Design of a declarative language for task-oriented grasping and tool-use with dextrous robotic hands

Sven Schneider

Publisher: Dean Prof. Dr. Wolfgang Heiden

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

September 2014

Technical Report 01-2014

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

Abstract

Apparently simple manipulation tasks for a human such as transportation or tool use are challenging to replicate in an autonomous service robot. Nevertheless, dextrous manipulation is an important aspect for a robot in many daily tasks. While it is possible to manufacture special-purpose hands for one specific task in industrial settings, a general-purpose service robot in households must have flexible hands which can adapt to many tasks. Intelligently using tools enables the robot to perform tasks more efficiently and even beyond the designed capabilities.

In this work a declarative domain-specific language, called Grasp Domain Definition Language (GDDL), is presented that allows the specification of grasp planning problems independently of a specific grasp planner. This design goal resembles the idea of the Planning Domain Definition Language (PDDL). The specification of GDDL requires a detailed analysis of the research in grasping in order to identify best practices in different domains that contribute to a grasp. These domains describe for instance physical as well as semantic properties of objects and hands. Grasping always has a purpose which is captured in the task domain definition. It enables the robot to grasp an object in a task-dependent manner. Suitable representations in these domains have to be identified and formalized for which a domain-driven software engineering approach is applied. This kind of modeling allows the specification of constraints which guide the composition of domain entity specifications. The domain-driven approach fosters reuse of domain concepts while the constraints enable the validation of models already during design time. A proof of concept implementation of GDDL into the GraspIt! grasp planner is developed.

Preliminary results of this thesis have been published and presented on the IEEE International Conference on Robotics and Automation (ICRA).

In memoriam of my cousins Kim and Nina.

Contents

List of Tables VIII

List of Figures IX

List of Algorithms XI

1. Introduction 1
   - 1.1. Contribution . . . 2
   - 1.2. Outline . . . 2
   - 1.3. Publication . . . 3

2. Problem formulation and task description 4

3. State of the art 6
   - 3.1. Grasp analysis and grasp evaluation . . . 6
   - 3.2. Grasp synthesis . . . 9
     - 3.2.1. Analytical approach . . . 9
     - 3.2.2. Empirical approach . . . 10
   - 3.3. Grasp semantics . . . 10
   - 3.4. Object annotations . . . 13
   - 3.5. Task knowledge . . . 15
   - 3.6. Environment . . . 16
   - 3.7. Grasp planning software . . . 17

4. Approach 18
   - 4.1. Modeling, metamodeling and domain specific languages . . . 18
     - 4.1.1. Model-driven engineering . . . 18
     - 4.1.2. Constraints . . . 21
     - 4.1.3. Domain-specific languages . . . 21
   - 4.2. Tools . . . 22
     - 4.2.1. Eclipse Modeling Framework (EMF) . . . 22
     - 4.2.2. Object Constraint Language (OCL) . . . 23
     - 4.2.3. Xtext . . . 23
   - 4.3. Architecture . . . 24
   - 4.4. Assumptions and limitations . . . 26

5. Domain analysis and domain modeling **27**
5.1. Units domain 27
5.1.1. Units domain analysis 27
5.1.2. Units domain model 27
5.1.3. Units language 29
5.2. Geometry domain 31
5.2.1. Geometry domain analysis 31
5.2.2. Geometry domain model 31
5.2.3. Geometric relations semantics language 32
5.3. Object domain 33
5.3.1. Object domain analysis 33
5.3.2. Object domain model 34
5.3.3. Object domain language 35
5.4. Task domain 36
5.4.1. Task domain analysis 36
5.4.2. Task domain model 40
5.4.3. Task domain language 42
5.5. Hand domain 42
5.5.1. Hand domain analysis 42
5.5.2. Hand domain model 43
5.5.3. Hand domain language 46
5.6. Composition of domain models 48
5.6.1. Composition model 48
5.6.2. Composition language 49

6. GDDL integration with a grasp planner **51**
6.1. Grasp planner evaluation 51
6.2. GDDL interface to grasp planner 52

7. Discussion and evaluation **54**
7.1. Model-driven engineering 54
7.1.1. Domain modeling 54
7.1.2. Domain-specific languages 55
7.2. Constraints 56
7.3. GDDL-grasp planner interface 59
7.4. Run-time architecture 59

8. Conclusion **61**

9. Future research **63**

Bibliography 66

A. URDF model 74

B. Contents of the attached CD 76

C. ICRA workshop paper 77

List of Tables

4.1. Comparison of features in CompleteOCL and OCLinEcore 24

5.1. Different manipulation tasks 37
5.2. Real and virtual regions and their usage semantics 39

6.1. Features of different grasp planners 51

List of Figures

2.1. Overview of the tasks in this thesis (a) and a robot using a spray bottle (b). Based on the spray bottle use case several concepts in the grasping domain can be shown. 5

3.1. The three types of contacts (a)–(c) used in finger-object contact analysis [76]. f_n denotes the contact normal, f_{tx} and f_{ty} are forces that the finger can exert on the object due to friction. The soft-finger contact can also exert a torque m_z on the object. 7
3.2. A hand grasping a mug (right side). The green polygon in the upper left part of the image shows the torque subspace of this grasp while the force subspace is depicted in the lower left part of the image. 8
3.3. Three grasp types: (A) *Power grasp* which involves several finger contact regions and the palm, (B) *precision grasp* with minimum contacts on the finger tips and (C) *lateral grasp* which involves the thumb and the side of the index finger [64]. 11
3.4. The grasp taxonomy by Cutkosky [17]. 12
3.5. The three major opposition types: *Pad opposition* where the finger tips form the opposition, *palm opposition* where some fingers and the palm are in opposition and *side opposition* in which the thumb and the side of a finger are in opposition [50]. 12
3.6. The same hand posture but with different functionalities [74]. 13
3.7. Each hand posture can be composed of several subconfigurations. This grasp consists of a two-finger precision configuration of thumb and index finger as well as a support configuration for the other fingers [73]. 13
3.8. Hand frames (**H**), grasp frames (**G**) and task frames (**T**) in two grasp scenarios. It can be observer that the position and orientation of the hand frames depend on the hand's preshape and the desired task [81]. 14
3.9. A proposal for annotating objects with semantics such as joints and links or forbidden regions [102]. 15

4.1. On the left side the diagram of a metamodel is shown that represents grasp taxonomies. The right side is an excerpt of Cutkosky's grasp taxonomy [17] and represents a model that conforms to the metamodel. Associated elements are highlighted with the equally colored box. Two exemplary constraints on the bottom can validate that the grasp taxonomy's name is set and all categories are unique. 20
4.2. The OMG's four-layer metamodel hierarchy. A domain specific language is defined in the layer **M2**. Layer **M1** comprises the domain models created by the domain experts. [53] 21
4.3. The architecture of a grasp planner which is configured based on GDDL . 25

5.1. The diagram of the units model. 28
5.2. The diagram of the object model. 34
5.3. Frames attached to a kitchen knife (based on [49]). 39
5.4. The diagram of the TFF model 40
5.5. The diagram of the task evaluation model 41
5.6. The diagram of the task model 41
5.7. The diagram of the grasp taxonomy model 44
5.8. The diagram of the hand model 45
5.9. Examples of the major categories of robotic hands: Gripper, dextrous hand and anthropomorphic hand. 45
5.10. The diagram of the model composition 48

6.1. Two approaches to provide the GDDL configuration to the grasp planner . 52

7.1. Excerpt of the generated IDE which shows the auto-completion for GDDL 56
7.2. Atomic constraint to validate a manipulation task 57
7.3. Two composition constraints to validate a joint value 57
7.4. The proposed integration of the GDDL run-time with a task planner. The task plan consists of a sequence of operators (blue boxes). The *grasp* operator is refined. As inputs it receives the object and task identified from the task planner. The feedback is provided as signals. 60

A.1. The diagram of the URDF model. Derived from the URDF XSD schema at [71]. 75

List of Algorithms

1. Forward-kinematics-based simulation grasp planner 9
2. Inverse-kinematics-based grasp planner 10

3. Pseudo-code that returns all grasps which can perform a given task 59

Abbreviations

ASCII	American Standard Code for Information Interchange
CAD	Computer-aided design
CLI	Command-line interface
DOF	Degree of freedom
DSL	Domain-specific language
EMF	Eclipse Modeling Framework
GDDL	Grasp Domain Definition Language
GUI	Graphical user interface
GWS	Grasp wrench space
IDE	Integrated development environment
KDL	Kinematics and Dynamics Library
M2T	Model-to-text
MDE	Model-driven engineering
MOF	Meta-Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
OpenRAVE	Open Robotics Automation Virtual Environment
PDDL	Planning Domain Definition Language
RCA	Robot control architecture
ROS	Robot Operating System
SDH	Schunk Dextrous Hand
SI	The International System of Units
TCP	Tool-center point
TFF	Task frame formalism
TWS	Task wrench space
UML	Unified Modeling Language
URDF	Unified Robot Description Format
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1. Introduction

Grasping and dextrous manipulation are important aspects for a service robot in many daily tasks. While grasping is a common and hence easy task for a human, it is very challenging for a robot. On the one hand grasping can be seen as a search problem in a high dimensional space. For example, a human hand has 21 degrees of freedom (DOF) [59] in the fingers and additionally 6 DOF for the wrist pose (position and orientation). Robotic hands and grippers often are less complex for instance the Schunk Dextrous Hand (SDH) [35] has only three fingers with 7 DOF (and 6 DOF for the wrist pose) but nevertheless have high dimensional search spaces. In contrast to manipulators the search space is non-uniform as it consists of a joint sub-space to specify the finger configuration and a Cartesian sub-space to specify the wrist pose. In addition, for certain tasks it is not sufficient to move the finger tip to a contact point but instead a larger area of a finger should be in contact with an object to increase grasp stability.

On the other hand grasping is a heavily constraint problem. For example, assume a robot is instructed to water a plant with a spray bottle. When grasping the spray bottle for this task constraints arise from the *environment* such as obstacles that can block certain regions around the spray bottle. Also the *object* itself imposes constraints on the grasp. For instance, the spray bottle has a handle that is easier to grasp than the bigger body which might not fit into the hand. Besides the body and the handle the spray bottle consists of a cap and a trigger which are connected by joints. Thus, similar to a robot an object should also be described by its kinematic structure. An aspect that has been disregarded in most research is the *task* that should be performed with the object after grasping. The task of spraying a plant for example can only be accomplished by objects which have a spray property. In addition the task description includes the wrenches (forces and torques) that arise during the task execution. Finally, constraints are imposed on the grasp due to the *hand*. While a dextrous hand can perform the plant watering task, a gripper has to few fingers to hold the bottle and simultaneously activate the trigger. This use case already shows that grasping comprises several domains. In each of these domains there is a huge variability such as small objects like needles and complex functionality like power drills or tasks such as transportation, pouring or tool-use. Similarly, robotic manipulators reach from simple two-finger grippers to complex anthropomorphic hands.

With projects like RoboEarth [99] for world representation or the Semantic Robot Description Language SRDL [54] which attaches semantic information to robot hardware components there has been a recent trend towards formal declarative descriptions of aspects in robotics. In the field of task planning the Planning Domain Definition Language (PDDL) [33] is an approach to specify planning problems in a planner-independent and standardized manner. This master thesis presents a domain-specific language (DSL) called Grasp Domain Definition Language (GDDL) which has a similar objective as PDDL, namely to specify grasp planning problems in a declarative manner and independently of a specific grasp planner. GDDL is based on the previously conducted domain analysis and has the goal to make assumptions explicit. For example, grasping is often seen as an “unparameterized” task in which the following task is implicitly encoded in the manipulation implementation such as transport, hold or place. This manipulation task however should always be provided as a parameter to the grasp task.

1.1. Contribution

The contributions of this master thesis are:

- Identification of the domains in grasping
- Domain analysis of the object, task, hand and composition domain
- Formal models of the analyzed domains in a state-of-the-art modeling framework
- Extension of the models with formal constraints
- Description of GDDL’s formal grammar
- A full-fledged integrated development environment (IDE) for GDDL
- Conceptual integration of GDDL with a grasp planner

1.2. Outline

The master thesis is organized as follows. After this introduction, chapter 2 gives an overview of the identified problems and the task to be solved. Chapter 3 presents some background information and the state of the art in grasp research. Then, chapter 4 outlines the approach taken in this thesis. The domain analysis and domain modeling is discussed in chapter 5. For each domain model examples of the GDDL DSL are provided. Following is the description of a sample integration of GDDL into a grasp planner in chapter 6. Chapter 7 discusses and evaluates the results of the domain models and GDDL as well as the ideas for integrating GDDL into a robot control architecture (RCA). The conclusion in chapter 8 summarizes the thesis while finally chapter 9 outlines possible future research ideas for GDDL.

1.3. Publication

Preliminary results of this thesis have been presented on the Workshop on Combining Task and Motion Planning on the IEEE International Conference on Robotics and Automation (ICRA 2013):

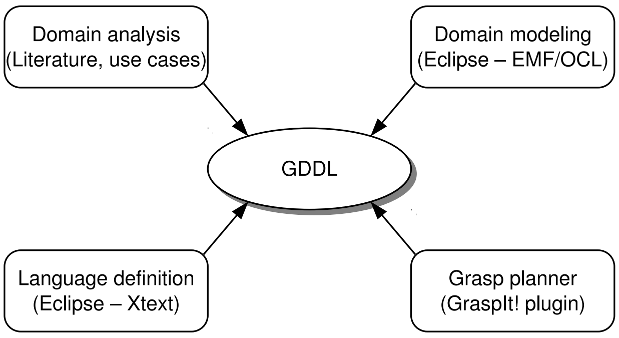
- Sven Schneider, Nico Hochgeschwender. Towards a Declarative Grasp Specification Language. In *IEEE International Conference on Robotics and Automation (ICRA) - Workshop on Combining Task and Motion Planning*, 2013.

2. Problem formulation and task description

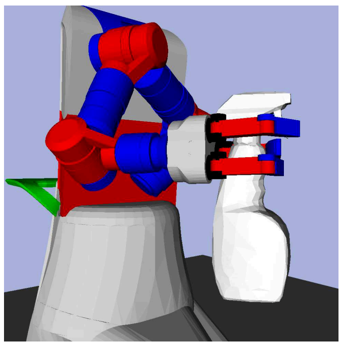
Many current grasp planning approaches apply empirical, machine learning algorithms. While these approaches work well in simple tasks like transporting or pick and place they usually are not aimed at complex tasks such as using objects as tools. Empirical approaches treat grasping as a black box that establishes implicit knowledge and hence does not foster the understanding of grasping itself and its concepts. Additionally, machine learning in grasping is often based on supervised learning which requires huge amounts of labeled grasp data. Labeling grasp data means that lots of grasps are generated. Then a human expert has to check for every grasp if it complies with certain constraints for instance the ones imposed by the task. However, this approach could be simplified if the expert instead described the task once and then the task description is used to automatically evaluate the grasps. Reusing the learned concepts is challenging as well. Therefore, instead of extending the existing knowledge the machine learner is retrained with new data to e.g. add a novel task. The opposite approach of empirical approaches are analytical approaches which allow grasp evaluation based on well-founded mathematical as well as physical models and principles. However, these methods are often neglected in empirical approaches. A systematic analysis of the grasping domain helps to identify relevant concepts which in the future might also enable the better combination of analytical and empirical approaches.

There exists no language which can specify grasp planning problems independently of a grasp planner. In task planning PDDL has been developed for such a unifying planning description. However, the description of the task planning domain is rather simple and inspired by logical systems used in mathematics. The basic domains which have been demonstrated in the spray bottle use case (see introduction for description and figure 2.1(b) for graphic) are much more complex as they require information in different *representations*. In spite of the huge variability in the domains, the entities in a domain share common features. Hence, representations of the different domains should allow a reusable specification of domain entities. When entities, such as the spray bottle or the spray task, are defined this independent information has to be *composed*. The composition should follow common patterns for all domains which allows to assert that certain constraints are met. For instance, the spray task has to specify that it is only compatible with objects that have a "spray" functionality.

Based on the outlined problems the tasks of this thesis are defined. At first, the general field of research in grasping is analyzed in order to identify the major domains and their concepts. The analysis also helps to determine the representations of the concepts which



(a) Requirements for the definition of GDDL



(b) The spray bottle use case

Figure 2.1.: Overview of the tasks in this thesis (a) and a robot using a spray bottle (b). Based on the spray bottle use case several concepts in the grasping domain can be shown.

are then formalized in a model-driven approach. The resulting models are the basis for the domain-specific language GDDL which describes the grasping domain concepts textually and independent of a specific grasp planner. An editor with features such as syntax-highlighting for GDDL should support developers in writing GDDL programs. Finally, GDDL is integrated with an existing grasp planner. Figure 2.1(a) summarizes these requirements.

3. State of the art

This chapter reviews the state of the art in grasp planning and the related domains. At first, the low-level interaction between objects and hands during a grasp is surveyed. The further domains are subdivided as proposed by Cutkosky [17] into hand, object and task. Additionally, a short overview about recent work on environmental constraints is given which has been neglected in Cutkosky's domain decomposition. Finally, some available grasp planning software is reviewed.

3.1. Grasp analysis and grasp evaluation

An important goal of grasping is the immobilization of the grasped object so that it does not move under external forces [88]. A hand achieves this by making contact with the object. Thus, hand-object contacts are a major evaluation criteria in grasping.

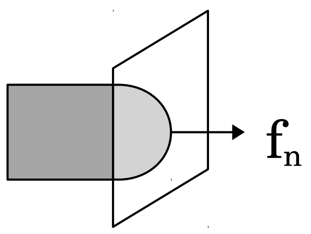
Depending on the surface materials of the hand and the object the contact interaction can be approximated by one of the following contact models: Frictionless point contact, point contact with friction and soft-finger contact [76]. These models are shown if figure 3.1. Each of these contacts constraints the object in a different manner. Frictionless point contacts can only resists motions along the contact's normal, a point contact with friction can additionally exert forces perpendicular to the normal. The soft-finger finally can also exert a torque on the object around the contact normal.

A common approximation of the friction for point contacts is the Coulomb friction model which states that all forces can be exerted through the contact which fulfill the constraint [92]

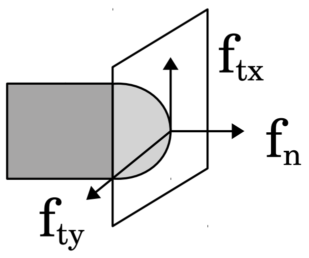
$$\sqrt{f_{tx}^2 + f_{ty}^2} \leq \mu f_N$$

where f_{tx}, f_{ty} and f_N are the force vector f's x, y and z coordinates. μ is the friction coefficient of the materials at the contact point. A graphical representation of this constraint is the *friction cone* which includes all valid friction vectors (see figure 3.1(d)). For soft-finger contacts similar approximation models exist [105][43][56].

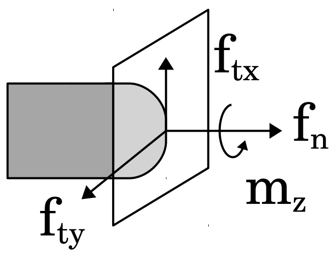
Up to here only one contact has been considered. For a stable grasp several fingers should be in contact with the object each being characterized by the friction constraint represented with respect to the contact's coordinate frame. Ferrari and Canny [29] have determined a unifying description of all wrenches that a given set of contact points can resist, the so called *grasp wrench space* (GWS). It is approximated by mapping each of the contact wrenches to the object's coordinate frame and then calculating the



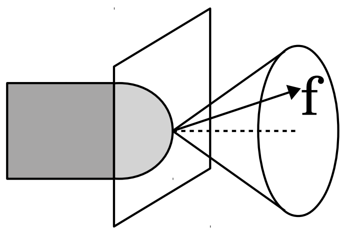
(a) Point contact model without friction



(b) Point contact model with friction



(c) Soft-finger contact model



(d) Friction cone

Figure 3.1.: The three types of contacts (a)–(c) used in finger-object contact analysis [76]. f_n denotes the contact normal, f_{tx} and f_{ty} are forces that the finger can exert on the object due to friction. The soft-finger contact can also exert a torque m_z on the object.

