Development of a Graphics Ontology for Natural Language Interfaces

by

Mehdi Niknam

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Department of Computer Science
University of Manitoba
Winnipeg

Thesis advisor

Dr. Christel Kemke

Author

Mehdi Niknam

Development of a Graphics Ontology for Natural Language Interfaces

Abstract

The overall context of this thesis research is to explore natural language as a medium to interact with computer software in the graphics domain, e.g. programs like MS Paint or OpenGL. A core element of most natural language understanding systems is an ontology, which represents concepts and items of the underlying domain of discourse. In this thesis, I developed an ontology for the graphics domain based on several resources, including documentation and textbooks on graphics tools and systems, existing ontologies, and - most importantly - a collection of natural language instructions to create and modify graphic images. The ontology was developed and improved in several development phases, and finally tested as part of a natural language interface to a graphics tool. This complex natural language interface accepts verbal instructions in the graphics domain as input and creates suitable graphic images as output, via a mapping through the ontology. The results of our tests indicate an accuracy of the system in the area of 80%, based on a subset of the collected sentences as input and an assessment of the suitability of the created images through a feature-based measurement as well as human users.

Contents

List of Figures

Acknowledgments

This research work would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Dr. Christel Kemke who was abundantly helpful and offered invaluable assistance, support and guidance. My gratitude are also due to the members of the reviewing committee, Dr. Kevin Russell and Dr. Dean Jin who helped me with their knowledge and assistance to complete this thesis successfully. Special thanks go also to all my graduate friends who helped and supported me through discussion and advice. I would also like to convey thanks to the Ministry of Advanced Education and Literacy of Manitoba and Faculty of Science for providing some financial means to conduct my research. I would like to thank Jennifer Rausch for her enormous support and assistance through this thesis. I wish to express my love and gratitude to my family for their understanding and endless love through the duration of my studies.

This thesis is dedicated to my parents.

Chapter 1

Introduction

Currently, computer users have various graphics software and tools available. Some of the tools offer a mouse-based interface and are relatively easy to use for a manually-skilled person, while others are more complicated and require the user to learn a specific graphics programming language. It would be very convenient and helpful for a lot of users if graphics programs like Paint or Draw in Microsoft Windows worked with verbal instructions, instead of having to use a mouse and menu interface or a complicated programming language. The overall context of this thesis research is to explore natural language as a medium to interact with computer software in the graphics domain, e.g. programs like MS Paint. A core element of most natural language understanding systems is an ontology, which represents concepts and items of the underlying domain of discourse.

Nowadays, ontologies play an important role in natural language processing. First, they provide a concept dictionary as basis for the natural language processing. Second, they can be used to derive a semantic representation providing a deeper under-

standing of the user's verbal input. Third, their role is to present adequate knowledge of the specific domain of discourse, i.e. in this case the domain of interacting with graphics tools, which helps to clarify ambiguities and vagueness in natural language sentences and map the verbal input to specific, well-defined operations in terms of the graphics tool.

According to Gruber (Gruber, 1995), an ontology is an explicit specification of a conceptualization. Conceptualization refers to different objects and concepts and the relationships between them that are supposed to exist in the domain. For example, in the graphics domain, *line*, *square*, *circle*, etc. are core objects; they can have certain attributes like *colour* and *size*, and spatial relations with each other, like *left of*, *right of*, *below*, and so on. Most importantly, there are different actions in the graphics domain such as *draw*, *erase*, and *position* to create graphic images.

Significant work and major developments in the area of ontology research and the use of ontologies in natural language processing have taken place in the past ten years or so. However, there was no ontology available for the graphics domain, and there has been only marginal research done on natural language interaction in connection with graphics systems.

In this thesis, I developed an ontology for the graphics domain based on several resources, including documentation and textbooks on graphics tools and systems, existing ontologies, and - most importantly - a collection of natural language instructions to create and modify graphic images.

The ontology was developed and improved in several phases, and finally tested as part of a natural language interface to a graphics tool. This complex natural

language interface accepts verbal instructions in the graphics domain as input and creates suitable graphic images as output, via a mapping through the ontology.

The results of the tests indicate an accuracy and correctness of the system generated images in the range of 75% to over 90%. The tests used a subset of the collected sentences as inputs and assessments of the suitability of the created images were performed through a feature-based measurement as well as through an evaluation by human users.

This thesis work includes the development of a useful application, particularly for people with physical restrictions, who cannot easily use standard manual interfaces, and those who are not familiar enough with computers to work easily and directly with graphics software.

I believe that this research provides a major contribution towards the development of ontologies and of natural language interfaces for the graphics domain, since the ontology can be used in other natural language interfaces for graphics systems, it can be easily adapted to various graphics tools, and it could even be modified for other domains, like computer aided design systems or geographical information systems.

Chapter 2

Background

Based on the thesis topic, I reviewed literature in the area of ontology research to obtain a clear understanding of the concepts and terminology it uses, and the different methods for building an ontology. The following sections provide a summary of this review.

2.1 Ontology—Basic Concepts

In this section, basic concepts of ontology research are discussed, including definitions of ontology, various types of ontologies, and the purposes of ontologies.

2.1.1 Definitions of Ontology

In the field of philosophy, ontology is the study of things that exist. Research about ontology started in computer science in the 1980s. However, computer scientists have not come to an agreement about the definition of ontology. The most accepted

definition of ontology among computer scientists was given in 1993 by Thomas R. Gruber. Gruber (Gruber, 1995) defines ontology as "an explicit specification of a conceptualization." He defines conceptualization as an abstract, simplified view of a part of the world that we want to represent. The conceptualization is based on the objects, concepts, and other entities that are supposed to exist in a domain and the relationships between them.

Guarino (Guarino, 1997) presents six different definitions of ontology that are used in the related literature besides Gruber's definition. He argues and compares those definitions to present his definition of ontology as "a logical theory that constrains the intended models of a logical language."

Chandrasekaran et al. (Chandrasekaran et al., 1999) state that "ontologies are content theories about the sort of objects, properties of objects and relations between objects that are possible in a specified domain of knowledge." They describe two different views of the meaning of ontology in artificial intelligence. Firstly, an ontology is "a representation vocabulary, specialized to some domain or subject matter." To be more precise, the conceptualizations that are associated with the terms in the vocabulary constitute the ontology. Secondly, ontology can be seen as "a body of knowledge describing some domain." In this view, the terms in the representation vocabulary are used to describe facts in the domain.

2.1.2 Ontology Types

There are various classifications of ontologies in computer science according to different aspects of ontologies. Uschold (Uschold, 1996) classified ontologies based

on the level of formality. Highly informal ontologies use natural language to express knowledge. Structured informal ontologies use a restricted form of natural language. Semi-formal ontologies use an artificial formal language. Finally, rigorously formal ontologies define terms in a formal language with formal semantics.

Chandrasekaran et al. (Chandrasekaran et al., 1999) divide ontologies into two types based on the level of generality: General-purpose ontologies and domain ontologies. A general-purpose ontology includes the whole world, all objects, concepts and the relationships between them. The Upper CYC ontology (Cycorp, 2008) and the Suggested Upper Merged Ontology (SUMO) (The Standard Upper Ontology Working Group, 2008) are examples of general ontologies. A domain ontology includes information about a specific domain. An example of a domain ontology is TOVE (University of Toronto, 2008), which is designed for the commercial and public enterprises domain.

Fensel (Fensel, 2004) provides another classification of ontologies into five different categories. First, domain ontologies contain knowledge for a specific domain (e.g. electronic, medical). Metadata ontologies contain vocabulary to describe the content of information sources (e.g. Dublin Core (The Dublin Core Metadata Initiative Organization, 2009)). Generic ontologies and common sense ontologies contain general knowledge about the world, providing basic concepts for things like time, space, state, event, etc., that can be used in different domains. Representational ontologies provide representational entities without stating what should be represented. (e.g. Frame-Ontology (Gruber, 1993), which provides concepts such as frames, slots, and slot constraints). Lastly, method and task ontologies provide terms for specific tasks

(e.g. "hypothesis" is related to the diagnosis task ontology) and problem-solving methods (e.g. "correct state" is related to the propose-and-revise method ontology).

The ontology that I developed is a domain ontology since it covers the concepts and relationships of a specific domain, the graphics domain. Also, I utilized Protégé-Frame editor to implement the ontology. Protégé-Frame editor uses a formal language to formalize the ontology. Consequently, the ontology is a rigorously formal ontology.

2.1.3 Purpose of Usage

The major purpose of object-oriented theory in software design is to reuse objects and their attributes in other applications. Similarly, one main aim of using ontologies in knowledge representation is to be able to share and reuse knowledge about a domain. A well-developed ontology can be used by different applications, which, thus, can share the stored knowledge.

2.2 Ontology Development Methodologies

There are different approaches used to assist or guide the development of an ontology. However, there is currently no standard methodology for developing ontologies. In this section, I describe the prevalent methods, which have been designed so far for ontology development.

Lenat and Guha (Lenat and Guha, 1990) developed the Cyc ontology from their experience during the development of the Cyc knowledge-base system. First, they manually extracted the knowledge, and then proceeded to code the knowledge that was stored in the knowledge-base system. The Cyc ontology was developed using the

most abstract concepts from the knowledge base as an upper-level general ontology. In addition, the Cyc ontology represents knowledge for different domains. Lenat and Guha implemented the ontology in the language CycL (Cycorp, 2008).

Uschold and King (Uschold and King, 1995) introduce an enterprise methodology for developing ontologies. Their method is divided into four parts: identifying the purpose, building the ontology, evaluation, and documentation. "Identifying the purpose" refers to answering questions such as why the ontology should be built and what are the uses of the ontology. Recognizing the range of users who will use the ontology is also one of the aims of identifying the purpose. The next step is building the ontology, which is divided into three phases: ontology capture, ontology coding, and integrating existing ontologies. "Ontology capture" means recognizing the key concepts and relationships in the domain, creating unambiguous text definition for the key concepts and the relationships among them, and identifying terms for referring to key concepts and relationships. In the coding phase, we choose a representation language to create a code for the explicit representation of the concepts that are captured in the previous phase. The next phase in the building step is integrating existing ontologies. An enterprise ontology can be developed from an existing ontology. The third step is evaluation, which makes certain that the ontology covers all requirements of the domain. Uschold and King suggest adopting the evaluation approaches in the field of knowledge-base systems for ontologies.

Grüninger and Fox (Grüninger and Fox, 1995) present a methodology that they used for developing the TOronto Virtual Enterprise (TOVE) ontology. The methodology comprises six steps. The first step is to recognize the motivating scenario,

that is, finding the reasons for developing the ontology and identifying the intended users. The next step is to design questions related to the motivating scenario. These questions are called "informal competency questions", since the ontology should be able to answer them. The third step is to create objects, attributes and relationships in first order logic. After that, the developer converts the informal competency questions to formal ones using first order logic. The fifth step is specifying axioms. Axioms define terms and constraints of the domain in first order logic. The axioms have to be sufficient to answer the competency questions. The last step is to develop completeness theorems that guarantee that there will be complete solutions to the competency questions.

Uschold (Uschold, 1996) offers a unified methodology for building ontologies by reviewing the Enterprise and the TOVE building methodologies. The unified methodology has four phases: purpose, level of formality, scoping and building the ontology. In the purpose phase, the developer answers questions about the kinds of users that will use the ontology. Also, similar to the TOVE ontology methodology, the developer modifies motivating scenarios and competency questions to clarify specific uses and mechanisms of the ontology. Uschold believes that identifying the purpose of ontology is extremely important for creating a reasonable ontology. The next step is recognizing the level of formality for describing the ontology. The level of formality depends on the type of purpose and the kind of user who want to use the ontology. In the scoping step, by using motivating scenarios, competency questions, brainstorming and trimming, the developer collects a set of concepts and terms that the ontology should characterize. For the building step, Uschold recommends four approaches.

The first approach is to ignore the previous steps and start to build the ontology with an ontology editor that is suitable for designing prototypes and small ontologies. The second approach is to perform the previous steps and use the results for formal coding; this approach is suitable for larger ontologies. The third approach is to create an intermediate document that needs formal coding to derive the final ontology. The fourth approach is to create a set of formal terms from the informal ones in order to define the axioms and definitions for the ontology. Uschold suggests some general criteria such as clarity, consistency and reusability to evaluate ontologies as well as a specific evaluation by making sure that the ontology is able to answer all competency questions.

Fernández López et al. (López et al., 1999) present a methodology called "Methontology" for building an ontology. Methontology includes three major activities: project management, development-oriented activities, and support. In project management, the first activity is planning an outline for all tasks in order to perform the project. Next is controlling the performance of the planned tasks. Finally, there needs to be quality assurance to make sure that all the outputs are acceptable. Development-oriented activities are divided into four sub activities. Specification is recognizing a purpose and the users of an ontology. Conceptualization is creating a conceptual model for the domain of interest. Formalization is converting the conceptual model to a formal model. Implementation is building the computational model from the formal model. In the end, the process of maintenance will keep the ontology up-to-date. Support activities are performed to coincide with the development-oriented activities. Support activities include knowledge acquisition,

evaluation, integration, documentation, and configuration management. The knowledge acquisition activity gathers knowledge from the domain. The evaluation creates a technical judgment of the proceedings for each phase. The integration part uses an existing ontology to build a new one. The documentation registers all details of the activities in different phases. Finally, configuration management records all versions of documentation to control new changes.

Sugumaran and Storey (Sugumaran and Storey, 2002) offer the four step heuristics-based ontology-creation methodology. The first step is identifying basic terms in the domain, which can be done by finding synonyms and using use cases. Use cases are text descriptions for concepts in the domain. In the next step, the developer identifies the three types of relationships between the terms in the domain: generalization, synonym and association. Also, if there are sub-ontologies, identifying the relationships between the terms in the sub-ontologies is necessary. The third step is identification of basic constraints. There are four types of constraints: pre-requisite constraints, temporal constraints, mutually inclusive constraints and mutually exclusive constraints. The last step is identification of higher-level constraints capturing domain knowledge. The aim of this step is to recognize constraints related to terms in the domain and constraints related to several terms or relationships.

Noy and McGuinness (Noy and McGuinness, 2001) present a methodology for ontology development that has seven steps:

1. Determine the domain and scope of the ontology: Noy and McGuinness suggest identifying the domain of coverage by the ontology, the reason for using the ontology, questions that the ontology should provide answers for, and the users

and maintainers of the ontology.

2. Consider reusing existing ontologies: The purpose of this step is to search for existing ontologies in the same domain in order to import them into the development environment.

3. Enumerate important terms in the ontology: This phase builds the terminology for the ontology.

4. Define the classes and the class hierarchy: The classes correspond to concepts in the domain. We will define those classes by selecting significant terms that have been identified in the pervious step. For example, the class polygon represents the concept polygon in the graphics domain, which has "square" as sub-class and "two-dimensional object" as super-class. There are three methods for constructing the hierarchy: top-down means starting from the most general classes, bottom-up means starting from the most specific classes and middle-out means starting from the important class in the middle and extending the hierarchy in both directions.

5. Define the properties of classes through slots: Some terms that have been identified in phase three describe properties of classes and will be defined using slots. However, these terms are properties of class that should be defined as slots. For example, "size" or "colour" can be used to describe the properties of graphics objects; we can define size and colour as slots for different classes of graphics objects such as the polygon class.

6. Define the facets of the slots: During this phase, the developer should define

different features of each slot such as the value type, allowed values, etc. For the slot "colour", we can define as value type the set of colour-names or numbers describing RGB values.

7. Create instances: The developer defines an instance for each class by choosing the class, creating an instance for the class, and filling in values for the slots.

In 2001, Staab et al. (Staab et al., 2001) presented a new methodology for building an ontology. Their method has five phases. The first phase is a feasibility study. The purpose of this study is to recognize problems and different possible solutions for building an ontology. The second phase is the kickoff, which is identifying the ontology's goal, domain and scope, applications, knowledge sources, users and usage guides, and competency questions. Finally, finding any reusable ontologies in the area of interest is necessary in this phase. In the refinement phase, Staab et al. suggest producing the ontology from the outcome of the kickoff phase. Three sub fields of the refinement phase are creating an informal baseline taxonomy, developing a seed ontology that has concepts and the relationships among them based on knowledge sources, and converting the seed ontology to a final ontology using formal representation languages. In the evaluation phase, the developer ensures that the ontology fulfills all of the requirements and answers all of the competency questions. Also, the evaluation phase provides an opportunity to test the ontology in a real environment and to get feedback from the users. The developer may go back from the evaluation phase to the refinement phase several times to achieve a proper result. In the maintenance phase, the developer creates changes in the ontology according to possible changes in the real world. Staab et al. suggest that the maintenance should be guided

by a policy.

Niles and Pease (Niles and Pease, 2001) explain the process of developing the SUMO ontology. The Suggested Upper Merged Ontology (SUMO) (The Standard Upper Ontology Working Group, 2008) is an upper-level ontology that is a product of Teknowledge Corporation and is sponsored by the IEEE. The first step of the development is to recognize all existing ontological content from various knowledge-based systems and convert this content to the SUO-KIF language (The Standard Upper Ontology Working Group, 2008). In the next step, Niles and Pease divided the concepts into two categories: higher-level concepts and lower-level notions. The final step was to align the new concepts with the existing ones, for which Niles and Pease used one of the following methods:

- Create some intermediate concepts between new concepts and existing concepts;
- Remove the new concepts from the merged ontology; or
- Maintain both the new and the existing concepts.

2.3 Summary of Development Methodologies

Although both Cyc and SUMO are counted as successful and comprehensive top-level ontologies, their development methodologies are not really helpful for building a new ontology. As can be seen, the development of the Cyc ontology does not have a clear outline. Also, for the SUMO ontology, the developers merged existing ontologies in the process of ontology development. The enterprise methodology has a clear outline, but the lack of a documentation process limits its usage to develop-

ing small domain ontologies only. The TOVE methodology and the heuristic-based methodology have the same weakness. The unified methodology anticipates the documentation step in the process of development. Noy and McGuinness's methodology is suitable for creating an ontology, also for non-professional developers, because there are detailed explanations of the steps, along with clear examples.

Methontology and the methodology of Staab et al. are the most mature among all of the methodologies reviewed because they consider the steps of documentation and maintenance in development the process. Maintenance keeps the ontology up-to-date and will make the ontology's life longer.

As a result of the literature study, I discovered that the methodologies for ontology development have common steps, which are listed below and gave rise to the steps in Figure 2.1.

- Identification of users and the use of the ontology;
- Identification of domain concepts and the relationships between them;
- Building an ontology by converting concepts to a formal form;
- Evaluating an ontology; and
- Maintaining an ontology;

I followed the common steps mentioned above to develop my ontology.

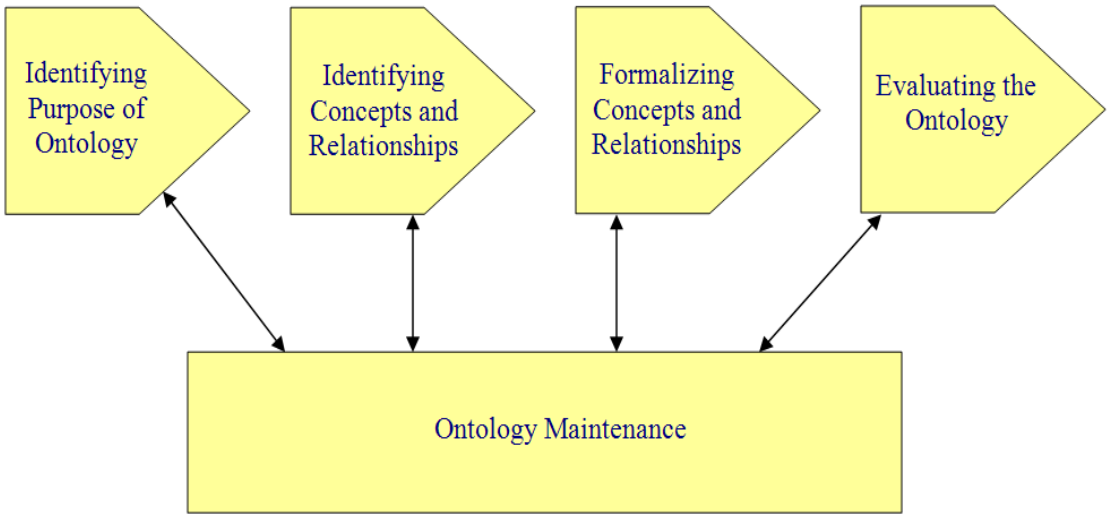

Figure 2.1: Common Steps in Ontology Development

2.4 Ontology Evaluation

Brank et al. (Brank et al., 2005) provide a nice survey of ontology evaluation approaches. They divide the approaches into four categories: 1) comparing the ontology with a Gold Standard, 2) using the ontology in an application and assessing the results, 3) comparing the ontology with a source of data related to a specific domain, and 4) evaluation of the ontology by humans who assess how well the ontology meets requirements.

Brank et al. also categorize the evaluation approaches into six different groups, based on the level of evaluation: the lexical, vocabulary, or data level focuses on the concepts in the ontology; the hierarchy or taxonomy level focuses on sub-class/super-class relationships between concepts; the layer of other semantic relations focuses on other kinds of relations other than *is-a* relationships in the ontology; the context or

application level focuses on the use of the ontology in an application; the syntactic level focuses on the formal language that is used to build the ontology; and the structure, architecture, and design level deals with the organization of the ontology and its suitability for further development. Brank et al. conclude that the only method to evaluate an ontology on all six levels is the evaluation by humans. However, application-based evaluation approaches give an acceptable evaluation on the first four levels mentioned above.

For the evaluation of the graphics domain ontology, I used the ontology as part of a natural language interface to a graphics tool. This form of evaluating the ontology was appropriate, since there is a close relationship between concepts in an ontology and natural language terms (Carroll, 1956).

2.5 Existing Ontologies

One way of finding concepts and relationships is to use existing ontologies. In this section, I reviewed some of the most known and used existing ontologies, which were also useful to develop my ontology.

The Suggested Upper Merged Ontology (SUMO) (The Standard Upper Ontology Working Group, 2008) is an upper-level ontology owned by IEEE that is available under general public license. The original purpose of SUMO was to cover general entities, but now SUMO includes a mid-level ontology and eighteen different domain ontologies. The mid-level ontology connects the abstract contents of SUMO to the detail contents in the domain ontologies. SUMO employs SUO-KIF, a language which has a similar syntax as LISP, to formalize concepts and relationships. Also,

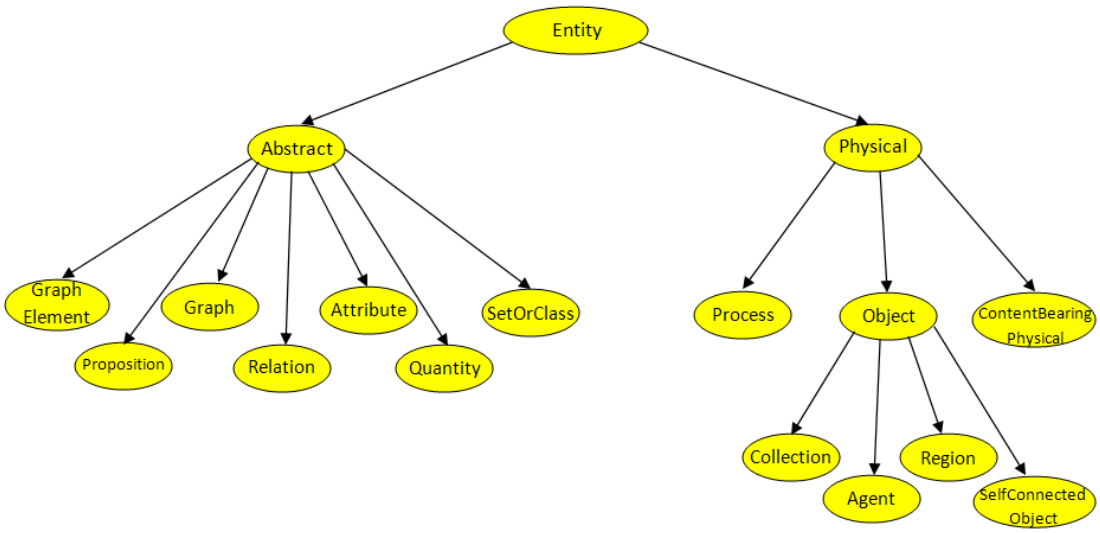

Figure 2.2: Suggested Upper Merged Ontology (SUMO)

SUMO uses the Sigma knowledge engineering environment as a system for developing, viewing and debugging theories in first order logic. SUMO has been mapped to the WordNet (Princeton University, 2008) lexicon, and has 20,000 terms and 70,000 axioms when all domain ontologies are combined. The major usage of SUMO is for research and applications in search, linguistics, and reasoning. Figure 2.2 indicates upper levels of SUMO (The Standard Upper Ontology Working Group, 2008).

Cyc (Cycorp, 2008) is an AI project started in 1984 to perform human-like reasoning using an ontology and a common sense knowledge base. Cyc's original ontology and knowledge base license are owned by Cycorp, Inc., but a small version of Cyc called OpenCyc was released under an open source license. Recently, Cyc became available under a research-purposes license as ResearchCyc for researchers. The Cyc ontology is a general ontology that includes more than 300,000 concepts. Cyc uses CycL as formal language which is a declarative language based on first-order logic.

Cyc has various extra tools such as an inference engine, a natural language parser, a natural language query tool, a template-based Fact Entry Tool, and more. The major usage of Cyc is reasoning in various applications.

Protégé (Stanford University, 2008a) is an open-source platform that provides a tool to develop domain models and knowledge-based applications with ontologies. The Protégé platform includes two ways of modeling ontologies, using the Protégé-Frame and using the Protégé-OWL editor. The Protégé-Frames editor allows users to build frame-based ontologies adhering to the Open Knowledge Base Connectivity protocol (OKBC). The Protégé-OWL editor supports the Web Ontology Language (OWL) (W3C, 2008) to build an ontology for the semantic web. The Protégé ontology library (Stanford University, 2008b) contains several different domain ontologies that are mostly designed for OWL.

The review showed that there is no existing ontology for the graphics domain that I could readily use to identify domain concepts and relationships. Although SUMO and Protégé contain several domain ontologies, none of them is designed for the graphics domain. However, reviewing these ontologies, especially SUMO, was really helpful to get ideas for organizing concepts in a taxonomic hierarchy. Also, I utilized the Protégé-Frame editor to implement the graphics domain ontology due to its superb documentation and the support provided by its developers at Stanford University.

2.6 Spatial Relations

There is a huge number of publications on spatial relations, spatial cognition, and spatial language, which I cannot describe in all detail here. I thus give references to important work and review the work on spatial relations with a focus on how to describe and implement spatial relations for my purposes, as outlined below.

There is a specific set of linguistic terms, which refer to spatial relations; these are mostly prepositions like "in", "on", "under", "above", or composites like "left of", "right of", "on top of" etc. I used such terms, which refer to spatial relations, in order to position graphical objects on a virtual canvas. It is possible to employ a relatively simple model of the core spatial relations, based on splitting the canvas in regions around a graphic object in question. Certain regions then correspond to specific spatial relations.

It is important to notice the difference between my use of spatial relations and that of most other researchers: 1) For my purposes, the representation of simple, static spatial relations was sufficient - my work does not currently involve objects, which move on their own. 2) For my application, I transferred linguistic terms for spatial relations into graphic images, and not vice versa - whereas most other research deals with the analysis of given images (or data) and the task of providing a verbal description of these images.

Early work on the linguistic analysis of spatial relations, which was later also used as basis of research on computational models of spatial relations, was done by Annette Herskovits (see e.g. (Herskovits, 1986)). In her later paper "Language, Spatial Cognition, and Vision" (Herskovits, 1997), she explains spatial relations based

on the grounding of linguistic terms in vision and spatial cognition. In order to provide a semantic representation for spatial relations, she starts out with the classic "figure-ground" distinction by identifying a "located object" (figure) and a "reference object" (ground); the ground is in general a larger, less mobile object in relation to the figure. Herskovits distinguishes two major categories of spatial relations, which are location-based and motion-based relations, and classifies spatial prepositions into these categories.

Some significant, relatively recent work has been done in the "Spatial Cognition Priority Program" (University of Bremen, 2009) and described in three resulting books (Freksa et al., 1998, 2000, 2003). Relevant in this context is also earlier work by Freksa (Freksa and Röhrig, 1993), who provides a computationally feasible model for some spatial relations.

In the context of the Spatial Cognition Priority Program, significant work has been done by Benjamin Kuipers, when he developed "The Spatial Semantic Hierarchy" (Kuipers, 2000). This work is partly embedded in the context of autonomous vehicles and robotic applications, and focusses on spatial knowledge related to motion and navigation, e.g. trajectory and path representations. Other research aspects dealt with are cognitive maps and spatial memory, and in-depth examinations of specific linguistic terms and cognitive issues, e.g. the concept of "corner" and related linguistic terms.

Similarly, Nualláin (Nualláin, 2000) presents a collection of papers addressing various topics in spatial cognition, ranging from philosophical aspects to geographical information systems.

Significant work dealing with computational models of spatial relations has been done by Terry Regier (Crawford et al., 2000; Regier, 1996; Regier and Carlson, 2001; Regier and Zheng, 2007) and by Kenny Coventry (Cangelosi et al., 2005; Coventry et al., 2002, 2005; Coventry and Garrod, 2002; Coventry and Olivier, 2002; Joyce et al., 2003). Regier's research focusses on neuropsychological models of spatial cognition and language and their implementation as a connectionist model. Coventry developed a psychologically-oriented semantic framework for spatial relations, which takes the functionality of involved objects into account.

Levinson and Wilkins (Levinson and Wilkins, 2006) provide a hierarchical classification of spatial relations used in natural language descriptions (see Figure 2.3). Due to its relevance for this thesis, their work is described below in more detail.

