Modeling Shapes and Graphics Concepts in an Ontology

Mehdi Niknam[1] and Christel Kemke[1]

[1] Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada
{mhniknam, ckemke}@cs.umanitoba.ca

Abstract. This work presents the development of a graphics ontology for natural language interfaces. In a first phase, the ontology was developed in a standard way, based on documentations and textbooks for graphics systems as well as existing ontologies. In the second phase, we collected sets of natural language instructions to create and modify graphic images from human subjects. In these sentences, people describe actions to create or modify images; graphic objects and shapes; and features of shapes, like size, colour, and orientation. When analyzing these sentences, we found that some concepts associated with shape features needed to be added to or modified in the ontology. The ontology was then integrated with a natural language interface and a graphics generation module, yielding the Lang2Graph system. The Lang2Graph system accepts verbal instructions in the graphics domain as input and creates corresponding images as output. We tested the Lang2Graph system using a subset of the collected sentences as input. We determined the correctness of the created output images using two methods: an objective, feature-based measurement of the goodness of fit of the created image, and a corresponding objective evaluation by human users. The results of the tests showed that the system performed at an accuracy level of ~80% and over.

Keywords: Graphics Domain, Graphics Shapes, Ontology, Graphics Ontology, and Ontology Development.

1 Introduction and Background

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. This 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.

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 understanding 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 [1], 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. There are different approaches to develop an ontology [2, 3, 4]. One of the most important steps in ontology development is the identification of domain concepts and the relationships between them. The standard approach to identify concepts in a domain is to study existing resources and ontologies describing the domain. There are some ontologies that model part of the concepts in a graphics domain, e.g. human shapes [5] and digital shapes [6]. However, to the best of our knowledge, there is no complex, general ontology of the graphics domain.

In this paper, we present an approach for the development and evaluation of a graphics ontology. We designed the first draft of the graphics ontology using concepts extracted from graphics books and software as well as existing ontologies (see section 2). Since there is a close relationship between concepts in an ontology and natural language terms [7], we evaluated the first ontology based on the analysis of a set of natural language instructions collected from web users, who were asked to describe or modify presented images. It turned out that some concepts associated with shape features had to be modeled differently, and the ontology was modified accordingly in phase2 of the ontology development. Section 3 outlines the sentence collection process and section 4 describes the ontology refinements in phase 2.

One of the common methods to evaluate an ontology is to use the ontology in an application and assess the results [8]. Therefore, we designed and implemented a graphics drawing tool with a natural language interface called *Lang2Graph* (see section 5). We tested the ontology in combination with the Lang2Graph system using a subset of the collected sentences as input. We assessed the suitability of the created images through an objective feature-based measurement as well as through a subjective evaluation by human users (see section 6).

2 Ontology Development Phase 1

We provide a brief explanation of the terminology used in the description of the ontology. This terminology is closely related to the Protégé-Frame editor [9], which we used to implement the ontology and the ontology was saved as standard text files

format. The term “class” is equivalent to “concept” and they both represent a group of similar entities in a domain. The term “instance” represents an actual member of a class. The term “slot” is equivalent to “feature” and they both represent properties of classes. In Protégé, a slot can be defined using different types like symbol, integer, float, string and instance. Also, each slot can be defined to accept single value or multiple values. A “facet” represents the properties of each slot such as value types, default value, and allowed values. Finally, a “constraints” represents a specification for a class.

In the first step, we examined the available resources such as graphics books, websites, and tools in order to find concepts in the graphics domain. We collected a list of graphics and geometric shapes from [10] and [11]. To refine and complete the list of shapes, we also analyzed graphics tools like MS Paint, the drawing tool in MS Office, and Java's graphics packages [12]. We also obtained common features of the shapes such as colour, size, and position from these graphics tools. For the first version of the ontology, suitable concepts were selected and organized into a taxonomic hierarchy.

Figure 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 drawing area; and “Area” representing different areas on the canvas.