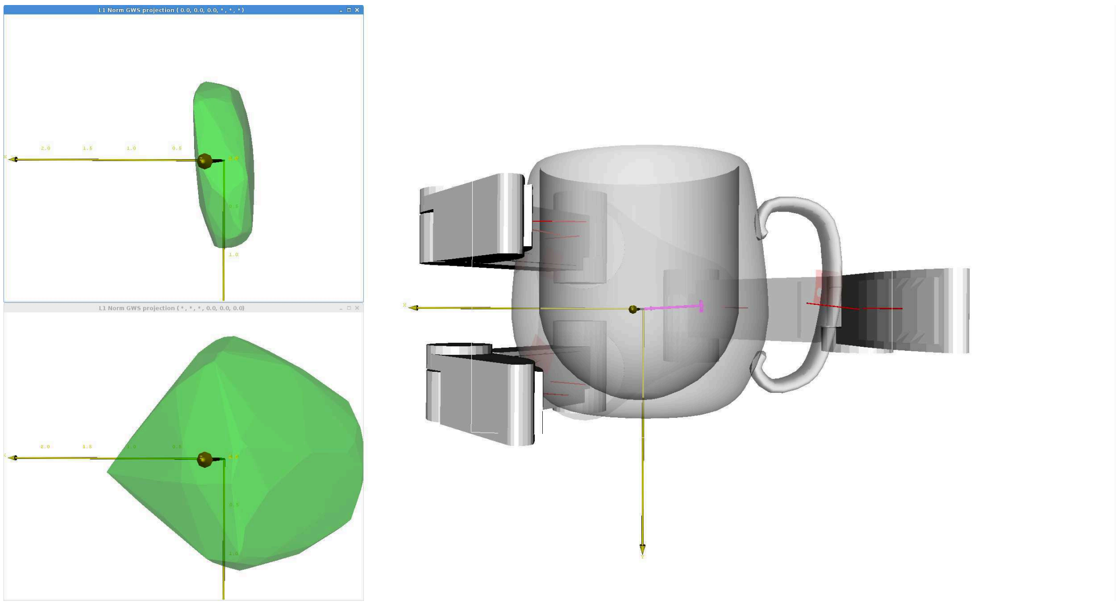


Figure 3.2.: A hand grasping a mug (right side). The green polygon in the upper left part of the image shows the torque subspace of this grasp while the force subspace is depicted in the lower left part of the image.

convex hull of these wrenches. Just like a wrench vector, the grasp wrench space has six dimensions (three force and three torque dimensions). Figure 3.2 shows an example of a grasp with the associated force and torque subspaces of the grasp wrench space. Many grasp evaluation criteria are based on this grasp wrench space. One example is the *force-closure* property which is for example reviewed in [11]. Force-closure means that a hand can increase the normal force vector f_N arbitrarily by squeezing the object so that any external disturbances on the object can be resisted [82]. A grasp is force-closed when the origin is inside of the GWS [69] which is for example the case in figure 3.2. Since force-closure is a Boolean property (a grasp is force-closed or not) it cannot be used to compare different force-closed grasps. Therefore, other evaluation scores have been introduced. Kirkpatrick *et al.* [52] proposed to inscribe a sphere in the GWS where a larger sphere is equivalent to a better grasp.

The previous criteria only analyze a grasp without considering the task requirements especially the wrenches that occur during the task execution. When these wrenches are known a *task wrench space* (TWS) can be constructed analog to the GWS. By comparing the GWS and the TWS scores for the performance of a grasp can be derived [58][15][38]. One problem that can be identified in the task-dependent approaches is the difficulty of modeling the TWS since this requires a wrench-based description of the task.

3.2. Grasp synthesis

The previously described methods lay the mathematical and physical basis for evaluating a given grasp's contact set. However, these methods are non-generative i.e. they cannot create grasps. To this end, grasp synthesis approaches are applied. The survey of Sahbani *et al.* [88] subdivides grasp synthesis into two broad classes: Analytical and empirical approaches.

3.2.1. Analytical approach

Analytical approaches determine grasps from kinematic and dynamic specifications and evaluate them based on one of the criteria outlined above. A further classification of grasp planning approaches in this category is the way in which hand-object contact points are determined. Algorithm 1 shows the *forward-kinematics-based* simulation approach. It simulates grasping by sampling start poses for the hand then approaching the object and finally closing the hand. The resulting contacts are then evaluated. This approach has been used for instance in [10][103][37]. The benefit is that all found contact points are reachable and precision as well as power grasps can be found. The *inverse-kinematics-based* approach (see algorithm 2) in contrast directly samples contact points on the object's surface and evaluates them. Then it applies the hand's inverse kinematics in order to reach the contact points. Depending on the hand and the configuration of the contact points an inverse kinematics solution does not necessarily exist. This approach can only find precision grasps and requires some sort of inverse kinematics for the usually kinematic deficient fingers. However, when contact points have been identified e.g. by a surface analysis this method can better reach these points. This inverse-kinematics-based approach can amongst others be found in [30][61][85].

The analytical approaches have in common that they need a detailed object and hand model. Besides this drawback Sahbani *et al.* [88] have identified the problem of modeling a task and the computational effort required for analytical approaches. However, from a software engineering point of view the different modules can be decoupled well.

Algorithm 1 Forward-kinematics-based simulation grasp planner

```
$grasps \leftarrow \emptyset$
**for** $i = 1 \rightarrow MAX\_ITERATIONS$ **do**
    Initialize hand's pose and finger configuration
    $pose \leftarrow$ Approach object
    $(configuration, contacts) \leftarrow$ Close hand
    $score \leftarrow$ Evaluate$(contacts)$
    $grasps.append(score, pose, configuration)$
**return** $grasps$
```

Algorithm 2 Inverse-kinematics-based grasp planner

```
$grasps \leftarrow \emptyset$
for $i = 1 \rightarrow MAX\_ITERATIONS$ do
    $contacts \leftarrow$ Sample points on object's surface
    $score \leftarrow$ Evaluate($contacts$)
    $(pose, configuration) \leftarrow$ InverseKinematics($grasp$)
    if $configuration$ exists then
        $grasps.append(score, pose, configuration)$
return $grasps$
```

3.2.2. Empirical approach

The empirical approaches focus on human observation and imitation as well as object observation. For generalizing from the observations to new situations machine learning techniques are applied. A common approach is the grasp transfer from known to unknown objects by means of shape matching of the objects' surfaces which is for instance applied in [36] and [57]. After identifying candidate grasps from a database the authors in [57] fall back to analytical methods in order to rank grasps for their task compatibility. Hillenbrand and Roa [40] present a method to transform grasps on objects from the same category which is based on object surface analysis. In contrast to using the complete object as reference the authors in [25] match only parts of an object which allows them to grasp novel objects.

A difficulty for empirical approaches is the inclusion of task knowledge and the generalization of it [88]. Additionally, the machine learning techniques often treat grasping like a black box that does not offer insights into the grasping domain. As empirical approaches are usually composed of tightly coupled components from different fields of research such as machine learning, object detection or object shape analysis the software architecture is less clean. An example of such an architecture can be found in [36].

3.3. Grasp semantics

Analytical grasp evaluation and synthesis require concrete shape and material models of the hand as well as the object to evaluate contacts. To get a better understanding of the grasping domain not only the numerical aspects need to be considered but also the meaning or semantics of grasps. The semantics of grasps capture general patterns and thus allow an abstract and hand-independent grasp description. Additionally, the semantics can be exploited to decrease the search space for grasps.

Analysis of human grasping shows that humans don't use the complete dexterity of their hands but instead rely on a limited set of standard finger configurations. Napier [72] categorizes human grasps into *prehensile* grasps, which enclose the grasped object partly

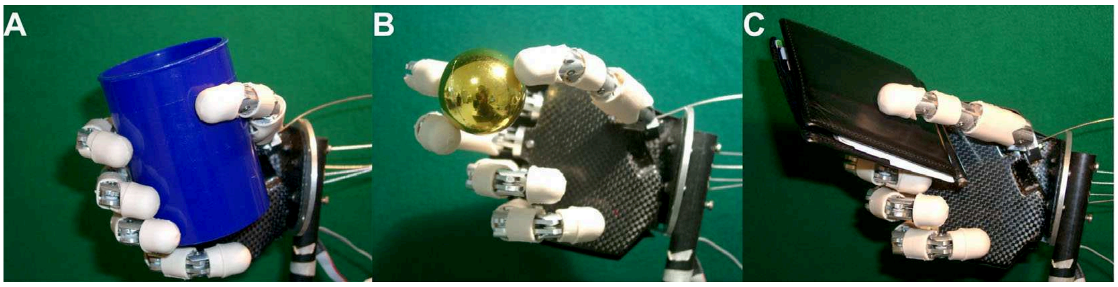


Figure 3.3.: Three grasp types: (A) *Power grasp* which involves several finger contact regions and the palm, (B) *precision grasp* with minimum contacts on the finger tips and (C) *lateral grasp* which involves the thumb and the side of the index finger [64].

or completely, and *non-prehensile* grasps which are for example used to push an object without enclosing it. He further subdivides prehensile grasps into *precision grasps* (point contacts with the finger tips) and *power grasps* (area contacts with fingers and the palm) as shown in figure 3.3. By further refining the subdivision along common grasp patterns a hierarchical representation of grasps can be derived, the *grasp taxonomy* [18][28]. Figure 3.4 depicts the grasp taxonomy by Cutkosky [17].

Another classification scheme for grasps is the opposition space [48]. The opposition space describes which parts of a hand exert forces in a grasp (see figure 3.5). In *pad opposition* the tip of the thumb is in opposition with the tip of the fingers, *palm opposition* means that the fingers or the thumb are in opposition with the palm while *side opposition* denotes that the thumb is in opposition with the side of the fingers. In many grasps several fingers perform the same functionality such as the index, middle and ring finger in figure 3.3 (A). In such a situation the fingers can be combined to a *virtual finger* [47].

Grasp classifications provide an abstract and symbolic representation of grasps which can be mapped to various hand kinematics but they are a static description of the final grasp state only. Thus, they can be used by a human to classify a given grasp but it is difficult to synthesize a grasp from the classification. Therefore, a complementary approach to classifying the grasp is the analysis of the hand's *preshape* as proposed by [101]. The preshape describes the fingers' posture before grasping when the hand is open.

Nguyen and Stephanou [73] have introduced a mapping between symbolic and numeric grasp representations. They distinguish *a*) the hand posture *b*) the hand functionality *c*) the task requirements which they call task functionality and *d*) the resulting grasp called prehensility. The authors have identified two representations of the hand posture. The *topological representation* is the convex combination of four terminal postures while the *geometric representation* is described in terms of the hand's kinematic and the virtual shape formed by the hand in a certain posture. Figure 3.6 depicts the difference between

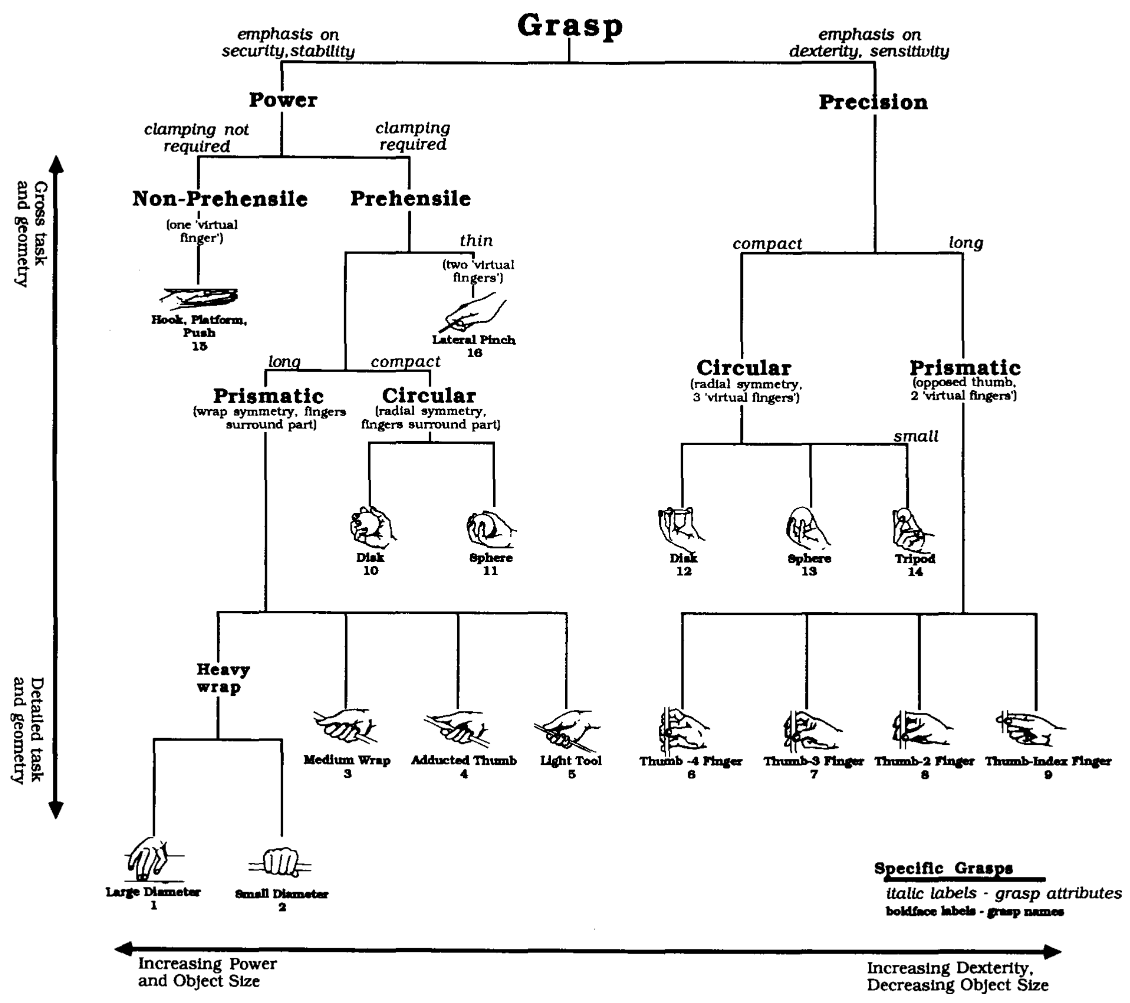


Figure 3.4.: The grasp taxonomy by Cutkosky [17].

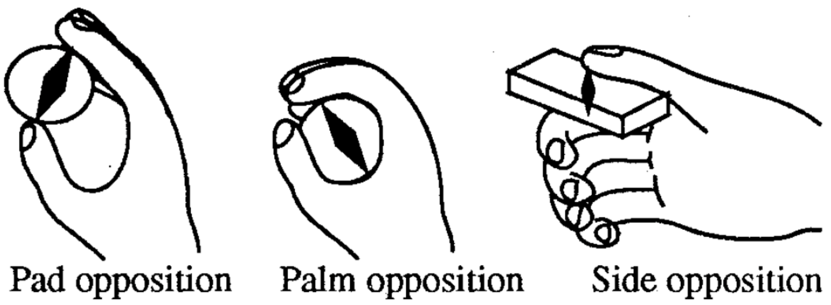


Figure 3.5.: The three major opposition types: *Pad opposition* where the finger tips form the opposition, *palm opposition* where some fingers and the palm are in opposition and *side opposition* in which the thumb and the side of a finger are in opposition [50].

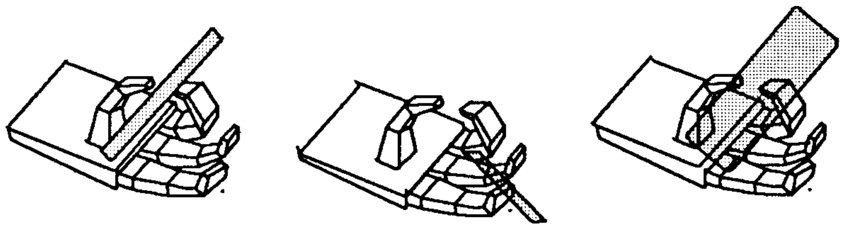


Figure 3.6.: The same hand posture but with different functionalities [74].

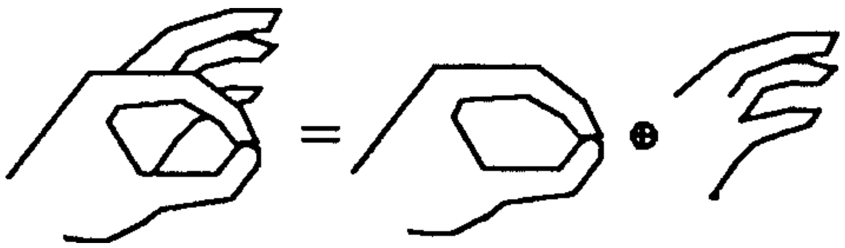


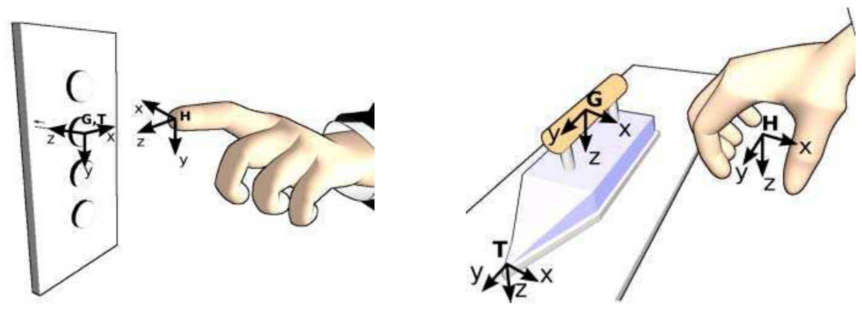
Figure 3.7.: Each hand posture can be composed of several subconfigurations. This grasp consists of a two-finger precision configuration of thumb and index finger as well as a support configuration for the other fingers [73].

the hand posture and the hand functionality. It can be observed that the same posture can result in different contacts and thus different grasps. To further differentiate a grasp the authors have identified the concept of *subconfigurations*. A subconfiguration consists of a subset of fingers involved in a grasp. The subdivision is performed along the task that the fingers perform in the grasp. For instance, the grasp in figure 3.7 consists of a thumb-index finger in opposition subconfiguration while the remaining fingers are in a support subconfiguration.

Prats *et al.* [80][81] have identified the problem that the grasp planner and later on the arm's motion planner require a reference frame to specify the approach towards an object. However, this reference frame depends on the hand's shape. For example, the frame for a one-finger "grasp" to push a button is located at the finger tip while the frame for a cylindrical power grasp is located inside of the cylinder formed by the hand as shown in figure 3.8. Thus, each grasp should be annotated with a hand frame while possible grasp locations on objects are marked by grasp frames. Additionally, many tools require a reference frame, the task frame, to specify the task in.

3.4. Object annotations

A robot can encounter known, familiar or unknown objects. For known objects the robot has a complete description (for a certain domain) available. When an object is unknown



(a) Task frame for a one-finger grasp (b) Task frame for a power grasp

Figure 3.8.: Hand frames (**H**), grasp frames (**G**) and task frames (**T**) in two grasp scenarios. It can be observer that the position and orientation of the hand frames depend on the hand's preshape and the desired task [81].

this knowledge is not available. An object is familiar to the robot when the robot does not have information about this specific object but about the category into which the object can be classified. There has been recent progress in grasping unknown objects based on machine learning techniques [89][13] or heuristics [44] but these approaches usually don't allow to use the object in tasks more complex than pick-and-place or transporting. Thus, these approaches are not considered in this thesis. For familiar objects though there are methods to transfer grasp knowledge from a known object [40]. It needs to be investigated further, if this approach also map more abstract task knowledge between a template and the familiar object.

The work by Xue *et al.* [102] is among the few that annotate objects with knowledge that is required to perform more complex tasks and tool-use. Proposed object semantics are show in figure 3.9 and include adding kinematics (i.e. joints and links) to the object representation and also regions that the hand should not enter in certain tasks. For instance, during pouring from a cup the hand should avoid the area around the spout. However, these proposals have not been implemented. Another paper with a similar scope is presented by Baier and Zhang [6]. The authors evaluate grasps based on how well they can be used in the tasks pour-in, pour-out, handover and movement. The following grasp evaluation criteria are applied: Grasp stability based on the change in the center of mass (pouring), forbidden regions (pouring), free regions (handover) and force closure (movement). Prats *et al.* [80] model the kinematics of doors, drawers and attached handles. From this description they derive how the handle should be grasped and which forces need to be applied. Additionally, the authors simplify the links which compose the overall object. Such an approach to approximating an object by primitive shapes is also found for example in [68] or [46]. The primitive shapes can guide the grasp planner by associating a shape with a certain grasp or by limiting the approach directions

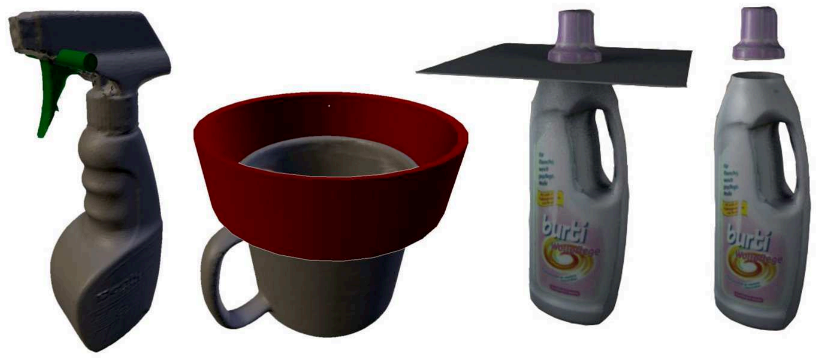


Figure 3.9.: A proposal for annotating objects with semantics such as joints and links or forbidden regions [102].

for grasping.

While the previous specifications represent certain numeric constraints they are not able to capture the functionality of an object. An abstract, symbolic representation of functionality are *affordances*. The term affordance has been coined by Gibson [34] and describes a relationship between an object, an action and a the effects of applying the task to the object. Norman [75] includes the capabilities of an agent, especially perception, in the definition of affordances. In robotics affordances are applied to derive objects' functionality from visual cues (e.g. [39]) but in general they describe action possibilities of objects. An approach to associating an affordance with an action from a repository is described in [96]. An example of some affordance of a spray bottle are: Pushable, gripable, liftable or sprayable. In grasping, affordance are often associated with machine-learning-based approaches [20][7]. The Affordance Network (AfNet) and its extension AfNet for Robotics (AfRob) [98] provides an ontology of common affordances. Besides the definition of these affordances it establishes the mapping between the affordance and the visual cue that helps identifying the affordance in perceived objects.

3.5. Task knowledge

Some task-oriented grasp planning approaches have already been outlined in section 3.1. They are only based on the wrenches that occur during the task execution and thus require the specification of the TWS. Two problems arise from this task-oriented grasp planning approach. On the one hand the predicted wrenches must be acquired and modeled. On the other hand a task has requirements beyond wrenches.