Levinson and Wilkins divide spatial relations into two major groups: static and dynamic. Static relations are relations which do not involve any movement in the spatial relation between the figure and the ground. For example, the spatial relation "on", as in the sentence "the book is on the table", does not involve any movement or change of the spatial relation between the book (figure) and the table (ground). In dynamic relations, some form of motion is involved in the relation. For instance, the preposition "into" in the sentence "The man goes into the house" involves a movement and a change of the location of the man (figure) with respect to the house (ground). Sub-categories of static relations are non-angular (topological) relations and angular (frame of reference) spatial relations. In non-angular relations, the location of the figure can be recognized without any coordinate system, while in angular relations, the location of the figure is specified using a coordinate system, and a so-called $frame\ of$

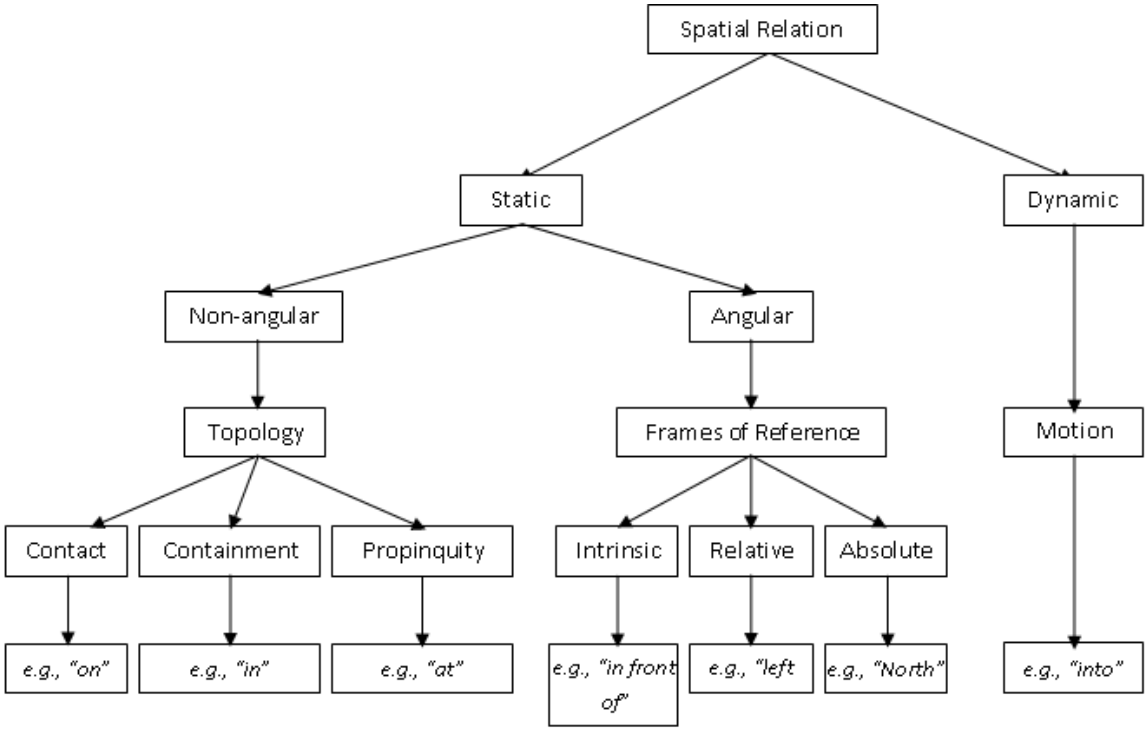

Figure 2.3: Categorization of spatial relations in natural language descriptions

reference. Topological relations are divided into three different categories according to the spatial connection between the figure and the ground. For example, in sentence "the book is on the table", the figure (the book) has physical contact to the ground (the table), and the relation is thus called contact. In the sentence "the apple is in the bowl", the ground (the bowl) contains the figure (the apple) and the spatial relation is thus called containment. In the sentence "the playground is at the school", the figure (the playground) is in the proximity of the ground (the school) and the spatial relation is called propinquity.

Topological spatial relations are relations determined based on the figure and ground objects themselves; the location of the viewer is not important in order to

specify the figure's location. For example, in the sentence "the book is on the table", it does not matter where the viewer is located.

Angular relations, where a coordinate system is used to specify the angle of the figure with respect to the ground, are sub-categorized based on the frames of reference they use. There are three different frames of reference: 1) an *intrinsic* frame of reference, where the location of the figure is described in relation to a known, inherent orientation or part of the ground; for example, in the sentence "the statue is in front of the church", the relation is intrinsic since the region "in front of" the church is inherent to the church and is thus known. 2) a *relative* frame of reference, where the coordinate system is based on the viewer; for instance, in the sentence "the cat is to the left of the tree", the left region of the tree itself is not known, but it depends on the observer's position. 3) an *absolute* frame of reference, where fixed bearings are used to specify the location of the figure; an example is the sentence "the coast is north of the mountain", since the direction "north" is independent of the viewer or any part or orientation of the figure or ground etc. This last category is specifically relevant to geographical information systems (see also (Frank, 1996) and (Pullar and Egenhofer, 1988)).

Finally, *dynamic* relations involve some form of motion and are usually described based on a trajectory, which shows the motion or the change of the spatial relation between the figure and the ground. Some typical spatial prepositions in this category are "into", "along", "toward", "around". For example, in the sentence "Sam drags the tree toward the car", the spatial relation between the figure (the tree) and the ground (the car) changes and the direction of the movement (the trajectory) of the tree can

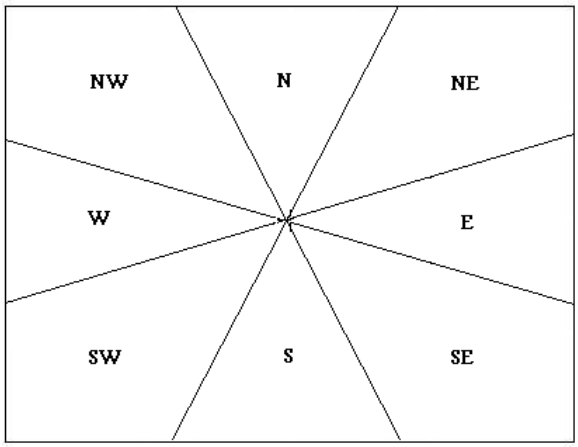

Figure 2.4: Cone-shaped cardinal directions

be specified with respect to the position of the car. If the trajectory is directed from the tree to the car, then this corresponds to the spatial relation "toward".

It should be noticed that figure and ground objects are treated as points in this view. In addition, a lot of spatial relations and thus the linguistic descriptions of them (in particular prepositional expressions) are somewhat under-determined. This means that, while we usually can specify an area or field in which the figure is located, an exact position is often not derivable and we must therefore resort to common sense or default values (see e.g. (Herskovits, 1997) and (Mukerjee, 1998)).

Frank (Frank, 1996) presents a mathematical model for a direction and distance system in a geographical scale for spatial relations. Frank suggests a model called "cone-shaped cardinal directions", shown in Figure 2.4. A similar cone-shaped model can also be used to describe angular spatial relations like "left of", "in front of", etc.

As mentioned earlier, the purpose of my work is to create images from natural language descriptions. Thus, I have to position objects on a canvas as described

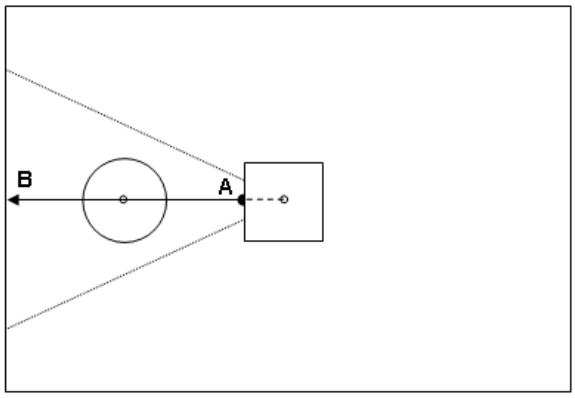

Figure 2.5: Positioning a circle left of the square

by users' instructions, which involve spatial relations. For instance, to fulfill the instruction "draw a circle left of the square", the position of the circle on the canvas has to be determined in the "left of" region of the square. This "left of" region corresponds roughly to the "West (W)" region in Figure 2.4. In addition, in my system, an exact position of the circle somewhere in this area has to be determined. Based on related work (Clay and Wilhelms, 1996; Freksa and Röhrig, 1993), in my approach the circle is located on a virtual horizontal line, which connects the center of the square and the left border of the canvas. The center of the circle is positioned on this virtual line, in the middle between the left edge of the square at A and the left border of the canvas at B (see Figure 2.5).

In addition, it is ensured that certain physical constraints and conditions are satisfied, e.g. that all objects are located completely inside the canvas.

Pullar and Egenhofer (Pullar and Egenhofer, 1988) describe spatial relations especially for use in geographical information systems in the following dimensions:

- Directions, e.g. *north*, *northwest*;

- Topological characteristics, e.g. *disjoint*, *touches*;
- Ordinal types, e.g. *in*, *at*;
- Distance types, e.g. *far*, *near*; and
- Fuzzy types, e.g. *next to*, *close*.

Distances and fuzzy types are also relevant for this thesis work, since users may employ such terms in natural language instructions for a graphics system, for example "Move the circle close to the square".

The reviewed research work about spatial relations is a good basis for the design of the position concepts in the graphics ontology (see Section 4.2.5).

Chapter 3

Sentence Collection

In this section, the experimental design for the sentence collection from users is described. An important tool used for the sentence collection was the Amazon Mechanical Turk (Amazon, 2009), which allowed me to collect sentences through a webpage by posting tasks for users. I describe the images created for these tasks and the different methods of sentence collection I used.

3.1 Sentence Collection Experiment

In order to refine the ontology and design the natural language interface, I had to collect a corpus of verbal instructions in the graphics domain from potential users of a graphics system or its natural language interface, respectively.

There are different ways of collecting such a natural language corpus. Gupta and Hennacy (Gupta and Hennacy, 2006) use the Open Mind Indoor Common Sense (OMICS) system (Gupta and Kochenderfer, 2004) in their research. OMICS is a

web-based system for collecting sentences from web users describing tasks in different application domains. Even though the approach of collecting and using sentences by Gupta and Hennacy is too simplistic and was thus not adopted here, the idea to use a web-based system to collect linguistic material seemed very attractive and promised to lead to a large corpus of sentences useable for the development of the natural language interface and for evaluating and improving the ontology as well.

The collection of the corpus involved two parts:

1. the design of a set of suitable tasks in the graphics domain, which are to be described by human users as verbal instructions, e.g. to create or modify images

2. the administration of these tasks to a set of users through a web page to collect the respective natural language instructions

The resulting corpus was used to elicit different words and terms referring to graphics concepts and actions, which were integrated into the ontology in the ontology development process (see Section 4.3). The collected sentences were also used in the design of the natural language interface.

3.1.1 Tasks Description

For the purpose of collecting verbal instructions from users, I designed a set of sample images and asked people to explain in natural language how to create these images. This method provided basic information about how users describe graphic shapes, their size, colour, and position. A second set of tasks was used to elicit how users refer to changes of images, e.g. through moving or deleting graphic objects, and

also how they describe spatial relations between objects. For this purpose, I presented pairs of images to the subjects and asked them to describe how to modify the first image in order to create the second one.

I posted a related set of tasks on the Amazon Mechanical Turk (Amazon, 2009). The Amazon Mechanical Turk (AMT) is a website that provides possibilities for businesses or researchers (so-called "requesters") to access a wide variety of users of this website (called "workers") who can perform certain tasks for them. Often, these tasks are concerned with web searches and the evaluation of search results, but in principle there is no limit to the variety of tasks as long as they can be described and issued through a web interface. The AMT also enables workers to select a diversity of tasks to complete at their convenience. In the AMT, each task is a question that can be answered by one or more workers. A task is also called a HIT (Human Intelligence Task) in AMT terminology. Workers will be rewarded with a small amount of money for completing a HIT.

I posted the images created for the sentence collection as HITs on the AMT. Figure 3.1 shows the webpage for one of those posted HITs.

3.1.2 Sentence Collection Phase 1

In the first phase of the sentence collection, I provided 20 images as HITs. The general idea behind these images is to find out how users describe the action, size, position and colour of a shape. The shape used in all images in this group was a square.

All of the 20 images correspond to a create-action and thus the users were asked

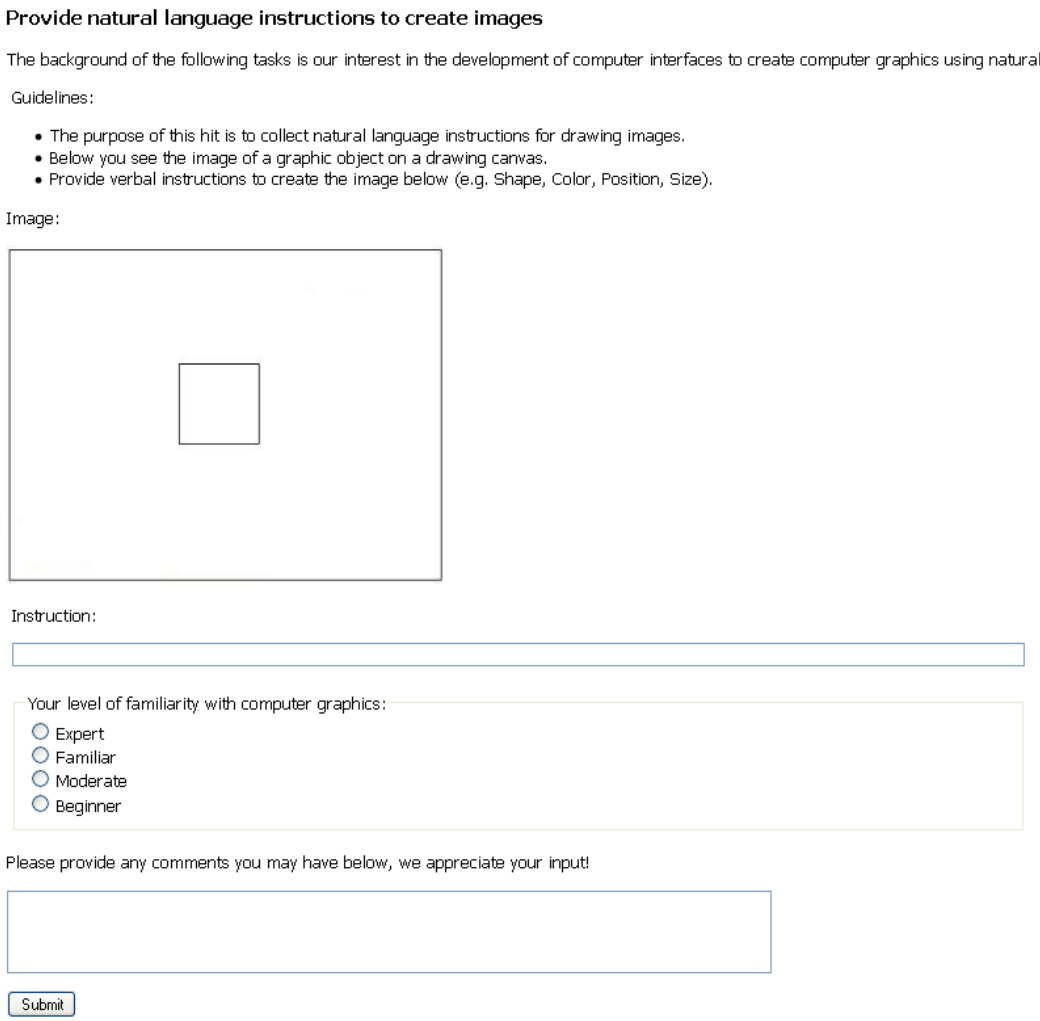

Figure 3.1: Sample HIT posted on the Amazon Mechanical Turk

to give an instruction on how to create the image. Three of the images are designed in order to extract terms to describe the size of the shape. The different sizes of the square in the images are small, medium and large, using the respective Size-values in the ontology.

To find verbal terms describing the position of a shape, I provided 9 images with a square in one of the 9 areas of the canvas. These images cover the absolute positioning

of the shape, since there is no other shape on the canvas and the users have to describe the position of the shape in relation to the canvas.

For the colour feature, I created 9 images containing a square with 9 different standard colours. The colours are: white, black, red, green, blue, yellow, aqua, magenta, and gray. Also, I added 3 images with 3 different intensities of the colour blue: light blue, dark blue, and blue, in order to observe how the user would describe the intensity of the colour.

After posting the images on the AMT website, I set the number of users for this group of HITs to 20, i.e. each task (image) can be processed by 20 workers. While reviewing the sentences, I realized that almost half of the users' inputs were not a natural language instruction. The inputs were either a format similar to a programming language instruction, for example "shape: square, size: small, colour: blue, position: center" or they were meaningless, like "shape, colour, size". Consequently, I modified the instruction-section of the HITs for the second and third group of HITs and emphasized that the input has to be a natural language instruction.

3.1.3 Sentence Collection Phase 2

In the next phase of the sentence collection, I created two groups of images. The first group included 16 images. The main idea behind these images was to investigate the terms that users use for shapes. I also chose different colours and positions for each shape in the HITs in order to examine the terms for colour and position of the shape as well.

I used 12 different geometric shapes including diagonal-line, vertical-line, horizontal-

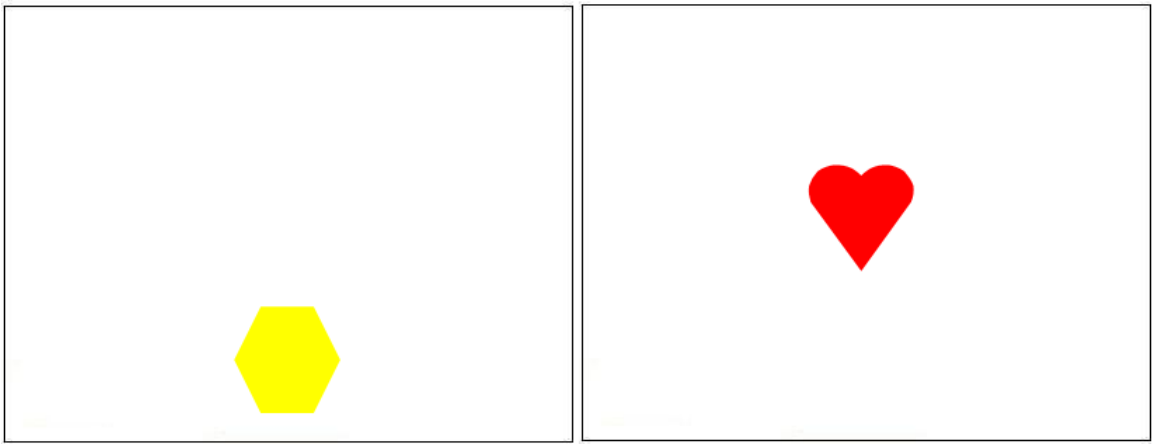

Figure 3.2: Sample Images of Shapes

line, equilateral-triangle, right-triangle, rectangle, diamond, trapezoid, regular-pentagon, regular-hexagon, and ellipse, as well as 4 predefined shapes including heart, star, crescent-moon, and sun. Figure 3.2 shows two sample images in this group.

The second group included 42 images posted as pairs in 21 HITs. In this group, the users were asked to modify the first image in order to obtain the second image. The main idea for this group of HITs was to study how users describe actions like move, colour change, size change, rotate, and erase as well as relative positioning of a shape.

Two HITs were designed for the colour change action. For the first HIT, the shape in the second image had a different colour than the first image. For the second HIT, the shape in the second image had a different border colour than the first image.

One HIT was designed for the erase action. In this HIT, there is a shape in the first image while in the second image there is no shape.

There are 9 HITs designed for the relative positioning of the shape. In the first

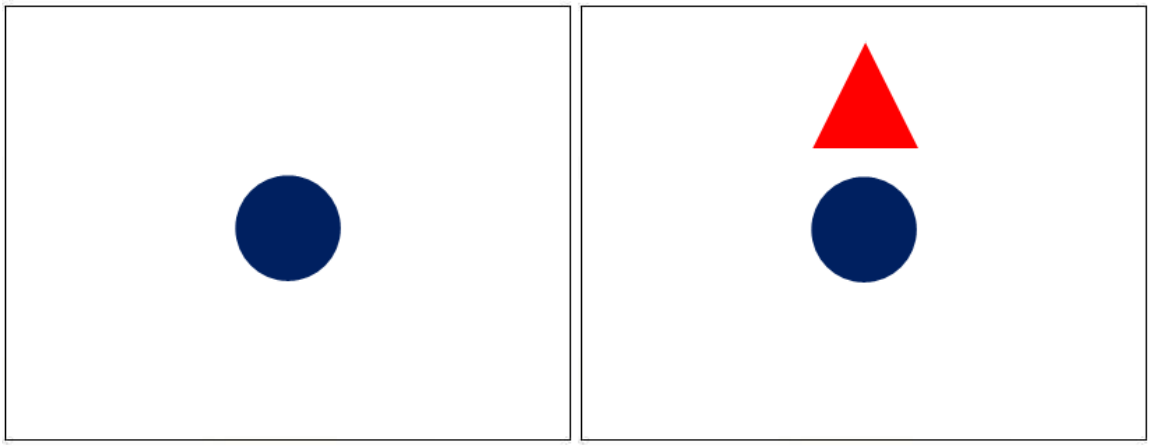

Figure 3.3: Sample Images for Relative Positioning

8 HITs, the first image included a shape in the center. The second image included the same shape as well as a new shape in one of the 8 different areas around the first shape (except center since this is the location of the first shape). An example is shown in Figure 3.3. In the 9th HIT, the first image included two shapes: one in the left and another one in the right half of the canvas, and the second image has a new shape located between the first two shapes. An example is shown in Figure 3.4. As mentioned, these 9 HITs were designed to investigate how users describe the position of the shape with respect to another shape or other shapes.

Two HITs were designed for a resize-action. For the first HIT, the first image includes a shape and the second image includes the same shape with a smaller size. For the second HIT, the first image has a shape and the second image has the same shape with a larger size. In Figure 3.5 the images for a sample HIT are shown.

Two HITs were designed for rotate-actions. The first image for the first HIT included a shape and the second image included the same shape but rotated 45

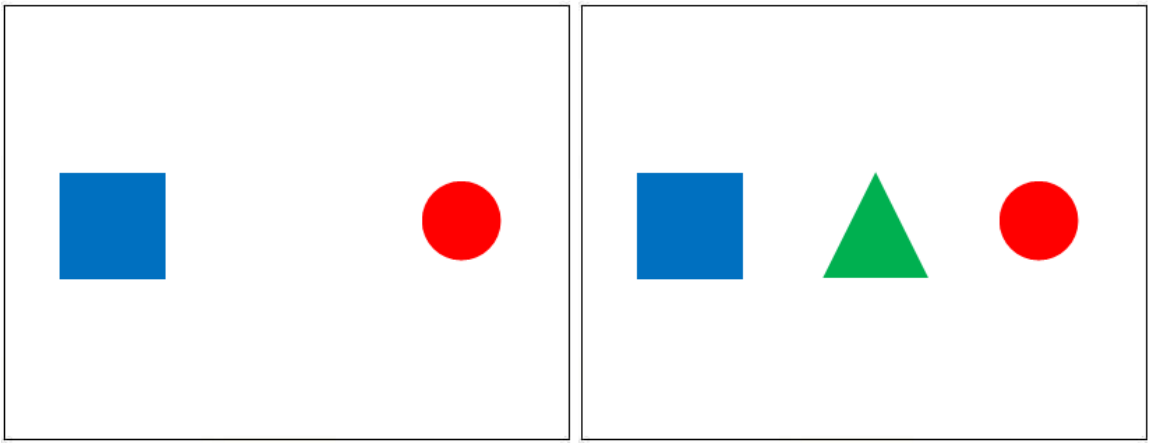

Figure 3.4: Sample Images for Relative Positioning

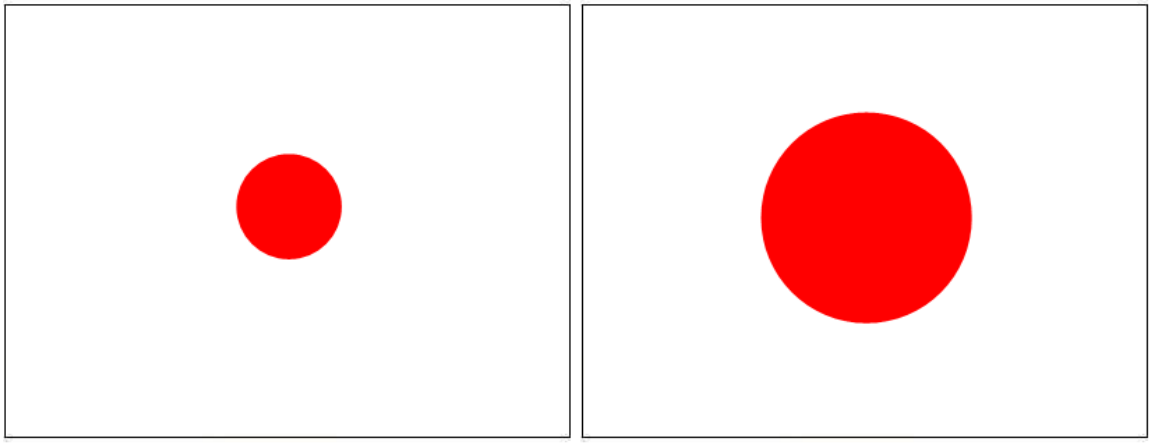

Figure 3.5: Sample Images for Resizing

degrees clockwise. In the second HIT, the second image has the same shape as the first image but the shape is rotated 90 degrees clockwise. In Figure 3.6 the images for a sample HIT are shown.

There are 5 HITs designed for the move-action. In the first 3 HITs, the second image has the same shape as the first image but the shape is moved to the left by 3 different distance sizes. In the 4th HIT, the first image includes two shapes that are

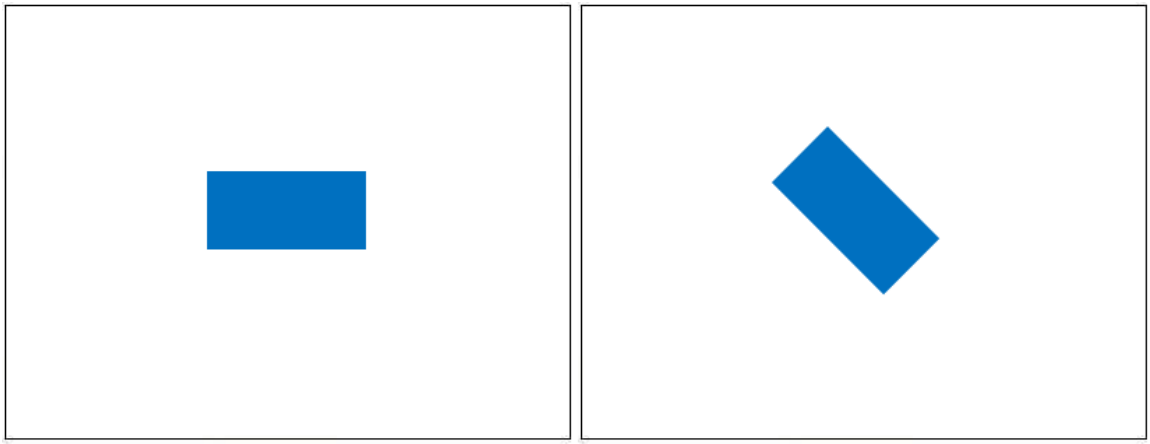

Figure 3.6: Sample Images for Rotation

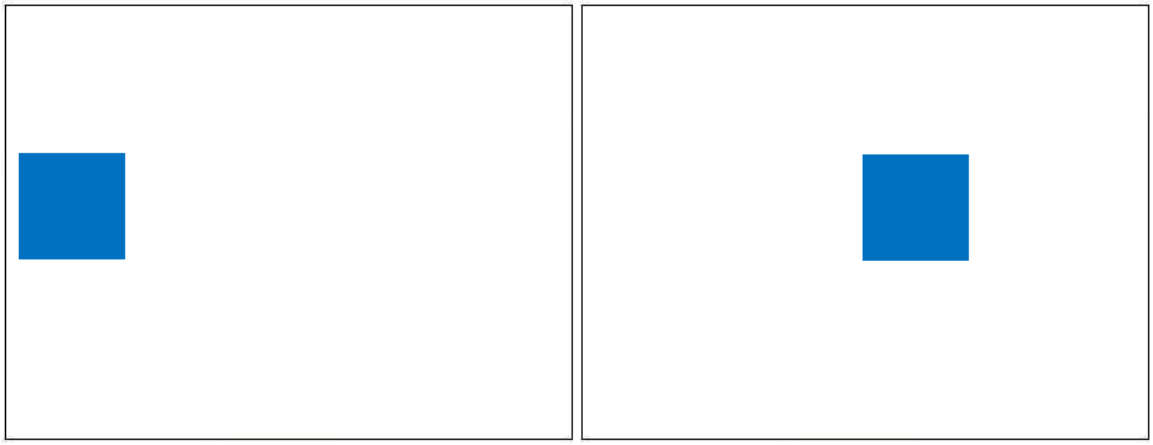

Figure 3.7: Sample Images for Moving

close to each other and the second image has the same shapes but away from each other. The 5th HIT is exactly opposite of the 4th HIT which means that in the first image two shapes are far from each other but in the second image they are close. With these two HITs, I investigated how users would describe the new position of the shape according to the other shape. In Figures 3.7 and 3.8, two images for two sample HITs in this group of HITs are shown.

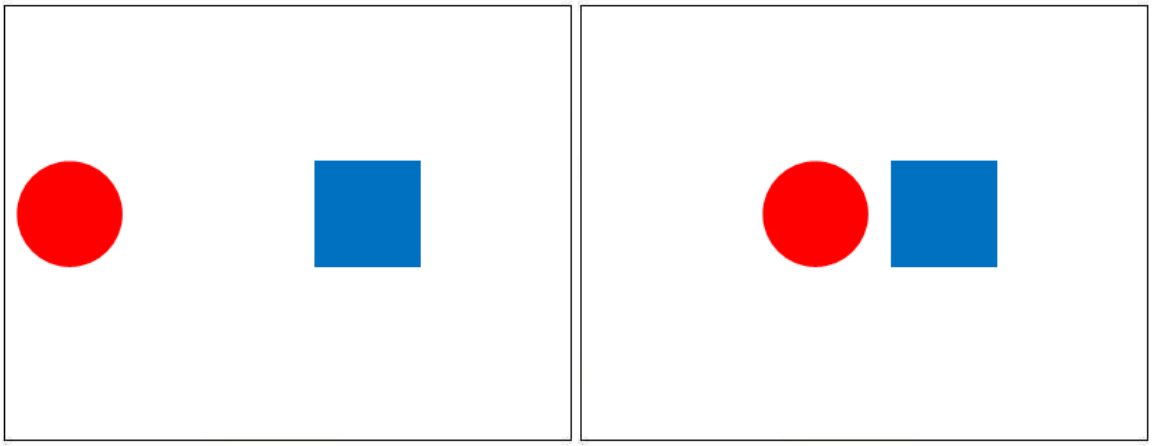

Figure 3.8: Sample Images for Moving

I published both groups of the HITs on the AMT website. The number of users for these two groups of HITs was set to 10. As mentioned, I also modified the instruction of the HITs by emphasizing that I am looking for a natural language instruction for the HITs. This time, the majority of the users provided proper natural language sentences.

3.1.4 Sentence Collection Phase 3

According to the data collected from the users in the HITs, more than 70% of the users selected that they are familiar with computer graphics or experts. Therefore, I decided to perform the experiment also with "regular" users, who have limited knowledge of computers, in order to obtain a variety of natural language inputs coming from different types of users. I designed a website containing all the tasks of the first and second phases for phase 3 of the experiment. This time, I asked 4 users to perform the tasks.