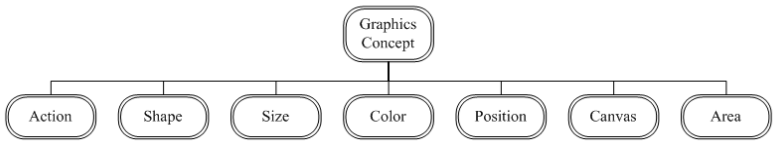

Figure 1: The top levels of concepts in the basic ontology

The Action-class includes three different subclasses: 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 (see more details in [13]).

Figure 2 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 relation between the shape and other shapes. Each shape has specific size slot, e.g., 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 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 using a constraint.

The LineComposed-class is a superclass for Line and Polygon. 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: EqualEdgeNumber, 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 values 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 EquilateralTriangle-class, since all its angles are equal as well. The Quadrate-class has Parallelogram and Trapezoid classes as its subclasses. Also, Pentagon and Hexagon are the other subclasses of Polygon. Similar facets and constraints mentioned in Triangle-class are defined to model other subclasses of Polygon-class. 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.

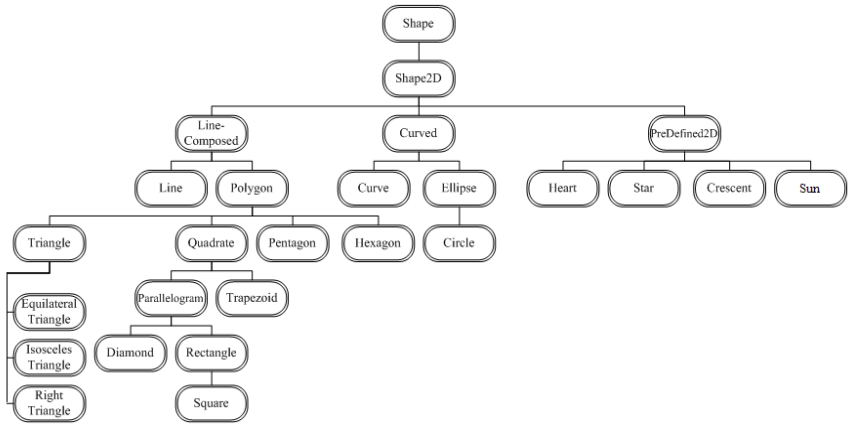

Figure 2: The hierarchy of the Shape-class and its subclasses

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". 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 FrameOfReference-slot for the Relative-class is an instance of the Shape-class only. The other subclasses of GraphicsConcept-class are described in details in [13].

3 Sentence Collection and Analysis

In order to refine the ontology and design the Lang2Graph system, we collect a corpus of verbal instructions in the graphics domain from potential users of a graphics system. There are different ways of collecting such a natural language corpus. Gupta and Hennacy in [14, 15] use a web-based system for collecting sentences from web users describing tasks in different application domains. The idea to use a web-based system to collect linguistic material seemed very attractive. An important tool we used for the sentence collection was the Amazon Mechanical Turk (AMT) [16], which allowed us to collect sentences through a webpage by posting tasks for users. AMT is a website that provides possibilities for businesses or researchers to access a wide variety of users of this website who can perform certain tasks for them.

We designed a set of sample images and asked people to explain in natural language how to create these images as tasks. 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 shapes, and also how they describe spatial relations between shapes. For this purpose, we 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. Figure 3 shows two sample images in the first set of tasks and Figure 4 shows sample images for the second set of tasks.

