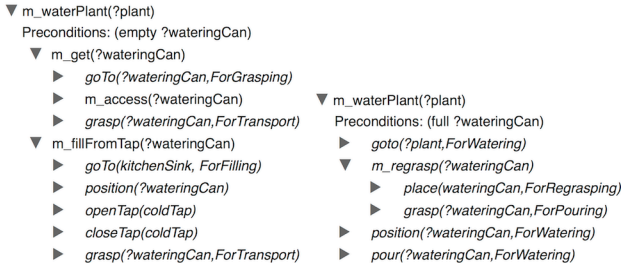

Figure 2: Sample decomposition of the plant-watering task

The need to transform the plans as well can be seen in the case a tea kettle is substituted for the watering can. Two additional actions are necessary to remove the kettle's lid, and replace it during filling (as seen in Figure 1). These are absent from the fill method for the watering can seen in Figure 2. Once again, the solution lies in the modeling of the methods and operators in DL. This is detailed in the section below on expanding the domain.

Generating the planning problem

Having shown how we model our domain, we now show how it allows us to generate a constrained planning problem. By matching a user's goal to a task, we are able to query our KB for all methods and operators which could decompose the given task. This is one of the benefits gained from using a hierarchical approach.

The approach used in (Hartanto 2011) explicitly specifies, for each method and operator, a useState variable which contains a list of states to be included within the initial state. For example, a navigation domain would include only rooms with open doors in the initial state. This additional domain knowledge is what enables the generation of a constrained planning problem.

A plan is then generated, at least, that is the best-case scenario: we have all the information we need to plan. Unfortunately, things can and do often go wrong. The most likely scenario is that a method or operator requires that a precondition be met and the KB lacks the information to determine if it is or isn't. For example, it contains no information on whether or not there exists an instance of a clean cup, even if it knows that all known instances are dirty.

Humans faced with the same situation would still attempt to proceed with their plan and assume that they will adapt as need be. As long as there are instances of the object, we attempt to generate a plan. If the locations of the instances are unknown, this flexibility is achieved by querying a semantic map and creating dummy instances as needed at the most probable location. For example, a dummy instance of clean(cup) may be instantiated in(cupboard4) in the ABox. This allows the planning process to proceed and increases the chances of success. If the knowledge within our KB and semantic map does not favor the odds, then, a search for substitutes is triggered.

Assuming we have sufficient information within our KB

to determine if preconditions are met or not, and even with the help of these dummy instances, the planner may still fail to generate a plan due to a failed precondition. For example, all cups may in fact be dirty. In such a case, humans would consider either washing a cup or making a substitution that would allow them to get the job done. They may use a mug for instance. Here, we assume that if a method that would allow the robot to wash a cup while making tea was desirable to the user, it would have been included in the tea-making method decomposition and `dirty(teacup)` would be the filter condition that would allow that branch to be taken. However, in our domain, washing the cup is undesirable (because our robot simply can't wash the cup and the dishwasher would take too long) and so the agent has no choice but to attempt a substitution.

If we recall how our planning problem was generated, the initial state is constrained and only contains `Teacup` instances, as that is what the methods and operators specify. The domain needs to be expanded to include the next best possible substitute for a teacup. Then, a means to enable the new object to bind to the variables in the existing methods and operators needs to be found. This is necessary to handle cases such as the one described above where filling a kettle to water plants involves additional actions which are missing from the original plant-watering domain. This is detailed in the following section.

Expanding the Domain

The expansion process can be seen as climbing a "flexibility ladder" where constraints are iteratively decreased in a structured way to provide the flexibility with which to choose substitutions. Substitutions are attempted when faced with a situation where the plan generation process has failed. The planner outputs explanations which include a reason for the failure, such as the failed preconditions, the bindings which have been made, as well as the task network up to the point of failure.

The Control module is responsible for providing the guidelines to expand the domain. To do this, it combines functional affordances and conceptual similarity to reason about the appropriate substitutes. If the goal specified the use of a unique instance (e.g. my teacup), the domain is expanded to include other instances of the class which the given object belongs to, for example, instances of a teacup would be included in the new problem's initial state.

If it fails to find such instances and the creation of a dummy variable is not supported by the semantic map (due to low or no probabilities), then the domain is expanded to include instances of objects which satisfy the next set of constraints: objects with the same functional affordance as the original object and high conceptual similarity, for example a mug. This seems to be in line with our own preferences.