3.2 Corpus Preparation

In the first step after collecting the sentences, I looked through the sentences for each task in order to match the sentence with the task and checked to see if the input is a natural language sentence. In the first phase of the sentence collection, there were sentences that followed a format similar to the programming languages. For example, in the input "shape: square, size: small, colour: blue, position: center", the shape is considered as a variable and square is assigned to it as a value. Here are some other such sentences: "Shape=Square, colour=Light-Blue, Boundary=Black, Position=Center, Size=100 pixels", "Small, Sky Blue, Square, Centre". Since my approach is to parse the sentences with a natural language parser and perform the semantic representation based on the parser's output, this group of sentence were removed from the corpus. Also, some users provided inputs that were not related to the task like "shape, size, colour". Such inputs were removed from the corpus as well. In the next phases of sentence collection, the number of such illegitimate inputs dropped dramatically due to a modification of the instruction provided for the tasks.

After the filtering, a total of 684 sentences (12 sentences per task) remained. I used almost 60% (399 sentences) of the sentences to refine the ontology and to design the natural language interface. The remaining 40% (285 sentences) of the sentences were used for test and evaluation.

Chapter 4

Ontology Design

In this section, I describe the design and development of the graphics domain ontology. First, I provide a brief explanation of the terminology used in the description of the ontology. This terminology is closely related to the Protégé-Frame editor, which I used to implement the ontology. Next, the ontology development phases are described in detail. In the first phase, a basic ontology was established through identifying graphics domain concepts and relationships and organizing them into a hierarchy based on different resources such as graphics books, documentation for graphics tools, and online graphics resources. In the next phase, more concepts were extracted and some concepts or their descriptions were modified based on the collected corpus of sentences (see Chapter 3). Some statistics regarding the size of the ontology, e.g. the number of concepts and features in the final version of the ontology, are given. Relevant information regarding the implementation of the ontology and the reuse of the ontology by others closes this chapter. The complete concept hierarchy of the final ontology can be seen in Appendix A.

4.1 Terminology

The term "class" is equivalent to "concept" and they both represent a group of similar entities in a domain. For example, the class "Square" represents a group of shapes in the graphics domain that have specific features like four equal edges and the angles between the edges are equal to 90 degrees. The term "instance" represents an actual member of a class. For example, the instance "Square1" is an instance of class "Square" with an edge size of, for example, two centimetres.

The term "slot" is equivalent to "feature" and they both represent properties of classes. For example, the class Square has a slot Color. In Protégé, a slot can be defined using different types like symbol, integer, float, string and instance. The symbol-type is used when a slot has a limited number of possible (symbolic) values. For example, the slot QuantitativeSize is of type symbol with possible values: very-small, small, medium, large and verylarge. The types integer and float are used for slots with a numeric value. For example, the slot Height in the Canvas-class needs to be an integer representing the height of the canvas in pixels. The string-type is used for slots with a text value. An example is the Synonyms-slot for classes containing different synonym names for a specific concept, like different names for a colour (e.g. Lightblue and Babyblue). The type instance is used, when the slot refers to a specific instance of a class. For example, the ShapeColor-slot defined for the class Shape refers to an instance of the Color-class representing a specific colour.

A "facet" represents the properties of each slot such as value types, default value, and allowed values. For example, the value type of the slot Color in the class Square is an integer representing an RGB value; the default value is zero representing the

colourblack. The facet represents a specification for a single slot. We also have "constraints", which represent a specification for a class. For example, a circle is a specific type of ellipse, with the constraint that the major radius size is equal to the minor radius size, in other words: it is perfectly round. In order to implement this specification in the ontology, the class Circle is a subclass of the class Ellipse. In addition, a constraint is included to define that the MajorRadiussize slot value and the MinorRadiusSize slot value are equal. Protégé uses the so-called Protégé Axiom Language to formulate constraints. The Protégé Axiom Language is a superset of first-order logic and has a syntax similar to KIF.

Since the ontology is developed for natural language processing purposes, synonyms of terms denoting concepts in the ontology are added. For this purpose, a meta-class is used, which has all the features of a normal class plus a multiple-value slot of type string called Synonyms. The Synonyms-slot of a class stores all the synonyms for the term denoting the concept in the ontology. All the classes in the ontology are generated from this Meta class.

4.2 Ontology Design Phase 1

In the first step, I examined the available resources such as graphics books, websites, and tools in order to find concepts in the graphics domain. I collected a list of graphics and geometric shapes from (Wolfram Research Inc., 2009) and (Wikipedia, 2009). To refine and complete the list of shapes I also analyzed graphics tools like MS Paint, the drawing tool in MS Office, OpenGL and Java's graphics packages (Sun Microsystems Inc., 2009). I also obtained common features of the shapes such as

colour, size, and position from these graphics tools. In addition, I listed the general actions common to all graphics tools, like draw, resize, rotate, erase, and move that were proposed to be represented in the ontology. For the first version of the ontology, suitable concepts were selected and organized into a taxonomic hierarchy.

In some cases, it was not obvious whether a set of entities should be modelled as a class or as slot values for a class. For example, the size of a shape could be simply modelled through a slot with a set of values; however, it is more appropriate to model the size as a class on its own, since there are different types of size (qualitative and quantitative) and each type can then be represented through a subclass of the class Size.

I also added some mid-level classes between very general superclasses, like Shape, and specific subclasses, like Triangle, in order to obtain a better, logical classification. For example, the class Shape2D has two subclasses: Curved and LineComposed. These subclasses are used to categorize different types of 2D-shapes, like Circle as a subclass of Curved and Square as a subclass of LineComposed.

Figure 4.1 shows the top two levels of concepts in the ontology hierarchy. The most abstract concept in the ontology is the class "GraphicsConcept". This class includes five subclasses: "Action" representing action-concepts; "Shape" representing shapes; "Size" for types of values for the size of shapes; "Color" representing values for colours; "Position" representing values for the location-feature of shapes; "Canvas" representing the canvas or drawing area; and "Area" representing different areas on the canvas. These most relevant concepts are described in more detail below.

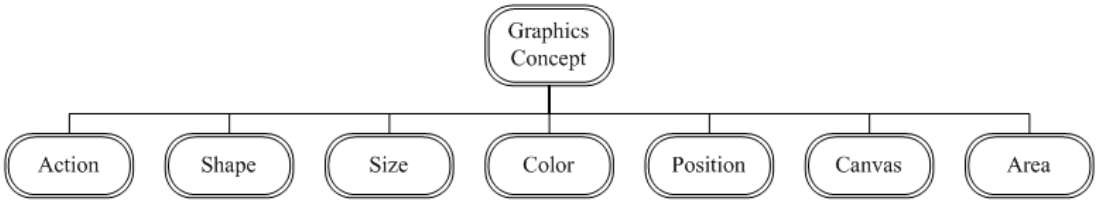

Figure 4.1: The top levels of concepts in the ontology (Phase 1)

4.2.1 The "Action" Hierarchy

Figure 4.2 shows the hierarchy of the Action-class and its subclasses. The Action-class has a slot named ShapeTo. The ShapeTo-slot contains a value of an instance of the class Shape. This instance represents the specific shape-object, on which the graphic-action is performed. The ShapeTo-slot is inherited to all the subclasses of the Action-class. Originally, the Action-class included six different subclasses: Draw, Move, Resize, Rotate, ColorChange and Erase. Later, the hierarchy was reorganized and three different subclasses were inserted as medium level actions: Create, Change, and Delete. The effect of either of the actions Move, Resize, Rotate, and ColorChange denotes a change of one of the shape's features. Consequently, the logical classification is to subsume all these actions under one superclass called Change. This improved the categorization and made search and classification in the hierarchy easier.

The Create-class has a subclass Draw. The Draw-class represents draw-actions in the graphics domain. In the future, other types of create-actions such as creating a canvas can be added as a subclass of the Create-class.

The Change-class contains the subclasses Move, Resize, Rotate, and ColorChange. The Move-class represents move-actions or position-changes of shapes. An additional

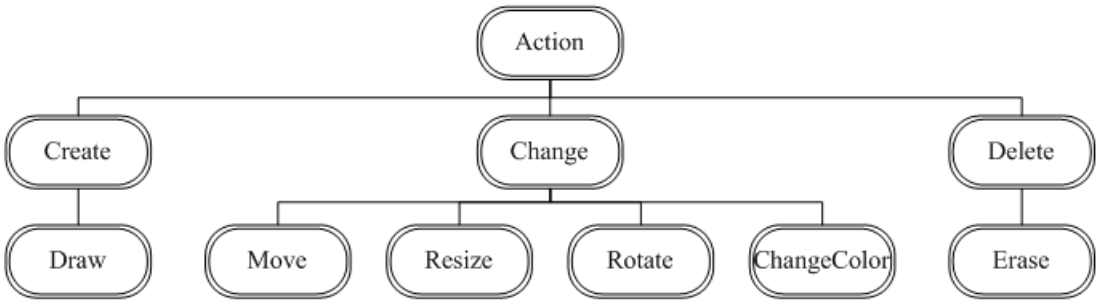

Figure 4.2: The hierarchy of the Action-class and its subclasses

slot of the Move-class is NewPosition, which is an instance of the Position-class. The Resize-class represents actions to change the size of a shape; it has a slot called New-Size, which is an instance of the class Size. The Rotate-class represents rotations of shapes in the graphics domain. The Rotate-class has a slot called Angle. The Angle-feature represents the angle of rotation for a shape. The last subclass of the Change-class is the ColorChange-class. The ColorChange-class has a slot called New-Color, which is an instance of the Color-class. The Delete-class has a subclass called Erase, which represents removing a shape from the scene in the graphics domain. Similar to the Create-action, the Delete-action could have other subclasses in future, such as deleting a canvas.

4.2.2 The "Shape" Hierarchy

Figure 4.3 shows the hierarchy of the Shape-class and its subclasses. The Shape-class has the following slots: ShapeColor as an instance of Color; ShapeLocation as an instance of Position; ShapeSize as an instance of Size; Dimension as a symbol representing the dimension of the shape; and SpatialRelation representing the spatial

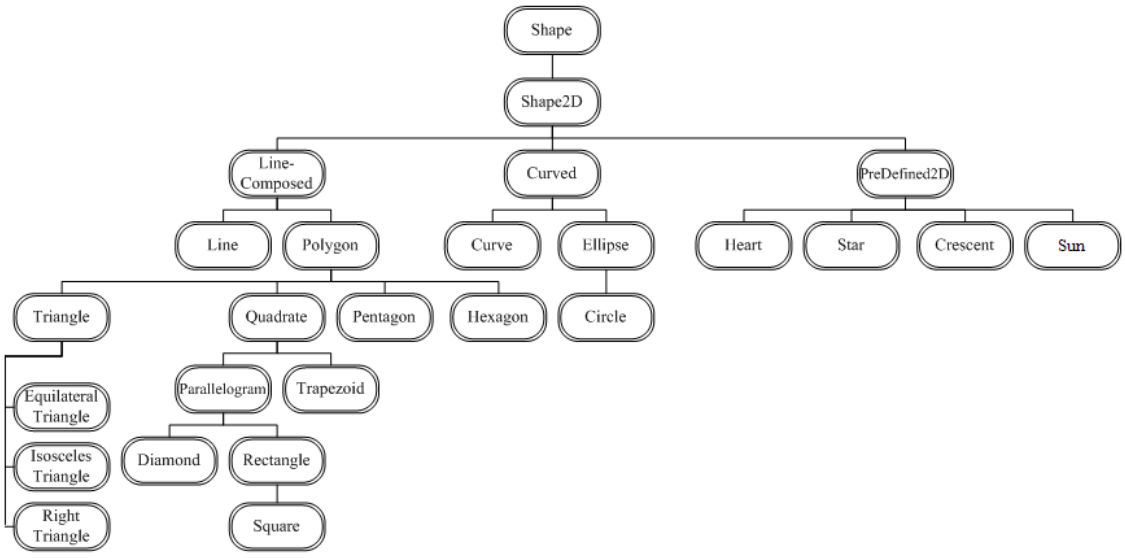

Figure 4.3: The hierarchy of the Shape-class and its subclasses

relation between the shape and other shapes. The ShapeSize-slot was later removed from the Shape-class, since shapes can have different types of size-feature. For example, the size of a polygon can be specified by the size of its edges, while the size of a circle can be specified by its radius.

The Shape-class has Shape2D as its subclass. The allowed value for the Dimension-slot of the Shape2D-class is set to 2. The Shape2D-class has three direct subclasses: Curved, LineComposed, and PreDefined2D. The Curved-class includes the subclasses Curve and Ellipse. The Curve-class has a CurveSize-slot as its size-feature. The Ellipse-class has MajorRadiusSize and MinorRadiusSize as its size-features. The Circle-class is a subclass of the Ellipse-class, since the circle is a specific type of ellipse with its major radius size equal to its minor radius size. In the Circle-class, the equality of the values for MajorRadiusSize and MinorRadiusSize is specified us-

ing a constraint. The LineComposed-class is a superclass for Line and Polygon. The Line-class has the LineSize-slot as its size-feature.

The Polygon-class has a slot called EdgeNumber, which represents the number of edges in a polygon. Another slot is the EdgeSize, representing the size of each edge of the polygon. The EdgeSize is a multiple-value slot, which accepts instances of the Size-class as value. The advantage of using a slot with multiple values is to represent an arbitrary number of edges. However, the problem is that the slot-values of the multiple-value slot cannot be used to formulate a constraint. In order to formulate constraints for subclasses of the Polygon, I added the following slots: EqualEdge-Number, ParallelEdgeNumber, EqualAngleNumber, and RightAngleNumber. These slots will be explained separately for the respective subclasses. The last slot in the Polygon-class is Angles, which is a multiple-value slot of integers representing the values of the angles between the edges of the polygon.

There are four direct subclasses under the Polygon-class: Triangle, Quadrate, Pentagon, and Hexagon.

The Triangle-class allows only the value 3 for the EdgeNumber-slot. The Triangle-class has three subclasses representing three different specific types of triangle. The EquilateralTriangle-class represents triangles, whose edges have all the same size. In order to formulate equally sized edges, the maximum and minimum value of the EqualEdgeNumber-slot of the EquilateralTriangle-class are facets with the value 3. The same setting is performed for the EqualAngleNumber-slot of the Equilateral-Triangle-class, since all its angles are equal as well. A similar approach is applied to formulate the specification the next subclass of the Triangle-class, the IsoscelesTriangle-

class, which represents triangles with two equal edges. The last subclass of the Triangle-class is RightTriangle, which represents triangles with one right angle. In this case, the maximum-value and the minimum-value of the RightAngleNumber-slot are set to 1.

The Quadrate-class has Parallelogram and Trapezoid classes as its subclasses. The allowed value of the EdgeNumber-slot for these classes is set to 4. The Parallel-EdgeNumber of the Trapezoid-class is set to 2, because there are two parallel edges in the trapezoid. The Diamond- and Rectangle-classes are subclasses of the Parallel-ogram. Both slots ParallelEdgeNumber and EqualEdgeNumber of the Diamond-class are set to 4 in order to represent the diamond shape. For the Rectangle-class, the slots ParallelEdgeNumber and RightAngleNumber are set to 4. The only subclass of Rect-angle is Square. In order to represent the square shapes, the slots EqualEdgeNumber, ParallelEdgeNumber and RightAngleNumber are set to 4.

The next subclasses of Polygon are Pentagon and Hexagon. The allowed values of the EdgeNumber-slots of these classes are set to 5 and 6, respectively. Also, the EqualEdgeNumber-slots for these classes are set to 5 for the Pentagon and 6 for the Hexagon, since they represent regular pentagons and hexagons.

The last subclass of the Shape2D-class is PreDefined2D. There are four subclasses under the PreDefined2D class: Sun, Crescent, Star, and Heart classes representing corresponding predefined shapes. The size-feature for the PreDefined2D-class is rep-resented in the PredefineShapeSize-slot, which is an instance of the Size-class.

The default shape in the ontology is an instance of the Square-class.

4.2.3 The Class "Color"

The Color-class represents the colours of shapes. There are three slots associated with the Color-class: ColorName of type string, which stores the name of the colour; the RGBColorCode of type string, which contains the hexadecimal RGB value of the colour; and Synonyms, which is a multiple-value slot of type string containing synonyms for the colour-name. The Color-class has 140 instances representing different colours. For example, the Color-instance DarkBlue has the ColorName "DarkBlue", the RGBColorCode "00008B", and Synonyms "Dark_Blue" and "Dark-Blue". The default colour in the ontology is the White-instance of the Color-class.

4.2.4 The Class "Size"

Figure 4.4 illustrates the hierarchy of the Size-class and its subclasses. The Size-class originally had no subclasses but included two slots: SizeValue and SizeUnit. The SizeValue-slot is a float number and the SizeUnit is a symbol with allowed values: centimetre, inch, and pixel. In the refinement phase, I added QualitativeSize and QuantitativeSize as subclasses of Size, since it is possible to describe the size with qualitative and quantitative values. The mentioned slots were moved to the QuantitativeSize-class. For the QualitativeSize-class, I added the slots QualitativeSizeValue and ActualSize. The QualitativeSizeValue is of type symbol and has allowed values verysmall, small, medium, large, and verylarge. The ActualSize is an integer representing the number of pixels corresponding to the QualitativeSizeValue. For example, if the QualitativeSizeValue is small, then the ActualSize value is 20. The ActualSize-values are determined in relation to the size of the canvas. The default

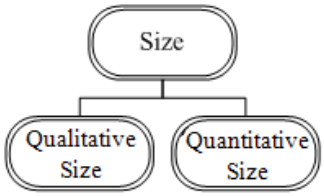

Figure 4.4: The hierarchy of the Size-class and its subclasses

size in the ontology is medium.

4.2.5 The Classes "Position" and "Area"

Figure 4.5 illustrates the hierarchy of the Position-class and its subclasses. The Position-class describes the locations of shapes on the canvas. The Position-class has three slots: FrameOfReference, which is an instance of either the Canvas-class or the Shape-class; X, which is an integer representing the horizontal value of the location on the canvas (in pixel); and Y, which is an integer representing the vertical value of the location on the canvas (in pixel).

The Position-class has the classes "Absolute" and "Relative" as subclasses. The description for the location can be either in relation to the canvas or in relation to other shapes on the canvas. Since the canvas is a single, fixed object, we talk in the first case of "absolute position". For example, in the sentence, "Draw a square in the center of the canvas", the location is described using the canvas as frame of reference, and therefore it is considered as an absolute position. The Absolute-class represents the absolute position on the canvas. The allowed value of the FrameOfReference-slot for this class is an instance of the Canvas-class only. The allowed value of the

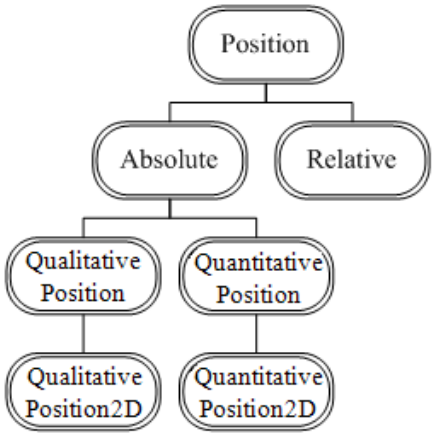

Figure 4.5: The hierarchy of the Position-class and its subclasses

FrameOfReference-slot for the Relative-class is an instance of the Shape-class only. For example, in the sentence "Draw a square left of the circle" the location of the square is described relative to another shape, i.e. the circle.

The Absolute-class has two subclasses: QuantitativePosition and QualitativePosition. The QuantitativePosition-class has a subclass QuantitativePosition2D. The class QuantitativePosition2D has only the inherited slots X and Y, since they represent the quantitative value of the location on the canvas (in pixel). The QualitativePosition-class has the subclass QualitativePosition2D. The QualitativePosition2D-class has two additional slots: HorizontalArea, which has as value type instances of the AreaHorizontal-class, and VerticalArea, which has as value type instances of the AreaVertical-class.

Figure 4.6 illustrates the hierarchy of the Area-class and its subclasses. AreaHorizontal and AreaVertical are subclasses of the Area-class. The AreaHorizontal-class represents three areas of the canvas in the horizontal direction (left, center, and

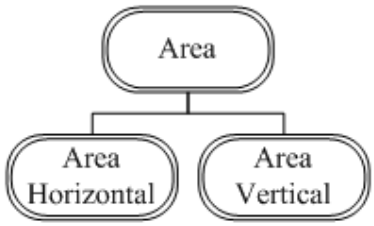

Figure 4.6: The hierarchy of the Area-class

right) and the AreaVertical-class represents three areas of the canvas in the vertical direction (top, center, and bottom). For example, in the sentence "Draw a square in the top left of the canvas" the value of the HorizontalArea-slot is "left" and the value of the VerticalArea-slot is "top". The combination of the horizontal and the vertical areas gives 9 different areas of the canvas (3x3 grid). The default value for the AreaHorizontal- and the AreaVertical-slots is "center".

The Relative-class has a slot called ValueofRelativePosition, which is of type symbol and has allowed values "leftof", "rightof", "below", and "above". These values represent four different areas relative to a shape as frame of reference.

4.2.6 The Class "Canvas"

The Canvas-class represents the drawing area in the graphics domain. It has four slots: BGColor, which is an instance of Color representing the canvas background-colour; Height, which is an integer representing the height of the canvas in pixels; Width, which is an integer representing the width of the canvas in pixels; and Shapes, which is a multiple-value slot accepting instances of the class Shape. The Shapes-slot represents the list of shapes on the canvas. In the ontology, there is a predefined

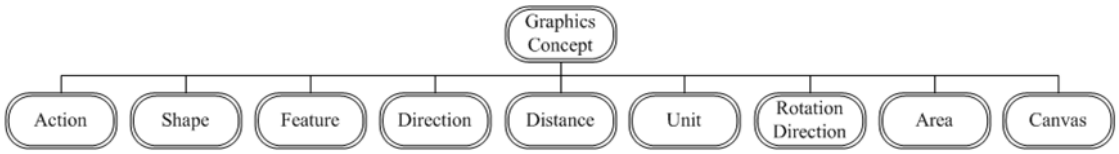

Figure 4.7: The top two levels of concepts in the ontology (Phase 2)

instance of the Canvas-class called "maincanvas".

4.3 Ontology Design Phase 2

In this section, I describe the changes made to the first version of the ontology according to the results from the analysis of the collected sentences. I divided the changes in the ontology into two categories: 1) adding or modifying classes or features, 2) rearranging the hierarchy. Figure 4.7 shows the top two levels of concepts in the second version of the ontology.

4.3.1 Adding or modifying classes or features

Modifying the Action-class

Figure 4.8 shows the Action-class hierarchy of the second version of the ontology. Due to sentences like "Change the color of the square to red", I added a slot called FeatureToChange to the Change-class. The FeatureToChange-slot has as value an instance of the feature, which needs to be changed; in this case, this value is an instance of Color, which is a subclass of the new Feature-class (see the hierarchy rearrangement in Section 4.3.2). In the example sentence above, the FeatureTo-

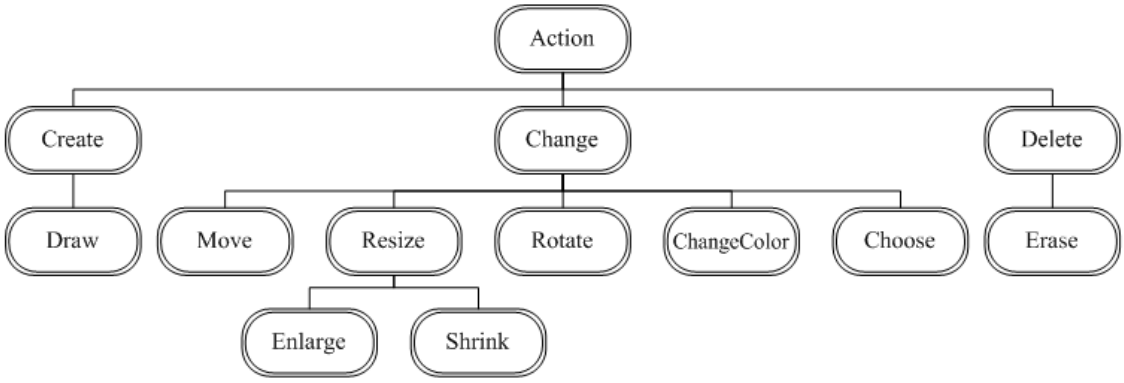

Figure 4.8: The Action-class hierarchy in the second version of the ontology

Change slot will be assigned the Red-instance of the Color-class, since the feature of the square to be changed is its colour and the new value is red, represented by the instance Red of the class Color. I also restricted the allowed values of this slot in the other subclasses of Change to instances of the respective feature-class. For example, in the Resize-class the allowed values for the FeatureToChange slot are only instances of the Size-class.

I added a class Choose as a subclass of Change in order to deal with sentences like "Draw a small square. Choose blue color for it." The Choose-action is similar to the Change-action, since it takes a feature of the shape and assigns a new value to it.

The Resize-class is divided into the classes Enlarge and Shrink, in order to deal with sentences like "Shrink the red square by 50%" or "Enlarge the circle twice its size". For these actions, the NewSize-slot is not sufficient anymore, since the amount of shrinking or enlarging has to be represented. I thus added a slot called EnlargeSize in the Enlarge-class and a slot called ShrinkSize in the Shrink-class. Both slots accept instances of the class QuantitativeSize as values. The default values for these slots

are 50% for the ShrinkSize and 200% for the EnlargeSize.

Through the sentences related to the move action it became clear that users describe the move-action in two different forms. They specify either a new position of the shape, or the direction of movement and the distance the shape should be moved. The sentences "Move the square to the center." and "Move the square to the left by 2 inches." are examples of two different forms of describing a the move-action. The NewPosition-slot of the Move-class only addresses the move in the first description. In order to address the second form of description, I added MoveDirection- and MoveDistance-slots to the Move-class. The MoveDirection is an instance of the Direction-class and the MoveDistance is an instance of the Distance-class. The Direction- and Distance-classes are explained below.

The Direction-class represents the main directions (north, south, west, and east). The only slot for this class is the DirectionType, which is a symbol with "north", "south", "west", and "east" as allowed values. The Direction-class has two subclasses: DirectionHorizontal and DirectionVertical. The DirectionHorizontal-class is subdivided into the classes WestDirection and EastDirection; their DirectionType-slots are set to "west" and "east", respectively. The DirectionVertical-class is subdivided into NorthDirection and SouthDirection; their DirectionType-slots are set to "north" and "south", respectively. Since I use the Synonyms of each class, when searching the ontology for a suitable concept matching a natural language input, additional terms are added as Synonyms to the Direction-classes. For example, the Synonyms-slot of the WestDirection has the values "west" and "left". If the input sentence is, for example, "Move the blue square to the left", the concept WestDirection will be found

through search with the term "left" and an instance of WestDirection will be set as value for the MoveDirection-slot in the Move-class.

The Distance-class represents the concept of distance in the ontology. The only slot in the Distance-class is DistanceSize, and its value is an instance of Size. The Distance-class has two subclasses: QualitativeDistance and QuantitativeDistance. The values of the DistanceSize-slot in the QuantitativeDistance-class are limited to instances of the QuantitativeSize-class, and in the QualitativeDistance-class, they are limited to instances of the QualitativeSize-class. For example, in the sentence "Move the square slightly to the right", the term "slightly" describes the distance qualitatively. I considered the term "slightly" to be equivalent to VerySmallSize, and thus "slightly" is added to the Synonyms of the VerySmallSize-class. Then, an instance of the class VerySmallSize is set as value of the DistanceSize-slot in this example.

I added five subclasses to the QualitativeSize-class in order to enhance the search in the ontology. The classes are: VerySmallSize, SmallSize, MediumSize, LargeSize, and VeryLargeSize. The QualitativeSizeValue-slot of each of these classes contains just one allowed value representing the respective size. For example, the allowed value for the VerySmallSize-class is "verysmall".

For the same reason, I added subclasses to the AreaHorizontal- and AreaVertical-classes. Left, Right, and CenterHorizontal are the new subclasses of the AreaHorizontal-class and Top, Bottom, and CenterVertical are the new subclasses of the AreaVertical-class.

The sentences involving rotate-actions showed that users described the rotation

not only by using the angle of rotation but also by specifying the direction of rotation. For example, in the sentence "Rotate the square 45 degrees clockwise", the user described the rotation by giving the angle (45 degrees) as well as the direction of rotation (clockwise). The Rotate-class has already a slot for the angle of rotation but a slot called RotateDirection had to be added. The RotateDirection-slot has as value an instance of the RotationDirection-class.

The RotationDirection-class has the subclasses Clockwise and CounterClockwise. It has only one slot called RotationDirectionType, which is of type symbol with "clockwise" and "counterclockwise" as allowed values.

Modifying the Shape-class

In order to represent the spatial relation of a shape with other shapes, I used a slot called SpatialRelation in the Shape class. For example, consider a scenario, in which a square is on the canvas and the user wants to add a circle left of the square. The SpatialRelation-slot of the circle will be set to "leftof" and the frame of reference will be the square. Similarly, the SpatialRelation-slot of the square will be set to "rightof" and the frame of reference will be the circle. This setting exactly reflects the current spatial relation between the square and the circle. However, when the user removes the square in a later modification of the scene, these spatial relations will not be valid anymore. In order to address this problem, I replaced the SpatialRelation-slot with a slot called ActualPosition. The ActualPosition has as value an instance of the Point2D-class representing the actual location of the shape on the canvas. The Point2D-class is added under the Shape2D-class; it has X- and Y-slots with integer

values, which represent the location of a point-shape on the canvas (in pixel). By using the ActualPosition of the shape at any given time, the spatial relations between a shape and other shapes can be determined through simple calculations.

In the collected sentences, I found that users describe among other features the orientation of a shape. For example, in the sentence "Draw a diagonal line" the term "diagonal" refers to the orientation of the shape. I thus added a ShapeOrientation-slot to the Shape-class, which represents the original orientation of the shape. The ShapeOrientation-slot has as value an instance of the Orientation-class (see below), for example horizontal, vertical, or giving a specific angle.

The Orientation-class is added to the ontology to represent the orientation of shapes. It has a slot called Angle, which represents the angle between the shape and the horizontal axis going through the center of the canvas. It accepts integer values between 0 and 360. The Orientation-class has three subclasses: Horizontal, Vertical, and Diagonal. The Angle-slot for the Horizontal-class is set to 0 and for the Vertical-class to 90. The Diagonal-class has two extra slots called HorizontalDirection (an instance of the DirectionHorizontal-class) and VerticalDirection (an instance of the DirectionVertical-class) in order to represent the type of diagonal orientation. I chose these extra slots according to the description of the users for a diagonal line in sentences like "Draw a black diagonal line in the middle of the window sloping down to the right" or "Draw a line going to the right lower corner". As can be seen, users describe the type of diagonal orientation according to a vertical and a horizontal direction. In the first sentence, "down" represents the vertical direction and "right" represents the horizontal direction. The same directions are used in the

second sentence. For modelling this, I added two subclasses called SlopingDown and SlopingUp to the Diagonal-class. In the SlopingDown-class, the HorizontalDirection-slot is set to "east" (right) and the VerticalDirection-slot is set to "south" (down) representing a 45 degree angle between the line and the horizontal axis of the canvas. In the SlopingUp-class, the HorizontalDirection is set to "west" (left) and the Vertical-Direction is set to "north" (up), thus representing a 135 degree angle between the line and the horizontal axis of the canvas.