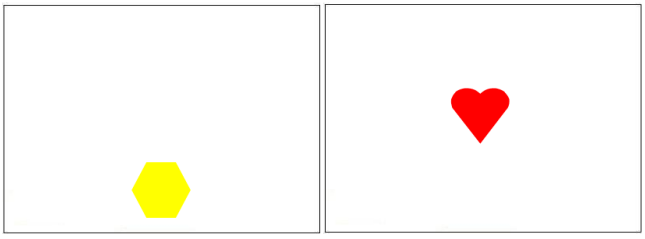

Figure 3: Sample Images of Shapes

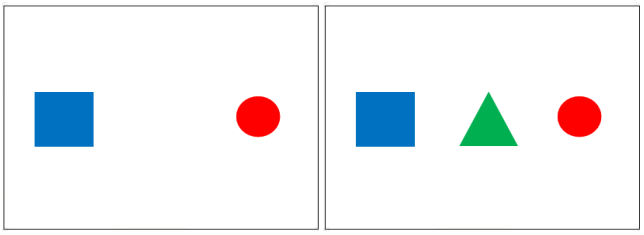

Figure 4: Sample Images for Relative Positioning

Finally, a total of 684 sentences (12 sentences per task) were collected. We used almost 60% (399 sentences) of the sentences to refine the ontology and to design Lang2Graph system. The remaining 40% (285 sentences) of the sentences were used for test and evaluation.

During the sentence analysis, we 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. Consequently, we trained the parser in order to achieve a correct syntactical analysis for such sentences.

4 Ontology Development Phase 2 (Ontology Refinement)

In this section, we describe the changes made to the first version of the ontology according to the results from the analysis of the collected sentences. We made changes by: 1) adding or modifying classes or features, 2) rearranging the hierarchy.

4.1 Adding or modifying classes or features

We updated the Action-class as follows. The Resize-class is divided into the classes Enlarge and Shrink, in order to deal with sentences like “Shrink the red square by 50%". The 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 the move-action. Therefore, we added the new slots MoveDirection and MoveDistance into the Move-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 e.g., “Rotate the square 45 degrees clockwise”. As a result, we added a slot called RotateDirection to the Rotate-class.

Next, we describe the modifications to the Shape-class. In order to represent the spatial relation of a shape with other shapes, we 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, we replaced the SpatialRelation-slot with a slot called ActualPosition. 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, we 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. We thus added a ShapeOrientation-slot to the Shape-class, which represents the original orientation of the shape. 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”, 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. Therefore, we added the concepts SlopingDown and SlopingUp into the ontology in order to deal with such sentences.

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”. A slot called ShapeBorder was 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". We 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.

4.2 Hierarchy Rearrangement

In order to obtain a more logical classification of the ontology and to ease the implementation of the natural language interface, we rearranged some of the classes in the hierarchy. We created a new class called Feature with no slots, and moved all the classes representing features of the Shape-class under this class. Subclasses of the Feature-class are: Color, Size, Position, Orientation, Transparency, and Border.

The development of the complete concept hierarchy and the final ontology are documented in detail in [13].

5 The Lang2Graph System

Figure 5 shows the overall architecture of the Lang2Graph system.

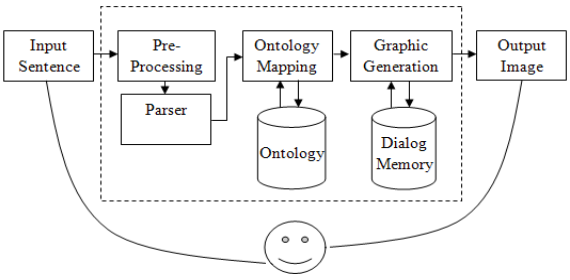

Figure 5: Architecture of the Lang2Graph System Diagram

First, the input sentence is pre-processed in order to be properly parsed and mapped to the ontology. For the syntactical analysis as part of the natural language interface, we chose the Stanford parser [17], which we trained using 60% of the collected sentences. The output from the parser is used to map the terms in the sentence onto the corresponding concepts in the ontology. Then, the graphics generation module provides the graphics scene based on the mapped ontology concepts.

We describe the operation of each component of the Lang2Graph system using a simple sentence "a blue square." In the pre-processing phase, a verb "Draw" has to be added to the input sentence which does 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. Now the sentence entering the parser is "Draw a blue square." 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. The typed dependencies provide an easy to understand description of the grammatical and semantic relationships in a sentence. The following is a collection of the typed dependencies produced for the input sentence:

det(square; a)
amod(square; blue)
dobj(draw; square)

The "det" dependency shows that the determiner "a" is a determiner for the noun "square". The "amod" shows that the adjective "blue" 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").These typed dependencies are used to create the ontological representation.