The next higher level would remove the constraint that the substitute should be conceptually similar, relying only on a shared functional affordance, for example, a vacuum flask. Should the agent not find such objects and given the old adage that "form follows function" (the form of objects is based on their function), conceptual similarity is then used to identify those objects which do not share the same functional affordance and yet are conceptually similar, for example a measuring cup; or in the case of the watering can, a tea kettle.

As previously mentioned, a direct substitution of objects may not produce a sound plan. Therefore, in addition to the objects being included in the newly expanded domain, the methods and operators which use them are also included. For example, the `m_fillFromTap(?kettle)` method with its additional steps replaces the original `m_fillFromTap(?wateringCan)` method.

To accomplish this, a successful decomposition of the original task is needed. The domain expansion process therefore involves two stages. In the first stage, the failed preconditions/bindings are 'corrected' (e.g. by using a placeholder instance of a clean `teacup`) and sent back to the planner so that it may successfully generate a plan. This process may be repeated until a plan is generated. Once a decomposition is available, method and operators that have the substitute object type as a variable are included in the newly expanded domain description along with instances of the substitute object. The preconditions (and variables) that referred to the original object, e.g. `have(?teacup)`, are replaced with the substitute, e.g. `have(?mug)`, and the planning process is triggered once more to enable the substituted object to be used in the new plan.

Current Status and Future Work

JSHOP2 has been extended to provide the task network, precondition(s) which resulted in the failed planning process and the bindings. In addition, the state transitions for the actions, together with the execution monitoring information allow a temporary component to keep track of the changes in the world. The KB used for the planning process (currently in the form of a domain file in JSHOP2 syntax) is also updated when planned sensing actions are performed. Once this is done, the planner proceeds to generate a plan with this updated domain. Modular SMACH (Field 2011) states (state machine-like execution scripts) for each operator can be called dynamically to execute the plans. Individual monitoring actions have been designed and included in the SMACH scripts to enable the robot to monitor the execution of its actions, thereby providing additional robustness and enabling the system to re-plan upon failure (Shpieva and Awaad 2013).

In our ongoing research, we are continuing with the modeling of the domain knowledge in the OWL-DL ontology which currently includes objects, parts and functional affordances.

Acknowledgments.

Iman Awaad gratefully acknowledges financial support provided by a PhD scholarship from the Graduate Institute of Bonn-Rhein-Sieg University. The authors thank Elizaveta Shpieva and Christian Tiefenau for their help in implementing some of the ideas presented here. The authors also thank the reviewers for their valuable feedback which has helped to improve this manuscript.

References

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1123–1128. AAAI Press.

Field, T. 2011. SMACH documentation. Online at http://www.ros.org/wiki/smach/Documentation.

Fritz, G.; Paletta, L.; Dorffner, G.; Breithaupt, R.; and Rome, E. 2006. Learning predictive features in affordance based robotic perception systems. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 3642–3647.

Gärdenfors, P. 2004. How to Make the Semantic Web More Semantic. In *Proceedings of the Third International Conference (FOIS 2004)*, 17–34.

Gibson, J. J. 1979. *The ecological approach to visual perception*. Houghton Mifflin (Boston).

Hartanto, R., ed. 2011. *A Hybrid Deliberative Layer for Robotic Agents: Fusing DL Reasoning with HTN Planning in Autonomous Robots*. Berlin, Heidelberg: Springer-Verlag.

Hartson, H. R. 2003. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & IT* 22(5):315–338.

Ilghami, O., and Nau, D. S. 2003. A General Approach to Synthesize Problem-Specific Planners. Technical Report CS-TR-4597, UMIACS-TR-2004-40, University of Maryland.

Levihn, M.; Kaelbling, L. P.; Lozano-Perez, T.; and Stilman, M. 2013. Foresight and reconsideration in hierarchical planning and execution. In *IEEE/RSJ - International Conference on Intelligent Robots and Systems (IROS): Workshop on Cognitive Assistive Systems*.

McKean, E., ed. 2005. *The New Oxford American Dictionary*. Oxford University Press.

Norman, D. 2002. *The psychology of everyday things*. Basic Books (New York).

Schneider, S. 2013. Design of a declarative language for task-oriented grasping and tool-use with dextrous robotic hands. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, Grantham Allee 20, 53757 St. Augustin, Germany.

Shpieva, E., and Awaad, I. 2013. Integrating the planning, execution and monitoring systems for a domestic service robot. In *Workshop on Roboterkontrollarchitekturen at Informatik*.

Tenorth, M., and Beetz, M. 2009. KnowRob - knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems (IROS), 2009 IEEE/RSJ International Conference on*, 4261–4266.