The spray bottle use case contains an example of such an additional requirement in the form of a position-based constraint: No part of the hand should be in front of the nozzle during spraying. Position-based constraints complement the previously explained annotated regions that are attached to an object. The task specification determines how these regions should be used [102][6]. Dang and Allen [19] associate task knowledge with local geometry of an object. Task knowledge includes the avoidance of certain regions or grasping the handle of an object. Due to the task-geometry association they can transfer task knowledge between object of the same class.

Just as the annotated regions are exploited in the task description, so are the attached task frames. A task can be expressed by the relation between frames on the object and frames on the hand. With the attachment of frames to the object Prats *et al.* [80] have introduced Mason's task frame formalism (TFF) [62] into the grasp domain. The TFF is an intuitive way of specifying compliant motions of a robot manipulator [16]. From a task specification in the TFF it is possible to predict the wrenches that can occur during the execution of a manipulation task. Therefore, it is an ideal candidate to bridge the gap between analytical grasp planning approaches and task specifications.

Quite recently, Bohg *et al.* [14] have integrated task knowledge into a grasp planning system using a Bayesian network. The Bayesian network includes the evaluation of grasps with respect to a certain task and allows to generalize the grasps so that they can be used on similar objects. For training, a grasp planner generates a set of stable grasps for a certain object. Then a domain expert labels these grasps if they are applicable for a given task. An additional task-oriented criterion that has been integrated into the Bayesian network is the object's free surface volume in a grasp. A large free surface is required in hand-over tasks so that the opposing person can easily grasp the object. A major deficit of the approach is the huge amount of grasps that must be labeled manually per task, object and hand. For instance, in the presented four tasks with 50 objects approximately 2000 grasps are labeled for one hand. Additionally, the trained tasks are rather simple so that it remains unclear if this approach would be able to use for example a spray bottle.

3.6. Environment

While the previous information can be specified in advance during design time, the environment must usually be evaluated at run-time. Therefore, it is not the focus of this thesis. Nevertheless, for completeness some research is presented here shortly because the environment is also one of the domains that contributes to a grasp in general.

Berenson *et al.* [10] calculate a clearance score for several points on the object's surface. In order to determine the clearance score the authors align a cone with the surface's normal at a certain point. The more obstacles are found within this cone the lower the clearance value. Therefore, an object should be grasped from a direction with high clearance to avoid collisions between the hand and an obstacle.

In [86] the authors present an approach that combines the kinematic reachability of a manipulator with off-line calculated grasps. Their approach is also able to consider obstacles and required free space for a specific task by projecting task trajectories and obstacles into the planar surface of a table. From the projected information they can derive the feasibility of a grasp. When no feasible grasps for a task exist in the current robot's configuration the objects can be moved first an then regrasped. By determining the stable positions in which an object can be placed, possible poses of the object in the environment can be predicted and modeled. This information allows to calculate possible regrasp operations off-line [104].

The previous approaches rely on the avoidance of obstacle. Dogar *et al.* [27] instead investigate how the interaction with obstacles as well as the desired object can be exploited to improve grasps. By simulating the physics of obstacles and graspable objects a robot is able to predict the reaction of objects towards contacts. The goal is to push away all obstacles while simultaneously pushing the desired object into the hand.

3.7. Grasp planning software

Several of the analytical grasp analysis methods have been implemented in free software. The most complete and therefore often used software is GraspIt! [67]. GraspIt! offers a plugin architecture to extend its functionality. More recently grasp planning and analysis software has been integrated into the Open Robotics Automation Virtual Environment (OpenRAVE) [26] and the Simox library [97].

The point-cloud-based grasp planner for unknown objects by Hsiao *et al.* [44] is available as a package for the Robot Operating System (ROS)[1] [83]. It uses heuristics to predict the quality of a grasp. The hand is only modeled as a set of simple boxes.

[1]`http://ros.org/wiki/pr2_gripper_grasp_planner_cluster`

4. Approach

This chapter outlines the approach taken to develop the grasp planning specification language GDDL. The model-driven software development approach as well as the chosen tools for the implementation are explained and how they can be applied to grasp planning. Finally, the chapter describes the coarse architecture as well as assumptions and limitations of the approach.

4.1. Modeling, metamodeling and domain specific languages

Models and ontologies are different views on the same subject of *knowledge representation*. Aßmann *et al.* [5] distinguish ontologies and models based on the concepts description and prescription. An ontology usually describes domains (descriptive) while models prescribe systems (prescriptive). Here *descriptive* means "describe what already exists" [94] and *prescriptive* means "prescribe a system that does not exist, and reality is constructed from it" [94]. In robotics the scope of ontologies is to represent the knowledge for automated *reasoning*. Models, on the contrary, are used to exploit the knowledge for *software development*.

4.1.1. Model-driven engineering

In general, a model is an abstract representation of the real world. In many mechanical and electrical engineering disciplines models are used to describe common patterns in the domain, test ideas in simulation and enable a common vocabulary. Model-driven development (MDD) or model-driven engineering (MDE) are an approach to transfer the established approach of modeling from engineering domains to software engineering. A developer should no longer implement a system in terms of a programming language but instead specify the functionality on a more abstract level. Certain aspects of the system can be generated from the models [4].

The models in MDE capture concepts in the *problem domain* which includes all aspects that are relevant to solving a specific problem i.e. it specifies *what* is required. In contrast the *solution domain* describes *how* a problem should be solved. While the solution domain may change frequently for instance due to new technology the problem domain is rather stable. By differentiating between these two domains it is also possible to foster the separation of concerns as the problem domain represents the knowledge in a field while the solution domain specifies how the knowledge is applied.

The concepts of problem domain and solution domain can be exemplified along the grasp taxonomy in figure 3.4. Here, the problem domain describes the required elements to describe the relation of different grasps. Some concepts are:

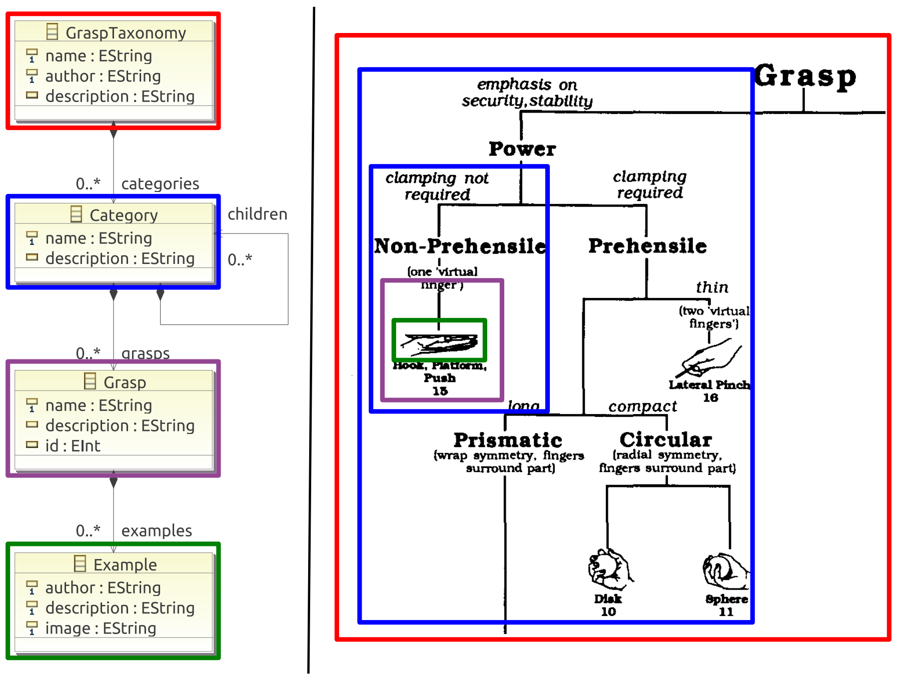
- The grasp taxonomy in general
- The different categories into which a grasp can be sorted
- The grasps themselves
- Examples of grasps

The solution domain depends on the problem that should be solved. One problem could be the representation of a grasp taxonomy in a computer. In this case a mapping from the concepts in the problem domain to appropriate data structures in a programming language must be determined. For each programming language these data structures may look differently while the general concepts remain.

The Object Management Group's (OMG) Meta-Object Facility (MOF) [77] is a proposal to formally describe models in MDE. Originally, MOF was used to describe the Unified Modeling Language (UML) but it can also define other models and languages. MOF is defined by a subset of UML's class diagrams and conforms to itself i.e. it is defined using its own elements. MOF has the deficit that it cannot capture behavior but only structure. Although operations can be defined by their name and signature they must be implemented in a general purpose programming language. The OMG proposes a four-layered approach to define models as shown in figure 4.2. These layers are called **M0**–**M3** where the MOF as meta-metamodel is situated on layer **M3**. The definition of for example UML conforms to the MOF and would be located on layer **M2**. It is called metamodel. A user-defined UML model would be situated on layer **M1**. The implementation resides on layer **M0**.

Figure 4.1 shows how a custom metamodel can be defined in MOF. The left side of the figure shows a diagram of a metamodel that describes grasp taxonomies. On the right side of the figure an excerpt of Cutkosky's grasp taxonomy [17] depicts an example of a model that conforms to the grasp taxonomy metamodel. In the OMG's model hierarchy this is represented as follows:

- **M3**: Ecore meta-metamodel
- **M2**: The grasp taxonomy metamodel
- **M1**: Cutkosky's grasp taxonomy model
- **M0**: Implementation in a specific programming language



context GraspTaxonomy
inv: **self**.name.size() <> 0
inv: **self**.categories->isUnique(name)

Figure 4.1.: On the left side the diagram of a metamodel is shown that represents grasp taxonomies. The right side is an excerpt of Cutkosky's grasp taxonomy [17] and represents a model that conforms to the metamodel. Associated elements are highlighted with the equally colored box. Two exemplary constraints on the bottom can validate that the grasp taxonomy's name is set and all categories are unique.

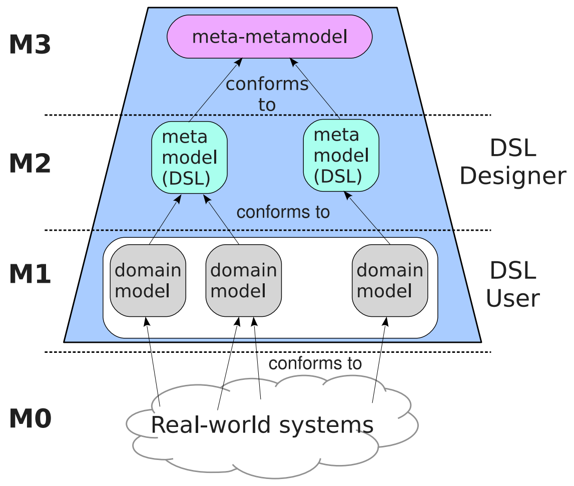


Figure 4.2.: The OMG's four-layer metamodel hierarchy. A domain specific language is defined in the layer **M2**. Layer **M1** comprises the domain models created by the domain experts. [53]

However, the grasp attributes of Cutkosky's taxonomy could also be directly encoded in MOF e.g. in an enumeration. In this case Ecore would act as a metamodel on layer **M2**. Thus, the layers should preferably only be treated as relative descriptions. The benefit of specifying a custom **M2** metamodel is that it can capture patterns in a domain in general and therefore is more flexible. For example, the grasp taxonomy metamodel above also describes the grasp taxonomy by Feix [28]. A drawback of this approach is that constraints can no longer be specified for a specific model but only for the metamodel.

4.1.2. Constraints

With the Object Constraint Language (OCL) [78] the OMG has defined a declarative language to extend models with formal constraints. OCL constraints are applied to instances of model elements i.e. objects. Figure 4.1 contains two invariant constraints to validate some aspects of the model. Further constraints are pre- or postconditions of operations as well as bodies of operation (implementation). However, the implementation of operations is limited as OCL is only a declarative language. Constraints can navigate along class associations and can easily handle different types of collections such as sets or sequences. This makes OCL a precise and powerful formalism.

4.1.3. Domain-specific languages

While the metamodel defines the concepts and semantics of a domain an additional approach is required to populate the model with individual objects that conform to the metamodel. Depending on the domain, different representations to populate the model are desirable. For instance, a graphical representation such as the diagram of a grasp taxonomy (see figure 3.4 may be easier to use and understand for new users while a textual representation can help to standardize the domain vocabulary. The textual representations are called *domain-specific languages* (DSL). In contrast to general purpose programming language like Java or C++ a DSL is a lightweight programming language that only focuses on the description of concepts in the problem domain. When a DSL is built on top of an existing general purpose language it is called an internal DSL. An external DSL in contrast uses its own syntax and is developed from scratch [32].

4.2. Tools

4.2.1. Eclipse Modeling Framework (EMF)

The Eclipse Modeling Framework[1] (EMF) [95] is a state-of-the art framework for MDE that implements many of the previously outlined proposals by the OMG. It is part of the Eclipse[2] project and thus free and open software that is constantly being developed further. EMF's meta-metamodel is called *Ecore* and aligned with MOF. Using a model-to-text (M2T) transformation different types of textual model representations can be generated e.g. Java code or Extensible Markup Language (XML) documents. The EMF4CPP [91] project implements several EMF components, including code generation, in and for C++. However, it is not actively maintained. EMF features several powerful modeling and development tools such as textual and graphical editors or "execution environments" in which for instance OCL constraints can be tested. With the XML Schema Definition (XSD) there exists a metamodel for XML documents. Such XSD metamodels can be converted to Ecore metamodels automatically. Therefore, from the programmer's point of view models an XML document can be accessed just as models which are instances of the Ecore metamodel.

These features oppose the deficits that can be found in several EMF tools. Selecting EMF as modeling framework results in a lock-in to the Eclipse universe because many tools are tailored to be used in Eclipse. Additionally, official documentation and tutorials are limited so that one has to gather information from several EMF-unrelated wikis. Certain features are unstable due to containing software bugs, especially when they are used in more complex situations. This applies in particular to the OCL and DSL tools. Other drawbacks arise when the models and DSLs are integrated into an existing project that does not rely on Eclipse or Java tools. For instance, the Eclipse build system does not (easily) support out-of-source builds[3] which pollutes the source code directory. Due

[1]http://www.eclipse.org/modeling/emf/
[2]http://www.eclipse.org/
[3]Out-of-source build means that the source code is compiled and linked in a different file system

to this, version control of EMF projects is challenging. As the build system is closely tied to Eclipse it is also not (easily) possible to replace it or trigger it from other build systems. Finally, certain (configuration) files contain generated content but must also be adapted manually which can lead to software bugs during development or run-time.

4.2.2. Object Constraint Language (OCL)

Eclipse Ecore models can be validated by OCL constraints. To this end, Eclipse offers to ways of specifying the constraints. With *CompleteOCL* an additional file contains contains the OCL statements. During run-time this file is loaded and applied to the model. *OCLinEcore* on the contrary embeds the constraints directly into the model as annotations. Table 4.1 compares the features of CompleteOCL and OCLinEcore. The benefits of the former are a better separation of concerns by separating the structure (i.e. the model) from the constraints. Syntax-wise CompleteOCL is closer to the OMG's OCL specification although some extensions have been implemented in CompleteOCL's grammar. One such extension is the support for helpful, human-readable error message that can not only show that an error occurred but also what the problem is. Although this feature can be activated in OCLinEcore it requires changing generated code. CompleteOCL specifications are interpreted by the application when the constraints are loaded. Thus, the usability is better in comparison to OCLinEcore where code must be re-generated every time a constraint is changed.

Benefits of OCLinEcore are the integration into existing code because constraint checking is activated by default whereas the CompleteOCL interpreter must be activated manually. OCLinEcore also allows a better reuse of implemented operations as these operations are directly included in the model. CompleteOCL files cannot include other OCL files. When an operation is implemented in CompleteOCL this implementation must be copied to other files that use the operation.

Both OCL implementations contain software bugs. While OCLinEcore does not support all Ecore features, especially generics in operation parameters, CompleteOCL fails to validate complex models for instance models that contain multiple packages. Due to this latter problem the OCLinEcore approach has been selected to implement OCL constraints.

4.2.3. Xtext

Xtext[4] is an Eclipse tool to define DSLs. These DSLs can then populate a domain model. The domain model can either be derived from the language's grammar or an existing manually created model. Xtext uses a notation similar to the Extended Backus-Naur Form (EBNF) [2] which is a metasyntax to describe the grammar of a formal

directory than the source code directory.

[4] `http://www.eclipse.org/Xtext/`

Table 4.1.: Comparison of features in CompleteOCL and OCLinEcore

Feature	**CompleteOCL**	**OCLinEcore**
Separation of concerns	$\oplus$	$\ominus$
Standardized OCL syntax	$\oplus$	$\ominus$
Nice error message	$\oplus$	$\ominus$
M2T after model changed	✗	✓
Ease of integration	$\odot$	$\oplus$
Reusability	$\ominus$	$\oplus$
Quality	$\odot$	$\odot$
Supports complex models	$\ominus$	$\odot$

language. Based on the grammar a full-fledged integrated development environment (IDE) is generated in ordered to help DSL users to write their programs. The IDE is based on Eclipse and supports features such as syntax highlighting, auto-completion or refactoring assistance. When model constraints are violated the IDE highlights the corresponding elements and can show an easily human-readable error message, in case one has been defined. A further helpful feature is that models described in other well-formed formats can be referenced from the generated language. One example of such a well-formed model is an XML document for which an XSD schema is available. The code generation, i.e. the M2T transformation, can be triggered from the graphical user interface (GUI) of the IDE but with few extension a command-line interface (CLI) can be developed as well.

4.3. Architecture

The Grasp Domain Definition Language (GDDL) is a DSL implemented in Xtext. A requirement for the development is the analysis of the grasping domain which is formalized in domain models. Figure 4.3 depicts the static architecture of GDDL and the interaction with an instance of a sample grasp planner. A GDDL program consists of different models that represent the *grasping-related* domains and models that describe the *composition* of these domains. The former models are:

- The *object* model that describes an object by its physical and semantic properties such as its surface but also the anticipated functionality.
- The *task* model defines how a task will be performed by an object after it has been grasped. This includes the task type and expected motions during the task execution.
- The *hand* model which represents a specific hand and its capabilities. It provides a

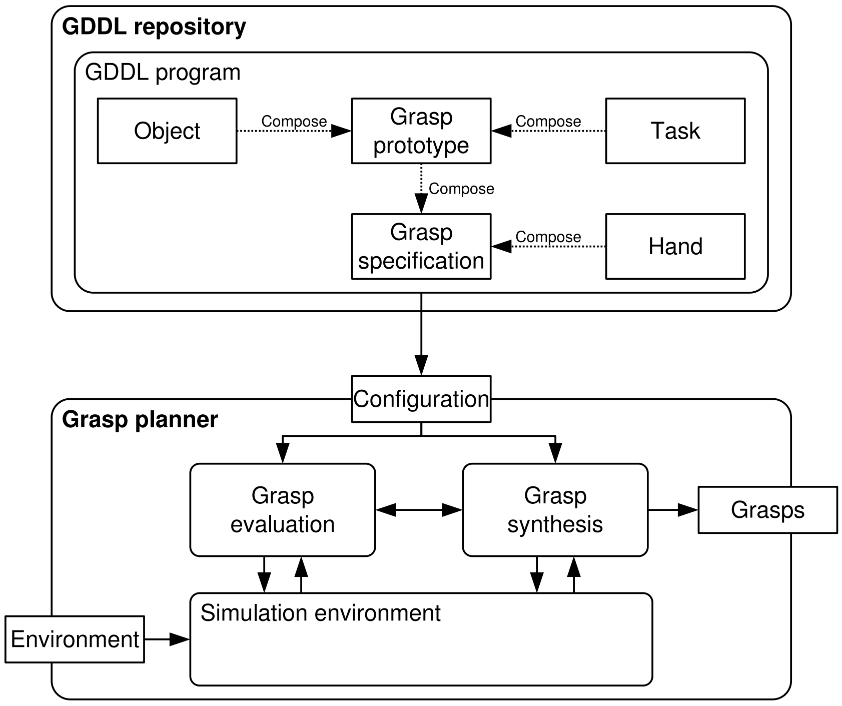


Figure 4.3.: The architecture of a grasp planner which is configured based on GDDL

mapping from abstract grasp semantics to the grasp realization with the concrete hand.

The latter models are mainly:

- The *grasp prototype* which composes the object model and the task model. Thus, it is a hand-independent representation of a manipulation task with a certain model.
- The *grasp specification* composes the grasp prototype with a concrete hand and therefore comprises all information required for grasp planning and evaluation.

Such a GDDL program is passed to a grasp planner as configuration. Based on this configuration the grasp planner synthesizes and evaluates grasps. Optionally, the environment can be considered in this process. Proper grasps are returned and can for example be passed on to a reactive grasp controller for execution. As GDDL and the grasp planner usually assume a nominal environment the controller should be able to handle certain imprecisions or errors.

4.4. Assumptions and limitations

The implemented approach on identifying and formalizing domain concepts is based on a forward kinematics simulation grasp planner. Nevertheless, many of the concepts are reusable in other grasp planners. The reason for this selection is simplicity of the grasp planner as it does not require the hand's inverse kinematics. Additionally, this type of grasp planners can generate power grasps as well as precision grasps. The goal is to analyze the nominal behavior and aspects of grasping. Thus, error handling or imprecisions that occur during run-time such as noise in the object localization or uncertainty in the shape of an object are currently not included into GDDL. The objects that should be grasped are assumed to be rigid bodies whose shape and surface are known and described as three-dimensional computer-aided design (CAD) mesh.