The first step in the ontology mapping phase is to find the appropriate action concept 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. In this example the Draw-class is the matching concept which represents the creation of a shape in the ontology. Consequently, an instance of Draw-class has to be generated. This instance gets an identifying name, for example "Draw1".

The instance "Draw1" 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. 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. We find the Square-class corresponding to the word "square" and an instance of Square-class called Square1 will be created. The instance Square1 will be assigned as filler to the ShapeTo-slot of the instance "Draw1".

The next step is to fill the slots of the "Shape1". 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. In our example, "blue" is the second element of the "amod". Then we search the ontology to find the concept corresponding to the "blue". We find "Blue" which is an instance of the Color-class in the ontology. Since there is no more information in the sentence, the rest of the slots of "Shape1" are filled with the default values (e.g., Size-slot with Medium-instance and Position-slot with Center-instance).

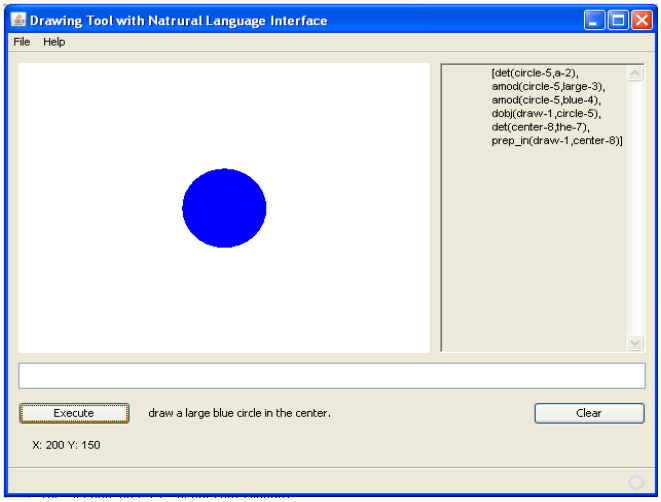

Figure 6: Screenshot of the Demo-version of the Lang2Graph system

The graphics generation component 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. Properties listed for the concepts in the ontology have matching properties in the respective graphics classes. 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. Based on our example, the graphics generation component generates a canvas with medium size blue square in its center.

Finally, the dialog memory stores different shapes in the graphics scene for later use. 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. A detailed description of the Lang2Graph system can be found in [13].

We implemented an online version of the Lang2Graph system for demo purposes (see Figure 6), and a batch processing system for testing as well.

6 Results

The Lang2Graph was used for the evaluation of 40% of the originally collected sentences. The images produced by the system were then compared against the original images or the input sentences, respectively. In the first evaluation, each created image was compared to the corresponding original image and checked for shape and feature matches. This led to some negative evaluation results, which were mostly caused by an under-specification of the image descriptions in the collected sentences. In other words, users often did not specify feature values, like the size "large" for a shape and thus the system created a shape with the default size "medium". We performed subsequently a second assessment, in which the given sentences and the created images were compared. Table 1 summarizes the results of both assessments.

In the second evaluation, the assessment was performed by human subjects. We asked 10 subjects (5 regular and 5 expert users) to compare the sentences and the matching created images, and judge each pair as Satisfactory, having a Minor Mistake, Major Mistake or being Unsatisfactory (see Figure 7).

Table 1. Results for the first (row1) and the second (row2) assessment of Evaluation1

Images	Shape	Fill	Border	Size	Position	Orientation
100% (285)	97% (279)	93% (264)	86% (244)	34% (96)	94% (268)	97% (278)
100% (285)	99% (284)	96% (275)	89% (255)	97% (278)	94% (269)	99% (282)