The next change in the Shape-class was to add a slot called ShapeTransparency. This slot is used to model the transparency of a shape, for example, the term "solid" in the sentence "Draw a solid blue square in the center". The ShapeTransparency can have instances of the Transparency-class as values. The Transparency-class has a slot called TransparencyType of type symbol with allowed values "solid" and "transparent". In addition, the Transparency class has Solid and Transparent as subclasses, with TransparencyType-values solid and transparent, respectively. This is done again in order to allow the addition of synonyms to these classes and thus to enhance the search process.

I added a new class called Border to the ontology, in order to model borders of shapes properly. The Border-class has two slots: BorderColor, which has an instance of Color as value with Black being the default value; and BorderThickness, which is of type symbol with allowed values "thin" and "thick" and default value "thin". A slot called ShapeBorder was then added to the Shape-class, which has as possible values instances of the Border-class. This represents borders of shapes explicitly, and allows the interpretation of sentences like "Create a 1 inch white square with a thin

black border".

Since users sometimes referred to the start point and the end point of a line, I added two slots to the Line-class: StartPoint and EndPoint. The StartPoint- and EndPoint-slots are filled with instances of the Point2D-class. I also replaced the EdgeSizes-slot of the Polygon-class with a new slot called Edges. The Edges-slot is a multiple-value slot accepting instances of the Line-class. This change helps to represent the polygon-shape as a collection of line segments, i.e. edges.

Modifying the Size-class

The QuantitativeSize-class has a UnitofSize-slot of type symbol. In order to allow the use of synonyms, I changed the type of the UnitofSize-slot to an instance of a new class called Unit. The Unit-class has a slot called ValueUnit, which is of type symbol and has allowed values "centimetre", "inch", "pixel", "degree", and "%". The Unit-class has five corresponding subclasses: Centimeter, Pixel, Inch, Degree, and Percent. This expansion allows the use of various unit-types in the interpretation of sentences.

Modifying the Position-class

Figure 4.9 shows the Position-class hierarchy in the second version of the ontology. For the sentence collection, I designed some tasks, in which the users had to add a new shape between two existing shapes. In order to interpret related instructions, the area between shapes needed to be modelled in the ontology. I thus added two subclasses under the Relative-class, called OneShape and TwoShape. The OneShape-class represents the relative position of a shape related to another shape as frame of reference, and the TwoShape-class represents the relative position of a shape with

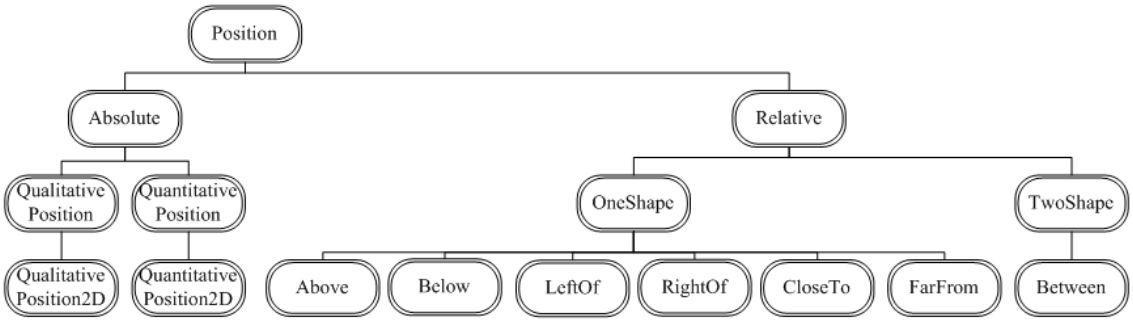

Figure 4.9: The Position-class hierarchy in version 2 of the ontology

respect to two other shapes. Subsequently, I changed the FrameOfReference slot in the Position-class to a multiple-value slot so that it can accept more than one frame of reference. I added under the TwoShape-class a new class called Between. I also added 6 new classes for relations under the OneShape-class: Above, Below, LeftOf, RightOf, CloseTo and FarFrom. A respective symbol is set as value of the slot ValueofRelativePosition of these classes. For example, the allowed value of the ValueofRelativePosition-slot for the Above-class is "above". The reason to add these extra classes under OneShape and TwoShape is to enable the use of synonyms in the ontology. In order to deal with sentences like "Move the circle close to the square" and "Move the circle away from the square", I added the classes CloseTo and FarFrom to represent the corresponding concepts.

4.3.2 Hierarchy Rearrangement

In order to obtain a more logical classification of the ontology and to ease the implementation of the natural language interface, I rearranged some of the classes in

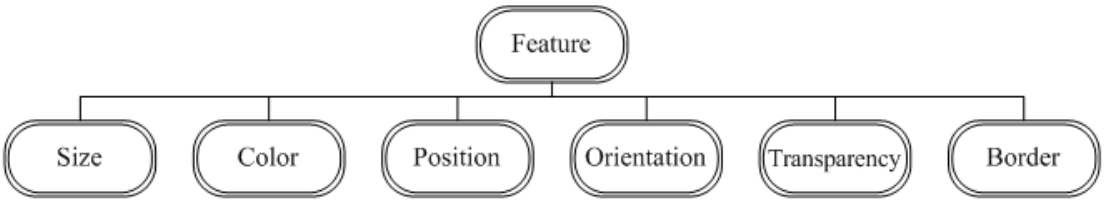

Figure 4.10: The Feature-class hierarchy in version 2 of the ontology

the hierarchy.

I created a new class called Feature with no slots, and moved all the classes representing features of Shape-classes under this class (see Figure 4.10). Subclasses of the Feature-class are: Color, Size, Position, Orientation, Transparency, and Border.

4.4 Ontology Statistics

In this section, I give a brief statistics regarding the ontology, such as the number of classes, features, and instances.

In total, the ontology contains 108 classes, 77 slots, and 170 pre-defined instances. 33 of those 108 classes have an abstract role, and 75 have a concrete role. 16 out of the 77 slots are of type integer; 4 slots are of type string; 4 slots are of type float; 10 slots are of type symbol; and 43 slots are of type instance. 140 of the 170 pre-defined instances are instances of the Color-class; 12 are instances of the Size-class; and 4 are instances of the Direction-class.

The complete concept hierarchy according to version 2 of the ontology is shown in Appendix A.

4.5 Ontology Reuse

The ontology was created using the Protégé-Frame editor. The Protégé-Frame editor is a free open source editor that can be downloaded from the Protégé website (Stanford University, 2008a). The ontology can be viewed and edited with the Protégé-Frame editor by other users. The ontology is saved in three files that are in a CLIPS-based text format. The main file is *GraphicsOntology.pprj* file. It contains information specific to the Protégé-frame editor. The *GraphicsOntology.pont* file is a text file containing information about the classes and slots. The *GraphicsOntology.pins* contains information about the instances.

Since Protégé is written in Java, the ontology can be easily used in a Java program. The Protégé Application Programming Interface (API) can be downloaded from the Protégé website. By adding the related JAR file to the program, the user can access the ontology and use its knowledgebase. Detailed guidelines for accessing the ontology through a program can be found on the Protégé website (Stanford University, 2009).

Chapter 5

Natural Language Interface Architecture

5.1 Natural Language Processing

A natural language processing system accepts natural language text as input and creates a corresponding computer understandable version of the natural language input sentences. Firstly, the sentence is tokenized and each token is assigned with a part-of-speech tag based on the word's definition and context. For example, in the sentence "The book has 100 pages" the part-of-speech tag assigned to "the" is DT (determiner) and "book" is tagged in this case as NN (noun), but the part-of-speech tag for "book" in the sentence "Book the flight." is VB (verb). The decision to assign NN or VB to "book" is made by the part-of-speech tagger according to the context, i.e. the other surrounding words or tags. After the sentence is tagged, a syntactical analysis is performed in order to find syntactical relationships between the words

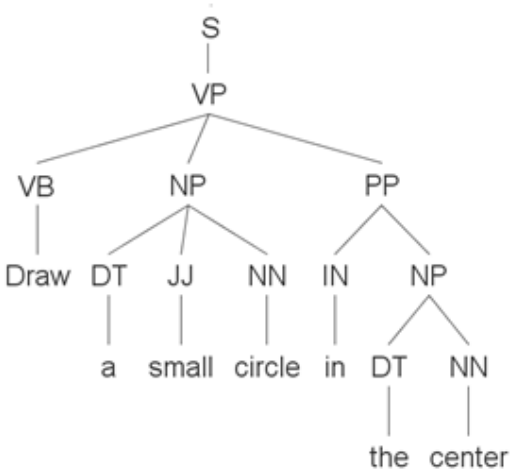

Figure 5.1: Parse Tree

and to group them accordingly into different phrases. This syntactical analysis is performed by the "parser". The output of the syntactical analysis is a parse tree, constructed based on an underlying grammar. The grammar consists of structural rules describing how to compose words into phrases. As an example, the output of the syntactic analysis for the sentence "Draw a small circle in the center." is shown in the parse tree in Figure 5.1.

Part-of-speech tags in this example are the pre-terminal nodes, i.e. VB (verb), DT (determiner), JJ (adjective), NN (noun) and IN (preposition). The parser groups related words into different phrases according to the natural language grammar. There is a verb phrase (VP) which contains a verb (VB, Draw), a noun phrase (NP) and a prepositional phrase (PP). The noun phrase contains a determiner (DT, a), an adjective (JJ, small), and a noun (NN, circle). The prepositional phrase contains a preposition (IN, in) and a second noun phrase, which consists of a determiner (DT, the) and a noun (NN, center). Now, it is easier to derive a semantic representation

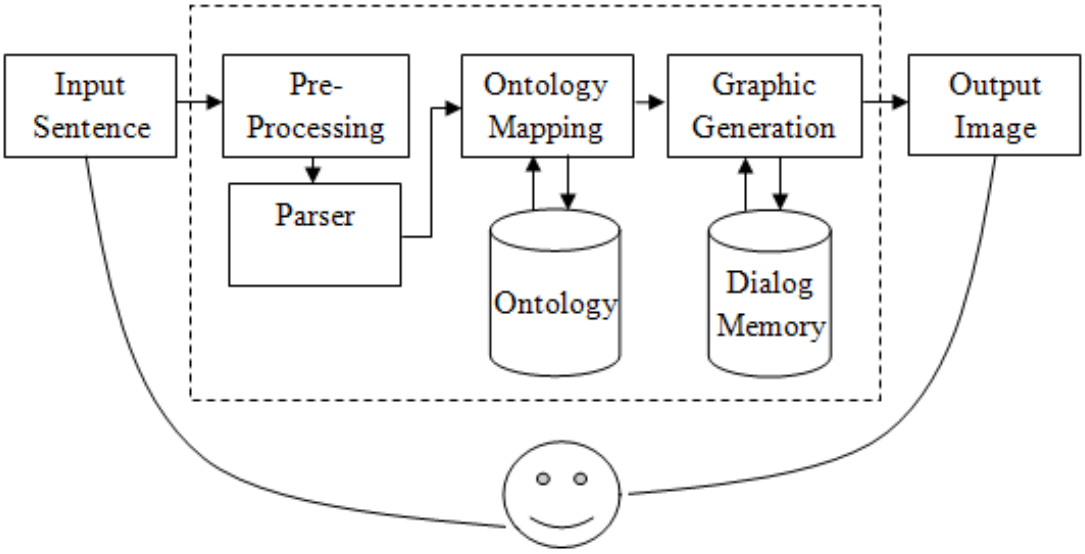

Figure 5.2: Natural Language Interface Architecture Diagram

of the sentence, since the relationships between the words are clear. For example, the adjective "small" describes an attribute of the noun "circle". With additional information from the ontology, we can derive that "small" refers to the value of the size-feature of a shape of type "circle" (see Section 5.2).

Figure 5.2 shows the architecture of the natural language interface implemented to test the ontology. The different components of the natural language interface are described in separate sections below.

5.1.1 Pre-Processing

In some sentences, parts needed to be added or changed in order to allow these sentences to be properly parsed and / or mapped to the ontology. For example, a verb "Draw" has to be added to input sentences which do not contain a verb in order

to obtain a correct parse tree. Such sentences appear occasionally, when users try to abbreviate the input by just describing the shape to be created, for example "blue square in the center". Following is a description of such simple changes made to sentences during the pre-processing phase.

Inputs, that do not include any verb, have to be recognized and the verb "Draw" has to be added. I utilized the parser to find out whether the input sentence has a verb. The parser has a built-in part-of-speech tagger (discussed in Section 5.1.2) that assigns a part-of-speech tag to each word in the sentence based on the word's definition and context. The presence of a verb can be determined by searching the list of the part-of-speech tags of the words in the sentence. If there is no "VB" tag in the list, the verb "Draw" will be added in the beginning of the sentence.

For the sentence "Draw a light blue square in the center", the parser would recognize "light" and "blue" as adjectives for the noun "square". In the further processing, "light" would be ignored, since there is no concept in the ontology corresponding to "light". However, "light" is an adverbial modifier for the adjective "blue" and they both should be mapped to the "LightBlue" instance of the Color-class in the ontology. This problem can be solved either by assigning suitable part-of-speech tags to the single words, or by fusing both words together into one word. I decided to fuse both words together, since the parser then can recognize "lightblue" as an adjective for "square", and the mapping process can find the corresponding colour-instance "lightblue" in the ontology. The same operation is performed for the following words, which might be found before a colour-name: "dark", "bright", "sky", and "baby".

In sentences like "Draw a one inch square in the center", which contain a num-

ber word (e.g. "one"), the number word is replaced by a corresponding digit. This was done to standardize the writing of numeric values and because the slot in the QuantitativeSize-class in the ontology accepts only numeric values. The same operation is performed for the following words: "two", "three", "four", "five", "half", and "ninety".

Similarly, fractions are translated into a standard format. For example, for a sentence like "Draw a square about one fourth of the canvas size.", the following changes are made during the pre-processing: "fourth" is changed to "/4"; "one" is changed to "1"; and then both are replaced with "0.25". The same operations are performed on the following words: "third", "3rd", and "eighth". In some sentences, like "Draw a 1-inch green square in the middle of the canvas" and "Draw a small white square in the top-left corner of the window", the users attached two words together using a hyphen "-". In such cases, the two attached words need to be separated, since each of them has a corresponding concept in the ontology. In the term "1-inch", "1" represents the size value and "inch" represents its unit. The pre-processing is responsible to separate "1" and "inch" and remove the "-". The same operation is performed on the following words: "-right", "-left", "-inch", "-inches", "-pixel", "-pixels", "-cm", "-centimetre", and "-degrees".

In sentences like "Draw a 2 x 2 cm square on top of the canvas." and "Draw a 1 inch / 1 inch square in the center with red.", the users used either "x" or "/" to separate two numbers or measurements. The "x" and "/" are replaced by "by" since "x" and "/" do not belong to a particular part-of-speech tag. Since the "/" is recognized as a separate token due to a space between "/" and previous and next

tokens, it would not affect the "/" in the token "1/4".

5.1.2 Parsing Process

Parser

For the syntactical analysis as part of the natural language interface, I chose the Stanford parser (The Stanford Natural Language Processing Group, 2009). The Stanford parser is an open source parser developed by a well known natural language processing research group at Stanford University. The Stanford parser parses sentences with a high accuracy. The parser is well documented and thus relatively easy to use and to integrate with other parts of the system. The parser is a probabilistic context free grammar parser, and can thus be trained for a specific corpus of sentences. The possibility of training the parser was useful in my case, since the corpus contains a lot of sentences that have an unusual syntactical structure. By training the parser in a suitable manner, the correct parse trees of these sentences can be obtained.

The Stanford parser has three different built-in parsers: a high accuracy unlexicalized parser (Klein and Manning, 2003), a lexicalized dependency parser, and a Factored Model parser (Klein and Manning, 2002). The high accuracy unlexicalized parser does not consider the lexicon while calculating and assigning probabilities to the grammar rules. The Factored Model parser uses the parse trees created by the high accuracy unlexicalized parser and the lexicalized dependency parser in order to infer the best parse tree for a sentence.

Probabilistic Context Free Grammar Parser A Probabilistic Context Free Grammar (PCFG) parser parses sentences based on a probabilistic context free grammar. A probabilistic context free grammar is a weighted context free grammar, where the weights assigned to the grammar rules represent probabilities of the occurrence of use of this grammar rule during parsing. It is often possible to have more than one matching parse tree for a sentence. Using a weighted CFG, the best parse of a sentence is the derivation, which has the maximum weight among all derived parse trees of this sentence. The probabilities are estimated during a training process from the number of occurrences of specific grammar rules in an annotated corpus, like the "Penn Treebank". The Penn Treebank is a collection of parsed sentences represented as so-called "Penn trees" (University of Pennsylvania, 2009).

How the Parser Works The parser accepts a sentence or a list of tokenized words as input. A token is a word in the sentence that has a corresponding entry in the lexicon, e.g. "NN" → "circle". The process of converting the sentence into a list of tokens is called tokenization. The part-of-speech tagging is performed during the tokenization. Each token is assigned a part-of-speech tag according to the lexicon. Therefore, the tokenization has to match with the treebank-format used to create the grammar.

In the next step, the parser parses the sentence based on the grammar rules and the probabilities assigned to them. The parse tree with the highest probability is returned as result of the parsing process. The output of the parser will be used to create a semantic representation of the sentence.

The parser contains the following files:

1) A Lexicon that consists of lexical entries, one per line, each of which is of the form $A \rightarrow a$ ("A" is a non-terminal and "a" is a terminal) and followed by a raw count. These are three examples of lexical entries:

"VB" → "draw" SEEN 100.0

"JJ" → "white" SEEN 39.0

"NN" → "square" SEEN 204.0

For example, the first entry means that the word "draw" has been recorded (SEEN) as a verb VB in the Treebank for 100 times.

2) A Unary Grammar consisting of unary rewrite rules, one per line, each of which is of the form $A \rightarrow B$ ("A" and "B" are non-terminals), followed by the negative normalized log probability. Below two unary grammar rules are shown.

"S" → "VP" -0.77361953

"NP" → "NN" -3.5216703

For example, the first rule means that the normalized log probability that the non-terminal "S" is replaced by a "VP" during parsing is -0.77361953.

3) A Binary Grammar consisting of binary rewrite rules, one per line, each of which is of the form $A \rightarrow B\ C$ ("A", "B", and "C" are non-terminals), followed by the negative normalized log probability. Following are examples of binary grammar rules.

"S" → "VP" "." -0.8193418

"VP" → "VB" "NP" -1.4193943

"NP" → "DT" "NN" -1.2467369

The first rule says that the normalized log probability that "S" is replaced by "VP" followed by "." is -0.8193418.

Stanford Typed Dependencies The parser produces two types of output: a parse tree in the Penn tree format and in addition a structural representation of the parsed sentence using the set of Stanford Typed Dependencies (de Marneffe and Manning, 2009). The typed dependencies provide an easy to understand description of the grammatical and semantic relationships in a sentence. For example, the following is a collection of the typed dependencies produced for the sentence "Draw a small square":

$det(square, a)$

$amod(square, small)$

$dobj(draw, square)$

The "det" dependency shows that the determiner "a" is a determiner for the noun "square". The "amod" shows that the adjective "small" is an adjective modifier of the noun "square". Finally, the "dobj" represents the direct object relationship between the verb and the noun (the noun "square" is a direct object of the verb "draw"). The complete reference for the Stanford Typed Dependencies can be found in (de Marneffe

and Manning, 2009). We use the collection of typed dependencies as an input for the ontological representation, since it provides a comprehensive collection of grammatical and semantic relationships, which are easier to interpret and to use in the ontology mapping process.

Training the Parser

I trained the Stanford Unlexicalized PCFG parser using parse trees in the Penn tree format. As basis for this training, I took 60% of the sentences which were collected in the sentence collection phase (see Section 3).

However, it turned out that for some sentences the top result, i.e. the parse tree with the highest probability, was not accurate or not suitable with respect to the structural relations as well as the typed dependencies. During the sentence analysis, I discovered that a significant number of the sentences follow a specific unusual structure. The sentence starts with a verb and continues with an object. Later in the sentence, the user gives more specifications and modifications for the object or the verb in the form of an adjectival phrase or an adverbial phrase. For example, in the sentence "Draw a blue square in the center of the canvas, about an inch wide.", the phrase "about an inch wide" modifies the object "square" even though it comes after the prepositional phrase, at the end of the sentence. Another example is "Create a 1 inch square in the middle of the page with cyan.", where the phrase "with cyan" is an adjectival phrase to modify the object "square". The parser tries to bind a part of the late modification of the object or verb to the prepositional phrase, which leads to a wrong parse tree structure. The default standard parser thus assigns the

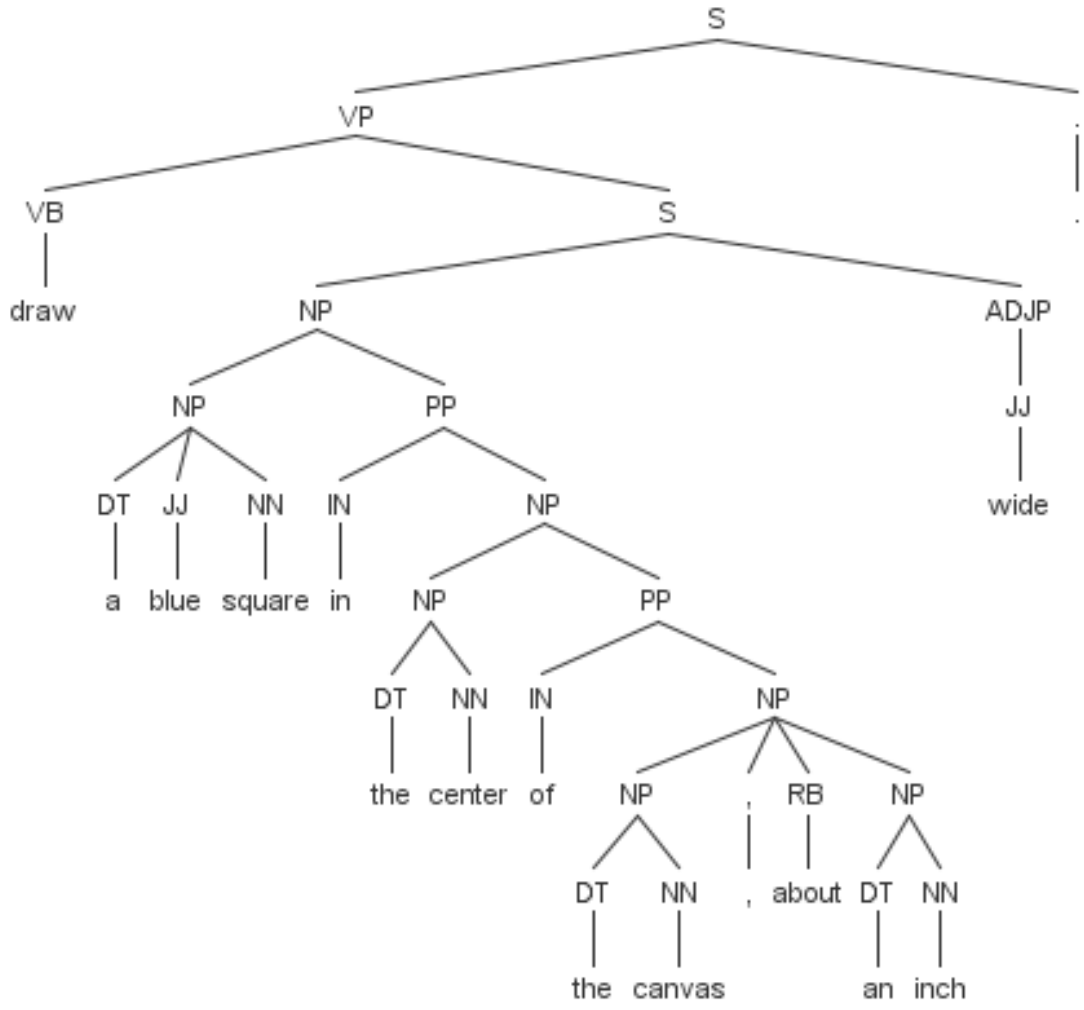

Figure 5.3: The wrong parse tree

highest probability to a wrong parse tree and generates a partially wrong set of typed dependencies.

For example, Figure 5.3 shows that the NP "about an inch" is separated from its ADJP "wide" and none of them is related to the respective noun "square", which should be modified be the complete ADJP "about an inch wide".

The phenomenon of late adjectival or adverbial phrases occurred very often, and thus the parser had to be specifically trained for this type of sentence. In order to produce a new adjusted parser based on training the default Stanford parser, I needed to provide a treebank in the Penn tree format with the correct parse trees for these uncommon sentences. For creating this training tree bank, I used 60% of the sentences in the collected corpus. The Stanford unlexicalized PCFG parser was then utilized to create the appropriate Penn tree for each of these sentences. The Stanford parser provides an option to view the k best parse trees for a sentence. Setting k=20, I could look through the best 20 parse trees of each sentence in order to find the appropriate parse tree. In some cases, the desired parse tree structure was not among those top 20 parse trees and I had to create it manually. Each of these selected parse trees is then added to the tree bank. Finally, the new tree bank could be used to create an improved and adjusted parser through training. This modified and corpus-trained parser was used to analyze the test sentences in the ontology evaluation phase (see Chapter 6).

5.2 Ontology Mapping

As mentioned in section 5.1.2, the second output of the parser, besides the parse tree, is a set of typed dependencies. These typed dependencies are subsequently used to create the ontological representation. In this section, the mapping of the typed dependency structure to the corresponding concepts in the ontology is described. A typed dependency has a name and two elements. For example, in $amod(square, red)$ the name is "amod", the first element is "square", and the second element is "red".

The "amod" dependency represents an adjective modifier relation between the first element and the second element. It means that "red" is an adjective modifier of "square".

There are some important typed dependencies used in the description of the ontology mapping process, which need to be explained here.

The "dobj" dependency represents a direct object relation between the main verb and the object in the sentence. For example, the corresponding "dobj" dependency for "Draw a square." is $dobj(Draw, square)$. The "square" is a direct object for the verb "draw". The "det" dependency represents the determiner relation between the first and the second element. The "det" dependency for the sentence above is $det(square, a)$, which means "a" is a determiner for "square". The "nn" dependency represents a noun compound modifier relation among its elements. For example, in the sentence "Draw a blue outline square.", "square" and "outline" are nouns and "outline" modifies "square". The corresponding "nn" for the sentence is $nn(square, outline)$. The "num" dependency represents the numeric modifier relation. For example, the corresponding "num" dependency for the sentence "Draw a 1 inch square" is $num(inch, 1)$. The "advmod" dependency represents the adverbial modifier relation. For example, in the sentence "Rotate the square clockwise" the "advmod" dependency is (rotate, clockwise), meaning that "clockwise" modifies the verb "rotate". Other important dependencies are different "prep_x" dependencies, that represent a prepositional modifier relation between the elements. The x can be replaced by any preposition. For example, the "prep_in" dependency for the sentence "Draw a square in the center." is $prep\_in(draw, center)$. It shows that a "draw"

action will be performed in the "center".

The output of the ontology mapping process is a list of Shape-instances, which are created based on the ontology. This list will be converted and transferred into the dialog memory. The diaolog memory is a stack that stores the existing graphics shapes, which are currently on the canvas (see Section 5.3.1). The graphics shapes in the dialog memory can be accessed by other components, in particular by the graphics generation component to create the updated canvas (see Section 5.3).

The ontology mapping process is described in detail in the sections below.

5.2.1 Finding the Action

The first step in the ontology mapping process is to find the appropriate action in the ontology. The action is identified by the central verb in the sentence. The central verb in the corpus sentences typically has a direct object - the graphics object or feature, which is affected by the action. Consequently, the verb can be found in the "dobj" type dependency. The first element in the "dobj" dependency is the central verb of the sentence and denotes the main action. In order to find the concept corresponding to the verb in the ontology, a search will be performed by browsing through the action-branch of the ontology. This process scans the action-names for all subclasses of the Action-class in the ontology, until a matching action has been found.

If the found action-concept is a subclass of the Create-class, an extra operation for disambiguation has to be performed to further identify the action-type. The reason for this extra operation is explained in the following example. In the sentence "Fill

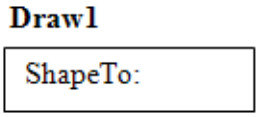

Figure 5.4: Draw1 instance

a red square in the center", the verb "fill" represents a create-action, while in the sentence "Fill the square with blue" the verb "fill" represents a change-action. The best way to clarify this ambiguity is to check the object of the verb. If the object has "a" as a determiner, the action type is create, since the unspecific determiner "a" indicates that the object does not yet exist.

Otherwise, if the object has "the" as a determiner or the object is a pronoun like "it", the action is a change-action, which is applied to a pre-existing object on the canvas. In the example, both actions are change-color action. After finding the correct subclass of the Action-class, an instance of this class has to be created in order to assign appropriate values to its slots.

5.2.2 Create-Action

If the action is a Create-action, an instance of the class "Draw", which represents the creation of a shape in the ontology, has to be generated. This instance gets an identifying name, for example "Draw1" (see Figure 5.4).

Finding Shape

The class "Draw" has a slot called "ShapeTo", which accepts as filler an instance of the Shape-class. The ShapeTo-slot thus links to the shape, which is the object

of the action. Now, the shape mentioned in the sentence has to be identified. We find this shape as the second element in the "dobj" dependency. For example, in the sentence "Draw a square.", the "dobj" dependency is $dobj(Draw, square)$ and "square" represents the shape. The corresponding concept for the shape can be found in the ontology by searching the Shape-concepts and their Synonyms-lists. If we find a Shape-class with a name corresponding to the word used to denote the shape in the input-sentence, an instance of this class will be created. This instance will be assigned as filler to the ShapeTo-slot of the instance "draw1" as well as to the Shapes-slot of the "maincanvas" (see Section 5.3.1 on "dialog memory").

If no Shape-class corresponding to the word for the shape in the input-sentence can be found in the ontology, a default instance of the class Square will be created and assigned to the ShapeTo-slot. At the same time, the shape will be added to the dialog memory, where it is available in case of future modifications. The next step is to fill other slots of the created Shape-instance.

Finding Colour

The sentence analysis showed that users typically describe the colour of a shape either by using a colour-name as adjective before the shape-name, or by adding the colour-name using the preposition "with" after the shape-name. Both cases have to be dealt with when a Create-action is instantiated.

In the first case, i.e. if the colour-name is in a pre-noun, adjective position, we need to find all the "amod" dependencies, whose first element is the shape-name. The second element of the "amod" is an adjective that represents a feature of the shape.

Then, we check in the ontology, whether there is a corresponding instance of Color using its Synonyms-slot. If there is a corresponding instance of Color, i.e. a specific colour like "Lightblue", it will be assigned to the FillColor-slot of the Shape-instance.