5. Domain analysis and domain modeling

This chapter analyses the domains which are relevant for grasping. In the MOF hierarchy these domains correspond to the **M1** and **M2** layers. The first two analyzed domains, the *units* domain and the *geometry* domain, are generally usable and therefore implemented as standalone models. In contrast the *object*, *task* and *hand* domains are specifically targeted at grasping, as is the *composition* domain which composes the previous three domains and attaches further information. The grasping-related domains are integrated in one model. It is important to note that the grasping domain models only represent the view on the aspects that are relevant for grasping. In general, those domains are much wider. For instance, a general task may additionally include perception or navigation aspects.

5.1. Units domain

5.1.1. Units domain analysis

A common issue when working on software in robotics is that the meaning of numbers in code or configuration files is unclear. For example when the kinematics of a robot are specified in GraspIt! by convention length are provided in millimeters. By contrast, in URDF and most of the ROS universe the convention is to specify lengths in meters. These conventions are either explained in the documentation or in the source code that interprets this data. Thus, in GDDL all physical quantities should carry information about their units. This approach is inspired by the Boost.Units library [90] that extends `C++` in order to perform checks on physical quantities. Similar projects that provide ontologies of units are QUDT [42] and UnitDim [84].

5.1.2. Units domain model

The goal of the units domain in GDDL is to model quantities in systems such as the International System of Units (SI) [1]. Figure 5.1 depicts a diagram of the units domain model which is based on the well-known concept of dimensional analysis (see e.g. [8]). In dimensional analysis a unit is represented as a vector in a n-dimensional space. The International System of Units is based on seven dimensions:

- Length
- Mass

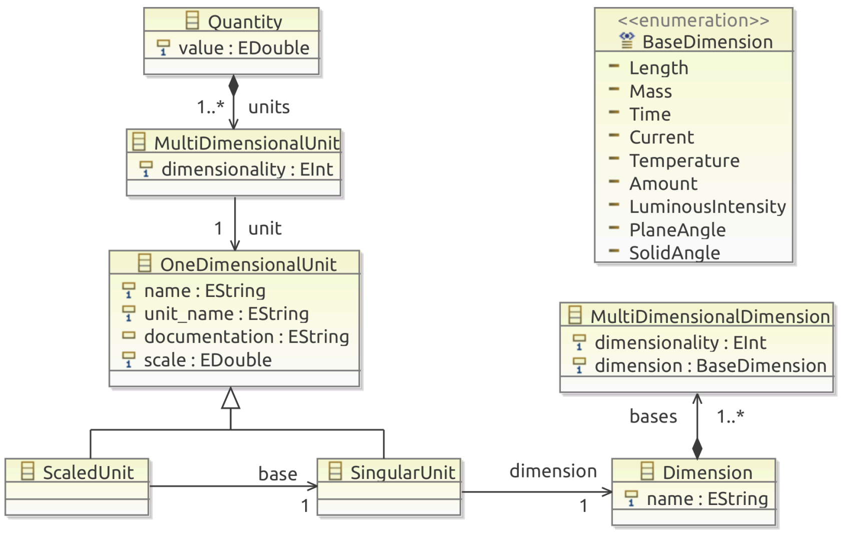


Figure 5.1.: The diagram of the units model.

- Time
- Electric current
- Temperature
- Amount of substance
- Luminous intensity

The Boost.Units library adds planar angles and solid angles in order to describe angles. These dimensions can be found in the *BaseDimensions* enumeration in figure 5.1. Take a force as an example. In the diagram a force would be encoded in the *Dimension* class. A force is defined as mass times length per squared time $\left(\frac{ML}{T^2}\right)$ i.e. it consists of three components, the mass, distance and time dimension. Each of these dimensions conforms to the *MultiDimensionalDimension* class in the diagram. While the mass and the distance component have a dimensionality of 1 the time component has a dimensionality of -2.

Let a force be specified as

$$1\,\mathrm{N} = 1\,\frac{\mathrm{kg{\cdot}m}}{\mathrm{s}^2} = 1\,\mathrm{kg{\cdot}m{\cdot}s}^{-2}$$

Such a specification conforms to the *Quantity* class which is composed of a value and three unit components i.e. kg^1, m^1 and s^{-2}. Each unit component is represented by an instance of the *MultiDimensionalUnit* class. The kg and the m component have a dimensionality of 1 and the s component has a dimensionality of -2. The unit component without the dimensionality e.g. m or kg is called a *OneDimensionalUnit* consisting of a symbol name like "m, the unit name like "meter" and a scale. An optional documentation can be added to a one dimensional unit. For *SingularUnits* such as "meter" the scale defaults to 1 while *ScaledUnits* should specify a different scaling factor. For instance, "kilogram" has a scale of 1000 with respect to the base unit "gram".

5.1.3. Units language

While Boost.Units is a helpful library in the context of `C++` it does not follow the SI notation. A major deficit is that it does not support the composition of units. For instance, to define a velocity of $1\,\frac{\mathrm{m}}{\mathrm{s}}$ or $1\,\mathrm{m{\cdot}s^{-1}}$ one has to write 1 *meter_per_second* where *meter_per_second* is a previously defined type. GDDL's units language however does not have to rely on `C++` features that lead to these limitations.

The SI notation is ambiguous and thus cannot be parsed completely. An example is mN which could either mean meter times newton or millinewton [31]. This has to be considered when designing the units language and leads to a two-step approach. First, the user has to define or include only the required dimensions (see listing 5.1 for an example) and units (see listing 5.2 for an example). Then, these units can be included and used in other models. By only defining a subset of all possible units as well as not supporting the slash (/) as division symbol most ambiguities can be avoided. Multi-dimensional units are separated by a dot (.) similar to the proposal in [31]. Listing 5.3 shows some valid quantities for the dimensions and units defined in the previous listings.

```
package de.hbrs.units.my_dimensions {
  length: Dimension {
    Length: 1
  }

  mass: Dimension {
    Mass: 1
  }

  time: Dimension {
    Time: 1
  }

  force: Dimension {
    Mass: 1
    Length: 1
    Time: -2
  }
}
```

Listing 5.1: Definition of the base dimensions length, mass and time as well as the composed force dimension.

```
import de.hbrs.units.my_dimensions.*

```

```
package de.hbrs.units.my_units {
  m: Unit {
    name: meter
    dimension: length
    documentation: "The␣distance␣travelled␣by␣light␣in␣vacuum␣in␣1/299,792,458␣
        second"
  }

  g: Unit {
    name: gram
    dimension: mass
    documentation: "One␣one-thousandth␣of␣the␣kilogram"
  }

  kg: ScaledUnit {
    name: kilogram
    base: g
    scale: 1000.0
    documentation: "The␣mass␣of␣the␣International␣Prototype␣Kilogram"
  }

  s: Unit {
    name: second
    dimension: time
    documentation: "The␣duration␣of␣9192631770␣periods␣of␣the␣radiation␣
        corresponding␣to␣the␣transition␣between␣the␣two␣hyperfine␣levels␣of␣the␣
        ground␣state␣of␣the␣caesium␣133␣atom"
  }

  N: Unit {
    name: newton
    dimension: force
  }
}
```

Listing 5.2: Definition of the units meter, gram, kilogram and second along the base dimensions. Newton is defined as a unit along the force dimension.

```
import de.hbrs.units.my_units.*

1 m
1 kg
1 s^-1
1 m.s^-1
1 N
1 N.m^-1.m
```

Listing 5.3: Some examples of valid quantity definitions.

5.2. Geometry domain

5.2.1. Geometry domain analysis

In many domains geometric relations such as positions and orientations or forces and torques between rigid bodies must be specified. Only recently, the semantics of commonly used relations have been described and implemented as a `C++` software library [22][21]. By requiring the developer to explicitly specify the otherwise hidden assumptions the semantics of operations on the relations can be checked which helps avoiding common mistakes. This geometric relations semantics library extends existing third-party geometric libraries such as the Kinematics and Dynamics Library [93] (KDL) so that their numerical algorithms are reused. During development and writing of the thesis only the `C++` library implementation is freely available but no formal model is offered. Therefore, a subset of the library has been modeled in Ecore. The following relations are required by GDDL:

- Position, orientation and pose (combination of position and orientation)
- Force, torque, wrench (combination of force and torque)

There are different methods to acquire the pose of a rigid body. For instance, the kinematic model of a robot can specify fixed poses based on a tool's mounting like the pose between the robot's end-effector and the hand. Another source of poses is the dynamically calculated pose of the end-effector with respect to the manipulator's base using the forward kinematics. Finally, also the object localization returns a pose of an object with respect to the camera.

5.2.2. Geometry domain model

The geometric relations semantics domain model consists of three major packages.

- *Geometric primitives* label rigid bodies, points, orientation frames and displacement frames.
- The primitives define the semantics of the *geometric relations*. The relations also implement the constraint checking on relations' operations like for example the composition of poses and orientations or the inverse of a pose. However, in this thesis the operations are not required and therefore have not been implemented.
- The *coordinate representation* describes the numerical data of a geometric relation. One geometric relation can have multiple representations. For example, a rotation can be represented as a rotation matrix, Euler angles, quaternions and so on. In contrast to the `C++` library the Ecore model that was created in this thesis is extended by previously presented units model. This enables the constraint checking

on individual elements. So, a position vector only accepts length quantities but can handle different scaling like meters or millimeters.

It should be noted that there is a clear separation between symbolic labels, semantics and numerical representations. For instance, a frame is only a label while its semantics is specified by a pose. The numerical information is then encoded in the representation like a homogeneous transformation matrix.

5.2.3. Geometric relations semantics language

The authors propose a notation for each geometric relation. From this notation an Xtext grammar has been derived and implemented in order to integrate the geometric relations semantics into GDDL. In the original paper [22] the authors only specify the notation of primitives using short, one-letter names. However, in practice it is more convenient to use more meaningful names consisting of more than one letter. For these cases we propose the following naming convention for geometric primitives that only use American Standard Code for Information Interchange (ASCII) characters.

- Point names consist of lower-case letters (a–z) or an underscore (_) such as
  - $bottle\_body\_origin$
- Orientation frame names consist of lower-case letters (a–z) or an underscore (_). They are delimited by square brackets like in
  - $[bottle\_body\_orientation]$
- Displacement frame names consist of lower-case letters (a–z) or an underscore (_). They are delimited by braces. For example
  - $\{bottle\_body\_frame\}$
- Rigid bodies are denoted by camel-case names consisting of upper-case and lower-case letters (a–zA–Z) such as
  - $SprayBottle$
- To attach a primitive to a body the vertical bar (|) is used as has been proposed in the original paper [22]. For instance
  - $bottle\_body\_origin|BottleBody$
  - $[bottle\_body\_orientation]|BottleBody$
  - $\{bottle\_body\_frame\}|BottleBody$

The example in listing 5.4 shows how geometric relations describe the spray bottle by attaching frames and bodies to it. The declaration of the two geometric primitives point and orientation frame is used to define a frame. In front of the nozzle a virtual body, called spray cone, marks the region where the liquid will be distributed while spraying. Here, the shape of this cone and also the shape of the bottle's body are irrelevant and thus they are only declared as a label. Finally, the listing defines the fixed pose of the spray cone with respect to the base frame.

```
bottle_body_origin: Point
[bottle_body_orientation]: OrientationFrame
{bottle_base}: Frame(bottle_body_origin, [bottle_body_orientation])
BottleBaseBody: Body

// ...

bottle_spray_cone_pose: Pose(
  {bottle_spray_cone}|BottleSprayConeBody,
  {bottle_base}|BottleBaseBody,
  [bottle_body_orientation],
  PositionVector(15 cm, 0 m, 30 cm),
  RollPitchYawAngle(0 rad, 1.57 rad, 0 rad)
)
```

Listing 5.4: Example of specifying the pose of the spray cone body with respect to the base frame.

5.3. Object domain

In comparison with the previous general purpose units and geometry domains the object domain is specially targeted at describing objects for grasping.

5.3.1. Object domain analysis

Several objects that are commonly found in households or workbenches and their functionality have been investigated based on the work presented in the state of the art section. Some categories of the analyzed objects are: Dishware, cookware, vessels, furniture or tools. A special focus has been put on the annotation of regions as proposed by Xue *et al.* [102] as well as Baier and Zhang [6], but also the attachment of frames to objects [80]. The patterns that have been identified in this investigation are formalized in the object domain model in the next section.

For annotated regions two major categories have been identified: Real regions and virtual regions. *Real regions* are complete links of an object as specified in the kinematic structure. For instance, the spray bottle consists of a body link, a cap link and a trigger link. Each of these could be real regions. In contrast the handle of the spray bottle's body or the handle of a cup usually are not represented as an independent link. In such a case a *marker region* can be superimposed over an existing link. The intersection of the marker and the CAD model is then the annotated region. Another concept that

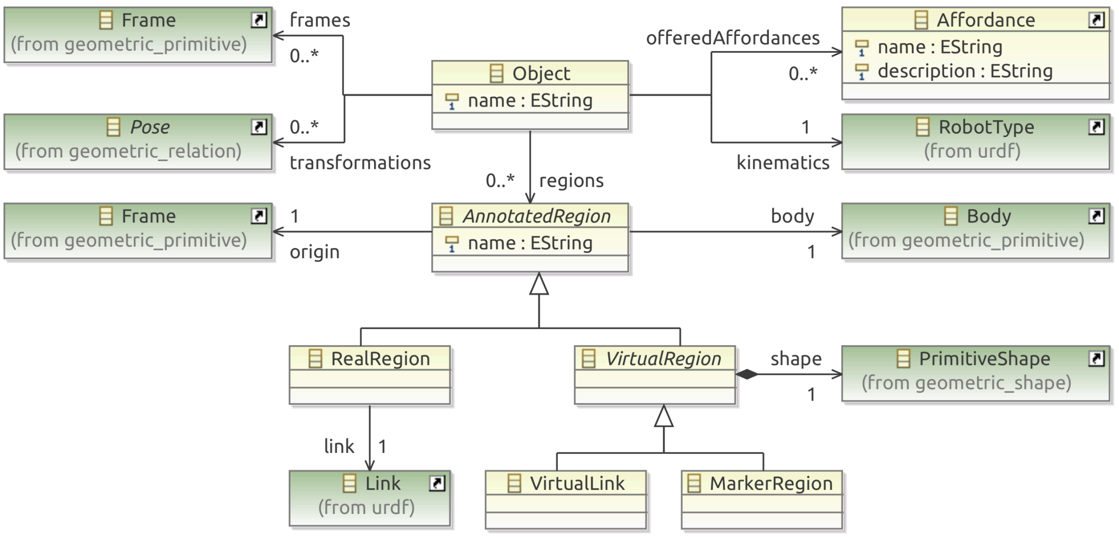


Figure 5.2.: The diagram of the object model.

denotes object semantics is the *virtual link*. A virtual link describes a region that should not be entered by the hand during certain tasks. Examples of these virtual links are the areas in front of the spray bottle's nozzle or the spout of a cup. They are attached to the existing object. Both, marker regions and virtual links are *virtual regions* and arise from the usage of an object. In GDDL they are described as primitives shapes.

5.3.2. Object domain model

Figure 5.2 depicts the diagram of the object model which has been derived from the previous conceptual considerations. Every object is identified by its *name* and has a *kinematic structure*. The kinematic structure can be as simple as a single rigid body but also more complex containing different types of joints and include dynamic properties such as mass or the inertia matrices of the bodies. To describe objects' kinematics and dynamics the existing Unified Robot Description Format (URDF) [65] is reused which is represented as XML. An XSD schema for URDF is available [71] and has been transformed to an Ecore model. Figure A.1 in the appendix shows this URDF-Ecore model. When an object contains non-fixed joints the transformations between the bodies must be calculated online. However, there can also be *frames* attached to an object such as for example a task frame. In this case the *transformation* must be specified offline in the object description. *Annotated regions* have to specify the name of their origin frame. By referencing a body from the geometric relations semantics they also establish the connection between the body label and geometric shape. While a *real region* references a link from the kinematics the *virtual regions* contain a primitive shape to describe their area.

Besides the kinematics and annotated regions an object references a set of *affordance* definitions that specify the functionality that the object provides. An affordance is encoded as a name and a description of the functionality. This representation of affordances as just a label, is an example of a view on a certain field. In general, affordances should also encode information about visual cues or effects of the associated action. However, this information does not contribute to grasping and therefore is not considered here. A list of affordances could for example be compiled from the AfNet/AfRob ontology [98]. The knowledge about the functional affordances enables the task description to check if the object can be applied in a certain task.

5.3.3. Object domain language

```
transport_affordance: Affordance {
  description: "Enables␣transporting␣of␣an␣object"
}

spray_affordance: Affordance {
  description: "Enables␣spraying␣of␣liquids␣with␣an␣object"
}
```

Listing 5.5: Definition of two affordances.

Listing 5.8 shows an example of a GDDL definition for a “transport” affordance and a “spray” affordance. A transport task should for instance check that an object has the “transport” affordance attached. For using the spray bottle the object should offer both, the “transport” and the “spray” affordance.

```
BottleBodyRegion: RealRegion {
  origin: {bottle_base}
  link: spray_bottle.base_link
  body: BottleBaseBody
}

BottleSprayConeRegion: VirtualLink {
  origin: {bottle_spray_cone}
  shape: Cone(50cm, 50cm)
  body: BottleSprayConeBody
}

// ...

spray_bottle: Object {
  kinematics: spray_bottle
  offered_affordances: [transport_affordance, spray_affordance]
  regions: [BottleBodyRegion, BottleTriggerRegion, BottleSprayConeRegion]
  frames: [{bottle_base}, {bottle_trigger}, {bottle_spray_cone}]
  transformations: [bottle_spray_cone_pose]
}
```

Listing 5.6: Example of the spray bottle description in GDDL.

In listing 5.6 the definition of two annotated regions is shown. The body region associates the geometric CAD information from the kinematics description with the body label from the geometric semantics relations. Additionally, the spray cone is defined which represents the area in front of the nozzle where the liquid is distributed. Finally, the object definition of the spray bottle composes the complete object description based on the previously separated knowledge.

5.4. Task domain

5.4.1. Task domain analysis

Task categories

Zöllner *et al.* [106] subdivide manipulation tasks into three basic classes: Transport, device handling and tool handling. Here, this classification has been refined by analyzing the previous such as vessels, furniture or tools in the context of their common application. Some use cases are opening or closing doors and drawers, pushing buttons or pouring from cups. The resulting task categories and exemplifying can be seen in table 5.1. Similar to [106] the tasks are: Direct manipulation, tool use and device usage.

The goal of *direct manipulation* is to move objects directly i.e. without any helping tools. The object sizes vary approximately between the size of a needle to the size of a parcel or bottle crate. A further subdivision is the categorization into "contact-oriented" tasks in which the manipulated object is in contact with the environment and "motion-oriented" tasks where it is not. Common "motion-oriented" tasks are: Pick, hold, transport or throw. Examples of "contact-oriented" tasks are: Push, pull, place, handover, squeeze or roll.

In *tool use* tasks a grasped object supports the tool user in achieving a goal that is difficult or impossible to achieve with the bare hand. Most tools can be grasped as well as operated by a single individual, often with a single hand, and range from the size of a needle to the size of an axe. Further examples of tools are knives, forks, screw-drivers or pens. Usually, during the use of a tool, the complete tool is moved. Tool-use often consists of two major phases:

1. Move the tool-center point to a predefined pose
2. Apply a force/torque or rotate/translate the tool-center point

The class of *device usage* tasks describes the interaction with devices. Like a tool, a device is an object that is used to achieve a goal that would be complicated to achieve with the bare hand. When a device is activated it performs or triggers some associated action on its own. Sizes of devices vary greatly. Some examples are a mobile phone or different type of machines. However, only small parts of the device are meant for interaction such as buttons or levers. These parts have about the size of a hand. Usually, in contrast to tool use, during the use of a device, the device remains stationary. An abstraction of device usage tasks is to only consider the kinematics of the device and describe the usage in terms of “activation” of joints or links. Buttons can for instance be modeled as prismatic joints that are activated by pushing or pressing tasks, while revolute joints are activated by turning and rotate tasks. The term hybrid in this context describes a revolute joint that is activated like prismatic joint. An example of this are the keys of a piano.

Table 5.1.: Different manipulation tasks

Task	**Goal**	**Refinement**	**Actions**
Direct manipulation	Move an object	Motion-oriented	pick, hold, transport, throw
		Contact-oriented	push, pull, place, handover
Tool use	Move TCP to pose Apply wrench or twist	-	cut, spear, stir, write, hold, pour, screw
Device usage	Activate joint	Prismatic joint	push, pull, press
		Revolute joint	turn, rotate
		Hybrid	push, pull, press
	Activate link	Link	touch, slide, press, tap, strum

Task specification

All three previous types of manipulation tasks can be decomposed into primitive manipulation actions similar to [74]. These primitives are *translation* or *rotation* and the exertion of *force* or *torque*. Rotation and translation require an acceleration resulting in the application of a wrench on the object. A grasp must resist these wrenches. Therefore, one major requirement of the task domain is to specify all wrenches or provide a way to derive the wrenches that may occur during the task execution. A method that provides such a specification is the established task frame formalism (TFF) [62][16] for manipulation tasks. The TFF assumes that a frame is attached to the manipulator’s end-effector or to the tool-center point. For each of the six DOFs (translation and orientation) either a motion (linear or angular) or a force (force or torque) is provided. An example of a motion task and cutting task is shown in listing 5.7. A limitation of the TFF when applied to grasp planning is that it does not specify accelerations for motions. Therefore, a heuristic should be applied to derive accelerations from motion specifications e.g. that a low velocity implies a low acceleration and vice versa.

```
move_left = MotionSpec {
  xt: velocity 0 m.s^-1
  yt: velocity 10 cm.s^-1
  zt: velocity 0 m.s^-1
  axt: velocity 0 rad.s^-1
  ayt: velocity 0 rad.s^-1
  azt: velocity 0 rad.s^-1
}

cut_forward = MotionSpec {
  xt: velocity 0.1 m.s^-1
  yt: force 10 N
  zt: velocity 0 m.s^-1
  axt: force 0 N.m
  ayt: velocity 0 rad.s^-1
  azt: force 0 N.m
}
```

Listing 5.7: Specification of a motion and a push task in the TFF.

Wrench-based grasp evaluation

In the state of the art section 3.1 several wrench-based grasp evaluation criteria have been reviewed, some of them are also capable of considering wrench-based task requirements. The wrenches should either be specified directly by the user or derived from the task specification. The convex hull of the individual wrenches approximates the TWS that can be used to evaluate a grasp.

It is important to consider in which frames the wrenches occur. In physics a common graphical representation to analyze forces and torques that act on a body is the free body diagram (see e.g. [87]). Figure 5.3 depicts an example of different frames attached to a knife. The following wrenches with respect to the frames can be identified:

- The task is specified in the $\{task\}$ frame. This is the reference frame for the TFF specification "cut_forward" which is shown in listing 5.7.
- Gravity acts on the $\{center\_of\_mass\}$ frame. However, the direction of the gravity vector with respect to the object's base frame depends on the orientation of the object during task execution.
- When an object is grasped the fingers apply wrenches at the contact points. The individual wrenches can be transformed to the $\{grasp\}$ frame.

The task-related wrenches can be derived from the TFF specification, while the gravity must be specified by the developer explicitly. Contact wrenches on the contrary must be calculated at run-time in the grasp planner's simulation. The grasp simulation also provides the transformation between the hand's grasp frame and the object's base frame. Other transformations are specified as fixed poses in GDDL. Therefore, it is the task of the run-time system to transform all wrenches into a common frame so that the TWS and the GWS can be compared.

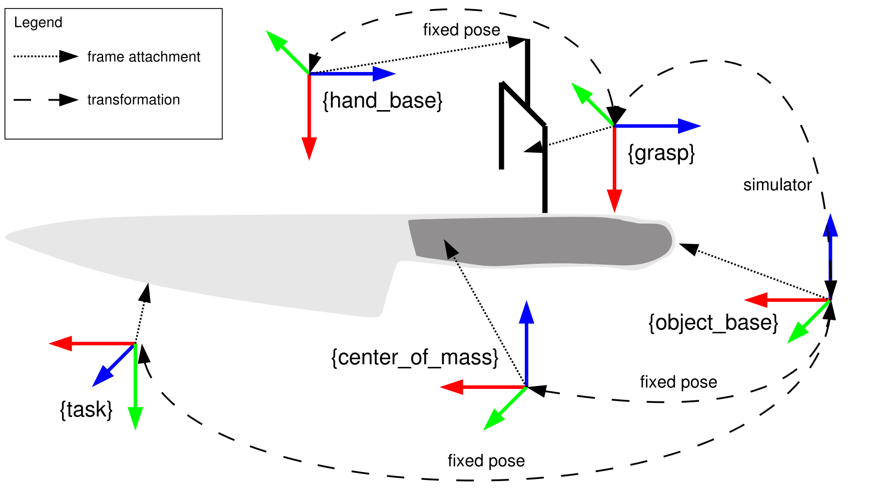


Figure 5.3.: Frames attached to a kitchen knife (based on [49]).

Task-based grasp evaluation

The major task-based evaluation criterion are the position constraints. The object model (see above) specifies real and virtual regions. In the task the desired usage is described. Three types of usage have been identified: Required, hint and forbidden. Each of these elements can be combined with a region (see table 5.2). From the spray flask use case the following combinations can be derived:

- Required real region: In order to pull the trigger, a part of the trigger must be touched.
- Real region as hint: Grasp the handle of the bottle's body
- Forbidden virtual region: While spraying keep out of the area in front of the nozzle

An example of a forbidden real region is the body of a hot frying pan as which should not be touched in order to prevent burns. Finally, a proximity sensor exemplifies a required virtual region. To activate such a sensor the hand must be placed "near" the sensor.

Table 5.2.: Real and virtual regions and their usage semantics

Region type	**Required**	**Hint**	**Forbidden**
Real region	Spray bottle trigger	Handle	Body of hot frying pan
Virtual region	Proximity sensor	–	Nozzle of spray bottle

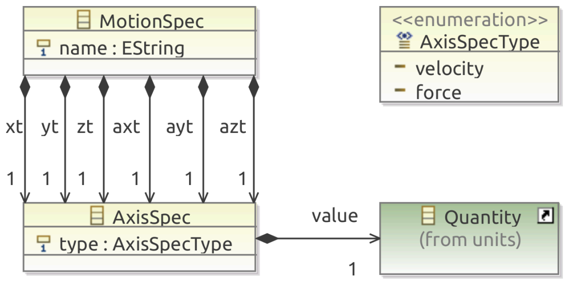


Figure 5.4.: The diagram of the TFF model

5.4.2. Task domain model

The diagram in figure 5.4 shows the model that formalizes data for motion specifications by the TFF. This model reuses the general structure and names from Klotzbücher's TFF model [53]. However, instead of representing the quantities as a double value and a unit encoded as a string, here the previously presented units model is reused. Thus, the axis specification benefits from the formal representation and validation of units.

Figure 5.5 is the diagram of different evaluation criteria classes. The model distinguishes the two categories *WrenchBasedCriterion* and *TaskBasedCriterion*. The epsilon metric (ϵ_{GWS}) is a wrench-based criterion and measures the worst-case quality of a grasp (see [29]). An average grasp quality measure is represented by the GWS volume measure [67]. Both criteria can be provided with a threshold to reject certain bad grasps. Another task-independent quality measure is the force-closure criterion. Force-closure should be used when no specification of a task can be provided like in many transportation tasks. The task-based evaluation criteria implement for example the previously discussed position constraints.

Finally, the task model in figure 5.6 models the three task categories *DirectManipulationTask*, *ToolUseTask* and *DeviceUsageTask*. The first two categories reference a TFF motion specification and associate this specification with a concrete frame. For instance, a *cut* specification as shown in listing 5.7 could be assigned to the $\{task\}$ frame in figure 5.3. Joint or link activation tasks on the contrary reference the element that should be activated. Each of these primitive tasks has to be evaluated based on at least one evaluation criterion. In practice, a *ManipulationTask* task can consist of several primitive tasks. An example of such a task decomposition is the spray bottle. One task is holding and transporting the spray bottle while the other task is the activation of the trigger to spray the liquid.

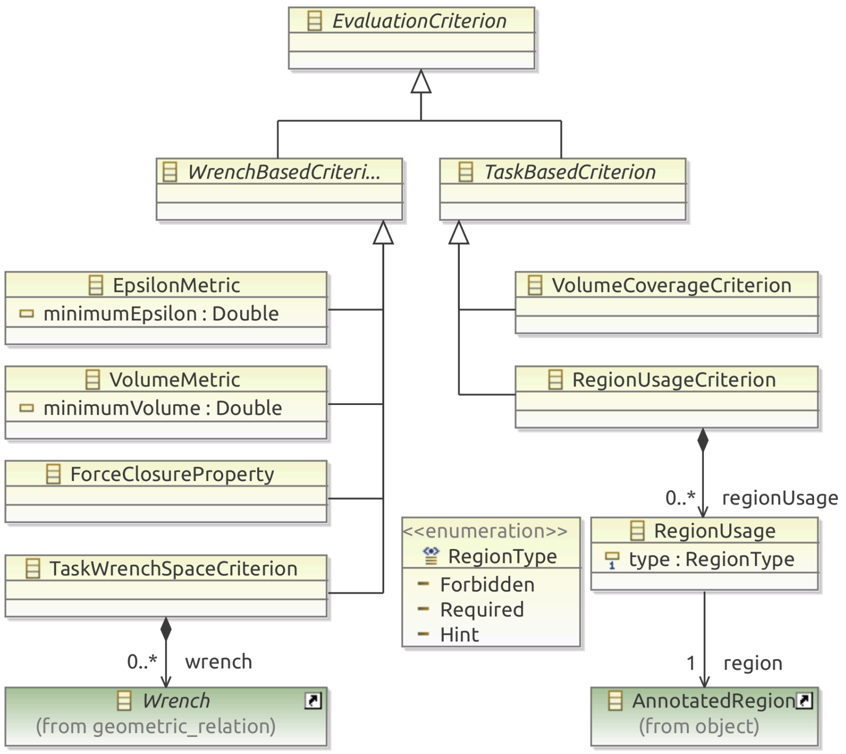


Figure 5.5.: The diagram of the task evaluation model

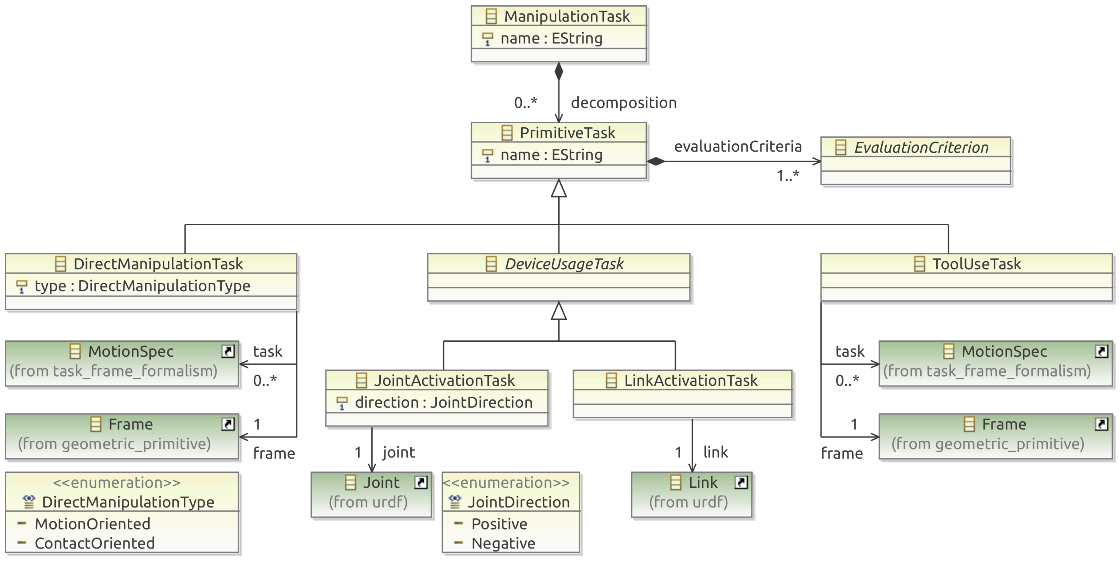


Figure 5.6.: The diagram of the task model

5.4.3. Task domain language

Here, the task domain is demonstrated with the specification of the spray task that describes how the spray bottle should be applied to spray liquids. To identify if an object is applicable for this task, at first the required object affordances are listed. In this case they are the transport affordance and the spray affordance. Then, the task is decomposed into the transport-specific and spray-specific primitive tasks. As shown in table 5.1 *transport* is a motion-oriented, direct manipulation task. For grasp evaluation the force-closure criterion is applied. *Pulling* the trigger is a device usage task that has to activate a joint. Although it is called "pulling" the goal is to activate a revolute joint, making this an example of a hybrid joint activation task (see table 5.1). Implementation-wise this difference does not matter so the revolute trigger joint is simply referenced. During spraying no hand part should be located in the spray cone in front of the nozzle and the trigger must be grasped. This region usage is the only evaluation criteria for the spray sub-task.

```
spray_task: ManipulationTask {
  required_affordances: [transport_affordance, spray_affordance]

  Decomposition {
    carry: DirectManipulationTask {
      Evaluation { ForceClosure }

      type: MotionOriented
      task_frame: {grasp}
      motion: [move_left, move_right, move_up, move_down]
    }

    spray: JointActivationTask {
      Evaluation {
        RegionUsage {
          SprayConeRegion: Forbidden
          TriggerRegion: Required
        }
      }
      direction: Positive
      joint: spray_bottle.trigger_joint
    }
  }
}
```

Listing 5.8: The spray task in GDDL.

5.5. Hand domain

5.5.1. Hand domain analysis

The hand model establishes a mapping between an abstract description of grasp semantics to a concrete realization of a grasp with a specific hand. On the one hand this requires a representation for grasp semantics and on the other hand the mapping must be specified. Based on the state-of-the-art analysis the following symbolic grasp semantics classes have been identified:

- Dexterity type: Dextrous and non-dextrous grasps
- Contact type: Precision, pinch and power grasp
- Thumb position: Abduction and adduction
- Opposition space: Pad, palm, side and object (in hook or platform) opposition
- Grasp shape: Circular (sphere, disk, tripod), prismatic (parallel, wrap), hook and platform
- Possible object size: Small, medium and large

These attributes can easily be attached to a grasp as a simple label. To find a suitable grasp realization the required semantics only have to be compared to the offered semantics. The virtual fingers are another semantic annotation that defines the minimum number of required fingers to perform a task. As a virtual finger can consist of more than one real finger the hand model should map real fingers to virtual fingers. The palm also counts as a real finger. Examples of the different number of virtual fingers are the hook or platform (one virtual finger), a precision grasp (two virtual fingers) or a power grasp (three virtual fingers, including the palm). Arranging these attributes and virtual fingers in a grasp taxonomy similar to [18] or [28] provides a powerful tool to developers of the task and hand model. The grasp taxonomy is easily understandable and good example taxonomies exist so that a developer can look up which grasp type is required and then specify the grasp's semantics in the task description. However, existing grasp taxonomies have not been designed to support the composition of grasps from subconfigurations as proposed in [74], instead they describe whole-hand configurations. As most grasp semantics' attributes can be applied to subconfigurations it is possible to easily extend grasp taxonomies for composition.

Nguyen and Stephanou [74] have identified the hand functionality as an additional attribute of a grasp. The hand functionality manifests in different hand-object contacts given a posture. This concept is difficult to capture in a grasp taxonomy, but based on the grasp frames [80] the major interaction of hand and object (and thus the contacts) can be approximated. Just like grasp taxonomies the grasp frames are designed for whole-hand configurations and therefore don't support subconfiguration composition by default.

5.5.2. Hand domain model

The grasp taxonomy model in figure 5.7 captures the semantics of grasps and resembles the model presented in section 4.1. It describes a grasp taxonomy by a hierarchy of

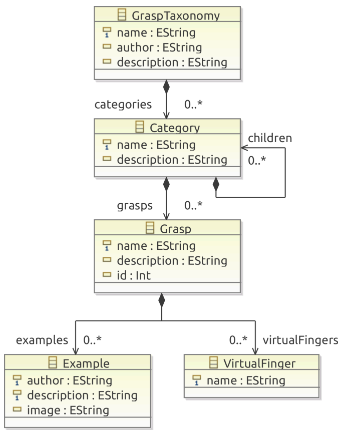


Figure 5.7.: The diagram of the grasp taxonomy model

categories. Each category can contain zero or more grasp descriptions which in turn specify examples from existing grasp taxonomies. In addition, a grasp establishes the association with a set of virtual fingers.

The GDDL model for hands as shown in figure 5.8 distinguishes three classes of manipulators (see also [12]). The first class are the simple *grippers* in which all fingers are associated with a single degree of freedom. *Dextrous hands* here describe manipulators that follow the design of the Salisbury hand [63]. They usually have three fingers, two of which can be spread. Finally, *anthropomorphic hands* resemble the functionality of the human hand and often have four to five fingers. A difference between dextrous and anthropomorphic hands is the orientation of the palm. In a dextrous hand the palm's orientation is aligned with the wrist. When the wrist faces towards the object so does the palm. In anthropomorphic hands the palm would be perpendicular to this approach direction. Figure 5.9 shows examples of the three classes.

The manipulator determines the grasp taxonomy it implements and also defines its kinematic structure. Like the object kinematics it reuses the URDF model. The hand's degrees of freedom are derived from the kinematics by only taking into account revolute or prismatic joints and ignoring for example fixed joints. The remainder of the classes in figure 5.8 can be classified into annotations of the kinematics (bottom left) and subconfiguration specifications (bottom right). Kinematic annotations assign a functional role to elements of the hand. This functional role can be the palm, consisting of several kinematic links, or digits like a thumb or a finger. Digits are specified by their root and tip link. Based on this information the kinematic chain for a digit can be extracted from the kinematic structure. Additionally, the kinematic annotation provides a name for the for the hand's origin frame.

The proposed extension of subconfigurations is implemented here. Each subconfigura-

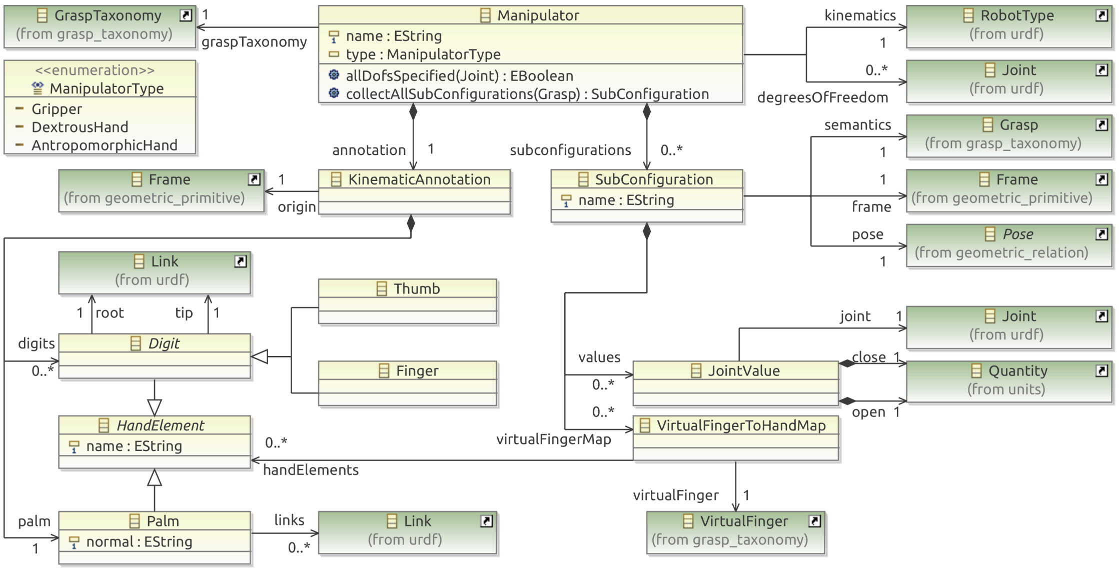
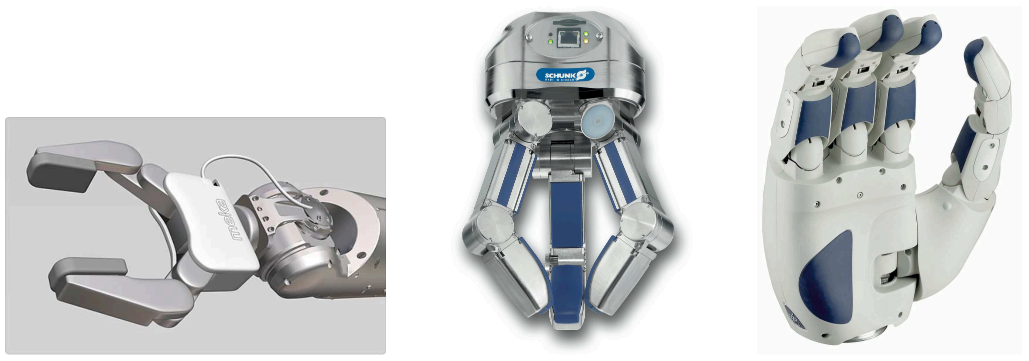


Figure 5.8.: The diagram of the hand model



(a) Gripper [60] (b) Dextrous hand [24] (c) Anthropomorphic hand [23]

Figure 5.9.: Examples of the major categories of robotic hands: Gripper, dextrous hand and anthropomorphic hand.

tion is associated with a semantic annotation, represented as a grasp description from a grasp taxonomy as well as the definition of the grasp frame and its pose. The subconfiguration's semantics enable the selection of subconfigurations based on task requirements while the grasp frame describes the interaction of the hand and the object. Additionally, the subconfiguration is defined by the open and close configuration of a subset of the hand's joints. Finally, for each subconfiguration the mapping between virtual fingers and hand elements is provided.

5.5.3. Hand domain language

Listing 5.9 shows an excerpt of a simple grasp taxonomy that conforms to the grasp domain model. In general a grasp taxonomy contains several grasps. However, to analyze the spray bottle use case only two grasps are required. Additional grasps are specified analog. The first grasp in this use case consists of cylindrical power configuration the grasp the bottle's handle. To shorten the listing the power and cylindrical category have been merged to one category. In this category the grasp is specified with the required virtual fingers and a reference to a similar grasp in the grasp taxonomy by Feix [28].

```
VF1: VirtualFinger
VF2: VirtualFinger
VF3: VirtualFinger

my_taxonomy: GraspTaxonomy by "Sven␣Schneider" {
  description: "A␣simple␣grasp␣taxonomy"

  power_cylindrical: Category {
    description: "Cylindrical␣power␣grasps"

    cylindrical_power_grasp: Grasp {
      description: "A␣simple␣cylindrical␣power␣grasp"
      id: 0
      virtual_fingers: [VF1, VF2, VF3]

      Example {
        author: "Thomas␣Feix"
        description: "Medium␣Wrap"
        image: "http://grasp.xief.net/images_grasps/i_3_1"
      }
    }
  }

  // ...
}
```

Listing 5.9: An example of a simple grasp taxonomy in GDDL.