In the second case, i.e. if the colour-word is added with a preposition after the shape-noun, the ontology search has to be performed using all "prep_with" dependencies, whose first element is the shape-name.

If no colour-feature is found through these operations, the default Color-instance will be assigned to the FillColor-slot of the Shape-instance.

If the user specifies a colour for the border of a shape, for example "Draw a black border square.", we need to check the "nn" dependencies. If the first element is the shape-name and the second element of the dependency is a word denoting a border, we need to create an instance of the Border-class and assign it to the ShapeBorder-slot of the Shape-instance. Then, we have to find the "amod" dependency, whose first element (denoting the object) is the same as the second element of the "nn" (denoting the border). In the next step, the second element of the "amod" (denoting the modifier) is used to search the ontology for a corresponding instance of the Color-class. In case the instance (a specific colour) is found, it has to be assigned to the BorderColor-slot of the Border-instance. The same operations will be performed in case of "prep_with" dependencies.

The dependencies for the sentence "Draw a black border square." are shown as an example below.

$det(square, a)$

$amod(border, black)$

$nn(square, border)$

$dobj(draw, square)$

In case the user does not specify a colour for the border, the BorderColor will be set to the same value as the FillColor of the shape. If the user does not mention any colour at all, default values are used for the FillColor (white) and the BorderColor (black).

Finding Size

The size is described by users either qualitatively, e.g. "small", or quantitatively, e.g. "1 inch wide". Dealing with these size-specifications is similar to processing colour-specifications as described above. In case of a qualitative size-specification, the user either adds the qualitative size-value before the shape-name, as in "Draw a small circle", or it is added after the shape-name using the preposition "with", for example "Draw a square with small size.". An approach analogue to the one described above is used to find the qualitative size of the shape but in this case, we search through the subclasses of the QualitativeSize-class. If a matching class is found, a respective instance of the class is assigned to the ShapeSize-slot of the Shape-instance.

If the user specifies the size of the shape using quantitative values, as in "1 inch wide", we need to look at the "num" dependency. The first element of the "num" dependency represents the unit of the size-value, e.g. "inch", and the second element

shows the numeric value, e.g. "1". In the next step, an instance of the corresponding Unit-class will be created and the Unit-instance as well as the numeric value will be assigned to respective slots of the instance of the QuantitativeSize-class.

In case the user did not specify a size, the qualitative default size "MediumSize" will be assigned to the ShapeSize-slot of the Shape-instance.

Finding Position

In order to find the position of a newly created shape, we first need to identify the frame of reference. The frame of reference is either the canvas, in which case we have an "absolute position", or it is another shape-object on the canvas, and we then have a "relative position". Example sentences are "Draw a circle in the center of the canvas." (absolute position) and "Draw a circle left of the square." (relative position). In order to find the frame of reference, we search the set of dependencies for specific prepositional dependencies.

If we find a "prep_of" dependency, we check the second element. If it refers to the canvas, then the position is absolute. Then, an instance of the QualitativePosition2D-class will be created, and the maincanvas-instance will be assigned to its FrameOf-Reference-slot.

If the second element of the "prep_of" refers to a shape, the position is relative. For example, in "Draw a circle left of the square.", the second element of the "prep_of" refers to an existing instance of the Square-class referenced by "the square". The first element of the "prep_of" refers to the Position-concept in the ontology (see Figure 4.9 for the hierarchy of the Position-class). The "prep_of" for the sample sentence above

is $prep\_of(left, square)$. The "left" corresponds to the Left-class in the ontology, which is a subclass of the AreaHorizontal-class.

If we do not find a "prep_of", we check the dependencies for the following: "-prep_above", "prep_below", "prep_over", "prep_under", "prep_beside", "prep_near", "prep_between", "prep_to" (for close to), and "prep_from" (for far from). If any of these dependencies is found, the position is relative.

In case of a relative position, the position of the shape is described with respect to an existing second shape. We find this second shape through the second element of the "prep_x" dependency. This second element should refer to an instance of a Shape2D, i.e. a shape, which exists already on the canvas. For example, for the sentence "Draw a circle beside the square.", the created dependencies are:

$det(circle, a)$

$dobj(draw, circle)$

$det(square, the)$

$prep\_beside(draw, square)$

In this example, it is assumed that an instance of the Square-class has been already assigned earlier to the Shapes-slot of the maincanvas-instance. Since the concepts for "beside" and "close to" are considered equivalent in the ontology, an instance of the CloseTo-class, which is a subclass of the OneShape-class, will be created.

Next, the shape-name will be found using the dialog memory. Since the name of the corresponding Shape-instance is stored in the dialog memory along with the

shape-name, the Shape-instance can be found and assigned to the FrameOfReference-slot of the CloseTo-instance.

In case there are two shapes of the same class on the canvas, and the user wants to add a third shape positioned relative to one of these existing shapes, the newer of the two existing shapes will be selected as the frame of reference. Then, the Shape-instance representing this selected shape, which is stored in the Shapes-slot of the maincanvas, will be used as frame of reference.

Currently, the system looks only for the shape-name in the dialog memory. This can be improved in the future versions of the system by using other features of the existing shape, which the user specified as additional criteria for searching the dialog memory. For example, for the sentence "Draw a red circle beside the blue square.", the system should look for a square with blue colour in the dialog memory.

The respective instance of either the QualitativePosition2D-class or the CloseTo-class created in the previous steps will be assigned to the ShapeLocation-slot of the Shape-instance. Also, the actual position of the shape, which are the x- and y-coordinates of the center of the shape, will be calculated and assigned to the slots "X" and "Y" of the Shape-instance. The calculation of the actual position is described in the graphics generation section (see Section 5.3).

If we do not find any of the dependencies mentioned above, we search for the following dependencies: "prep_in", "prep_at", "prep_on", "prep_into", "prep_near", and "prep_to". In these cases, the position is absolute.

If such a dependency is found and its first element is either a verb or an object, the second element will represent the position-value. We then need to search the

subclasses of the Area-class for a class corresponding to the second element, i.e. the position.

In some sentences, the second element does not represent any of the concepts in the Area-class. In this case, we have to search for an "amod" dependency, whose first element corresponds to the second element of the "prep_x" dependency mentioned above. Then, we use the second element of the "amod" to search for a matching class in the subclasses of the Area-class. After finding the corresponding class, we need to create an instance of it, and assign this instance to the ShapeLocation-slot of the Shape-instance.

For example, for the sentence "Draw a square in the top left corner", the created dependencies are:

$det(square, a)$

$dobj(Draw, square)$

$det(corner, the)$

$amod(corner, top)$

$amod(corner, left)$

$prep\_in(Draw, corner)$

The second element of the "prep_in" is "corner" which does not represent any of the concepts in the Area-class. In this case, we look at the "amod" dependencies, whose first element is the second element of the "prep_x" dependency ("corner"). In this example, we find the dependencies $amod(corner, top)$ and $amod(corner, left)$.

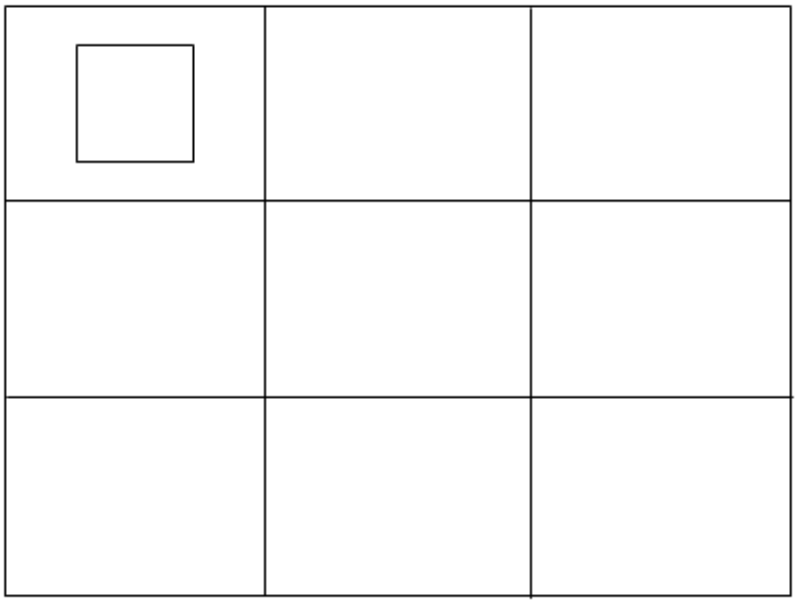

Figure 5.5: Example of absolute positioning

Then we need to search the subclasses of the Area-class for the class(es) corresponding to the second element of the "amod", i.e. "top" and "left". We find the corresponding classes, in this example the Top-class and the Left-class, which together represent the top-left grid as shown in Figure 5.5. Finally, we create instances of these classes and assign them to the ShapeLocation-slot of the Shape-instance.

Finding Orientation

In order to find the orientation of a shape, we need to search the adjectives that the users add before the shape name (object). For example, in the sentence "Draw a diagonal line.", the word "diagonal" refers to an orientation of the line. Similar to the other features, we need to find the "amod" dependency, whose first element is the shape-name. If there is such an "amod" dependency, for example

$amod(line, diagonal)$, we look at the second element of the dependency and search through the subclasses of the Orientation-class in order to find a corresponding concept. If there is a match, an instance of the matched class will be created and assigned to the ShapeOrientation-slot of the Shape-instance. Otherwise, the default orientation, which is an instance of the Horizontal-class, will be assigned to the ShapeOrientation-slot. The dependencies for the sample sentence above are shown below.

$det(line, a)$

$amod(line, diagonal)$

$dobj(draw, line)$

The second element of the "amod" dependency refers to the Diagonal-class in the ontology. Thus, an instance of this class will be created and assigned to the ShapeOrientation-slot of the Shape-instance.

5.2.3 Change-Action

The type of the action is identified in the first step of the ontology mapping process. Each Action-class has different types of slots, which need to be filled, according to the feature that it can change. The only common slot in the Change-class and its subclasses is the ShapeTo-slot, which represents the shape the action is performed on.

Finding Existing Shape

For Change-action sentences, the second element in the "dobj" dependency is usually the shape.[1] If the second element of the "dobj" dependency is "it", the last shape that was added onto the canvas is the sought existing shape.

Otherwise, the second element of the "dobj" dependency is used to search the ontology for a corresponding Shape-concept, accessing the shape-names in the Synonyms-slots of all the subclasses of the Shape2D-class. After finding the Shape-class, we need to search for a respective shape in the dialog memory. The shape in the dialog memory will have the same name as the corresponding Shape-instance in the ontology. If there is no corresponding Shape-concept found in the ontology, the program returns the last shape that was added onto the canvas as sought existing shape.

Then, the Shape-instance will be assigned to the ShapeTo-slot of the respective instance of the Action-class that was created during the finding-action step.

Finding New Colour

There are two types of sentences concerning the ChangeColor-action. First is the group of sentences, in which the shape-name is the direct object of the verb, for example "Fill the square with yellow". The second type is the group of sentences, in which the feature-name is the direct object of the verb, for example "Change the border colour to blue." or "Choose red colour.".

For the first group, the colour can be found by searching the "prep_with" dependency, whose first element is the shape-name. In the same way that was described

[1] There is an exception in some sentences regarding the colour-change, which will be described in Section 5.3.3.

in the finding-colour section, the Color-instance will be found and assigned to the FillColor-slot of the existing Shape-instance.

In the second group, we need to find the feature that has to be changed. Then, we assign the new value to the respective feature. In order to find the feature, we search for concepts, which correspond to the second element of the "dobj", in the subclasses of the Feature-class. For example, for "Change the border to red.", the second element of the "dobj" refers to the Border-class. Then, the new Color-instance ("red") will be assigned to the BorderColor-slot of the Border-instance, which represents the ShapeBorder of the Shape-instance.

If the verb is "change", the second element of a "prep_to" dependency refers to the new Color-instance. For example, in "Change the color to red.", the "prep_to" dependency is $prep\_to(change, red)$. The corresponding Color-instance for "red" can be found in the ontology.

If the verb is "choose", either the second element of the "dobj" or the second element of an "amod" dependency, whose first element is the second element of the "dobj" dependency, will refer to the Color-instance. Examples are "Choose blue." and "Choose blue color.". The dependencies for the sentence "Choose blue color" are shown below.

$amod(color, blue)$

$dobj(choose, color)$

As can be seen above, the second element of the "amod" dependency ("blue"),

whose first element ("color") is the second element of the "dobj" dependency, refers to the new Color-instance.

Finding New Size

We deal here with actions that shrink or enlarge an existing shape. To find out how much the shape needs to be resized, we need to look for the "num" dependency. The first element of the "num" dependency represents the unit of the value and the second element represents the value of the resizing. Normally, the unit is "%". For example, in the sentence "Shrink the square by 50%.", the action is an instance of the Shrink-class, and the amount of resize is 50%. In this case, an instance of the QuantitativeSize-class will be created; "50" will be assigned to its SizeValue-slot and an instance of the Percent-class will be assigned to its UnitofSize-slot. Then, the created QuantitativeSize-instance will be assigned to the ShrinkSize-slot of the Shrink-instance.

In case the user did not specify the amount of resize, default values will be used. The default value for the ShrinkSize is 50% and for the EnlargeSize it is 200%.

Finding New Orientation

To find the new orientation, we need to find an "advmod" dependency in order to find the rotation direction. If there is an "advmod" dependency, the second element of it refers to a subclass of the RotationDirection-class. For example, for the sentence "Rotate the rectangle 45 degrees clockwise.", the "advmod" dependency is $advmod(Rotate, clockwise)$. "clockwise" corresponds to the Clockwise-subclass of the RotationOrientation-class. An instance of the Clockwise-class will be created and

assigned to the RotateOrientation-slot of the Rotate-instance. In addition, we need to find the angle of rotation. The second element of the "num" dependency represents the angle of rotation. The "num" dependency for the example sentence above is $num(degrees, 45)$. The value "45" will be assigned to the Angle-slot of the Rotate-instance. If the direction or the angle of rotation is not specified in the sentence, an instance of the Clockwise-class and "45", respectively, are the default values, which will be assigned to the respective slots of the Rotate-instance.

Finding New Position

The users specified the Move-action in two different ways, either by specifying the new position, as in "Move the square to the right side.", or by specifying the direction and the distance of movement, as in "Move the square 2 cm to the right.".

If the sentence is of the first type, the same operations as for finding the position of a newly created shape (see Section 5.2.2) are used in order to find the new position of a shape as result of a Move-action. The found Position-instance will be assigned to the NewLocation-slot of the Move-instance.

If the sentence belongs to the second type, the direction and the distance of movement need to be extracted from the sentence. In order to find the direction, we need to find a "prep_to" dependency. The second element of the "prep_to" dependency represents the direction. The "prep_to" dependency for the sentence "Move the square to the left" is $prep\_to(move, left)$. The term "left" corresponds to the West-class, which is a subclass of the Direction-class, in the ontology. Consequently, an instance of the Left-class will be created and assigned to the MoveDirection-slot of the Move-

instance.

The next step is to find the distance of movement. Users can specify the distance qualitatively, like "Move the square slightly to the right.", or quantitatively, like "Move the square 2 cm to the right.". If the distance is qualitative, we can find the distance value in the "advmod" dependency. The "advmod" dependency for the first example sentence is $advmod(Move, slightly)$. The term "slightly" corresponds to the VerySmallSize-class, which is a subclass of the QualitativeSize-class, in the ontology. Therefore, an instance of the VerySmallSize-class will be created and assigned to the MoveDistance-slot of the Move-instance. If the distance is quantitative, the respective instance can be found with a similar method.

If the user did not specify the distance of the movement, an instance of Quanititative-Distance, which is 1/3 of the canvas width, will be assigned to the MoveDistance-slot of the Move-instance. The new position of the shape will be calculated based on the slot-values of the Move-instance, and will be assigned to the Position-value as well as the X- and Y-slots of the Shape-instance in the ontology (see also Section 5.3).

5.2.4 Delete-Action

For the delete-action, the existing shape will be found using a similar method as described for the change-action. The found Shape-instance will be assigned to the ShapeTo-slot of the Erase-action, which represents the removal of the shape from the canvas. In the next step, the Shape-instance will be removed from the ontology and later from the dialog memory.

Property Name	Description
Px and *Py*	Store the actual position of the center of the shape in the canvas
Size	Stores the actual size of the shape in the canvas
Fcol and *Bcol*	Store the fill-color and border-color
Angle	Stores the angle of the shape with horizon
Height and *Width*	Store the height and width of the box containing the shape
Name	Stores the name of the shape
InstanceName	Stores the name of corresponding instance of the shape in the ontology

Figure 5.6: Properties of the shape-class

5.3 Graphics Generation

This section outlines how concepts from the ontology are mapped onto graphics entities, in order to produce the graphic image. The Graphics Generation system is implemented in Java and uses the Java Graphics package.

The graphics generation is based on a hierarchy of Java-classes, which reflects the hierarchy of Shape-concepts in the ontology. Each Shape-concept in the ontology has a matching shape-class in this hierarchy.[2] Properties listed for the concepts in the ontology have matching properties in the respective graphics classes. For example, the concept "Square" in the ontology is represented in the graphics generation program by a class named "square", which has the same super- and subclasses as the Square-concept, and the same properties (inherited or genuine) as well, like "size", "border-colour", "fill-colour", "position" and "orientation". Figure 5.6 shows all the properties of the shape-class.

[2]Java-classes in the Graphics program are written with a small initial letter, like shape-class, while classes in the ontology start with a capital letter, like Shape-class.

In addition, each shape-class in the graphics program has an associated method called "draw". The draw-method associated with each shape contains code to draw the shape according to its property-values. The values of position, size, and orientation will be assigned to an affine transform matrix. Then the affine transformation will be performed on the shape in order to determine the respective position, size, and orientation of the shape. The colour-values are used to draw the shape.

The Graphics Generation system also has a canvas-class. The canvas-class has Height, Width, and BGcolor as its properties. The properties will be set using the related feature-values of the maincanvas-instance in the ontology. All the drawing will be done in the Graphics component using the maincanvas.

The draw-method for a shape-class describes in a generic form how to draw a corresponding image, i.e. how to create a respective shape on the canvas. For example, the draw-method of the square-class specifies how to create a generic square on the canvas. In order to create the image for a specific shape, the graphics generation needs access to concrete values for the properties of the shape to be created. These values are stored in the Shape-instance, which was created during the processing of the input sentence (see Section 5.2.2).

For example, if the input sentence is "Draw a small blue square.", a Square-instance, which has a BlueColor-instance and a SmallSize-instance as its slot-values, will be created in the ontology through the ontology mapping process. In the graphics generation phase, a corresponding square-instance will be created in the Java Graphics package and, through a conversion process, the slot-values in the Square-instance will be interpreted and assigned to the respective properties of the square-class. In the

example above, the RGB-value of Blue will be converted into a Color-type in the Java Graphics package. The ActualSize-slot of the SmallSize-instance contains the size in pixels; the size-values will be assigned to the size-properties of the square-instance. The square-instance will be added to the dialog memory, and the draw-method of the square-class will be called in order to create the square on the canvas.

Links to the current shape-instances, which are needed to create or update the graphics scene, are stored in the dialog memory. Every time the user enters a new input sentence or instruction, the natural language interface creates or modifies a shape-instance, and - if it is not yet available - creates a respective link to the shape-instance in the dialog memory. Then, the graphics generation system executes the draw-method for all the shapes in the dialog memory in order to create the shape-images on the canvas.

This process will be performed for all types of actions. In case of a Create-action, a new shape is added to the dialog memory and, after performing the mentioned operation, the new shape will be drawn on the canvas. In case of a Change-action, a feature-value of an existing shape is updated and the modified shape will be re-drawn on the canvas. In case of a Delete-action, the shape will be removed from the canvas, as well as from the dialog memory.

I also created specific functions, which convert the feature-values from the ontology into a suitable format, so that they can be assigned to corresponding features of the shape-class in the graphics generation program. In order to assign a colour-value to a shape, the RGB value of the color-instance in the ontology is converted into a Color-type in the Java Graphics package. Another function converts the size-values from

the ontology into corresponding values in pixels. For example, if the size specified through the ontology is 2 centimetres, it will be converted into 80 pixels. Similarly, values for qualitative sizes, like medium or small, are converted.

For the orientation, there is a function to calculate the new orientation of a shape according to the angle of rotation and the direction of rotation information specified in the ontology. For example, if the angle of rotation is 45 degrees and the rotation direction is counter-clockwise, the function assigns -45 to the angle-feature of the graphics-shape, since the rotation in the graphics part is performed clockwise.

For the position of the shape, we have a function that converts the Position-concepts in the ontology into specific pixel positions on the canvas. If the position in the ontology is specified according to the canvas, the function will find the center of the respective grid and assigns it to the position-feature of the shape.

I defined a specific model for each of the Relative Position values in the ontology, in order to find the actual position of a shape in the graphic image. For instance, to fulfill the instruction "Draw a circle left of the square.", the position of the circle on the canvas has to be determined in the "left of" region of the square. The actual position of the circle is located on a virtual horizontal line, which connects the center of the square and the left border of the canvas (see Figure 5.7). The center of the circle is positioned on this virtual line, in the middle between the left edge of the square at point A and the left border of the canvas at point B.

Similar models are defined to calculate the actual position of the shape for other Relative Position concepts of the ontology such as "right of", "above", and "below".

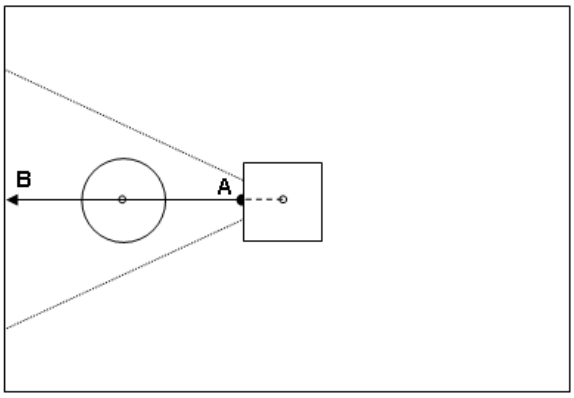

Figure 5.7: Positioning a circle left of a square

5.3.1 Dialog Memory

As mentioned in Section 4.2.6, there is an instance of the Canvas-class in the ontology called "maincanvas". The "maincanvas" stores the Shape-instances that have been created through user instructions and are currently on the canvas. Every time the system processes an instruction, the Shapes-slot of the "maincanvas" will be converted, updated and transferred into the dialog memory using functions mentioned in the graphics generation section (see Section 5.3). When a new instruction is being processed, the system either has to create a new shape, draw it on the canvas, and add it to the dialog memory, or, if it is an instruction to change an existing shape, it has to find the respective shape-instance in the dialog memory, and then it has to modify it by setting or changing specific feature-values.

For example, the instruction "Move the blue circle to the left." causes the system to search for a blue circle in the dialog memory and, after it is found, to change the value of its position feature. The blue circle will then be drawn in the new position. The canvas will be updated every time the system processes an instruction.

5.3.2 The "Lang2Graph" Demo and Test Systems

I implemented two different versions of the "Lang2Graph" (Language to Graphics) system, i.e. the natural language interface combined with the graphics generation component. One is an online system used for demo purposes; and the other one is a batch processing system for testing. The demo version accepts natural language inputs from the keyboard, while the test version reads the inputs from a file.

A screenshot of the demo-version of the system is shown in Figure 5.8. The window in the system contains a canvas, a text input, a panel containing the typed dependencies created for the input sentence and the buttons Execute and Clear. In the demo-version, the user can type a sentence containing one instruction. After pressing the Execute-button, the system processes the input sentence and draws a corresponding image on the canvas. The Clear-button will remove the shapes in the dialog memory as well as the corresponding instances from the ontology and clear the canvas.

For test purposes, I changed the system in a way that it accepts the sentences from a text file. Each line in the file contains one sentence, which has one instruction. In order to deal with multiple sentence instructions, I manually placed each of the instructions in a separate line. I added a delimiter to the last line finishing a sentence. This causes the system during processing to create an image corresponding to each sentence or instruction in a compositional manner. When the system reads the delimiter, the multi-sentence instruction has been processed completely, and it clears the dialog memory and removes the respective instances from the ontology. In this version, the image created by the system will be saved in a file, before the system is

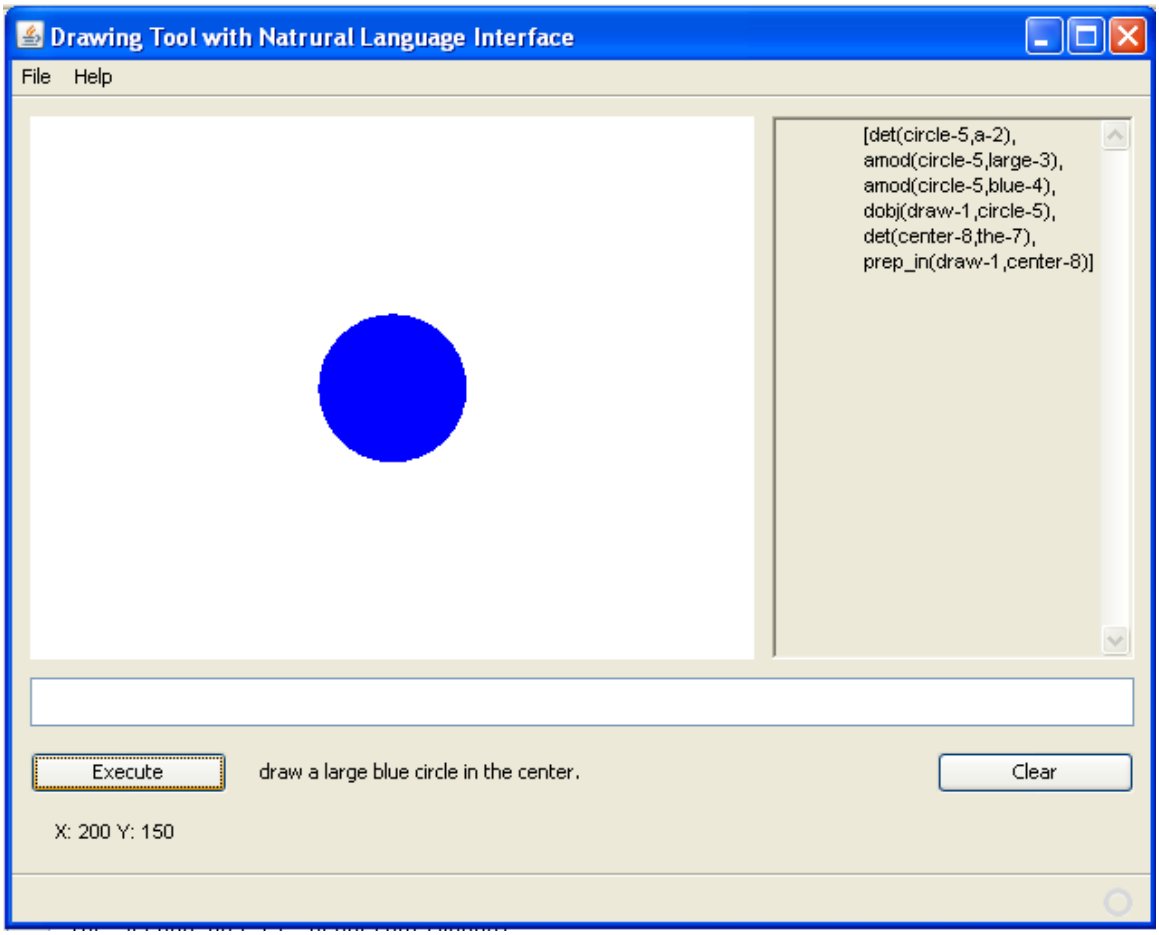

Figure 5.8: Screenshot of the Demo-version of the Natural Language Interface

cleared.

5.3.3 Sentences with Multiple Instructions

Currently, sentences containing more than one instructions, such as "Draw a large square and fill it with blue." cannot be processed at once. Such sentences need to be manually separated into several sentences with one instruction each. Then, each sentence is processed separately. The dialog memory is not cleared in between, in

order to achieve the effect of both instructions being displayed on the canvas.

Interestingly, due to the use of defaults for shapes and their properties as defined in the ontology, the outcome of a complex instruction is not necessarily the same as the output for a corresponding multi-instruction. For example, the multi-instruction "Draw a large square and fill it with blue." does not yield the same image output as the complex instruction "Draw a large blue square.". In the first case, the system creates a square with black border, by default, and then adds the fill-colour blue, while in the second case, the system creates a completely blue square. This means that the composition of two instructions does not necessarily yield the same result as a corresponding single complex instruction.

The screenshots representing the outcome for the mentioned complex instruction and multi-instruction are shown in Figures 5.9 and 5.10. As can be seen, the square in the figure corresponding to the first sentence is completely blue, while the square in the other figure has a black border and a blue fill-colour. The first sentence corresponds to a single, complex Create-action. The system assigns the colour specified by the user to both border- and fill-colour of the shape. For the multiple sentence instruction, the system creates a shape of unspecified colour first, and thus takes the default border-colour (black) and the default fill-colour (white) to draw the shape. In the next step, the system processes the second instruction "fill it with blue.". This instruction corresponds to the ColorChange-action and only the fill-colour of the shape will be changed - unless the user explicitly mentions the border-colour.