The grasp taxonomy is used by the manipulator definition in listing 5.9 which describes the Schunk Dextrous Hand (SDH). All elements of the kinematic structure are defined in a URDF document and can easily be referenced from here. This is for example used in the kinematic annotation that describes the elements of the palm or the thumb.

Two subconfigurations are defined of which the first one realizes the cylindrical power grasp. This is denoted by the reference to the cylindrical power grasp entry in the grasp taxonomy. Next, the grasp's coordinate frame as well as its pose are defined. The coordinate frame is the general purpose frame $\{grasp\}$. As the cylindrical power grasp requires at least three virtual fingers in this subconfiguration an mapping between the three virtual fingers and three real fingers (i.e. palm, thumb and first finger) is established. For each of the real fingers the subconfiguration defines the joint values as open and closed configuration. Again, the units DSL is reused so that physically meaningful quantities can be used.

```
sdh: Manipulator {
  kinematics: sdh
  grasp_taxonomy: my_taxonomy

  KinematicAnnotation {
    sdh_palm: Palm {
      links: [sdh.base_link, sdh.sdh_palm_link]
    }

    sdh_thumb: Thumb {
      root: sdh.sdh_thumb_1_link
      tip: sdh.sdh_thumb_3_link
    }

    // ...
  }

  SubConfigurations {
    small_cylindrical_power_grasp: SubConfiguration {
      semantics: my_taxonomy.power.cylindrical.cylindrical_power_grasp
      task_frame: {grasp}
      pose: sdh_small_cylindrical_power_grasp_pose
      VirtualFingerMap {
        VF1: [sdh_palm]
        VF2: [sdh_thumb]
        VF3: [sdh_finger_1]
      }
      Values {
        sdh.sdh_thumb_2_joint: JointValue { open: -0.9854 rad close: 0.0 rad }
        // ...
      }
    }

    hook_finger_1: SubConfiguration {
      // ...
    }
  }
}
```

Listing 5.10: The Schunk Dextrous Hand (SDH) modeled in GDDL.

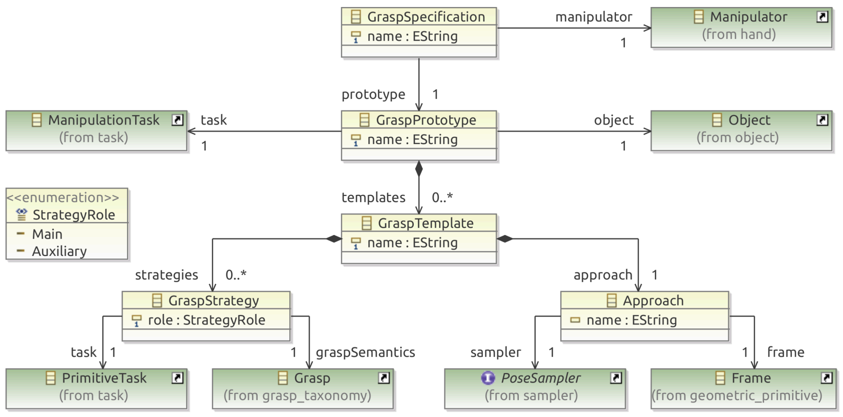


Figure 5.10.: The diagram of the model composition

5.6. Composition of domain models

5.6.1. Composition model

The previous domains have all been clearly separated. There are

- an *object* model such as the spray bottle
- a *task* model like spray
- and a *hand* model that represents e.g. the SDH.

However, the goal is to specify a task similiar to "use the *spray bottle* for *spraying* with the *SDH*". Therefore, the separate models must be composed. A diagram of the metamodel that specifies this composition is shown in figure 5.10. The composition starts at the bottom where the goal is the specification of an abstract grasp strategy (see [101]) that consists of a primitive task and the required grasp semantics to perform this primitive task. In the spray bottle use case this means for instance that the primitive task carry should be performed by a cylindrical power grasp. This does neither determine which hand should perform the grasp nor how the grasp should be realized. As the grasp strategy is also independent of the wrist's pose in the next step it is associated with an approach description. The approach description defines the configuration of a sampler which calculates wrist poses. The combination of grasp strategy and approach description is called the grasp template. Exactly one grasp strategy in a grasp template must be given the main role, other strategies are all auxiliary. During the composition with a concrete hand the main strategy determines which subconfiguration's grasp frame is active. Depending on the task an object can be grasped from several directions. This is specified in the grasp prototype (the concept is similar to [79]). The grasp prototype is hand-independent so that as a final step it is composed with a concrete manipulator description resulting in the grasp specification.

5.6.2. Composition language

Listing 5.11 shows an example of an approach pose sampler. Pose sampling is split into position sampling and orientation sampling. The position sampler calculates positions on the surface of the provided primitive shape which can e.g. be a plane, sphere or cylinder. In the example a planar sampler is used that samples x- and y-coordinates in the plane randomly from the provided range. Instead of a random values sampler for example also an iterative sampler could be chosen. When the position is determined, based on the known primitive shape the pose sampler calculates the orientation.

```
rear_approach_sampler: PlanarPoseSampler {
  position_sampler: PlanarPositionSampler {
    plane: Plane(PositionVector(0 m, 1 m), PositionVector(1 m, 0 m))
    v: RandomValueSampler(-10 cm, 10 cm)
    w: RandomValueSampler(-10 cm, 10 cm)
  }
}
```

Listing 5.11: Example of a random approach pose sampler

An example of defining a grasp prototype is shown in listing 5.12. This hand-independent grasp specification first references the object (the spray bottle) and then the task (the spray task). Only one grasp template is required because grasping from the rear of the handle usually fulfills the task requirements best. These task are a stable grasp around the handle, which is described in the first grasp strategy, and touching the trigger as described in the second grasp strategy. The previously described planar pose sampler returns poses with respect to the origin. As the desired approach poses are located behind the handle all sampled poses have to be transformed. This is achieved by attaching the sampler to a frame, in this case the frame of the handle.

```
spray_with_spray_bottle: GraspPrototype {
  object: spray_bottle
  task: spray_task

  GraspTemplates {
    rear_approach_strategy: GraspTemplate {
      Strategies {
        GraspStrategy {
          role: Main
          task: spray_task.carry
          configuration: my_taxonomy.power_cylindrical.cylindrical_power_grasp
        }

        GraspStrategy {
          role: Auxiliary
          task: spray_task.spray
          configuration: my_taxonomy.precision_support.hook
        }
      }

      Approach {
        sampler: rear_approach_sampler
        frame: {bottle_handle}
      }
    }
  }
}
```

Listing 5.12: Composition of the task and object domain into hand-independent grasp prototype

The last step is the composition of the hand-independent grasp prototype with a concrete hand. Listing 5.13 exemplifies this for the SDH and the "spray with spray bottle" grasp prototype.

```
sdh_spray_with_spray_bottle: GraspSpecification {
  manipulator: sdh
  prototype: spray_with_spray_bottle
}
```

Listing 5.13: Composition of the hand-independent grasp prototype and the SDH manipulator

6. GDDL integration with a grasp planner

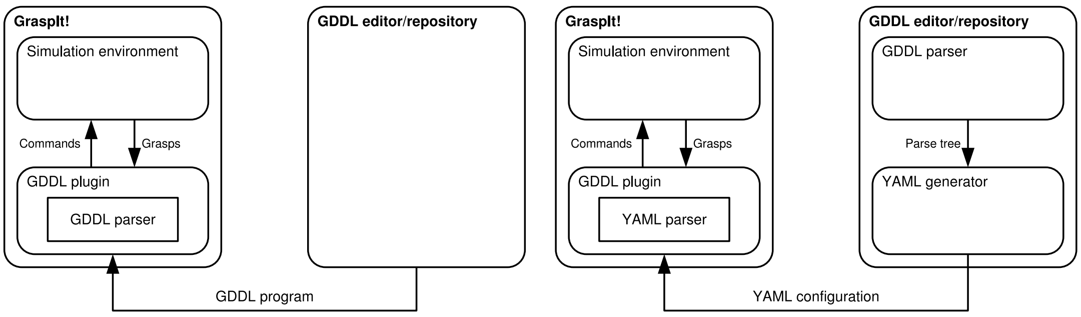
The metamodels and models or DSL programs respectively in the previous chapter are located on the layers **M2** and **M1** in the OMG's MOF modeling hierarchy. A proof-of-concept implementation situated on the **M0** layer is presented here.

6.1. Grasp planner evaluation

At first the grasp planners presented in section 3.7 are evaluated. The candidates are GraspIt! [67], OpenRAVE's grasp planner [26] and the grasp planner from the Simox library [97]. All of these grasp planners can perform a static analysis of a given grasp. However, only GraspIt! support the simulation of object dynamics. This means that when the fingers are closed in the simulation environment they don't stop on contact with the object but instead start moving the object. Especially, for grippers this is important as they usually only have a single DOF to control all fingers. Therefore, the first object contact stops the motion of both fingers and in most cases the object won't be in a stable grasp. A major deficit of GraspIt!'s dynamics simulation is its numeric instability. For prismatic joints it does not work at all and for revolute joints it only works in some cases. None of the grasp planners is able to handle objects with kinematic structures i.e. an object can only consist of one rigid body. The Simox software is well structured and designed. As it is relatively new software it has been tested only in few projects. OpenRAVE's grasp planner on the contrary does not follow well established practices in software engineering but has been used in several projects. GraspIt! is the most used grasp planning software. However, its software quality could be improved.

Table 6.1.: Features of different grasp planners

Grasp planner	GraspIt!	OpenRAVE	Simox
Static analysis	✓	✓	✓
Dynamics simulation	✓	✗	✗
Object kinematics	✗	✗	✗
Collision engines	Custom, PQP	Bullet, ODE, PQP	PQP
Code quality	$\odot$	$\ominus$	$\oplus$
Functionality	$\oplus$	$\odot$	$\odot$
Implementation type	Application	Library	Library



(a) Plugin which includes the GDDL parser. (b) Plugin using YAML configuration.

Figure 6.1.: Two approaches to provide the GDDL configuration to the grasp planner

For instance, the GUI is tightly coupled with functional code. Although GraspIt! can be extended by plugins it is not possible to use it as a library. Table 6.1 summarizes the result. Based on this evaluation GraspIt! has been chosen for the proof-of-concept implementation for GDDL.

6.2. GDDL interface to grasp planner

The GDDL interface to the grasp planner has to implement two aspects. On the one hand it must map the physical properties of hand and object from GDDL to the grasp planner's representation. On the other hand it must configure the grasp planner with the provided GDDL data. Ideally it would be possible to generate the grasp-planner-specific representation of objects and hands from a unified model or existing descriptions by a model-to-model transformation. Since this is out of scope for this thesis a different approach has been chosen: A YAML[1] (YAML Ain't Markup Language) configuration file maps the identifiers (i.e. the names) of objects or hands in the GDDL model to the correct representation for the grasp planner.

Figure 6.1 depicts two approaches to integrate the GDDL interface into a grasp planner. In the figure this is exemplified by the GraspIt! plugin, but it also holds for other grasp planners. The first approach, shown on the left side of the figure, integrates the GDDL parser directly into the plugin (e.g. as a software library). This enables the plugin to directly parse and interpret GDDL programs. It is however problematic that Xtext only generates a parser for Java but the GraspIt! plugin must be developed in `C++`. It has to be investigated further if the parser generators of the EMF4CPP project could be used to generate a GDDL parser in `C++`. Due to this drawback the second approach as shown in figure 6.1(b) has been favored in this thesis. Here, the GDDL IDE includes

[1]`http://www.yaml.org/spec/1.2/spec.html`

a code generator that transforms GDDL into an intermediate representation such as YAML files which are then in turn interpreted by the GraspIt! plugin. With Xpand[2] and Xtend[3] Eclipse offers two powerful tools for template-based code generation which would enable easy generation of the YAML files. As this is out of scope for this thesis and the EMF4CPP tools should be investigated first, for now the YAML files are created manually.

For grasp synthesis the proof of concept plugin implements a forward kinematics simulation-based grasp planning approach as outlined in algorithm 1. It evaluates grasps by the already existing wrench-based criteria that have been implemented in GraspIt!. Further criteria such as the collisions between the hand and virtual links is work in progress as it requires a second collision engine that runs in parallel to the simulation collision engine. While the simulation's collision engine prevents the hand from entering rigid bodies, forbidden regions may be touched during the closing of the hand but the hand may not remain inside this region in the final grasp.

[2]`http://wiki.eclipse.org/Xpand`
[3]`http://www.eclipse.org/xtend`

7. Discussion and evaluation

7.1. Model-driven engineering

7.1.1. Domain modeling

Model-driven software engineering offers several benefits to developers as well as users of the models. For instance, the model, especially the graphical representations, enables developers to communicate with domain experts more easily. In combination with a DSL it can even help to standardize terms, notations or the semantics of concepts within a domain.

Based on a domain analysis the relevant concepts are identified and the overall domain can be decomposed. In a well decomposed system it is easier to identify aspects that concern more than one domain. These aspects should be extracted from the special-purpose domain models into a general-purpose model which is reusable. In GDDL such general-purpose models are the units models, the geometric relation semantics model or the kinematics model. Some candidates that should be separated out are the object representation from which certain concepts are also required by e.g. the perception or a world model. By decomposing the domains and identifying their interfaces or interactions a more focused extension of the individual domains is enabled. While the goal of this thesis the identification of the domains related to grasping and describe their coarse models there are experts that concentrate only on one of the domains. For instance, an expert in grasp semantics can contribute to the manipulator domain and does not have to care about the formal models of the object.

In the models certain patterns can be identified as well as stable aspects of a domain. For example, forces and torques that occur during task execution will always be a major concern in determining grasp stability. A detailed domain analysis also reveals hidden assumptions or implicit conventions that are generally made or used in a domain. For example, GraspIt! assumes that all quantities of length are provided in millimeters while ROS assumes that lengths are meters. The domain analysis leading to GDDL's models for instance has shown that grasping does not just concern an object and a hand. Instead, grasping has always a purpose that is defined in a task. Many research efforts assume that the task is simple such as transporting or holding, but don't specify this explicitly.

A further features of formal domain metamodels is that they can specify structure on the one hand but on the other hand also behavioral aspects as formal constraints. Models that conform to the metamodels can therefore be validated automatically. This helps to increase the overall robustness as many constraints can be checked already at design time and errors can be found earlier in the software development process.

Another aspect on the software development side is the code generation that is enabled by formal metamodels. Although it has not been used in this thesis, code generated from metamodels increases software robustness as well as it decreases development time since common elements of software don't have to be programmed manually. Robustness is achieved by capturing best practices in code templates. These templates need to be tested and optimized only once and can then be reused over and over again instead of repeatedly reimplementing low-quality code. The distinction of metamodels and conforming models has the benefit that a program's code skeleton can be generated while the configuration of the skeletons is specified in the models. Different model representations can be chosen for example some kind of DSL.

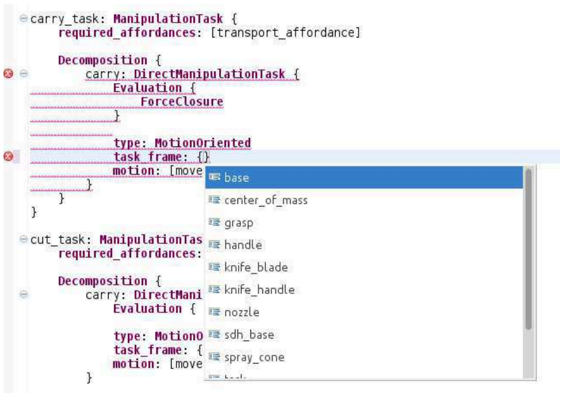
By explicitly capturing domain knowledge the knowledge representation can be extended and the models automatically adapted to these changes. Implicit representations that are often used in machine learning approaches have to be retrained completely in such situations. As knowledge in declarative languages like GDDL is specified explicitly no extensive amount of labeled data is required as for example in many machine-learning-based methods.

A major drawback of the model-driven engineering approach are the available tools. Many of the Eclipse tools are difficult to use and have software bugs. Additionally, some tools get replaced which then requires porting of models or code to the new replacement tools. When modeling looks like overhead to a developer anyway, he or she might be scared off by the available software tools.

7.1.2. Domain-specific languages

With Xtext a full-fledged IDE can be developed easily from the specification of language's grammar. The IDE supports features such as syntax highlighting, code completion, code navigation or hovering. An example of such an IDE for GDDL is shown in figure 7.1. Theoretically, the editor can show errors that occur during the model validation in a human-readable error message. While this works for smaller models which contain only one package, this is currently hindered by a bug in Eclipse. Besides the GUI, also a command line interface (CLI) can be developed easily. The code skeleton is generated together with the IDE and only needs to be configured accordingly. However, since this is not the focus of the thesis this feature has not been implemented yet. Models encoded in well-formed formats such as XML with an XSD schema can be referenced and reused from the generated DSL. An example is the URDF for robot hardware definitions that has been integrated with GDDL.

In GDDL physical quantities are not represented as pure values but instead as values and units. Due to this representation quantities with differently scaled units can easily



```
carry_task: ManipulationTask {
    required_affordances: [transport_affordance]

    Decomposition {
        carry: DirectManipulationTask {
            Evaluation {
                ForceClosure
            }

            type: MotionOriented
            task_frame: {}
            motion: [move
        }
    }
}

cut_task: ManipulationTas
    required_affordances:

    Decomposition {
        carry: DirectMani
            Evaluation {

            type: MotionO
            task_frame: {
            motion: [move
        }
```

Figure 7.1.: Excerpt of the generated IDE which shows the auto-completion for GDDL

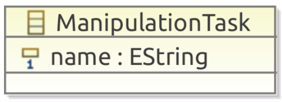
be combined and validated. For example, a length can either be specified in meters, centimeters or millimeters. In contrast to most common programming languages the meaning of a value is therefore directly encoded in the GDDL program so that a developer does not have to look up the semantics in the documentation or in the source code. The explicit model of physical quantities also enables the automatic validation of assignments. For instance, it can be checked that only angles are assigned to revolute joints while a prismatic joint only accepts lengths. Similarly, the metamodel of geometric relations can validate several geometric concepts.

7.2. Constraints

The formal models enable the specification of formal constraints in OCL. In contrast to *models* which capture the structure of a domain, constraints are part of the domain's *behavior*. As constraints can navigate the structure of a model and can easily handle sets as well as operations on sets they are a powerful tool. The possibility to specify constraints on models is a major difference to ontologies. Constraints can be subdivided into two categories [41]:

- *Atomic constraints* are specified based on one metamodel. An example is shown in figure 7.2. It validates the name of a manipulation task, which must consist of at least one letter.

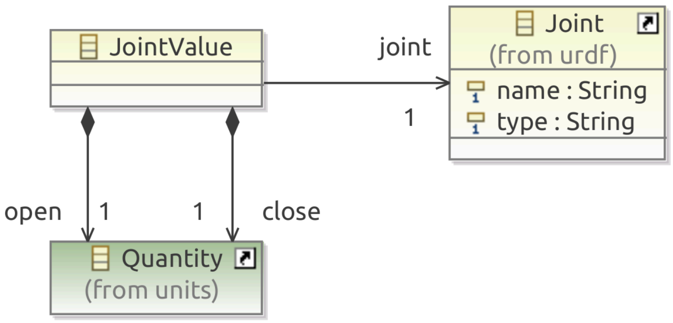
- *Composition constraints* arise from the composition of two or more domain models and therefore require the navigation between classes in the constraint. Figure 7.3 exemplifies two composition constraints for joint values of a hand. The first constraint asserts that no joint values are provided for fixed joints while the second constraint requires that only angluar values are specified for revolute joints.



```
context task::ManipulationTask

    inv NameMustBeValid('A valid name must be provided'):
        (self.name <> OclVoid) and (self.name.size() > 0)
```

Figure 7.2.: Atomic constraint to validate a manipulation task



```
context hand::JointValue

    inv NoFixedJoints('Fixed joint ' + joint.name + ' should not have a value'):
        self.joint.type <> 'fixed'

    inv RevoluteJointHasAngularOpenConfiguration(
            'The "open" configuration for the revolute or continuous joint '
            + joint.name + ' should be an angle'
      ):
        ((self.joint.type = 'revolute') or (self.joint.type = 'continuous'))
            implies self.open.isDimensionality(0, 0, 0, 0, 0, 0, 0, 1, 0)
```

Figure 7.3.: Two composition constraints to validate a joint value

Some of the main constraints in the grasping domain are outlined in the following. However, due to the erroneous OCL integration into EMF and Xtext with both, CompleteOCL as well as OCLinEcore, not all constraints have been implemented. In the context of the *units* metamodel, several OCL operations simplify the constraint checking. For instance, a quantity can be queried if it is of a specific type such as a length or time. All metamodels that reuse the units metamodel therefore immediately support the validation of quantities via constraints. An example is the *geometric relations semantics* metamodel in which all coordinate representations such as position vectors or force vectors are required to have a specific unit. For the *TFF* metamodel the immediate support of units is a feature that has previously not been supported by Klotzbücher's TFF meta model approach [53]. When TFF is used in contact-oriented primitive motion tasks, at least one axis specification should be a force (which arises from the environment contact).

For the *manipulator*, besides the previously described constraints for joint values, it is validated that prismatic joints are always provided with length quantities. It can also be checked if the provided values are within the joint limits. Additionally, it can be checked that the pose of a subconfiguration's frame is provided with respect to the hand's base frame.

In the domain *composition* metamodel several constraints have to be specified. The first one is concerns the grasp prototype, in which all templates must provide a grasp strategy for each primitive task that makes up the overall manipulation task. Additionally, all of these grasp strategies must be part of the manipulation task (not some other manipulation task). In the grasp template exactly one *main* grasp strategy must be specified. This main grasp strategy's frame must match the name of the manipulator's grasp frame in the grasp specification. One constraint check that is also relevant for components that are external of the grasping domain is the validation that a certain object $\mathcal{O}$ can be used in the task $\mathcal{T}$. This check is achieved by comparing the affordance requirements with the affordances offered by an object. Another powerful constraint is the validation that a robot with hand $\mathcal{H}$ can grasp object $\mathcal{O}$ for task $\mathcal{T}$. For instance, a simple two-finger gripper is not able to use a spray bottle for spraying because it cannot simultaneously hold the bottle and activate the trigger. Dextrous and anthropomorphic hands in contrast can perform this task. This constraint is implemented by checking if the set of grasps returned by algorithm 3 is empty or not. If it is empty, the hand cannot perform the task, else it can. The basic idea of the algorithm is to determine all task-matching subconfigurations, then calculate the Cartesian product of these subconfigurations and check if any product is a complete hand configuration.

Algorithm 3 Pseudo-code that returns all grasps which can perform a given task

```
primitive_tasks ← manipulation_task.decomposition
subconfigurations ← ∅
for each task in primitive_tasks do
    subconfigurations.append(hand.subconfigurations_with_semantics(task.semantics)
candidate_grasps ← subconfigurations.cartesian_product()
grasps ← ∅
for each grasp in candidate_grasps do
    if hand.all_dofs_specified(grasp) then
        grasps.append(grasp)
return grasps
```

7.3. GDDL-grasp planner interface

The implementation of the GDDL plugin for GraspIt! has shown that several required software components to evaluate certain domain concept have not yet been integrated into state of the art grasp planners. This includes for example a method to check if a hand is within a virtual region like the are in front of a spray bottle's nozzle. Another missing feature is the support for kinematics of graspable objects which would make it easier to evaluate if hand is in contact with a pre-defined part of an object. Additionally, a working physics simulation will help improving the quality of generated grasps. However, implementations for these features should not be added in an ad-hoc manner. Instead, a detailed analysis of grasp planning algorithms should be performed. Following the Best Practice in Robotics (BRICS) methodology the interfaces of these algorithms should be harmonized and the implementations refactor to software components [66]. In this context the missing features should be tackled. In this thesis the main domains and their *representations* have been identified that influence a grasp. The refactoring of grasp planning algorithms complements this by analyzing the *behavioral* aspects. In the refactored software, components such as collision detection, grasp evaluation or the dynamics simulation should be easily interchangable, so that a custom grasp planner is easily composable from reusable components. Via a common interface GDDL could then provide the configuration to these components.

7.4. Run-time architecture

With the GDDL-GraspIt! plugin a first part of the run-time architecture has been implemented. Figure 7.4 depicts the proposed architecture for integrating GDDL into a robot control architecture with a special focus on the interaction with a task planner. The task planner (upper part of the figure) provides an object identifier and a task identifier to the grasp operator (lower part of the figure). An additional input to the

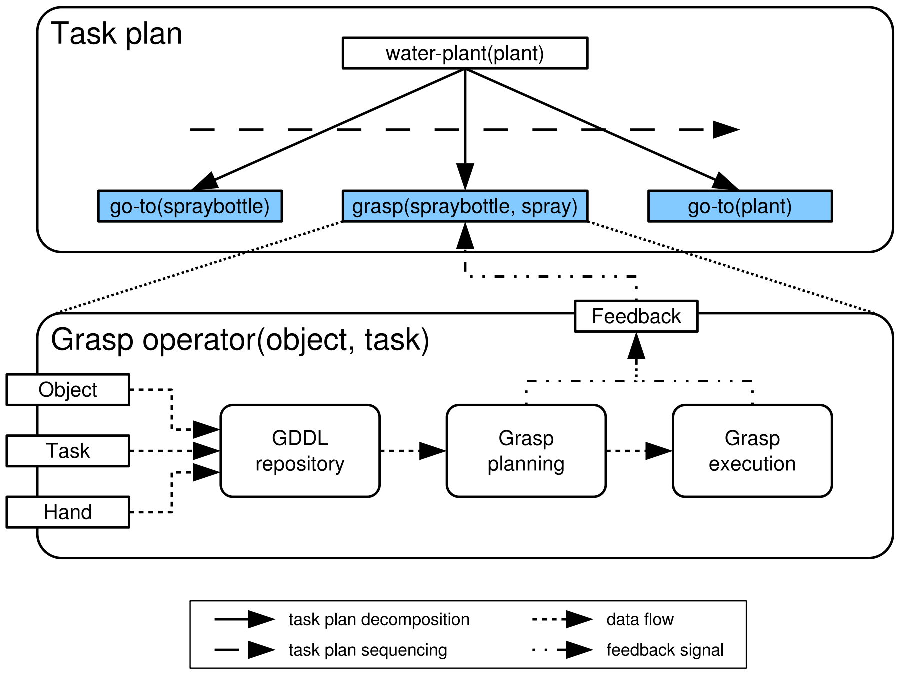


Figure 7.4.: The proposed integration of the GDDL run-time with a task planner. The task plan consists of a sequence of operators (blue boxes). The *grasp* operator is refined. As inputs it receives the object and task identified from the task planner. The feedback is provided as signals.

operator is the robot's hand which is background knowledge. Based on the object, task and hand the associated models are selected from the GDDL repository and composed. The resulting grasp specification is provided to the grasp planner (in this case GraspIt! with the GDDL plugin) as configuration. This configuration determines how grasps are synthesized and evaluated. The best grasp is sent to a reactive grasp controller which executes the grasp while handling errors that may arise from model imprecisions or noise in the object localization.

8. Conclusion

In this thesis a systematic analysis of grasping with robotic hands for tasks such as tool use has been presented. The results of the analysis have been formalized in domain metamodels which describe the most relevant domains that contribute to a grasp. The following domains have been identified:

- The *object* domain describes physical and semantic properties of object that should be grasped or used as tools. The object metamodel contains a surface description of objects such as a 3D CAD model or the surface material. Semantic properties are the kinematic structure, annotated regions or offered functionality.
- A *task* specifies how a grasped object will be used for instance in transportation or as a tool. The task is decomposed into primitive tasks of different type. A primitive task specifies evaluation criteria to determine if a grasp is applicable in a task or forces that are exerted on the grasped object during task execution.
- The *hand* domain describes how common grasps are realized by a concrete hand. Limitations due to the hand design can be derived from this metamodel. The semantics of a grasp are captured in grasp taxonomies.
- Finally, the *composition* domain describes how the previous independent domains are composed to perform a meaningful grasp with a given hand for a task.

For each of these domains representations have been developed or existing representations reused. Representations of physical quantities and geometric relations have been extracted out of the GDDL metamodels and therefore can be used in different metamodels as well. Besides fostering the reuse of domain concepts a model-driven approach in combination with code generation can increase the robustness of software. In addition, formal constraints guide and validate the composition of the grasping-related domains. Many constraints can already be checked at design time for instance if a concrete robotic hand is able to grasp an object in order to perform a given task. The Eclipse platform with the projects EMF and OCL have been selected to implement the metamodels and constraints.

Based on the formal domain metamodels the declarative domain-specific language GDDL has been specified. GDDL's objective is similar to PDDL as it should specify grasping problems independently of a grasp planner. GDDL is implemented with the

Eclipse Xtext project and provides a powerful IDE which supports developers in specifying GDDL programs. Some features of the IDE are syntax highlighting, code completion or refactoring assistance.

In a proof of concept, GDDL has been integrated into the grasp planner GraspIt!. The GDDL-GraspIt! plugin is the first step towards the complete run-time architecture which additionally will also include a repository of several domain models. A concept of integrating this GDDL run-time architecture into a robot control architecture with a task planner has been outlined.

9. Future research

This chapter proposes several improvements to the developed domain models and implemented or applied grasp planning software that should be performed in future work.

Handling model errors and imprecision: GDDL and most grasp planners assume that the complete state of the object that should be grasped is known in advance. However, these assumptions don't apply to a real robot which leads to errors and imprecisions in the robot's world model. Model imprecisions can have several sources [9]. For example, the 3D shape of the object on which the grasp was planned is different from the one that the grasp is executed on. This may be caused by a wrong object recognition or simply because the object database does not include the recognized object. Additionally, the object surface properties may have been detected incorrectly which results in wrong friction coefficients during the grasp planning. Another source of errors is the physics and contact simulation of the grasp planner which is only an approximation of the real world.

These problems can be overcome by different approaches. For instance, GDDL could be extended by error models to specify tolerances that may occur during the grasp planning and execution. Grasp planning with respect to pose uncertainty has e.g. been presented in [100]. Existing approaches to reactive grasping often use probabilistic models such as Hidden Markov Models that are updated based on tactile feedback or force-torque sensors in the arm or the hand [70][9][55][45]. On the one hand this allows the handling of unexpected contacts between the hand and an object. On the other hand this also enables fine adjustments of a grasp by local optimization methods. For instance, tilting fingers slightly might increase the contact area of a grasp and increase the grasp's stability. In this case the optimization method could be determined and configured by the extended GDDL specification for reactive grasping. In many common scenarios small object are lying on a support surface such as a table. When these objects are picked up by an enveloping grasp the hand necessarily gets in contact with the support surface. To this end Kazemi *et al.* [51] apply a compliant arm controller that establishes the hand-surface contact and also compensates for small wrist motions that are caused by the fingers pushing against the table during closing.

To summarize, model errors and imprecisions can and should be handled at different software layers. The most abstract approach is the specification of error models at design time for instance in a complementary language of GDDL. Also, the grasp planner can take uncertainty into account while on the lowest software layer compliant arm and finger controllers are able to handle unforeseen contacts during the execution of a grasp.

Besides the software also the hardware layer contributes to handling errors. Human hands for example have soft surfaces that passively adapt to the object. In robotics this concept has lately been applied to the design of the universal gripper [3]. It consists of a balloon filled with granular material that can enclose many objects passively. By applying a vacuum to the balloon, the material hardens and grips the object. Another approach are underactuated hands which have less degrees of actuation than degrees of freedom. Depending on the design the non-actuated finger parts also passively align with the object.

Extension of the analytical grasp planning models: The current models assume that all fingers start closing at the same time. In some situations this may result in undesired grasp results or collisions between the fingers. A solution that should be investigate to avoid these situations it the sequencing of subconfigurations which means that first all fingers of one subconfiguration are closed and then others can follow. An extension of sequencing certain subconfigurations is interleaving of arm motions and finger motions. Grasping a glass of water for transporting or pouring is an example of such an approach. At first, the glass is grasped and lifted. When there is sufficient space below, the little finger is moved underneath the glass and supports the grasp by increasing its stability.

Modeling of empirical grasp planning: Empirical grasp planning approaches offer advantages in generalization grasps especially on similar objects. Hence, much research effort has been investigated in these approaches, too. However, no extensive domain analysis has been performed to identify and formalize commonalities between different empirical approaches have been performed. In comparison with the metamodels identified in this thesis such an effort could lead to the integration of analytical and empirical grasping which fosters reuse and exchange between both approaches. For instance, GDDL could be used to represent the input for a machine learner.

Integration with a robot control architecture: The GraspIt! GDDL plugin is a first step of developing the run-time software for grasp planning. This plugin should be extended with additional features such as more grasp evaluation methods but also for interfacing with e.g. ROS so that it can be integrated into an existing ROS architecture. On the other side, the interface to the task planner must be implemented. The conceptual implementation of this interface has already been described in section 7.4.

Refactoring and extending grasp planners: The analysis of existing grasp planners (see section 6.1) has shown that their implementations on the one hand all planners share common approaches such as collision detection or grasp evaluation. But on the other hand some planners also lack required features or suffer from bad software quality. As proposed previously, these problems could be tackled by applying the BRICS methodology of harmonizing common interfaces and refactoring the implementations to composable and interchangeable components. Once, such a basis has been established additional features can be added like support for objects with kinematics or working dynamic simulations.

Code generation from metamodels and grammar: Currently, the GDDL repository and the GDDL grasp planner plugin communicate via an intermediate YAML configuration format. To interpret GDDL directly in the plugin `C++` code should be generated from the metamodels and also an according parser from GDDL's Xtext grammar. However, as of now, EMF only supports the generation of Java code. A candidate that should be investigated closer in this context is EMF4CPP project which includes tools to use EMF and Xtext in a `C++` environment.

Bibliography

[1] Iso 31: Quantities and units, 1993.

[2] Iso/iec 14977: Information technology - syntactic metalanguage - extended bnf, 1996.

[3] John R. Jr. Amend, Eric Brown, Nicholas Rodenberg, Heinrich M. Jaeger, and Hod Lipson. A positive pressure universal gripper based on the jamming of granular material. *IEEE Transactions on Robotics*, 28:341–350, 2012.

[4] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Software*, 20:36–41, 2003.

[5] Uwe Aßmann and Steffen Zschaler. *Ontologies for Software Engineering and Software Technology*, chapter Ontologies, Meta-models, and the Model-Driven Paradigm, pages 249–273. Springer, 2006.

[6] Tim Baier and Jianwei Zhang. Reusability-based semantics for grasp evaluation in context of service robotics. In *IEEE International Conference on Robotics and Biomimetics*, 2006.

[7] Carl Barck-Holst, Maria Ralph, Fredrik Holmar, and Danica Kragic. Learning grasping affordance using probabilistic and ontological approaches. In *International Conference on Advanced Robotics*, 2009.

[8] Grigory Isaakovich Barenblatt. *Scaling, self-similarity, and intermediate asymptotics*. Cambridge University Press, 1996.

[9] Yasemin Bekiroglu, Kai Huebner, and Danica Kragic. Integrating grasp planning with online stability assessment using tactile sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[10] Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, 2007.

[11] Antonio Bicchi. On the closure properties of robotic grasping. *International Journal of Robotics Research*, 14:319–334, 1995.

[12] Antonio Bicchi. Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation*, 16:652–662, 2000.

[13] Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergström, Danica Kragic, and Antonio Morales. Mind the gap - robotic grasping under incomplete observation. In *IEEE International Conference on Robotics and Automation, 2011 (ICRA 2011)*, 2011.

[14] Jeannette Bohg, Kai Welke, Beatriz Léon, Martin Do, Dan Song, Walter Wohlkinger, Marianna Madry, Aitor Aldóma, Markus Przybylski, Tamim Asfour, Higinio Martí, Danica Kragic, Antonio Morales, and Markus Vincze. Task-based grasp adaptation on a humanoid robot. In *10th IFAC Symposium on Robot Control*, 2012.

[15] Christoph Borst, Max Fischer, and Gerd Hirzinger. Grasp planning: How to choose a suitable task wrench space. In *IEEE International Conference on Robotics and Automation, 2004 (ICRA 2004)*, 2004.

[16] Herman Bruyninckx and Joris De Schutter. Specification of force-controlled actions in the "task frame formalism": A synthesis. *IEEE Transactions on Robotics and Automation*, 12:581–589, 1999.

[17] Mark R. Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5:269–279, 1989.

[18] Mark R. Cutkosky and Robert D. Howe. *Dextrous robot hands*, chapter Human grasp choice and robotic grasp analysis, pages 5–31. Springer-Verlag New York, Inc., 1990.

[19] Hao Dang and Peter K. Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[20] Charles de Granville and Andrew H. Fagg. Learning grasp affordances through human demonstration. Submitted to the Journal of Autonomous Robots, 2010.

[21] Tinne De Laet, Steven Bellens, Herman Bruyninckx, and Joris De Schutter. Geometric relations between rigid bodies (part 2) - from semantics to software. *IEEE Robotics & Automation Magazine*, 2012.

[22] Tinne De Laet, Steven Bellens, Ruben Smits, Erwin Aertbelien, Herman Bruyninckx, and Joris De Schutter. Geometric relations between rigid bodies: Semantics for standardization. *IEEE Robotics & Automation Magazine*, 2012.

[23] Deutsche Servicerobotik Initiative (DESIRE). Schunk anthropomorphic hand, 2007. [Online]. Available: `http://www.service-robotik-initiative.de/gfx/plattform/Produktbild_SAH_download_gross.jpg`. Last accessed on May 3nd, 2013.

[24] Deutsche Servicerobotik Initiative (DESIRE). Schunk dextrous hand, 2007. [Online]. Available: `http://www.service-robotik-initiative.de/gfx/plattform/Profi_SDH_download_gross.jpg`. Last accessed on May 3nd, 2013.

[25] Renaud Detry, Carl Henrik Ek, Marianna Madry, Justus Piater, and Danica Kragic. Generalizing grasps across partly similar objects. In *IEEE International Conference on Robotics and Automation, 2012 (ICRA 2012)*, 2012.

[26] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, 2008.

[27] Mehmet R. Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha S. Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and Systems*, 2012.

[28] Thomas Feix, Roland Pawlik, Heinz-Bodo Schmiedmayer, Javier Romero, and Danica Kragic. A comprehensive grasp taxonomy. In *Robotics, Science and Systems: Workshop on Understanding the Human Hand for Advancing Robotic Manipulation*, 2009.

[29] Carlo Ferrari and John Canny. Planning optimal grasps. In *IEEE International Conference on Robotics and Automation, 1992 (ICRA 1992)*, 1992.

[30] M. Fischer and G. Hirzinger. Fast planning of precision grasps for 3d objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 1997 (IROS 1997)*, 1997.

[31] Marcus P. Foster. Disambiguating the si notation would guarantee its correct parsing. *Proceedings of the Royal Society*, 465:1227–1229, 2009.

[32] Martin Fowler and Rebecca Parsons. *Domain Specific Languages*. Addison-Wesley, 2010.

[33] Malik Ghallab, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl - the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[34] James J. Gibson. The theory of affordances. In *Percieving, Acting and Knowing*. Lawrence Erlbaum Associates, 1979.

[35] SCHUNK Mobile Greifsysteme GmbH, 2012. [Online]. Available: `http://mobile.schunk-microsite.com/en/produkte/produkte/servoelektrische-3-finger-greifhand-sdh.html`. Last accessed on April 13th, 2013.

[36] Corey Goldfeder and Peter K. Allen. Data-driven grasping. *Autonomous Robots*, 31:1–20, 2011.

[37] Kensuke Harada, Kenji Kaneko, and Fumio Kanehiro. Fast grasp planning for hand/arm systems based on convex model. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[38] Robert Haschke, Jochen J. Steil, Ingo Steuwer, and Helge Ritter. Task-oriented quality measures for dextrous grasping. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 2005.

[39] Tucker Hermans, James M. Rehg, and Aaron Bobick. Affordance prediction via learned object attributes. In *IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*, 2011.

[40] Ulrich Hillenbrand and Maximo A. Roa. Predicting functional grasps through contact warping and local replanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[41] Nico Hochgeschwender, Luca Gherardi, Azamat Shakhirmardanov, Gerhard K. Kraetzschmar, Davide Brugali, and Herman Bruyninckx. A model-based approach to software deployment in robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013. [Submitted].

[42] Ralph Hodgson and Paul J. Keller. Qudt - quantities, units, dimensions and data types in owl and xml, 2011. [Online]. Available: `http://www.qudt.org/`. Last accessed on April 16th, 2013.

[43] Robert D. Howe and Mark R. Cutkosky. Practical force-motion models for sliding manipulation. *International Journal of Robotics Research*, 15:557–572, 1996.

[44] Kaijen Hsiao, Sachin Chitta, Matei Ciocarlie, and E. Gil Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010 (IROS 2010)*, 2010.

[45] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Task-driven tactile exploration. In *Robotics: Science and Systems Conference*, 2010.

[46] Kai Huebner, Kai Welke, Markus Przybylski, Nikolaus Vahrenkamp, Tamim Asfour, Danica Kragic, and Rüdiger Dillmann. Grasping known objects with humanoid robots: A box-based approach. In *International Conference on Advanced Robotics, 2009. (ICAR 2009)*, 2009.

[47] Thea Iberall. The nature of human prehension: Three dextrous hands in one. In *IEEE International Conference on Robotics and Automation, 1987 (ICRA 1987)*, 1987.

[48] Thea Iberall. Human prehension and dexterous robot hands. *International Journal of Robotics Research (IJRR)*, 16:285–299, 1997.

[49] Wikipedia: The Free Encyclopedia. Wikimedia Foundation Inc. File:kitchen knives.svg, 2009. [Online]. CC BY-SA 3.0. Available: `http://en.wikipedia.org/wiki/File:Kitchen_knives.svg`. Last accessed on May 2nd, 2013.

[50] Sing Bing Kang and Katsushi Ikeuchi. Grasp recognition using the contact web. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1992.

[51] Moslem Kazemi, Jean-Sebastien Valois, J. Andrew (Drew) Bagnell, and Nancy Pollard. Robust object grasping using force compliant motion primitives. In *Robotics: Science and Systems*, 2012.

[52] David Kirkpatrick, Bhubaneswar Mishra, and Chee-Keng Yap. Quantitative steinitz's theorems with applications to multifingered grasping. *Discrete & Computational Geometry*, 7:295–318, 1992.