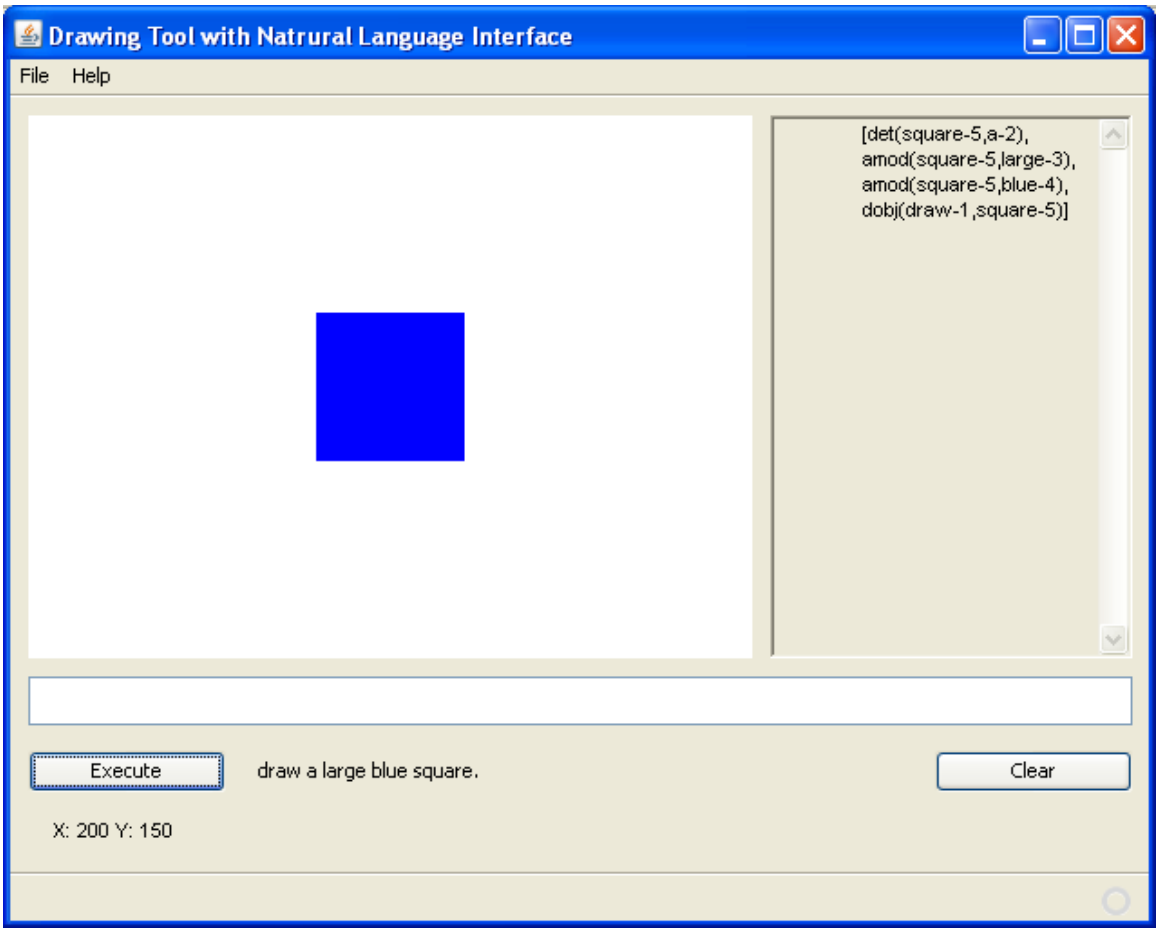

Figure 5.9: Screenshot of the image created for a complex instruction

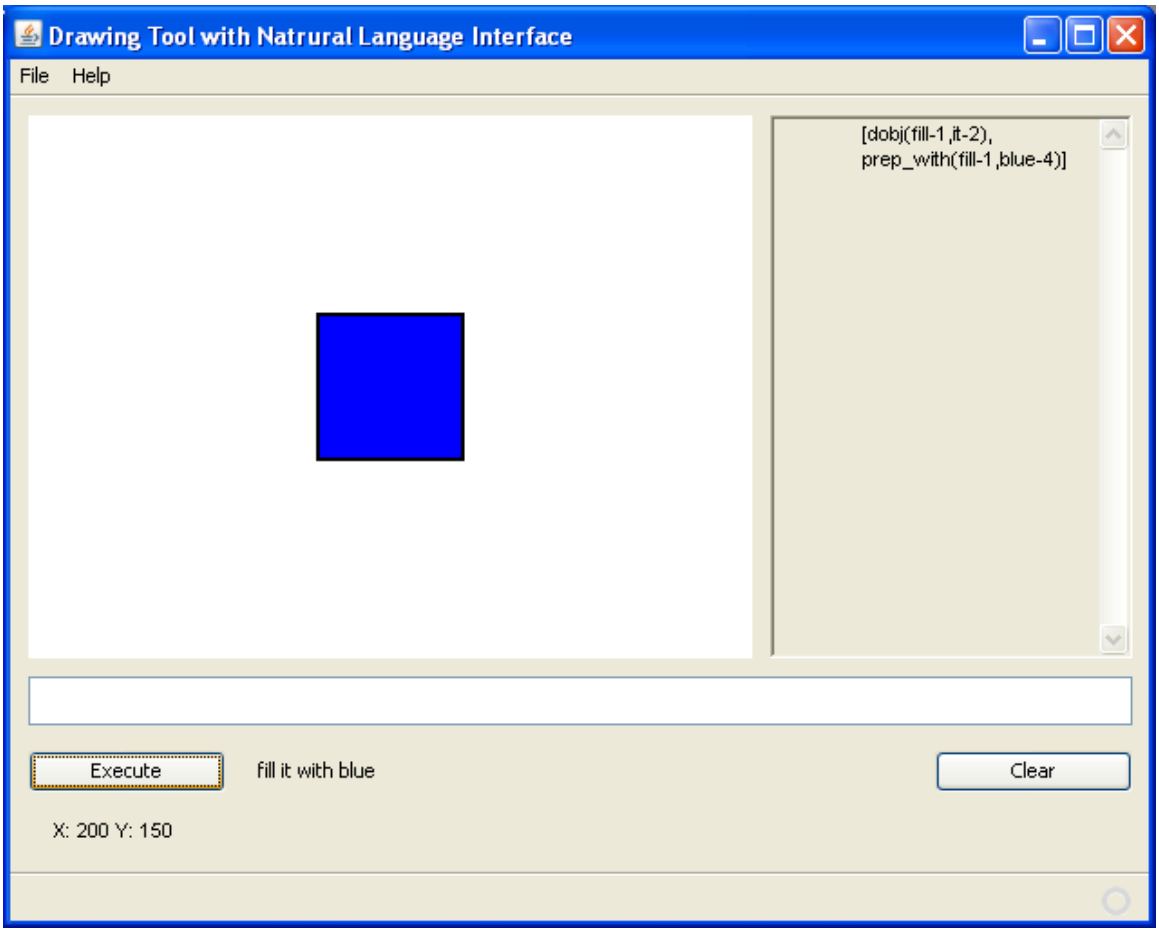

Figure 5.10: Screenshot of of the image created for a multi-instruction

Chapter 6

Evaluation

In order to evaluate the ontology, the ontology was integrated with a natural language interface (see Chapter 5). This combined system reads a natural language input instruction and produces a matching graphic image as output. It analyses the natural language input syntactically, then provides an interpretation of the sentence using a mapping of the dependency structure onto the ontology, and finally it creates an image based on the derived ontology structure.

The natural language interface was used for the evaluation of 40% of the originally collected sentences. The images, which were produced by the system, were then examined regarding their match with the original image or the input sentence, respectively.

In a first evaluation, each created image was initially compared to the corresponding original image, i.e. the image generated by the system was compared to the image shown to the users during the sentence collection. This method, however, led to some unjustified negative evaluation results. These negative evaluation results were in the

majority of cases caused by an under-specification of the image description in the collected sentences. In other words, users often did not specify certain feature values, like the size "large" for a shape in the original image. This led to the creation of a size "medium" shape by the system, which was in the first evaluation unjustly assessed as an error, since the created image reflected properly the instruction. There were some other negative results, which I will present and discuss below, together with possible remedies and solutions to the underlying problems. I performed subsequently a second assessment in this category, in which the given sentences and the created images were compared.

In the second evaluation method, the assessment was performed by human subjects, who are potential users of the system. In this evaluation, the sentences and generated images were shown to the subjects and they were asked to judge whether the corresponding image created by the system was a satisfactory representation of the verbal instruction. The results of this evaluation were overall relatively good, even though some problems showed up in this evaluation, which are discussed below.

6.1 Evaluation 1

6.1.1 Method 1 - Comparing the Original Image and the Created Image

In order to compare the original images shown to the users in the corpus collection phase and the created images produced by the natural language interface, the following six features of the images were used to determine their suitability:

- shape
- fill-colour
- border-colour
- size
- position
- orientation

Each of the mentioned features of an output image is assigned a score of 1 or 0, depending on whether the feature value is correct (1), or not (0) (see Appendix B). The evaluation was due to time constraints performed manually, although it could be easily automated and done by a program. The criteria used in deciding whether a feature is correct or not are discussed below.

Shape

If the shape in the final image is an exact match with the shape in the original image, the score is 1; otherwise, the score is 0. For example, a square in the original image matches with a square in the created image; it would not match with a rectangle.

Fill-Colour and Border-Colour

The fill- and the border-colour of the final image need to be in the same category as the corresponding colours in the original image in order to be scored as 1. Colour

categories according to the RGB scheme are the shades: white, gray, blue, green, yellow, brown, and orange. For example, if in the original image the shape's colour is red and in the final image the shape's colour is orange, the score is 1 since both colours are in the same shade-category. However, if the shape's colour in the final image is green, the score is 0, since green is not in the same category as the colour red.

Size

The score for size is 1, if the shape's size in the created image is equal to the shape's size in the original image + / - 20% of the shape's size in the original image. For example, if the shape's height and width in the original image are 40 pixels, the score is 1, if the height and the width of the shape in the final image are between 32 and 48 pixels. The 20% of the shape's size is selected since then there will be no overlap between the default sizes assigned to the qualitative sizes. For example, the default size for a "small" square in the standard canvas is 20 by 20 pixels; the actual size for a "medium" square is 40 by 40 pixels. Considering the above formula, the acceptable size range for the "small" square is between 16 and 24 pixels and the acceptable size range for a "medium" square is between 32 and 48 pixels. Thus, with a tolerance of +/- 20%, there will be no overlap of the qualitative sizes.

Position

As mentioned earlier, the canvas is divided virtually into a 3 by 3 grid with 9 fields. The position feature is scored with 1, if the center of the shape in the created image is located in the same field as the centre of the shape in the original image;

Total Images	Shape	Fill-colour	Border-colour	Size	Position	Orientation
100% (285)	97% (279)	93% (264)	86% (244)	34% (96)	94% (268)	97% (278)

Figure 6.1: Results for Evaluation Method 1

otherwise, the score is 0.

Orientation

If the shape's orientation in the created image is an exact match with the shape's orientation in the original image, the score is 1; otherwise the score is 0. For example, if the line is diagonal in the original image the line in the final image has to be diagonal.

Results for Method 1

Figure 6.1 shows the number of correct matches in each category for the 285 test cases after applying the method described above.

6.1.2 Method 2 - Comparing Sentence and Created Image

In the next step, I compared the sentences with their corresponding images to find out the sources of the problems in images with 0 scores and possible solutions to correct errors. The problems are categorized based on the affected features, i.e. "shape", "fill-colour", "border-colour", "size" etc.

Shape

In 6 cases, the shape in the created image does not match the shape in the original image. In 5 cases, the users refer to the square as "box". Since "box" is a synonym for rectangle in the ontology, which is assumed to be the correct interpretation, the system creates a rectangle instead of a square. The error can thus be assigned to a wrong description of the shape by the user and not a system mistake, since the created shape is a match with the user's sentence. Therefore, the score for the shape should be 1 in these cases.

In the sentence "draw and fill the image of a yellow sun" the direct object for the verb is "image". Since there is no shape in the ontology with name "image", the system creates the default shape, which is a square. A possible solution for this problem is to replace "image of" with a blank in the pre-processing phase, since the user specified the shape itself as an "image of the shape", which is redundant.

Fill-colour

There are 21 cases of incorrect fill-colours. In 11 cases, the users did not specify the fill-colour of the shape in the sentence and the system thus chose the default fill-colour (white) for the shapes. Since the problem here is again an underspecification by the user and the created images match the description in the input sentence, the score for the fill-colour in these cases should be 1.

In 4 cases, the typed dependencies created for the sentences do not consider the colour as an adjective modifier for the object. As an example, the typed dependencies for the sentence "Put a solid dark green isosceles triangle in the middle of the image"

are shown below. As can be seen, "darkgreen" is not an adjective modifier for the "triangle" but for "isosceles".

$det(triangle, a)$

$amod(triangle, solid)$

$num(isosceles, darkgreen)$

$amod(triangle, isosceles)$

$dobj(put, triangle)$

$det(middle, the)$

$prep\_in(put, middle)$

$det(image, the)$

$prep\_of(middle, image)$

This is a mistake initially caused by the parser, which generates a wrong parse structure and can be corrected by retraining the parser with similar sentences and appropriate parse trees and dependency structures.

In 3 sentences, the user used the verb "stroke" to describe the creation of a white shape with black border-colour, for example, "Stroke a one inch square with black near the lower right corner of the canvas". In such cases, the system creates a completely black square, since the only specified colour is "black" and the system did not find out that this was supposed to relate only to the border-colour and not the fill-colour of the shape. A solution to this problem is to create two subclasses of the create-action: one for drawing a shape just with its border (as in stroke), and a second one for

drawing a filled shape (like painting).

In 3 cases, the names used for colours in the input sentences were not yet included in the synonym lists of the respective colours in the ontology. This is easy to change by adding more colour names.

Border-colour

In 41 cases, the border-colour of the shape in the created image does not match the border-colour in the original image.

In 18 sentences, either the user did not specify any colour for the shape, or the system could not recognize any colour for the shape due to the problems mentioned in the previous section. Consequently, the system chose the default border-colour (black) for the shape. In this case, the sentences should be a match with the final image, since there is no specification for the border-colour and thus the border-colour might as well be black. The score for these images should thus be 1.

The original image and the final image created for the sentence "Draw a 2 by 2 cm square in the center of the canvas. Fill the square with red colour" are shown in Figure 6.2.

As can be seen, the fill-colour and border-colour for the square in the original image are both red while the border-colour in the final image is black. As mentioned, the sentences containing multiple instructions are separated into sentences with only one instruction for processing. In the sentence example above, the system first processes the first instruction "Draw a 2 by 2 cm square in the center of the canvas.". The outcome is a square with the default fill-colour white and border-colour black. In

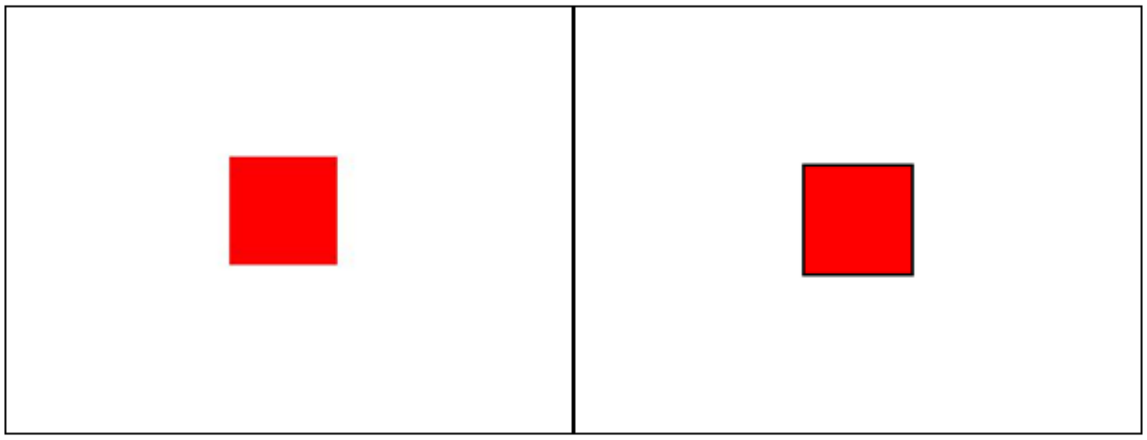

Figure 6.2: Square with different border-colour

the next step, the system processes the second instruction "Fill the square with red colour". According to the system, if the user does not specify red also as border-colour, the red colour will be assigned to the fill-colour of the shape and the border will stay black. In 23 cases, the border-colour is not correct due to the above mentioned reason. The solution for this problem is to change the description of the colour-change action in the ontology so that the new colour specified by the user is assigned to both the border- and the fill-colour of the shape.

Size

The number of images with unmatched sizes is artificially high, which is due to a problem with the experiment design. In the original images used during sentence collection, mostly large sizes were used, with the intention that the user can recognize the shapes easier. However, since all the shapes in this part of the sentence collection were large in size, the users did not specify the size in the natural language instructions. Thus, the system created in these cases typically shapes with the default size

medium. This is the reason that 151 images scored 0 in size. However, when comparing the sentences and the created images, the scores for the size should be 1, since there was no size specified in the sentences and thus size medium of the shape in the created images should be appropriate and match.

I found also that the description of the qualitative sizes in the ontology have to be modified. In 35 sentences, users describe the shape-sizes as "small" or "medium", while the size was "large" according to the ontology. However, the sentences in this category should get score 1, because comparing the sentences with the final images leads to a correct result.

There are 7 cases in which the user specified the size of the shape with quantitative values and this user-specified size is wrong. This might be due to different settings of the users' browser or screen display, or it could be a mistake in the size estimation by the user.

Position

In 17 cases, the position of the shape in the created image and the original image did not a match. In 6 of those 17 cases, the typed dependencies provided for the sentences by the parser are not correct regarding the dependency of the position description. For example, the typed dependencies for the sentence "Stroke a one inch square with black near the lower right corner of the canvas." shown below contain a wrong binding for the core position-element "corner". In this example, prep_near(black-8,corner-13) indicates the position. However, the dependency is wrong since it indicates that "black" is near the "corner" while the correct depen-

dency would be prep_near(square-6, corner-13) corresponding to the interpretation that the "square" is near the "corner".

$det(square, a)$

$num(inch, 1)$

$amod(square, inch)$

$dobj(stroke, square)$

$prep\_with(stroke, black)$

$det(corner, the)$

$amod(corner, lower)$

$amod(corner, right)$

$prep\_near(black, corner)$

$det(canvas, the)$

$prep\_of(corner, canvas)$

This problem can be solved by retraining the parser with similar sentences and adjusted parse trees and dependency structures. In 3 of the 17 cases, the sentence is too complex to extract the position information. For example, in the sentence "Stroke a one inch square with black centered horizontally in the canvas and near the top of it", the position information comes in two parts: first "centred horizontally in the canvas" and then "near the top of it". Processing this group of sentences is difficult due to their complex structure and would require some more work regarding the parser and the ontology mapping.

In 1 sentence, the user did not specify the location of the shape and the system selects the default position for the shape. In this case, the original image and the created image do not match but the sentence and the created image are matching and thus it should be considered as correct.

In 6 sentences related to the move-action, the description of the new position provided by the user is not a match with the original image. Either the distance of the movement is not estimated correctly by the user or the location is not specified correctly.

Orientation

In 7 cases, the orientation of the shape in the original image and the created image are not a match. 4 sentences are related to create-actions and the users did not specify the orientation of the shapes in the sentence. However, the sentences and the created images match, and thus the score for the orientation in these cases should be 1.

Similarly, in 3 sentences related to the rotate-action the users did not specify the angle of rotation in the sentence. As a result, the orientation of the shape in created image does not match with the orientation in the original image but it fits the description in the sentence. Thus, the value for rotation should be 1 and not 0.

In some sentences, users referred to the x-axis or y-axis in order to show the orientation of a shape. The concepts of x-axis and y-axis were not foreseen in the ontology at that point in time but it is added to the ontology in the third phase of the ontology development.

Total Images	Shape	Fill-colour	Border-colour	Size	Position	Orientation
100% (285)	99% (284)	96% (275)	89% (255)	97% (278)	94% (269)	99% (282)

Figure 6.3: Results for Evaluation Method 2

All 6 features correct	5 features correct	4 features correct	3 features correct
81% (231)	15% (42)	3.7% (11)	0.3% (1)

Figure 6.4: Number of correct features for Method 2

Figure 6.3 shows the statistics after modifying the scores to 1 for those cases, in which the user missed to specify a feature-value for the original image, which led to an unjustified 0 although the image matches the sentence. This indicates an average feature correctness of 96%.

Figure 6.4 shows the number of images with a certain number of correct features. For example, 81% of the created images have all 6 features correct, based on a match with the input sentences.

6.2 Evaluation 2

In the second evaluation, I asked 10 human subjects (5 regular users and 5 expert users) to compare the sentences with the final images. Since there were 5 sentences per each of the 57 original images, each sentence was tested with one regular user and one expert. The task was to compare the sentence with the created image and choose one of the following categories: Satisfactory, Minor Mistake, Major Mistake,

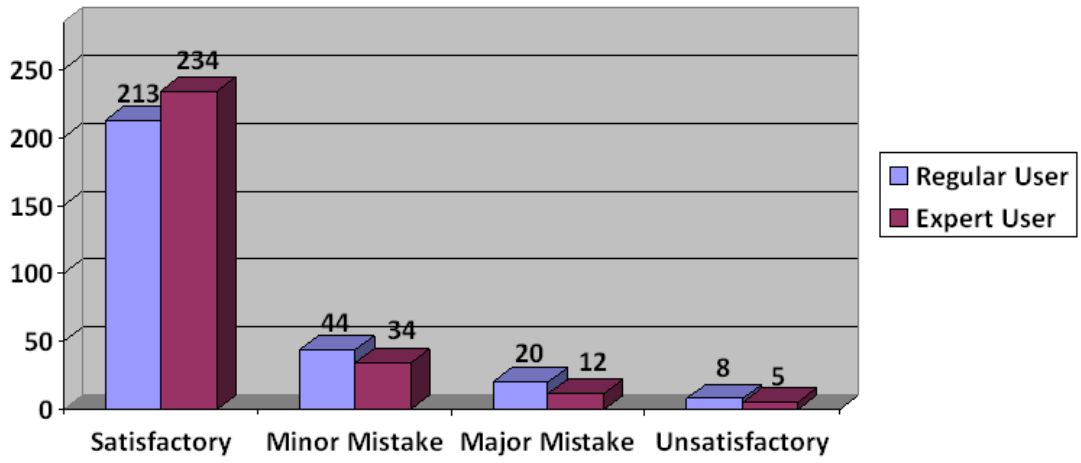

Figure 6.5: Results of Evaluation 2 with users (Chart)

Satisfactory		**Minor Mistake**		**Major Mistake**		**Unsatisfactory**	
Regular users	Expert users	Regular users	Expert users	Regular users	Expert users	Regular users	Expert users
75%	82%	15%	12%	7%	4%	3%	2%

Figure 6.6: Results of Evaluation 2 with users (Table)

Unsatisfactory. The results of the second evaluation are shown in Figures 6.5 and 6.6.

The regular users considered 8 cases as unsatisfactory. 4 of them are similar to the cases that the expert group considered as unsatisfactory. These cases are related to problems in creating the correct shape, position or orientation. In one case, an expert categorized the case as unsatisfactory, while a regular user categorized the same case as major mistake. The problem in this case is a failure of the system to recognize the correct position of the shape. The other 4 cases considered by the regular users as

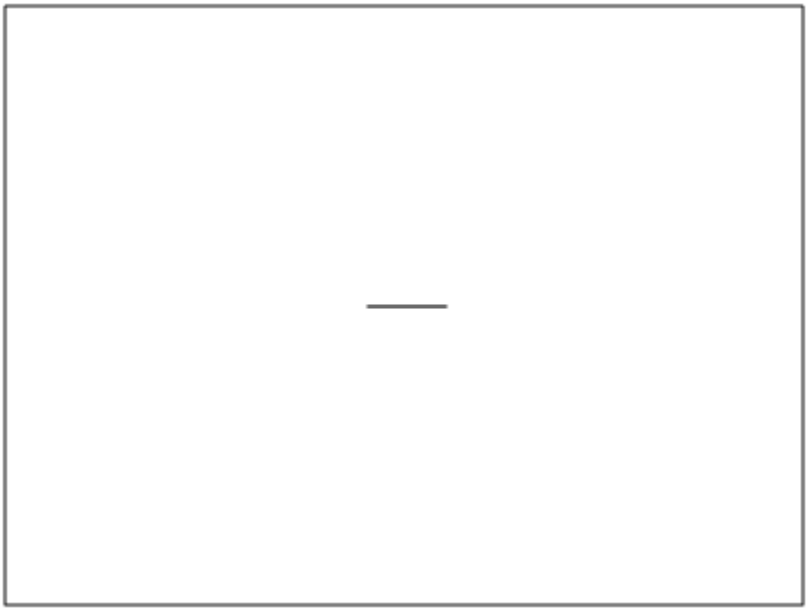

Figure 6.7: Sample image of horizontal line

unsatisfactory, were classified by the experts as satisfactory. The reason for this could be users' errors, since the result of the first evaluation also showed no incorrectness for these cases.

In the regular users group, 44 cases are considered as a minor mistake. 7 cases are similar to the experts group's evaluation. The reason for them is that the shape's border- and fill-colour are different. In 5 cases, the reason is recognizing a wrong position for the shape. The experts put the same cases in the major mistake category. The remaining 32 cases categorized by the regular users group as minor mistake were classified by the experts as satisfactory. The first evaluation did also not indicate any errors for them. The sentence "Place a horizontal black line in the middle of the page." is one of these 32 cases; the corresponding created image is shown in Figure 6.7.

The regular users group selected the major mistake category for 20 cases. The

Satisfactory or Minor Mistake		**Unsatisfactory or Major Mistake**	
Regular users	Expert users	Regular users	Expert users
90%	94%	10%	6%

Figure 6.8: Combined values for Evaluation 2

experts categorized 6 cases as minor mistake, which were classified by the regular users as major mistake. The problem was a wrong position or fill-colour. One case was categorized by both groups as major mistake because of a wrong fill-colour. The remaining 12 cases categorized by the regular users group as major mistake, were classified by the expert group as satisfactory. In 6 of them the border-colour was different than the fill-colour. In the other 6 cases, the experts' categorization was satisfactory, which was consistent with the first evaluation, which also indicated no errors.

In total, there are 49 cases categorized by one of the groups of users as satisfactory and by the other group as minor mistake. Also, there are 14 cases categorized by one group as satisfactory and by the other group as major mistake. Figure 6.8 shows the combined values for the categories "Satisfactory" and "Minor Mistake", and "Unsatisfactory" and "Major Mistake" of Evaluation 2.

The results from the expert group show that 82% of the created images were satisfactory. 94% of the images were classified as satisfactory or having a minor mistake. The result from the regular users is not significantly different but it shows a lower performance for the system. The reason for the lower performance ranked

by regular users could be their insufficient knowledge of geometrical shapes, which can lead to a wrong judgement. In addition, the users level of English could be a reason for a wrong judgement, since some sentences were complex regarding their syntactical structure, and the user has to understand the sentence in order to compare the sentence with the created image.

According to the expert users, the system created unsatisfactory images just for 2% of the sentences. This number is 3% for the regular users.

6.3 Summary of Evaluation Results

The system was evaluated using three methods. First, a feature-based method was used, in which the original images were compared to the created images. Since there was a flaw in the experiment for the sentence collection, and users subsequently underspecified the images, the performance particularly regarding the size feature was not satisfactory. In the second method, the sentences were compared with the created images. The correctness of features ranged from 89% to 99% per feature. 81% of the images have all 6 features correct; and only one image has only 3 features correct. The third evaluation was performed with human subjects in two groups of users, experts and regular users. The expert users found 94% of the created images either satisfactory (82%) or with a minor mistake (12%), while the regular users found 90% of the created images either satisfactory (75%) or with a minor mistake (15%).

Chapter 7

Conclusion

In this research work, I developed an ontology for the graphics domain and used it as part of a natural language interface. The ontology was developed based on different resources including documentation and textbooks on graphics tools, existing ontologies, and a corpus of natural language instructions to create or modify graphics images, all collected specifically for this thesis. The ontology was tested as part of a natural language interface in combination with a graphics generation system. This complex natural language interface accepts a natural language instruction as input and creates an appropriate image as output. The system was tested with a subset of the collected sentences, and the created images were assessed regarding their correctness or appropriateness using a feature-based evaluation method as well as judgements by human users. The overall correctness of the system was in the 80% range.

The sentence collection method used in this research is a novelty and very effective. In a short period of time, a huge number of sentences could be collected through the

Amazon Mechanical Turk with reasonable costs. I believe that this method is a great option, especially for researchers in natural language processing or other interfaces, who are looking for real-word data or evaluations through human users.

The same approach as in my thesis can be employed to develop an ontology and subsequently a natural language interface for a variety of different design systems, for example architectural design systems or electronics design systems. These design systems typically have a limited, pre-defined set of graphic objects and actions, e.g. add or remove specific ICs to/from a breadboard. In this example, the ICs would be the graphic objects, the breadboard corresponds to the drawing canvas, add and remove are typical actions, and modification involves replacing an IC.

There are not many well-developped methods to evaluate ontologies directly, or natural language interfaces. The prevalent method in similar research areas is to employ human subjects to evaluate a system. The approach I took to evaluate the ontology as part of a natural language interface integrates the use of human subjects to evaluate the system, with a structured approach for the evaluation, as well as an objective approach to measurement.

However, some caution has to be used when dealing with the evaluation results. There was a trade-off between the advantages and disadvantages of using expert users or regular users for the evaluation. Some inputs coming from expert users had a structure similar to a programming language instruction, and these inputs could not be used as sentences. Secondly, the experts have a better grasp and understanding of graphics concepts and terminology. This probably led to the case that regular users, unfamiliar with these terms, could not understand the sentences and thus provided a

misleading evaluation. This might be the reason for the fact that the evaluation by the experts indicated a better performance of the system than the judgement by the regular users.

Currently, the ontology covers the core elements of 2D-graphics. As future work, it is possible and would be interesting to expand the ontology for 3D-graphics by including concepts for 3D-shapes like sphere, cube, cone, and by adding 3D-actions like lighting. Users sometimes specify the size of a shape using a proportional value of another shape's size or a proportion of the canvas size. For example, in "Draw a red rectangle double the size of the blue rectangle." the size-value of the shape is specified with relation to another shape and its size. In future work, the concept of relative size could be added to the ontology. Modelling this concept would be similar to the concept of relative position in the ontology.

Appendix A

Concept Hierarchy of the Final Ontology

Figures A.1, A.2, and A.3 illustrate the complete concept hierarchy of the final ontology.