[53] Markus Klotzbücher, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Reusable hybrid force-velocity controlled motion specifications with executable domain specific languages. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011 (IROS 2011)*, 2011.

[54] Lars Kunze, Tobias Roehm, and Michael Beetz. Towards semantic robot description languages. In *IEEE International Conference on Robotics and Automation*, 2011.

[55] Janne Laaksonen, Ville Kyrki, and Danica Kragic. Evaluation of feature representation and machine learning methods in grasp stability learning. In *IEEE-RAS International Conference on Humanoid Robots*, 2010.

[56] Yanmei Li and Imin Kao. A review of modeling of soft-contact fingers and stiffness control for dextrous manipulation in robotics. In *IEEE International Conference on Robotics and Automation, 2001 (ICRA 2001)*, 2001.

[57] Ying Li, Jiaxin L. Fu, and Nancy S. Pollard. Data driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on Visualization and Computer Graphics*, 13:732–747, 2007.

[58] Zexiang Li and S. Shankar Sastry. Task-oriented optimal grasping by multifingered robot hands. *IEEE Journal of Robotics and Automation*, 4:32–44, 1988.

[59] John Lin, Ying Wu, and Thomas S. Huang. Modeling the constraints of human hand motion. In *Workshop on Human Motion. Proceedings*, 2000.

[60] Meka Robotics LLC. G2 compliant gripper, 2013. [Online]. Available: `http://mekabot.com/wp-content/uploads/gripper_a02.jpg`. Last accessed on May 3nd, 2013.

[61] Efrain Lopez-Damian, Daniel Sidobre, and Rachid Alami. A grasp planner based on inertial properties. In *IEEE International Conference on Robotics and Automation, 2005 (ICRA 2005)*, 2005.

[62] Matthew T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems Man and Cybernetics*, 11:418–432, 1981.

[63] Matthew T. Mason and J. Kenneth Salisbury. *Robot Hands and the Mechanics of Manipulation.* MIT Press, 1985.

[64] Giulia C. Matrone, Christian Cipriani, Emanuele L. Secco, Giovanni Magenes, and Maria Chiara Carrozza. Principal components analysis based control of a multi-dof underactuated prosthetic hand. *Journal of NeuroEngineering and Rehabilitation*, 7:1–13, 2010.

[65] Wim Meeussen, John Hsu, and Rosen Diankov. Urdf, 2013. [Online]. CC BY 3.0. Available at: `http://ros.org/wiki/urdf`. Last accessed on April 6th, 2013.

[66] Edmund Milke, Stefan Christen, Erwin Prassler, and Walter Nowak. Towards harmonization and refactoring of mobile manipulation algorithms. In *International Conference on Advanced Robotics (ICAR)*, 2009.

[67] Andrew Miller and Peter K. Allen. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11:110–122, 2004.

[68] Andrew T. Miller, Steffen Knoop, Henrik I. Christensen, and Peter K. Allen. Automatic grasp planning using shape primitives. In *IEEE International Conference on Robotics and Automation, 2003 (ICRA 2003)*, 2003.

[69] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

[70] Antonio Morales, Mario Prats, Pedro Sanz, and Angel P. Pobil. An experiment in the use of manipulation primitives and tactile perception for reactive grasping. In *Robotics: Science and Systems (RSS). Workshop on Robot Manipulation: Sensing and Adapting to the Real World*, 2007.

[71] Thomas Moulard. Urdf validator, 2012. [Online]. Available: `https://github.com/laas/urdf_validator/blob/3d3a4b1758c5193aabe09b154a929d33b574d0a7/urdf.xsd`. Last accessed on April 6th, 2013.

[72] John R. Napier. The prehensile movements of the human hand. *Journal of Bone and Joint Surgery*, 38:902–913, 1956.

[73] Thang N. Nguyen and Hany E. Stephanou. A topological algorithm for continuous grasp planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1990.

[74] Thang N. Nguyen and Harry E. Stephanou. *An introduction to intelligent and autonomous control*, chapter Intelligent robot prehension, pages 319–347. Kluwer Academic Publishers, 1993.

[75] Donald A. Norman. *The design of everyday things.* Doubleday, 1988.

[76] Allison M. Okamura, Niels Smaby, and Mark R. Cutkosky. An overview of dexterous manipulation. In *IEEE International Conference on Robotics and Automation, 2000 (ICRA 2000)*, 2000.

[77] Object Management Group (OMG). Omg meta object facility (mof) core specification. version 2.4.1, 2011.

[78] Object Management Group (OMG). Omg object constraint language (ocl). version 2.3.1, 2012.

[79] Nancy S. Pollard. Synthesizing grasps from generalized prototypes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[80] Mario Prats, Pedro J. Sanz, and Angel P. del Pobil. Task-oriented grasping using hand preshapes and task frames. In *IEEE International Conference on Robotics and Automation, 2007 (ICRA 2007)*, 2007.

[81] Mario Prats, Pedro J. Sanz, and Angel P. del Pobil. A framework for compliant physical interaction based on multisensor information. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2008.

[82] Domenico Prattichizzo and Jeffrey C. Trinkle. Grasping. In *Handbook on Robotics.* Springer, 2008.

[83] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation, 2009 (ICRA 2009). Workshop on Open Source Software*, 2009.

[84] Hajo Rijgersberg and Jan Top. Unitdim: an ontology of physical units and quantities. Technical report, Wageningen : Agrotechnology & Food, 2005.

[85] Máximo A. Roa and Raúl Suárez. Geometrical approach for grasp synthesis on discretized 3d objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007 (IROS 2007)*, 2007.

[86] Steffen W. Ruehl, Andreas Hermann, Zhixing Xue, Thilo Kerscher, and Rudiger Dillmann. Graspability: A description of work surfaces for planning of robot manipulation sequences. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[87] Andy Ruina and Pratap Rudra. *Introduction to Statics and Dynamics.* Oxford University Press, 2002.

[88] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60:326–336, 2011.

[89] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research (IJRR)*, 27:157–173, 2008.

[90] Matthias C. Schabel and Steven Watanabe. Chapter 34. boost.units 1.1.0, 2003. [Online]. Available: `http://www.boost.org/doc/libs/1_53_0/doc/html/boost_units.html`. Last accessed on April 16th, 2013.

[91] Andrés Senac González, Diego Sevilla Ruiz, and Gregorio Martinez Perez. Emf4cpp: a c++ ecore implementation. In *Desarrollo de Software Dirigido por Modelos (DSDM), Jornadas de Ingenieria del Software y Bases de Datos (JISBD)*, 2010.

[92] Karun B. Shimoga. Robot grasp synthesis algorithms: A survey. *International Journal of Robotics Research (IJRR)*, 15:230–266, 1996.

[93] Ruben Smits. Kdl: Kinematics and dynamics library, 2001. [Online]. Available: `http://www.orocos.org/kdl`. Last accessed on April 6th, 2013.

[94] Ahmet Soylu, Patrick De Causmaecker, Davy Preuveneers, Yolande Berbers, and Piet Desmet. Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: a meta-review. *International Journal of Metadata, Semantics and Ontologies*, 6:96–125, 2011.

[95] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework, Second Edition.* Addison-Wesley, 2008.

[96] Alexander Stoytchev. Behavior-grounded representation of tool affordances. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[97] Nikolaus Vahrenkamp, Manfred Kröhnert, Stefan Ulbrich, Tamim Asfour, Giorgio Metta, Rüdiger Dillmann, and Giulio Sandini. A robotics toolbox for simulation, motion and grasp planning. In *International Conference on Intelligent Autonomous Systems (IAS)*, 2012.

[98] Karthik Mahesh Varadarajan and Markus Vincze. Afrob: The affordance network ontology for robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[99] Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Haussermann, R.J.M. Janssen, J.M.M. Montiel, Alexander Perzylo, Bjoern Schiessle, Moritz Tenorth, Oliver Zweigle, and M.J.G. van de Molengraft. Roboearth - a world wide web for robots. *IEEE Robotics & Automation Magazine*, 18:69–82, 2011.

[100] Jonathan Weisz and Peter K. Allen. Pose error robust grasping from contact wrench space metrics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[101] David Wren and Robert B. Fisher. Dextrous hand grasping strategies using preshapes and digit trajectories. In *IEEE International Conference on Systems, Man and Cybernetics*, 1995.

[102] Zhixing Xue, Alexander Kasper, J. Marius Zoellner, and Ruediger Dillmann. An automatic grasp planning system for service robots. In *International Conference on Advanced Robotics, 2009. ICAR 2009*, 2009.

[103] Zhixing Xue, J. Marius Zoellner, and Ruediger Dillmann. Grasp planning: Find the contact points. In *International Conference on Robotics and Biomimetics*, 2007.

[104] Zhixing Xue, J. Marius Zoellner, and Ruediger Dillmann. Planning regrasp operations for a multifingered robotic hand. In *IEEE International Conference on Automation Science and Engineering*, 2008.

[105] Nicholas Xydas and Imin Kao. Modeling of contact mechanics and friction limit surfaces for soft fingers in robotics, with experimental results. *International Journal of Robotics Research (IJRR)*, 18:941–950, 1999.

[106] Raoul Zöllner, Michael Pardowitz, Steffen Knoop, and Rüdiger Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

A. URDF model

Kinematics of objects and manipulators reuse definitions from URDF XML documents. An XSD schema for URDF documents can be found at [71]. This XSD metamodel has been transformed into an Ecore metamodel. Figure A.1 depicts the most relevant elements of the Ecore metamodel.

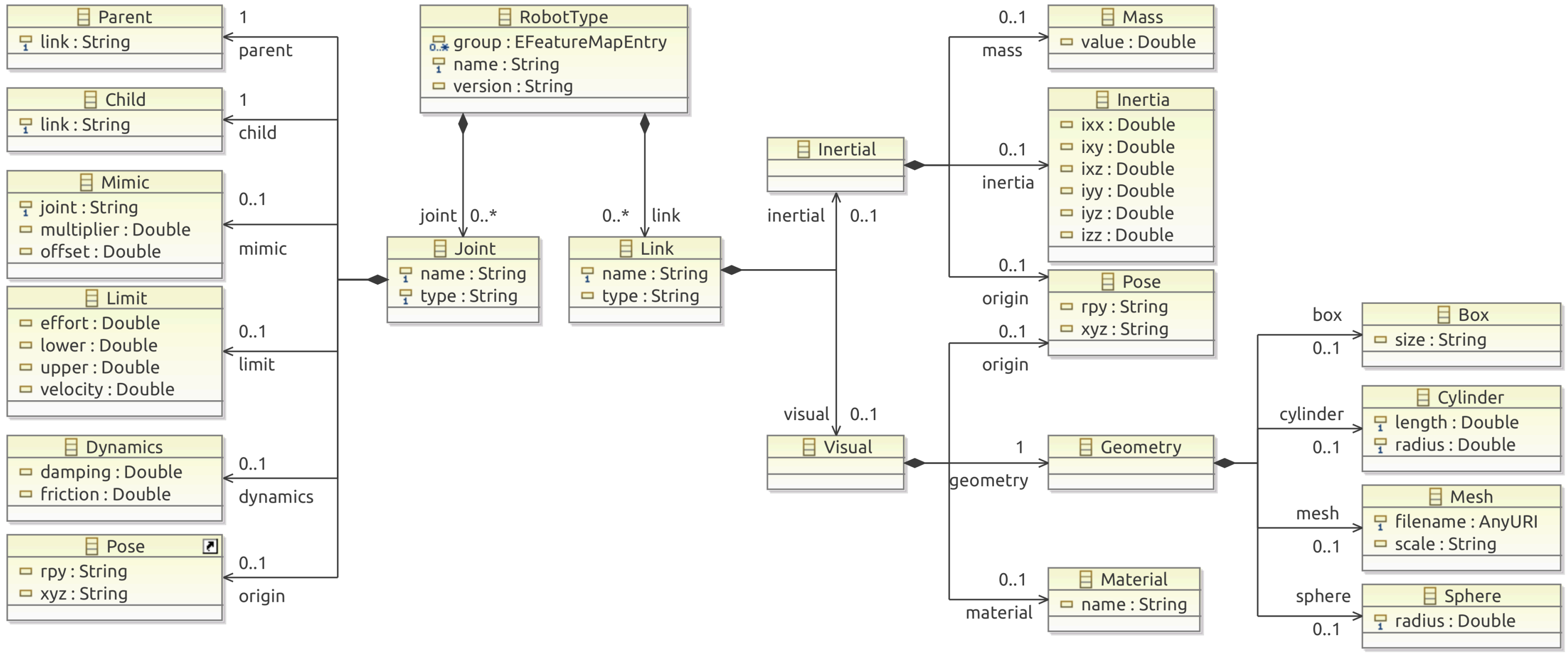


Figure A.1.: The diagram of the URDF model. Derived from the URDF XSD schema at [71].

B. Contents of the attached CD

- This document as PDF
- Eclipse projects containing the formal Ecore domain models with OCL constraints and Xtext grammars
- Proof of concept implementation of the GraspIt! plugin
- Document containing title, abstract, supervisors' names and student name
- BibTeX entry of the thesis

C. ICRA workshop paper

The following extended abstract paper has been accepted for a poster presentation on the Workshop on Combining Task and Motion Planning of the IEEE International Conference on Robotics and Automation (ICRA 2013).

Towards a Declarative Grasp Specification Language

Sven Schneider and Nico Hochgeschwender[1]

I. INTRODUCTION

With the advent of mobile manipulation platforms the integration of grasp planning techniques into robot control architectures becomes not only technically appealing but also a required approach in order to build service robots capable of performing a wide range of tasks. To grasp an object in a task-dependent manner a robot needs to compose knowledge from several domains. For example, a robot which is instructed to water a plant (the *task* domain) with a spray bottle needs to know where to grasp the bottle (the *object* domain) in such a manner that a certain hand configuration (the *hand* domain) is able to pull the trigger. In the following we will present our work in progress towards a declarative language which allows to encode information about the different domains for the sake of grasping. The resulting domain-specific language called Grasp Domain Definition Language (GDDL) allows to encode grasping problems and to check constraints (e.g., a spray bottle can't be pulled with a two-finger gripper) already at design time. The objective of GDDL is similar to PDDL [1], namely to specify problems in a planner-independent manner. Similarly to PDDL, GDDL can be used to specify benchmarking problems for a wide range of grasp planners and as an intermediate knowledge representation in task-oriented robot control architectures.

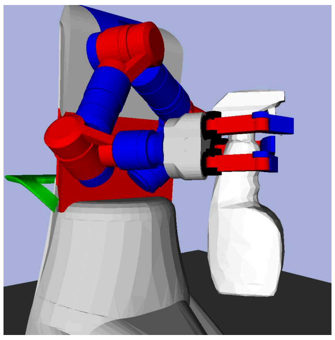


Fig. 1. The domestic service robot Care-O-bot 3 using a spray bottle in simulation.

II. PROBLEM FORMULATION

Research on grasping produced in the past a huge body of seminal work focusing on the different domains involved in grasping, e.g. [2] on grasp semantics or the combination of both the hand and the object domain (hand-object interaction) in [3][4]. Quite recently, the hand, object and task domain has been addressed in an integrated manner using machine learning methods (see Bohg *et al.* [5]). We argue that this *integrated approach* is required to build robots capable of performing a wide range of tasks. However, this integrated approach is challenging as there is a huge variability in each domain. Ranging from two-finger grippers to anthropomorphic hands to objects like needles and power drills and tasks such as transportation, pouring and tool-use. In spite of this variability, most entities in each domain have common features. Our *research hypothesis* is that by identifying and formalizing these features they can be reused more easily instead of developing specialized approaches per hand, object and task.

III. APPROACH

To describe the domains we apply a model-driven engineering approach using the Eclipse Modeling Framework (EMF)[1]. Here, each domain is specified in the form of a model. The next sections describe the domains and features that need to be captured by GDDL in more detail.

A. Object domain

Information about objects can be classified into two categories: Measurable physical properties and semantic properties. For grasping the object's surface description and its dynamics like mass and inertia are of main interest. The surface description consists of the object's shape as a computer-aided design (CAD) model and the material which includes friction coefficients. A grasp planner uses the physical properties to evaluate the stability of a grasp.

Semantic properties cannot be measured and hence are provided by domain experts. In many devices tool-use actions are associated with the activation of joints such as pulling the spray bottle's trigger. To simplify the description of these tasks the semantic properties include the object's kinematic structure like the trigger joint of the spray bottle. An additional semantic property are the object's affordances specifying the tasks in which the object may be applied. For example the spray bottle can be transported which implies a *movable* affordance and spray liquid i.e. it has a *sprayable* affordance. When the robot uses the spray bottle none of its components should be in front of the nozzle. This is described by attaching a primitive shape such as a cone to the nozzle that is labeled as *forbidden* area. In a similar manner certain links like the trigger can be marked as *required*

[1]Sven Schneider and Nico Hochgeschwender are with the Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences, Germany. Email: sven.schneider@inf.h-brs.de, nico.hochgeschwender@h-brs.de

[1]http://www.eclipse.org/modeling/emf/

regions. Figure 2 depicts an excerpt of the object domain model with the previously presented features.

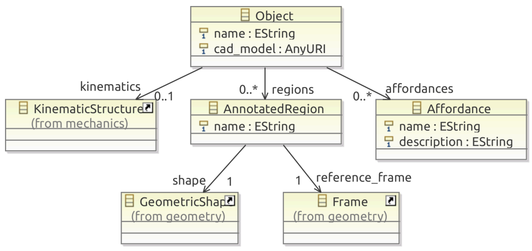


Fig. 2. A formal model showing an excerpt of the object domain.

B. Task domain

The task domain comprises all manipulation actions that the robot performs with the object after grasping. In the plant watering use case these manipulation actions are transporting the spray to a desired pose and pulling the trigger. For grasping the view on the manipulation actions is limited to wrenches (forces and torques) because a grasp must either resist external wrenches as in transport tasks or apply wrenches for example when pulling the trigger. Thus, in the task domain the manipulation actions are decomposed into primitive actions like *exert force* or *exert torque*. The task domain also specifies which affordances an object must provide to be used in the task.

C. Hand domain

Similarly to the object domain the hand's features can be categorized into physical and semantic properties. The physical properties are equal to those in the object domain i.e. the hand's shape description and its surface material. While the hand's kinematic structure is an important feature of the semantic properties the main feature is a mapping from an abstract grasp description to a grasp realization with a specific hand. An example of such a mapping is that a cylindrical grasp corresponds to a certain set of joint angles. Another feature that describes a grasp abstractly besides the shape is the dexterity and resistance to external disturbances such as precision or power grasps.

D. Composition of the domains

The previous three domains are independent of each other. In order to generate a grasp instance for a specific object, task and hand these independent domains are combined. The composition of the object domain with the task domain yields a hand-independent grasp specification which can be annotated with hand approach directions. For instance, to use the spray bottle it should be approached from the rear near the bottle's handle. Two finger sub-configurations are required: A cylindrical power grasp around the bottle's handle to achieve the *transporting* task and a hook grasp for *pulling* the trigger. This hand-independent description of finger configurations is included in the grasp specification as well. Composing the grasp specification with a hand entity from the hand domain results in the grasp instance with concrete finger-joint angles. This grasp instance is the input for a grasp planner such as GraspIt! [6] which simulates the grasp and evaluates the stability of the resulting hand-object contacts.

E. Constraints

Designing models of the previous domains makes hidden assumptions explicit. For example, in GDDL all physical quantities consist of a value and a unit to allow for automatic conversion. Additionally, these explicit models enable the specification of formal constraints on entities in each domain which can validate the entities already at design time. For instance, this assures that only angular joint values are provided for revolute joints. Furthermore, the affordances provided by an object must match those affordances that are required in a task. In addition, this approach can validate that a grasp is potentially executable because a hand has at least as many fingers as are required in the task description. For example a gripper has too few fingers to carry a spray bottle and simultaneously spray water with it.

IV. CONCLUSIONS AND FUTURE WORK

We have presented our work in progress towards GDDL, a declarative language for grasp specification. Features in three major domains (object, task and hand domain) have been identified and outlined that are relevant for grasping. A grasp can be composed out of entities defined in each domain. As next step we will develop a repository of GDDL descriptions and integrate it with a task planner which selects object and task specifications at runtime. Evaluation criteria for GDDL are the reusability of domain concepts, the grasp quality regarding the specified task and the portability to other grasp planners.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the on-going support of the Bonn-Aachen International Center for Information Technology and the B-IT foundation.

REFERENCES

[1] M. Ghallab, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl - the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.
[2] T. Iberall, "Human prehension and dexterous robot hands," *International Journal of Robotics Research (IJRR)*, vol. 16, pp. 285–299, 1997.
[3] C. Ferrari and J. Canny, "Planning optimal grasps," in *Proc. IEEE International Conference on Robotics and Automation (ICRA 1992)*, 1992.
[4] C. Borst, M. Fischer, and G. Hirzinger, "Grasp planning: How to choose a suitable task wrench space," in *Proc. IEEE International Conference on Robotics and Automation (ICRA 2004)*, 2004.
[5] J. Bohg, K. Welke, B. Léon, M. Do, D. Song, W. Wohlkinger, M. Madry, A. Aldóma, M. Przybylski, T. Asfour, H. Martí, D. Kragic, A. Morales, and M. Vincze, "Task-based grasp adaptation on a humanoid robot," in *10th IFAC Symposium on Robot Control*, 2012.
[6] A. Miller and P. K. Allen, "Graspit!: A versatile simulator for robotic grasping," *IEEE Robotics and Automation Magazine*, vol. 11, pp. 110–122, 2004.