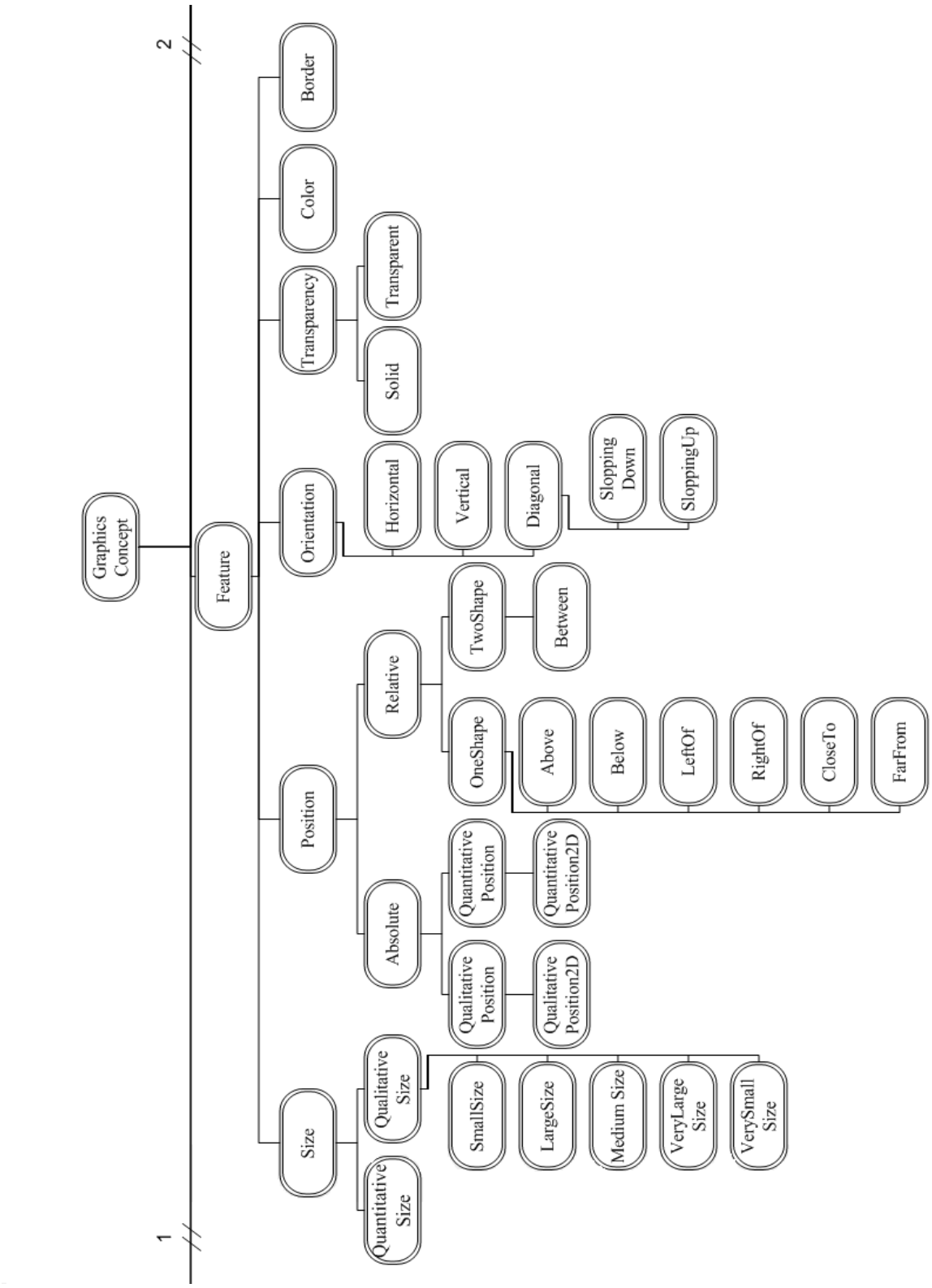

Figure A.1: Concept Hierarchy Centre Part

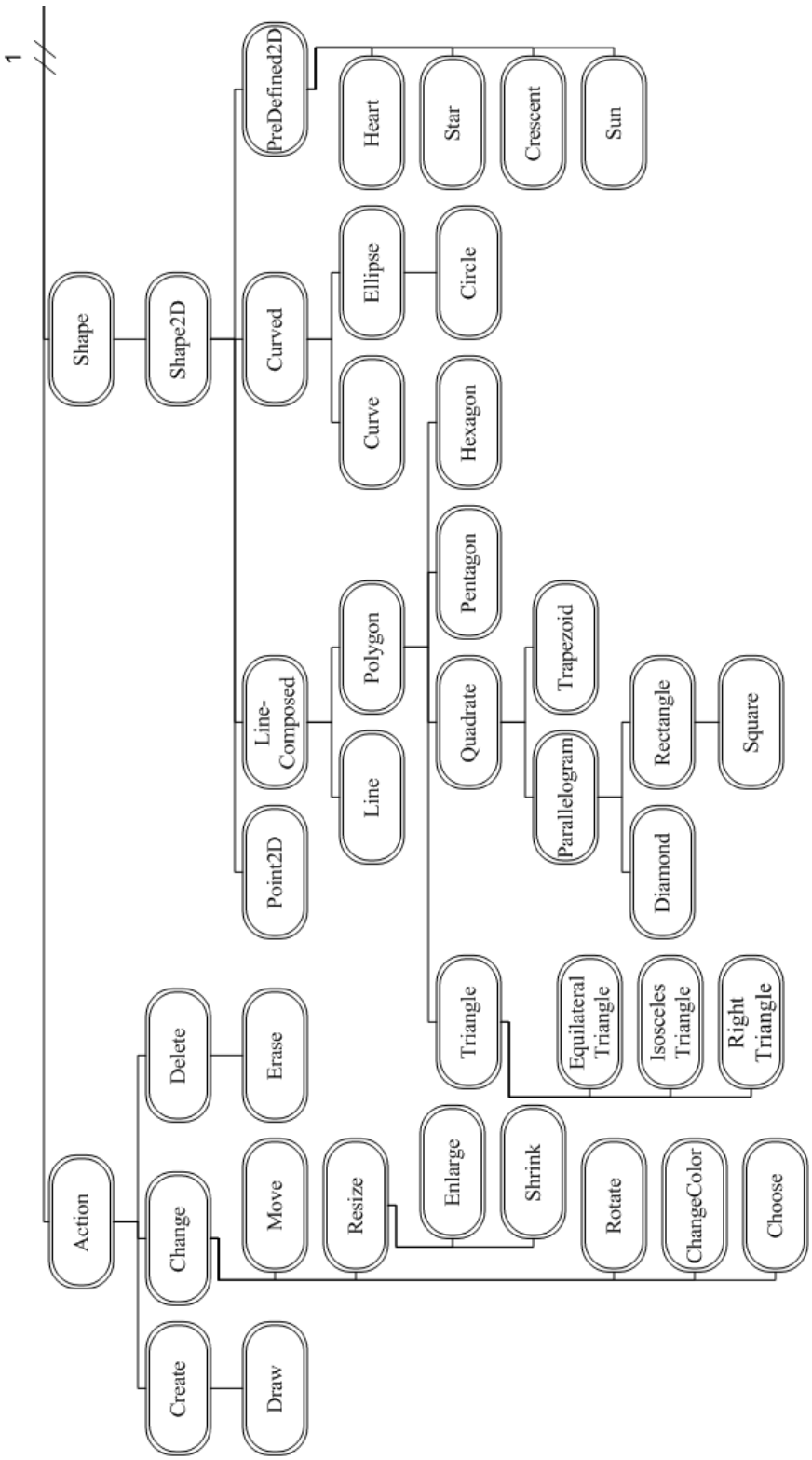

Figure A.2: Concept Hierarchy Left Part

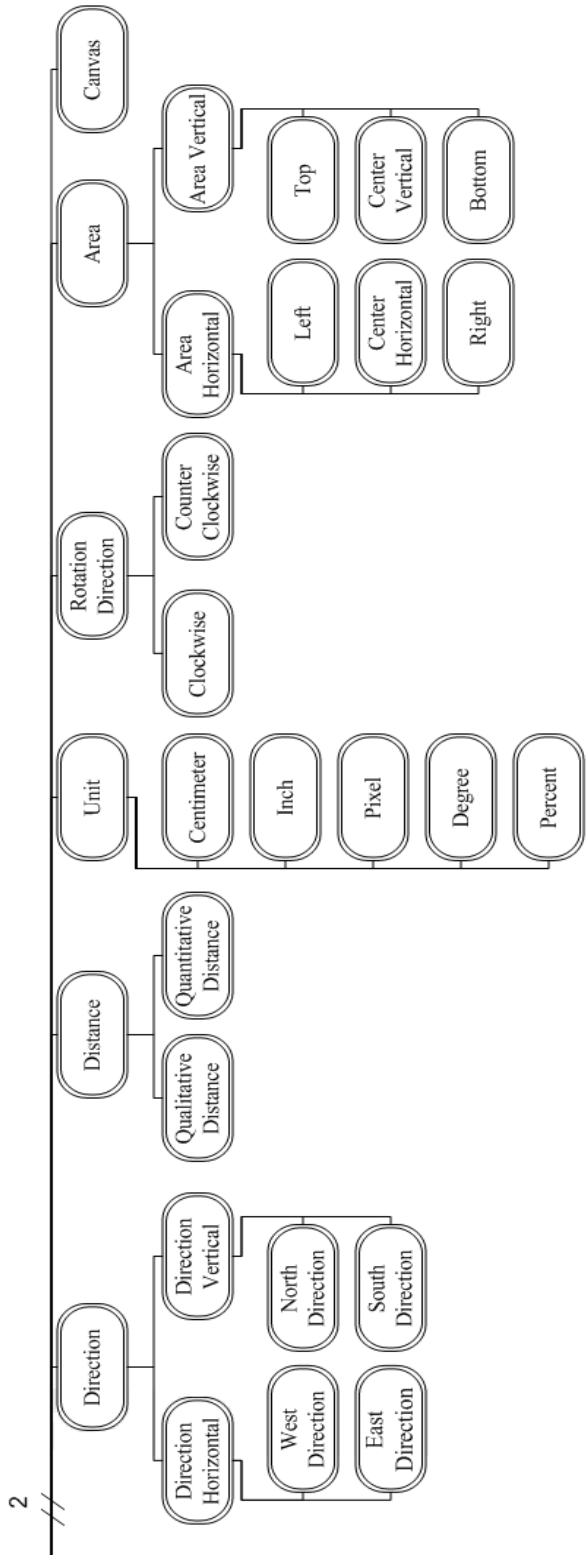

Figure A.3: Concept Hierarchy Right Part

Appendix B

Supporting Data

B.1 List of the Sentences Used for Evaluation

Figures B.1, B.1, B.2, B.3, B.4, B.5, B.6, B.7, and B.8 contain the list of the sentences used for the evaluation.

#	Sentence
1	Draw a diagonal line sloping down and to the right
2	draw a 4 cm diagonal line in the center of the canvas from left to right
3	draw a one inch diagonal line from the top left corner to bottom right corner at the center of the canvas
4	Draw a thin, black diagonal line in the center of the box with the higher end closest to the upper left-hand corner, sloping downward.
5	draw a slanted line in the center
6	Draw a thin, black horizontal line segment in the middle of the image.
7	place a horizontal black line in the middle of the page
8	Draw a 2-inch horizontal line in the middle of the field.
9	Draw a thin, black horizontal line in the center of the box. The line does not connect both sides of the box.
10	draw a 4 cm horizontal line in the center of the canvas
11	Draw a thin vertical line exactly in the middle of the field so that it takes up the middle third of the field.
12	Draw a short, vertical line in the center of the page.
13	Draw a vertical line of about half the height of the canvas, and center it both horizontally and vertically
14	draw a 4 cm vertical line in the center of the canvas
15	Draw a thin, black vertical line in the middle of the image.
16	Draw a curve in the top left corner of the canvas
17	draw a large arc in the left upper corner.
18	Put a curve in the top, left corner with the longer radius horizontal.
19	In the upper left-hand corner of the canvas, draw a thin, black arc that looks like a frown.
20	Draw an arc on the top left corner. The arc's shape has to be that of the top of the head of a human being.
21	Put a solid dark green isoceles triangle in the middle of the image.
22	Draw an equilateral triangle, point facing up, and color it with green.
23	draw a green equilateral triangle, each side 2 cm at the center of the canvas
24	draw and fill a green triangle
25	draw a triangle with three equal sides
26	Draw a right triangle, colored orange brown in the middle of the image.
27	draw a brown right triangle, each right side 2 cm at the center of the canvas
28	Create a right triangle with the legs parallel to the left and bottom edges of the screen
29	Draw a right angled triangle and fill it with brown color
30	Draw a brown right triangle in the center of the box. The right angle is on the lower left corner of the triangle.
31	Draw a blue rectangle in the center of the page, the longest sides are parallel to the x-axis
32	Draw a blue rectangle in the middle of the canvas
33	Put a solid blue rectangle with the long side horizontal in the center of the image.
34	Draw a box whose height is about half of its width, and color it blue.
35	Draw a rectangle, paint it blue
36	Draw a vertical gold diamond in the bottom left of the space.

Figure B.1: Test Sentences Page 1

#	Sentence
37	Draw a small yellow diamond at the lower left area of the canvas.
38	Draw a tall, yellow-orange diamond in the lower left.
39	draw and fill yellow diamond on the bottom left of the page.
40	Draw a yellow diamond in the lower left-hand corner of the box. The diamond is taller than it is wider.
41	Draw a shaded trapezoid in the center of the page.
42	In the center of the screen, draw a medium-sized, solid gray trapezoid with the wider side on the bottom.
43	an upright, gray trapazoid in the middle of the image.
44	Draw a gray trapezoid in the center of the box.
45	Draw a grey-filled trapezoid with its shorter side parallel to the x-axis
46	Draw a yellow-green pentagon resting in the middle.
47	Draw a small, green pentagon in the center of the box.
48	Put a solid green pentagon in the center of the image.
49	Draw a filled-in green regular pentagon resting on one side.
50	Draw a pentagon with it's bottom side parallel to the x-axis in the center of the canvas
51	Draw a yellow hexagon resting near the bottom, in the middle.
52	Create a yellow hexagon
53	In the center, bottom part of the page, draw a solid yellow hexagon.
54	draw and fill yellow hexagon in the bottom centre of page
55	Draw a hexagon centered horizontally, but close to the bottom of the canvas, such that two of its sides are parallel to the x-axis
56	Draw a navy oval in the center of the box. The oval should look like an egg resting on a table, rather than standing on one.
57	Draw an elipse in the center of the canvas with its longest side along the horizontal axis
58	draw and fill blue oval
59	Draw a blue oval in the middle of the page.
60	Draw an oval shape in the middle. For the orientation, it has to be flat at the top and bottom. Then colour it blue.
61	Please draw a heart in the middle of the page. Fill the heart with red.
62	Put a red heart in the center.
63	Draw a heart, and color it red.
64	Draw a red heart in the center of the box
65	Draw a heart in the center of the canvas, with the point oriented to the bottom
66	Put a yellow star in the center of the image.
67	Draw a yellow five-pointed star in the center of the canvas, such that one point only is pointing up.
68	Draw a yellow star in the center of the box. The star will have 5 points
69	star
70	Draw a 5-sided star in the middle, then color it with yellow.
71	Draw a moon such that the bulging side protrudes towards the left

Figure B.2: Test Sentences Page 2

#	Sentence
72	Draw a light-blue crescent moon, in the middle. Then squish it vetically slightly.
73	Please draw a cresent moon in the middle of the page. Fill the moon with blue.
74	Draw a baby blue crescent moon, so the opening is facing the right side of the box. It is a pretty thin crescent moon and should be in the center of the box.
75	Draw a blue cresent moon with the points facing right in the center
76	Draw a yellow sun in the center of the canvas
77	draw and fill the image of a yellow sun
78	Put a solid yellow sun icon with eight points in the center of the image.
79	Draw a sun shaped graphic in the middle of the page.
80	Create a yellow sun in the middle of the white box.
81	Draw a quadrat with size of 1 centimeter. Fill it with color light blue.
82	a light-blue solid square about half-inch each side
83	Create a square. Each side should be about 2.5 centimeters. Choose the color skyblue.
84	Create a small light blue square in the center of the space.
85	Draw a sky blue square at the center of the canvas.
86	Fill a one inch square in the middle of the canvas with dark blue.
87	a solid dark-blue square about half inch each side, at the center.
88	Draw a quadrat with size of 1 centimeter. Fill it with color dark blue.
89	Draw a solid dark blue square in the center about 1 inch wide
90	Little dark blue box in the middle.
91	Create a small purple square in the center of the space.
92	a medium, pink square in the center of the window
93	Draw a square with four sides. Fill in with hot pink or fuscia.
94	a purplish solid square about half-inch each side
95	a medium purple square in the middle
96	Half inch grey square in the middle.
97	Draw a solid light grey square about an inch wide, centered horizontally and vertically
98	a grey square in the center of the window
99	Draw a 2 by 2 cm square in the center of the canvas. Fill the square with grey colour.
100	draw a two centimeter grey square in the middle of a white page
101	create a one inch cyan square that is in the middle of the display
102	medium sized turquoise square in center of box
103	draw an aqua color square in the center
104	Draw a solid cyan square in the center about 1 inch wide
105	Produce a light blue square, measuring one inch by one inch in the middle of the canvas, with its sides parallel to the corresponding sides of the canvas.
106	In the center of the canvas draw a small square. Fill the square with the color yellow.

Figure B.3: Test Sentences Page 3

#	Sentence
107	a yellow square in the center of the window
108	a medium yellow square in the middle
109	Create a small yellow square in the center of the space.
110	Half inch yellow box in the center.
111	Draw a 2 by 2 cm square in the center of the canvas. Fill the square with red colour.
112	In the center of the canvas, create a red square about 1/5 the size of the canvase
113	Draw 30 pixels red quad in the middle
114	medium size red square in center of box
115	Create a small red square in the center of the space.
116	draw green color square in the center
117	a light green solid square about half-inch each side
118	a medium sized green square in the center
119	Small green box in the center.
120	In the center of the canvas draw a small square. Fill in the square with the color lime green.
121	In the center of the canvas draw a small square. Fill in the square with the color blue.
122	Half inch blue square in the middle.
123	Draw a square about 1 3rd of the length of the larger box with its center in the center of the larger box it should be royal blue.
124	Draw a 1 inch Blue Square in the center of a White Canvas. The square is not rotated.
125	Draw a solid blue square in the center about one inch wide
126	a black solid square about half-inch each side
127	small black square in middle
128	Draw a black 90 x 90 pixel square in the center of the canvas.
129	Create a small black square in the center of the space.
130	draw a solid black square in the center of the white canvas. The square should be much smaller than the canvas, but not small
131	a medium white square in the upper, right side of the window
132	small white square in top right corner
133	place a 1 inch transparent square in the top right coner of the canvas
134	Draw a 1 inch / 1 inch square near upper right corner of the canvas. Use black lines. Don't fill it with any color.
135	Make a half inch square in the top right corner.
136	a medium, white square in the upper, left side of the window
137	In the top left corner of the canvas draw a small square.
138	draw a black-lined square about half inch each side, located near the upper left hand side of the image
139	In the upper-left corner of the page, draw four lines to make a square.
140	Draw a 2 by 2 cm square at the top left corner of the canvas.
141	Draw a square at the top

Figure B.4: Test Sentences Page 4

#	Sentence
142	make a small square at the top
143	a medium white square in the top
144	Stroke a one inch square with black centered horizontally in the canvas and near the top of it.
145	a white square in the upper, center of the window
146	Create a small white square in the right center of the space.
147	small white square in far right, vertically in the middle
148	Draw a square halfway up the canvas on the right side.
149	Draw a 2 by 2 cm square at the middle right of the canvas.
150	a small white square in the middle, right of the window
151	2cm square on the left of a white page
152	a small white square on the middle, left side of the window
153	Stroke a one inch square with black centered vertically in the canvas and near the left side of it.
154	Draw a 100x100 pixel square on the left side of the canvas, 10 pixels from the edge.
155	Small transparent box on the left.
156	Draw a small transparent quad in the lower right corner of the screen.
157	a medium size square on the lower, right side of the window
158	Produce an open square in the lower right hand corner of your page.
159	Stroke a one inch square with black near the lower right corner of the canvas.
160	Draw an open square box about 1 inch wide, close to the bottom right corner
161	a medium, white square in the lower, left side of the window
162	Put a small clear box in the lower left hand corner.
163	Draw an open square box bout 1 inch wide, close to the bottom left corner
164	In the lower left hand corner of the canvas, draw a square
165	draw square in the bottom left position
166	On the bottom center, draw four lines to make a square.
167	a white square in the lower, center of the window
168	Draw a small white square in the middle of the canvas near the bottom border.
169	a black-lined square about half inch each side, located near the bottom of the image
170	create a one inch white square with a thin black line border at the lower center and 3% away from the edge of the display
171	Draw a medium sized white square at the center of the canvas.
172	draw a square in the middle of the canvas that's one-fourth the size of the canvas.
173	Create a large white square in the center of the space.
174	a black-lined square about an inch each side, located at the center.
175	Draw a 200x200 pixel square in the middle of the canvas.
176	Draw a 50 x 50 pixel square in the center of the canvas
177	Draw a small square in the centre of the page. It is a thin line, and not filled.

Figure B.5: Test Sentences Page 5

#	Sentence
178	Make a tiny square in the center.
179	a small white square in the center of the window
180	draw a black-lined square about quarter inch each side, located at the center.
181	change color blue to red
182	Change the blue square into a red square
183	Color the square with red
184	Change the color of the blue square to red.
185	convert blue square to red square
186	Change the outline of the yellow square from black to red.
187	Change the color of the box's outline from black to red.
188	Change the border of the square from black to red
189	Convert the border of the square to red
190	change border on yellow square to red
191	Remove the solid blue square from the center of the image.
192	Erase the blue square
193	Erase the square so the box is now blank
194	Erase it entirely.
195	remove the square
196	Draw a black circle to the left of the green diamond.
197	Place a black circle to the left of the green diamond.
198	Draw a solid black circle to the left of the green diamond.
199	Add a black circle with diameter that matches the diamond's height, to the left of the diamond
200	Draw a solid black circle to the left of the figure.
201	Draw a yellow triangle at the right of the circle
202	Draw a yellow triangle to the right of the blue circle. Horizontally align the triangle with the circle and space it exactly in between the circle and the edge of the page.
203	insert yellow triangle to the right of the blue circle
204	Draw a yellow triangle to the immediate right of the circle, halfway between the circle and the right side of the box.
205	Add a yellow triangle with the same size as the circle, to the right of the circle
206	Add a red triangle directly above the blue circle
207	Draw a red equilateral triangle above the circle and centered horizontally, such that one of its sides is parallel to the x-axis.
208	Add a red triangle above the circle, roughly the same size and pointing upwards
209	Draw a red equalateral triangle above the dark-blue circle.
210	draw triangle over circle
211	Place a blue square under the red circle
212	Draw a blue square, of the same size as the circle, directly below the circle. The two shapes will not touch.

Figure B.6: Test Sentences Page 6

#	Sentence
213	Add a blue square the same size as the red circle directly below the red circle.
214	Add a blue square directly below the red circle
215	Draw a square below the circle and color it with blue
216	Draw a green triangle between the blue square and the red circle.
217	Insert a green triangle of equal dimensions halfway between blue square and the red circle.
218	Draw a triangle in between the circle and square. Fill it with green color
219	draw green triangle between blue square and red circle
220	put a green triangle in between a blue square on the left and a red circle on the right.
221	Add a blue square to the top left of the canvas, such that there is some space between it and the existing image
222	insert a blue square with the same size in the upper left corner
223	fill and draw blue square to the top left corner of page
224	Draw a solid blue square near the top, left corner.
225	Draw a blue square in the upper left-hand corner of the box that is roughly the same size as the circle
226	draw and fill gray triangle to the top right corner of the page
227	Add a grey equilateral triangle with its bottom leg paralell to the x-axis, to the top right of the square, leaving some blank space between it and the existing image
228	Leave the green square in the centre intact. Then add a grey triangle to the top right.
229	Place a grey triangle in the top right whose size is similar to the green square.
230	fill and draw a gray triangle to the top right corner of page
231	Draw a yellow triangle to the lower right of the purple rectangle.
232	Draw a yellow triangle in the lower right-hand corner of the larger box.
233	Draw an upright yellow triangle in the lower-right corner.
234	Add a yellow up-pointing triangle in the bottom right corner of the canvas
235	Add a gold triangle in the bottom right corner of the image
236	add a green square in the lower left-hand corner of the box that is roughly the same size s the circle.
237	add a green 2cm by 2cm square in the bottom left corner of the canvas
238	Add a green square, whose side is equal to the diameter of the circle, to the bottom left corner of the canvas.
239	Leave dark red circle in the centre intact, and add a green square to the bottom left.
240	Draw a green square, whose sides are equal to the red circle's diameter, in the lower left.
241	Make the square smaller that becomes half of its original size.
242	Shrink the square in the center by half its dimensions.
243	reduce blue square by 50%
244	Shrink the blue square until it is about half the size of the original
245	Make the blue square smaller by half.
246	increase the size of the circle to double the size
247	Enlarge the circle to twice its diameter.
248	Enlarge the size of the red circle in the centre, until it becomes double its original size.

Figure B.7: Test Sentences Page 7

#	Sentence
249	Enlarge the red circle to about twice the size
250	Enlarge the circle by about 200%
251	Rotate clockwise 45 degrees.
252	Rotate the rectangle a quarter turn clockwise.
253	Rotate the blue rectangle 45 degrees clockwise.
254	tilt the blue rectangle to the right
255	Please rotate the rectangle at a 45 degree angle.
256	Rotate the green triangle ninety degrees clockwise.
257	Rotate the green triangle so that it points to the right.
258	Tilt the triangle so that the base side points at 90 degrees
259	Rotate green triangle
260	Rotate the green triangle ninety degrees clockwise.
261	Drag the square towards the right side for about 50 pixels
262	Move the blue square halfway to the middle of the image.
263	Move the blue square from the left side one-third of the screen width to the right.
264	Send the square horizontally to the center of the box
265	Move the square to the right along the x-axis, so that the square's right edge comes to the center of the canvas
266	Move the blue square right twice it's own width.
267	Move the blue square to the middle right of the the image
268	move the square to the right by 3cm
269	Move the square to the right of the canvas
270	Move the square along the x-axis to the right, so that the square's left edge is at the center of the canvas
271	Move the square to the right side.
272	Move the blue square from left side to right side.
273	move the square to the right by 5cm
274	Move the square maintaining its horizontal position to the right edge of the canvas
275	Move the square along a horizontal line to the side of the box.
276	Move the red circle next to the blue square.
277	Move the red dircle close to the blue square
278	Move circle next to square
279	move the circle 4cm towards the square
280	move the circle 3cm to the right
281	Move the red circle to the left a distance of 5cm.
282	move the circle far from the square
283	Move the red circle to the left center of the screen.
284	Move the red circle to the left until it almost touches the edge of the canvas, keeping it at the same horizontal level.
285	Move the red circle to the left side of the canvas.

Figure B.8: Test Sentences Page 8

B.2 Data for Evaluation 1 Method 1 - Comparing the Original Image and the Created Image

Figures B.9, B.9, B.10, B.11, B.12, B.13, B.14, and B.15 contain the result data for the Evaluation 1 Method 1 (Comparing the Original Image and the Created Image). "0" stands for an incorrect feature value and "1" stands for a correct feature value.

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
1	1	1	1	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	0	1	1
5	1	1	1	0	1	1
6	1	1	1	0	1	1
7	1	1	1	0	1	1
8	1	1	1	1	1	1
9	1	1	1	0	1	1
10	1	1	1	1	1	1
11	1	1	1	0	1	1
12	1	1	1	0	1	1
13	1	1	1	0	1	1
14	1	1	1	1	1	1
15	1	1	1	0	1	1
16	1	1	1	0	1	1
17	1	1	1	0	1	1
18	1	1	1	0	1	1
19	1	1	1	0	1	1
20	1	1	1	0	1	1
21	1	0	0	0	1	1
22	1	1	0	0	1	1
23	1	1	1	1	1	1
24	1	1	1	0	1	1
25	1	0	0	0	1	1
26	1	1	1	0	1	1
27	1	1	1	1	1	1
28	1	0	0	0	1	1
29	1	1	0	0	1	1
30	1	1	1	0	1	1
31	1	1	1	0	1	1
32	1	1	1	0	1	1
33	1	1	1	0	1	1
34	1	1	0	0	1	1
35	1	1	0	0	1	1
36	1	1	1	0	1	1
37	1	1	1	0	1	0
38	1	1	1	1	1	0
39	1	1	1	0	1	0
40	1	1	1	0	1	0
41	1	0	0	0	1	1
42	1	1	1	0	1	1
43	1	1	1	0	1	1
44	1	1	1	0	1	1
45	1	0	0	0	1	1

Figure B.9: Data for Evaluation 1 Method 1 Page 1

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
46	1	0	0	0	1	1
47	1	1	1	0	1	1
48	1	1	1	0	1	1
49	1	1	1	0	1	1
50	1	0	0	0	1	1
51	1	1	1	0	1	1
52	1	1	1	0	0	1
53	1	1	1	0	1	1
54	1	1	1	0	1	1
55	1	0	0	0	0	1
56	1	1	1	0	1	1
57	1	0	0	0	1	1
58	1	1	1	0	1	1
59	1	1	1	0	1	1
60	1	1	0	0	1	1
61	1	1	0	0	1	1
62	1	1	1	0	1	1
63	1	1	0	0	1	1
64	1	1	1	0	1	1
65	1	0	0	0	1	1
66	1	1	1	0	1	1
67	1	1	1	0	1	1
68	1	1	1	0	1	1
69	1	0	0	0	1	1
70	1	1	0	0	1	1
71	1	0	0	0	1	1
72	1	1	1	0	1	1
73	1	1	0	0	1	1
74	1	0	0	0	1	1
75	1	1	1	0	1	1
76	1	1	1	0	1	1
77	0	0	0	1	1	1
78	1	1	1	0	1	1
79	1	0	0	0	1	1
80	1	1	1	0	1	1
81	1	1	0	0	1	1
82	1	1	1	0	1	1
83	1	1	0	0	1	1
84	1	1	1	0	1	1
85	1	1	1	0	1	1
86	1	1	1	1	1	1
87	1	1	1	0	1	1
88	1	1	0	0	1	1
89	1	1	1	1	1	1
90	0	1	1	0	1	1

Figure B.10: Data for Evaluation 1 Method 1 Page 2

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
91	1	1	1	0	1	1
92	1	1	1	0	1	1
93	1	1	0	0	1	1
94	1	1	1	0	1	1
95	1	1	1	0	1	1
96	1	1	1	0	1	1
97	1	1	1	1	1	1
98	1	1	1	0	1	1
99	1	1	0	1	1	1
100	1	1	1	1	1	1
101	1	1	1	1	1	1
102	1	1	1	0	1	1
103	1	0	0	0	1	1
104	1	1	1	1	1	1
105	1	1	1	1	1	1
106	1	1	0	0	1	1
107	1	1	1	0	1	1
108	1	1	1	0	1	1
109	1	1	1	0	1	1
110	0	1	1	0	1	1
111	1	1	0	1	1	1
112	1	1	1	0	1	1
113	1	1	1	0	1	1
114	1	1	1	0	1	1
115	1	1	1	0	1	1
116	1	1	1	0	1	1
117	1	1	1	0	1	1
118	1	1	1	0	1	1
119	0	1	1	0	1	1
120	1	1	0	0	1	1
121	1	1	0	0	1	1
122	1	1	1	0	1	1
123	1	0	0	0	1	1
124	1	1	1	1	1	1
125	1	1	1	1	1	1
126	1	1	1	0	1	1
127	1	1	1	0	1	1
128	1	1	1	1	1	1
129	1	1	1	0	1	1
130	1	1	1	0	1	1
131	1	1	1	0	1	1
132	1	1	1	0	1	1
133	1	1	1	1	1	1
134	1	1	1	1	1	1
135	1	1	1	0	1	1

Figure B.11: Data for Evaluation 1 Method 1 Page 3

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
136	1	1	1	0	1	1
137	1	1	1	0	1	1
138	1	1	1	0	1	1
139	1	1	1	0	1	1
140	1	1	1	0	1	1
141	1	1	1	0	1	1
142	1	1	1	0	1	1
143	1	1	1	0	1	1
144	1	0	1	0	0	1
145	1	1	1	0	1	1
146	1	1	1	0	1	1
147	1	1	1	0	1	1
148	1	1	1	0	1	1
149	1	1	1	1	0	1
150	1	1	1	0	0	1
151	1	1	1	0	1	1
152	1	1	1	0	1	1
153	1	0	1	1	0	1
154	1	1	1	1	1	1
155	0	1	1	0	1	1
156	1	1	1	0	1	1
157	1	1	1	0	1	1
158	1	1	1	0	1	1
159	1	0	1	1	0	1
160	1	1	1	1	1	1
161	1	1	1	0	1	1
162	0	1	1	0	1	1
163	1	1	1	1	1	1
164	1	1	1	0	1	1
165	1	1	1	0	1	1
166	1	1	1	0	1	1
167	1	1	1	0	1	1
168	1	1	1	0	0	1
169	1	1	1	1	1	1
170	1	1	1	0	1	1
171	1	1	1	0	1	1
172	1	1	1	0	1	1
173	1	1	1	1	1	1
174	1	1	1	1	1	1
175	1	1	1	0	1	1
176	1	1	1	0	1	1
177	1	1	1	1	1	1
178	1	1	1	0	1	1
179	1	1	1	1	1	1
180	1	1	1	1	1	1

Figure B.12: Data for Evaluation 1 Method 1 Page 4

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
181	1	1	0	1	1	1
182	1	1	0	1	1	1
183	1	1	0	1	1	1
184	1	1	0	1	1	1
185	1	1	0	1	1	1
186	1	1	1	1	1	1
187	1	1	1	1	1	1
188	1	1	1	1	1	1
189	1	1	1	1	1	1
190	1	1	1	1	1	1
191	1	1	1	1	1	1
192	1	1	1	1	1	1
193	1	1	1	1	1	1
194	1	1	1	1	1	1
195	1	1	1	1	1	1
196	1	1	1	0	1	1
197	1	1	1	0	1	1
198	1	1	1	0	1	1
199	1	1	1	0	1	1
200	1	1	1	0	1	1
201	1	1	1	0	1	1
202	1	1	1	0	1	1
203	1	1	1	0	1	1
204	1	1	1	0	1	1
205	1	1	1	0	1	1
206	1	1	1	0	1	1
207	1	1	1	0	1	1
208	1	1	1	0	1	1
209	1	1	1	0	1	1
210	1	0	0	0	1	1
211	1	1	1	0	1	1
212	1	1	1	0	1	1
213	1	1	1	0	1	1
214	1	1	1	0	1	1
215	1	1	1	0	1	1
216	1	1	1	0	1	1
217	1	1	1	0	1	1
218	1	1	1	0	1	1
219	1	1	1	0	1	1
220	1	1	1	0	1	1
221	1	1	1	0	1	1
222	1	1	1	0	0	1
223	1	1	1	0	1	1
224	1	1	1	0	1	1
225	1	1	1	0	1	1

Figure B.13: Data for Evaluation 1 Method 1 Page 5

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
226	1	1	1	0	1	1
227	1	1	1	0	0	1
228	1	1	1	0	1	1
229	1	1	1	0	1	1
230	1	1	1	0	1	1
231	1	1	1	0	0	1
232	1	1	1	0	1	1
233	1	1	1	0	1	1
234	1	1	1	0	1	1
235	1	1	1	0	1	1
236	1	1	1	0	1	1
237	1	1	1	1	1	1
238	1	1	1	0	1	1
239	1	1	1	0	1	1
240	1	1	1	0	1	1
241	1	1	1	1	1	1
242	1	1	1	1	1	1
243	1	1	1	1	1	1
244	1	1	1	1	1	1
245	1	1	1	1	1	1
246	1	1	1	1	1	1
247	1	1	1	1	1	1
248	1	1	1	1	1	1
249	1	1	1	1	1	1
250	1	1	1	1	1	1
251	1	1	1	1	1	1
252	1	1	1	1	1	0
253	1	1	1	1	1	1
254	1	1	1	1	1	1
255	1	1	1	1	1	1
256	1	1	1	1	1	1
257	1	1	1	1	1	0
258	1	1	1	1	1	1
259	1	1	1	1	1	0
260	1	1	1	1	1	1
261	1	1	1	1	1	1
262	1	1	1	1	1	1
263	1	1	1	1	0	1
264	1	1	1	1	0	1
265	1	1	1	1	0	1
266	1	1	1	1	1	1
267	1	1	1	1	1	1
268	1	1	1	1	0	1
269	1	1	1	1	1	1
270	1	1	1	1	0	1

Figure B.14: Data for Evaluation 1 Method 1 Page 6

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
271	1	1	1	1	1	1
272	1	1	1	1	1	1
273	1	1	1	1	0	1
274	1	1	1	1	1	1
275	1	1	1	1	1	1
276	1	1	1	1	1	1
277	1	1	1	1	1	1
278	1	1	1	1	1	1
279	1	1	1	1	1	1
280	1	1	1	1	1	1
281	1	1	1	1	1	1
282	1	1	1	1	1	1
283	1	1	1	1	1	1
284	1	1	1	1	1	1
285	1	1	1	1	1	1
Total	**279**	**264**	**244**	**96**	**268**	**278**

Figure B.15: Data for Evaluation 1 Method 1 Page 7

B.3 Data for Evaluation 1 Method 2 - Comparing Sentence and Created Image

Figures B.16, B.16, B.17, B.18, B.19, B.20, and B.21 contain the result data for the Evaluation 1 Method 2 (Comparing Sentence and Created Image). "0" stands for an incorrect feature value and "1" stands for a correct feature value.

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	0	1	1
12	1	1	1	1	1	1
13	1	1	1	0	1	1
14	1	1	1	1	1	1
15	1	1	1	1	1	1
16	1	1	1	1	1	1
17	1	1	1	1	1	1
18	1	1	1	1	1	1
19	1	1	1	1	1	1
20	1	1	1	1	1	1
21	1	0	0	1	1	1
22	1	1	0	1	1	1
23	1	1	1	1	1	1
24	1	1	1	1	1	1
25	1	1	1	1	1	1
26	1	1	1	1	1	1
27	1	1	1	1	1	1
28	1	1	1	1	1	1
29	1	1	0	1	1	1
30	1	1	1	1	1	1
31	1	1	1	1	1	1
32	1	1	1	1	1	1
33	1	1	1	1	1	1
34	1	1	0	0	1	1
35	1	1	0	1	1	1
36	1	1	1	1	1	1
37	1	1	1	1	1	1
38	1	1	1	1	1	1
39	1	1	1	1	1	1
40	1	1	1	1	1	1
41	1	0	0	1	1	1
42	1	1	1	1	1	1
43	1	1	1	1	1	1
44	1	1	1	1	1	1
45	1	0	0	1	1	1
46	1	0	0	1	1	1
47	1	1	1	1	1	1
48	1	1	1	1	1	1

Figure B.16: Data for Evaluation 1 Method 2 Page 1

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
49	1	1	1	1	1	1
50	1	1	1	1	1	1
51	1	1	1	1	1	1
52	1	1	1	1	1	1
53	1	1	1	1	1	1
54	1	1	1	1	1	1
55	1	1	1	1	0	1
56	1	1	1	1	1	1
57	1	1	1	1	1	1
58	1	1	1	1	1	1
59	1	1	1	1	1	1
60	1	1	0	1	1	1
61	1	1	0	1	1	1
62	1	1	1	1	1	1
63	1	1	0	1	1	1
64	1	1	1	1	1	1
65	1	1	1	1	1	1
66	1	1	1	1	1	1
67	1	1	1	1	1	1
68	1	1	1	1	1	1
69	1	1	1	1	1	1
70	1	1	0	1	1	1
71	1	1	1	1	1	1
72	1	1	1	1	1	1
73	1	1	0	1	1	1
74	1	0	0	1	1	1
75	1	1	1	1	1	1
76	1	1	1	1	1	1
77	0	0	0	1	1	1
78	1	1	1	1	1	1
79	1	1	1	1	1	1
80	1	1	1	1	1	1
81	1	1	0	1	1	1
82	1	1	1	1	1	1
83	1	1	0	0	1	1
84	1	1	1	1	1	1
85	1	1	1	1	1	1
86	1	1	1	1	1	1
87	1	1	1	1	1	1
88	1	1	0	1	1	1
89	1	1	1	1	1	1
90	1	1	1	1	1	1
91	1	1	1	1	1	1
92	1	1	1	1	1	1
93	1	1	0	1	1	1
94	1	1	1	1	1	1
95	1	1	1	1	1	1
96	1	1	1	1	1	1

Figure B.17: Data for Evaluation 1 Method 2 Page 2

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
97	1	1	1	1	1	1
98	1	1	1	1	1	1
99	1	1	0	1	1	1
100	1	1	1	1	1	1
101	1	1	1	1	1	1
102	1	1	1	1	1	1
103	1	0	0	1	1	1
104	1	1	1	1	1	1
105	1	1	1	1	1	1
106	1	1	0	1	1	1
107	1	1	1	1	1	1
108	1	1	1	1	1	1
109	1	1	1	1	1	1
110	1	1	1	1	1	1
111	1	1	0	1	1	1
112	1	1	1	0	1	1
113	1	1	1	1	1	1
114	1	1	1	1	1	1
115	1	1	1	1	1	1
116	1	1	1	1	1	1
117	1	1	1	1	1	1
118	1	1	1	1	1	1
119	1	1	1	1	1	1
120	1	1	0	1	1	1
121	1	1	0	1	1	1
122	1	1	1	1	1	1
123	1	1	1	0	1	1
124	1	1	1	1	1	1
125	1	1	1	1	1	1
126	1	1	1	1	1	1
127	1	1	1	1	1	1
128	1	1	1	1	1	1
129	1	1	1	1	1	1
130	1	1	1	1	1	1
131	1	1	1	1	1	1
132	1	1	1	1	1	1
133	1	1	1	1	1	1
134	1	1	1	1	1	1
135	1	1	1	1	1	1
136	1	1	1	1	1	1
137	1	1	1	1	1	1
138	1	1	1	1	1	1
139	1	1	1	1	1	1
140	1	1	1	1	1	1
141	1	1	1	1	1	1
142	1	1	1	1	1	1
143	1	1	1	1	1	1
144	1	0	1	1	0	1

Figure B.18: Data for Evaluation 1 Method 2 Page 3

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
145	1	1	1	1	1	1
146	1	1	1	1	1	1
147	1	1	1	1	1	1
148	1	1	1	1	1	1
149	1	1	1	1	0	1
150	1	1	1	1	0	1
151	1	1	1	1	1	1
152	1	1	1	1	1	1
153	1	0	1	1	0	1
154	1	1	1	1	1	1
155	1	1	1	1	1	1
156	1	1	1	1	1	1
157	1	1	1	1	1	1
158	1	1	1	1	1	1
159	1	0	1	1	0	1
160	1	1	1	1	1	1
161	1	1	1	1	1	1
162	1	1	1	1	1	1
163	1	1	1	1	1	1
164	1	1	1	1	1	1
165	1	1	1	1	1	1
166	1	1	1	1	0	1
167	1	1	1	1	1	1
168	1	1	1	1	0	1
169	1	1	1	1	1	1
170	1	1	1	1	1	1
171	1	1	1	1	1	1
172	1	1	1	0	1	1
173	1	1	1	1	1	1
174	1	1	1	1	1	1
175	1	1	1	1	1	1
176	1	1	1	1	1	1
177	1	1	1	1	1	1
178	1	1	1	1	1	1
179	1	1	1	1	1	1
180	1	1	1	1	1	1
181	1	1	0	1	1	1
182	1	1	0	1	1	1
183	1	1	0	1	1	1
184	1	1	0	1	1	1
185	1	1	0	1	1	1
186	1	1	1	1	1	1
187	1	1	1	1	1	1
188	1	1	1	1	1	1
189	1	1	1	1	1	1
190	1	1	1	1	1	1
191	1	1	1	1	1	1
192	1	1	1	1	1	1

Figure B.19: Data for Evaluation 1 Method 2 Page 4

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
193	1	1	1	1	1	1
194	1	1	1	1	1	1
195	1	1	1	1	1	1
196	1	1	1	1	1	1
197	1	1	1	1	1	1
198	1	1	1	1	1	1
199	1	1	1	1	1	1
200	1	1	1	1	1	1
201	1	1	1	1	1	1
202	1	1	1	1	1	1
203	1	1	1	1	1	1
204	1	1	1	1	1	1
205	1	1	1	1	1	1
206	1	1	1	1	1	1
207	1	1	1	1	1	1
208	1	1	1	1	1	1
209	1	1	1	1	1	1
210	1	1	1	1	1	1
211	1	1	1	1	1	1
212	1	1	1	1	1	1
213	1	1	1	1	1	1
214	1	1	1	1	1	1
215	1	1	1	1	1	1
216	1	1	1	1	1	1
217	1	1	1	1	1	1
218	1	1	1	1	1	1
219	1	1	1	1	1	1
220	1	1	1	1	1	1
221	1	1	1	1	1	1
222	1	1	1	1	0	1
223	1	1	1	1	1	1
224	1	1	1	1	1	1
225	1	1	1	1	1	1
226	1	1	1	1	1	1
227	1	1	1	1	0	1
228	1	1	1	1	1	1
229	1	1	1	1	1	1
230	1	1	1	1	1	1
231	1	1	1	1	0	1
232	1	1	1	1	1	1
233	1	1	1	1	1	1
234	1	1	1	1	1	1
235	1	1	1	1	1	1
236	1	1	1	1	1	1
237	1	1	1	1	1	1
238	1	1	1	1	1	1
239	1	1	1	1	1	1
240	1	1	1	1	1	1

Figure B.20: Data for Evaluation 1 Method 2 Page 5

Sentence#	Shape	Fill color	Border color	Size	Position	Orientation
241	1	1	1	1	1	1
242	1	1	1	1	1	1
243	1	1	1	1	1	1
244	1	1	1	1	1	1
245	1	1	1	1	1	1
246	1	1	1	1	1	1
247	1	1	1	1	1	1
248	1	1	1	1	1	1
249	1	1	1	1	1	1
250	1	1	1	1	1	1
251	1	1	1	1	1	1
252	1	1	1	1	1	0
253	1	1	1	1	1	1
254	1	1	1	1	1	1
255	1	1	1	1	1	1
256	1	1	1	1	1	1
257	1	1	1	1	1	0
258	1	1	1	1	1	1
259	1	1	1	1	1	0
260	1	1	1	1	1	1
261	1	1	1	1	1	1
262	1	1	1	1	1	1
263	1	1	1	1	0	1
264	1	1	1	1	0	1
265	1	1	1	1	0	1
266	1	1	1	1	1	1
267	1	1	1	1	1	1
268	1	1	1	1	0	1
269	1	1	1	1	1	1
270	1	1	1	1	1	1
271	1	1	1	1	1	1
272	1	1	1	1	1	1
273	1	1	1	1	0	1
274	1	1	1	1	1	1
275	1	1	1	1	1	1
276	1	1	1	1	1	1
277	1	1	1	1	1	1
278	1	1	1	1	1	1
279	1	1	1	1	1	1
280	1	1	1	1	1	1
281	1	1	1	1	1	1
282	1	1	1	1	1	1
283	1	1	1	1	1	1
284	1	1	1	1	1	1
285	1	1	1	1	1	1
Total	**284**	**275**	**255**	**278**	**269**	**282**

Figure B.21: Data for Evaluation 1 Method 2 Page 6

B.4 Data for Evaluation 2 (Comparing Sentence and Created Image by Users)

Figures B.22, B.22, B.23, B.24, B.25, B.26, B.27, and B.28 contain the result data for the Evaluation 2 (Comparing Sentence and Created Image by Users). "X" in a cell shows the user selected the respective category for the task.

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
1	X				X			
2	X				X			
3	X				X			
4	X				X			
5	X				X			
6	X				X			
7		X			X			
8	X				X			
9	X				X			
10	X				X			
11	X						X	
12	X				X			
13			X			X		
14	X				X			
15	X				X			
16	X				X			
17		X			X			
18	X					X		
19		X			X			
20		X			X			
21	X				X			
22	X				X			
23	X				X			
24	X				X			
25	X				X			
26		X			X			
27	X				X			
28	X				X			
29	X				X			
30		X			X			
31	X				X			
32	X				X			
33	X				X			
34	X				X			
35	X				X			
36	X				X			
37	X				X			
38		X				X		
39		X			X			
40			X			X		
41	X						X	
42		X			X			
43	X				X			
44		X			X			
45			X			X		
46	X						X	

Figure B.22: Data for Evaluation 2 Page 1

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
47	X				X			
48	X				X			
49	X				X			
50	X				X			
51	X				X			
52	X				X			
53	X					X		
54	X				X			
55			X			X		
56		X			X			
57	X				X			
58	X				X			
59		X			X			
60		X			X			
61	X				X			
62	X				X			
63	X				X			
64	X				X			
65	X				X			
66	X				X			
67	X				X			
68	X				X			
69	X				X			
70		X				X		
71	X				X			
72	X				X			
73	X					X		
74	X					X		
75		X				X		
76	X				X			
77				X				X
78	X				X			
79	X				X			
80	X				X			
81	X				X			
82	X				X			
83			X		X			
84		X			X			
85			X		X			
86	X				X			
87	X				X			
88	X				X			
89	X				X			
90	X				X			
91	X				X			
92		X			X			

Figure B.23: Data for Evaluation 2 Page 2

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
93		X				X		
94	X				X			
95			X		X			
96				X	X			
97	X				X			
98	X				X			
99	X				X			
100	X				X			
101			X		X			
102	X				X			
103			X		X			
104	X				X			
105	X					X		
106	X				X			
107	X				X			
108		X					X	
109	X				X			
110		X			X			
111				X	X			
112		X					X	
113	X					X		
114	X				X			
115	X				X			
116	X				X			
117	X				X			
118	X					X		
119		X			X			
120	X					X		
121	X				X			
122	X				X			
123			X			X		
124	X				X			
125	X					X		
126	X				X			
127	X				X			
128	X					X		
129	X				X			
130		X			X			
131	X				X			
132	X				X			
133	X					X		
134		X			X			
135	X				X			
136	X				X			
137	X				X			
138	X				X			

Figure B.24: Data for Evaluation 2 Page 3

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
139	X				X			
140	X				X			
141	X				X			
142	X				X			
143	X					X		
144		X			X			
145	X				X			
146				X	X			
147	X				X			
148	X						X	
149	X				X			
150			X			X		
151	X						X	
152	X				X			
153			X		X			
154	X				X			
155			X		X			
156	X				X			
157	X				X			
158	X					X		
159			X				X	
160		X				X		
161	X				X			
162	X				X			
163	X					X		
164		X			X			
165	X				X			
166				X				X
167	X				X			
168		X					X	
169	X				X			
170		X				X		
171	X				X			
172		X					X	
173	X					X		
174	X				X			
175	X				X			
176	X				X			
177	X				X			
178	X				X			
179	X				X			
180	X				X			
181		X			X			
182		X			X			
183	X					X		
184		X			X			

Figure B.25: Data for Evaluation 2 Page 4

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
185		X				X		
186	X				X			
187	X				X			
188	X				X			
189	X				X			
190	X				X			
191	X				X			
192	X				X			
193	X				X			
194	X				X			
195	X				X			
196	X				X			
197	X				X			
198	X				X			
199	X				X			
200	X					X		
201	X				X			
202	X				X			
203	X				X			
204	X				X			
205		X			X			
206	X				X			
207	X				X			
208	X				X			
209	X				X			
210			X		X			
211	X				X			
212		X			X			
213		X			X			
214	X				X			
215	X				X			
216	X				X			
217	X				X			
218	X				X			
219	X				X			
220	X				X			
221	X				X			
222				X				X
223	X				X			
224	X				X			
225	X				X			
226	X				X			
227		X					X	
228	X					X		
229	X				X			
230	X				X			

Figure B.26: Data for Evaluation 2 Page 5

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
231			X					X
232	X				X			
233	X				X			
234	X				X			
235	X				X			
236	X				X			
237	X				X			
238	X				X			
239	X				X			
240	X				X			
241				X	X			
242	X				X			
243	X				X			
244	X				X			
245		X			X			
246	X				X			
247	X				X			
248	X					X		
249	X				X			
250			X		X			
251			X		X			
252				X				X
253	X					X		
254	X				X			
255	X				X			
256	X				X			
257		X			X			
258	X					X		
259	X				X			
260		X			X			
261	X				X			
262	X				X			
263	X				X			
264	X				X			
265			X		X			
266	X				X			
267		X			X			
268		X			X			
269	X				X			
270			X		X			
271	X				X			
272	X				X			
273	X				X			
274	X				X			
275	X				X			
276	X				X			

Figure B.27: Data for Evaluation 2 Page 6

Sentence#	Regular Users				Expert Users			
	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory	Satisfactory	Minor Mistake	Major Mistake	Unsatisfactory
277	X				X			
278	X						X	
279	X				X			
280	X				X			
281	X				X			
282	X				X			
283	X				X			
284		X			X			
285	X				X			
Total	**213**	**44**	**20**	**8**	**234**	**34**	**12**	**5**

Figure B.28: Data for Evaluation 2 Page 7

Bibliography

Amazon. Amazon Mechanical Turk. URL: `https://www.mturk.com/mturk/welcome`, Retrieved January 14, 2009.

J. Brank, M. Grobelnik, and D. Mladenic. A survey of ontology evaluation techniques. In *4th Conference on Data Mining and Data Warehouses (SiKDD'05) at the 8th Multi-Conference Information Society (IS'05)*, pages 166–169, Ljubljana, Slovenia, November 2005.

A. Cangelosi, K. R. Coventry, R. Rajapakse, A. Bacon, and S. Newstead. Grounding language into perception: A connectionist model of spatial terms and vague quantifiers. In *Modelling Language, Cognition and Action: Proceedings of the 9th Neural Computation and Psychology Workshop*, pages 47–56, May 2005.

J. B. Carroll, editor. *Language, Thought and Reality: Selected Writings of Benjamin Lee Whorf*. MIT Press, Cambridge, Massachusetts, 1956.

B. Chandrasekaran, J. R. Josephson, and V. R. Benjamin. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, January 1999.

S. R. Clay and J. Wilhelms. Put: Language-based interactive manipulation of objects. *IEEE Computer Graphics and Applications*, 16:31–39, March 1996.

K. R. Coventry and S. C. Garrod, editors. *Saying, Seeing and Acting. The Psychological Semantics of Spatial Prepositions*. Essays in Cognitive Psychology. Psychology Press, Hove and New York, 2002.

K. R. Coventry and P. Olivier, editors. *Spatial Language. Cognitive and Computational Perspectives*. Kluwer Academic Publishers, Netherlands, 2002.

K. R. Coventry, A. Cangelosi, D. W. Joyce, and L. Richards. Putting geometry and function together - towards a psychologically-plausible computational model for spatial language comprehension. In *Proceedings of the Twenty-fourth Annual Conference of the Cognitive Science Society*, page 33, Virginia, US, August 2002.

K. R. Coventry, A. Cangelosi, R. Rajapakse, A. Bacon, S. Newstead, D. Joyce, and L. V. Richards. Spatial prepositions and vague quantifiers: Implementing the functional geometric framework. In *Proceedings of Spatial Cognition Conference 2004*, pages 98–110, Germany, February 2005.

L. E. Crawford, T. Regier, and J. Huttenlocher. Linguistic and non-linguistic spatial categorization. *Cognition*, 75:209–235, June 2000.

Cycorp. The CYC ontology. URL: `http://www.cyc.com`, Retrieved January 6, 2008.

M.-C. de Marneffe and C. D. Manning. Stanford typed dependencies manual. URL: `http://nlp.stanford.edu/software/dependencies_manual.pdf`, Retrieved May 8, 2009.

D. Fensel. *Ontology: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, Berlin, 2004.

A. U. Frank. Qualitative spatial reasoning: cardinal directions as an example. *International Journal of Geographical Information Science*, 10:269–290, 1996.

C. Freksa and R. Röhrig. Dimensions of qualitative spatial reasoning. In *Qualitative Reasoning in Decision Technologies, Proc. QUARDET '93*, pages 483–492, Barcelona, Spain, June 1993.

C. Freksa, C. Habel, and K. Wender, editors. *Spatial Cognition - An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, volume 1404 of *Lecture Notes in Computer Science*. Springer, Berlin, 1998.

C. Freksa, C. Habel, and K. Wender, editors. *Spatial Cognition II - An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, volume 1849 of *Lecture Notes in Computer Science*. Springer, Berlin, 2000.

C. Freksa, C. Habel, and K. Wender, editors. *Spatial Cognition III - An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, volume 2685 of *Lecture Notes in Computer Science*. Springer, Berlin, 2003.

T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, November–December 1995.

M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *The 1995 International Joint Conference on AI (IJCAI'95), Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 6.1–6.10, Montreal, Canada, August 1995.

N. Guarino. Understanding, building, and using ontologies. *International Journal of Human and Computer Studies*, 46(2-3):907–928, February–March 1997.

R. Gupta and K. Hennacy. Finite state grammar transduction from distributed collected knowledge. In *Computational Linguistics and Intelligent Text Processing 7th International Conference Proceedings*, pages 343–354, Mexico City, Mexico, 19–25 February 2006.

R. Gupta and M. Kochenderfer. Common sense data acquisition for indoor mobile robots. In *Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 605–610, San Jose, CA, USA, 25–29 July 2004.

A. Herskovits. *Language and spatial cognition: An interdisciplinary study of the prepositions in English*. Cambridge University Press, Cambridge, UK, 1986.

A. Herskovits. Language, spatial cognition, and vision. In *Spatial and Temporal Reasoning*, chapter 6, pages 155–200. Springer, Netherlands, 1997.

D. Joyce, L. Richards, A. Cangelosi, and K. R. Coventry. On the foundations of perceptual symbol systems: Specifying embodied representations via connectionism. In *Proceedings of the Fifth International Conference on Cognitive Modeling*, pages 147–152, Bamberg, Germany, April 2003.

C. Kemke. Natural language communication between human and artificial agents. In Z.-Z. Shi and R. Sadananda, editors, *Agent Computing and Multi-Agent Systems, 9th Pacific Rim International Workshop on Multi-Agents (PRIMA 2006), Guilin, China, August 7-8, 2006*, volume 4088 of *Lecture Notes in Computer Science*, pages 84–93. Springer, August 2006a.

C. Kemke. An architectural framework for natural language interfaces to agent systems. In *Special Session "Natural Language Processing for Real Life Applications", IASTED International Conference on Computational Intelligence CI-2006*, pages 371–376, San Francisco, CA, November 2006b.

C. Kemke. An action representation formalism for natural language interfaces to agent systems. *International Journal of Convergence Information Technology*, 2 (2):30–36, 2007.

C. Kemke. An action ontology framework for natural language interfaces to agent systems. *Artificial Intelligence Review (AIRE)*, to be published, 2009.

D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, July 2003.

D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

B. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, May 2000.

D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1990.

S. C. Levinson and D. P. Wilkins. *Grammars of Space: Explorations in Cognitive Diversity*. Language, Culture, and Cognition. Cambridge University Press, Cambridge, 2006.

M. F. López, A. Gómez-Pérez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(1):37–46, January–February 1999.

A. Mukerjee. Neat versus scruffy: a review of computational models for spatial expressions. In P. Olivier and K. Gapp, editors, *Representation and processing of spatial expressions*, pages 1–35. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1998.

I. Niles and A. Pease. Towards a standard upper ontology. In *International Conference on Formal Ontology in Information Systems*, pages 2–9, October 2001.

N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, 2001.

S. O. Nualláin, editor. *Spatial Cognition: Foundations and Applications: Selected*

Papers from Mind III. Annual Conference of the Cognitive Science Society of Ireland. Benjamins Publishing Company, Berlin, 2000.

Princeton University. Wordnet - a lexical database for the English language. URL: `http://wordnet.princeton.edu`, Retrieved February 20, 2008.

D. Pullar and M. Egenhofer. Towards formal definitions of topological relations among spatial objects. In *Third International Symposium on Spatial Data Handling*, pages 225–243, Sydney, Australia, August 1988.

T. Regier. *The Human Semantic Potential: Spatial Language and Constrained Connectionism.* MIT Press, Cambridge, Massachusetts, 1996.

T. Regier and L. Carlson. Grounding spatial language in perception : An empirical and computational investigation. *Journal of Experimental Psychology*, 130:273–298, June 2001.

T. Regier and M. Zheng. Attention to endpoints : A cross-linguistic constraint on spatial meaning. *Cognitive Science*, 31:705–719, August 2007.

S. Staab, R. Studer, H. P. Schnurr, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, January–February 2001.

Stanford University. Protégé. URL: `http://protege.stanford.edu/`, Retrieved August 16, 2008a.

Stanford University. Protégé developer documentation. URL: `http://protege.stanford.edu/doc/dev.html`, Retrieved May 11, 2009.

Stanford University. The protégé ontology library. URL: `http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library`, Retrieved August 12, 2008b.

V. Sugumaran and V. C. Storey. Ontologies for conceptual modelling: Their creation, use, and management. *Data & Knowledge Engineering*, 42(3):251–271, September 2002.

Sun Microsystems Inc. Graphics documentation. URL: `http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics.html`, Retrieved May 9, 2009.

L. Talmy. How language structures space. In H. Pick and L. Acredolo, editors, *Spatial Orientation: Theory, Research, and Application*, pages 283–312. Plenum Press, NewYork, 1983.

The Dublin Core Metadata Initiative Organization. Dublin Core Metadata Initiative. URL: `http://www.dublincore.org`, Retrieved May 14, 2009.

The Standard Upper Ontology Working Group. The Suggested Upper Merged Ontology. URL: `http://www.ontologyportal.org`, Retrieved August 14, 2008.

The Stanford Natural Language Processing Group. The stanford parser: A statistical parser. URL: `http://nlp.stanford.edu/software/lex-parser.shtml`, Retrieved May 18, 2009.

University of Bremen. Spatial cognition priority program. URL: `http://www.cosy.informatik.uni-bremen.de/spp/`, Retrieved May 5, 2009.

University of Pennsylvania. The penn treebank project. URL: `http://www.cis.upenn.edu/~treebank/`, Retrieved May 10, 2009.

University of Toronto. The TOVE ontology project. URL: `http://www.eil.utoronto.ca/enterprise-modelling/tove`, Retrieved January 14, 2008.

M. Uschold. Building ontologies: Towards a unified methodology. Technical Report 197, Artificial Intelligence Applications Institute, University of Edinburgh, Scotland, 1996.

M. Uschold and M. King. Towards a methodology for building ontologies. Technical Report 183, Artificial Intelligence Applications Institute, University of Edinburgh, Scotland, 1995. Presented at: "Workshop on Basic Ontological Issues in Knowledge Sharing"; held in Conjunction with the 1995 International Joint Conference on AI (IJCAI'95), Montreal, Canada, August 1995.

W3C. The Web Ontology Language (OWL). URL: `http://www.w3.org/TR/owl-features/`, Retrieved August 18, 2008.

Wikipedia. List of geometric shapes. URL: `http://en.wikipedia.org/wiki/List_of_geometric_shapes`, Retrieved May 8, 2009.

Wolfram Research Inc. Wolfram mathematica. URL: `http://reference.wolfram.com/mathematica/guide/GraphicsObjects.html`, Retrieved May 8, 2009.