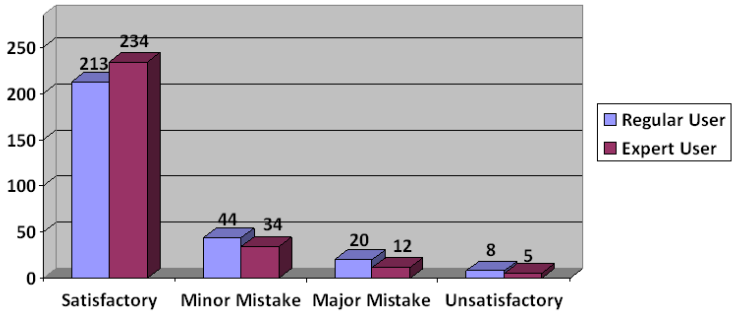

Figure 7: Results of Evaluation2 with users

The result for the regular users is not significantly different but it shows a lower classification rate of images as Satisfactory. The reason for this could be a lack of knowledge of geometrical shapes, which can lead to a wrong judgment.

7 Conclusions

In this paper, we presented the development of an ontology for the graphics domain. In a first phase, the ontology was developed in a standard way, based on documentations and textbooks for graphics systems as well as existing ontologies. In the second phase, we collected sets of natural language instructions to create and modify graphic images from human subjects. When analyzing these sentences, we found that some concepts associated with shape features needed to be added to or modified in the ontology. The ontology was then integrated with the Lang2Graph system. We tested the Lang2Graph system using a subset of the collected sentences as input. The results of our tests indicate an accuracy of the system in the area of 80%. Currently, the ontology covers the core elements of 2D-graphics. As future work, it 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. The same approach 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.

References

1. T. Gruber. Toward principles for the design of ontologies used for knowledge sharing.International Journal of Human-Computer Studies, 43(5-6):907-928, 1995.
2. V. Sugumaran and V. C. Storey. Ontologies for conceptual modelling: Their creation, use, and management. Data & Knowledge Engineering, 42(3):251-271, September 2002.

3. N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Tech. Rep. KSL-01-05, Stanford Knowledge Systems Laboratory, 2001.
4. S. Staab, R. Studer, H. P. Schnurr, and Y. Sure. Knowledge processes and ontologies. IEEE Intelligent Systems, 16(1):26-34, January-February 2001.
5. M. Gutierrez, D. Thalmann, F. Vexo, L. Moccozet, N. Magnenat-Thalmann, M. Spagnuolo. An Ontology of Virtual Humans: incorporating semantics into human shapes. In: Proceedings of Workshop towards Semantic Virtual Environments, pp.57-67, 2005.
6. B. Falcidieno, M. Spagnuolo, P. Alliez, E. Quak, C. Houstis, E. Vavalis. Towards the semantics of digital shapes: the AIM@SHAPE approach. In: European Workshop on the Integration of Knowledge, Semantic and Digital Media Technologies, IEE Royal Statistical Society, London, 2004.
7. J. B. Carroll, editor. Language, Thought and Reality: Selected Writings of Benjamin Lee Whorf. MIT Press, Cambridge, Massachusetts, 1956.
8. B. Grobelnik, and D. Mladenic. A survey of ontology evaluation techniques. In 4th Conference on Data Mining and Data Warehouses at the 8th Multi-Conference Information Society, pp. 166-169, Ljubljana, Slovenia, November 2005.
9. Stanford University. Protégé developer documentation. URL: http://protege.stanford.edu /doc/dev.html, Retrieved 2010.
10. Wikipedia. List of geometric shapes. URL: http://en.wikipedia.org/wiki/List_of _geometric _shapes , Retrieved 2009.
11. Wolfram Research Inc. Wolfram mathematica. URL: http://reference.wolfram.com/math ematica/guide/GraphicsObjects.html, Retrieved 2009.
12. Sun Microsystems Inc. Graphics documentation. URL: http://java.sun.com/j2se/1.4.2/docs / api/java/awt/Graphics.html, Retrieved 2010.
13. M. Niknam, Development of a Graphics Ontology for Natural Language Interfaces. MS thesis. The University of Manitoba, Winnipeg, 2010. Print.
14. 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.
15. 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.
16. Amazon. Amazon Mechanical Turk. URL: https://www.mturk.com/mturk/welcome, Retrieved 2009.
17. 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.
18. C. Kemke. An action ontology framework for natural language interfaces to agent systems. Artificial Intelligence Review (AIRE), 2009.
19. T. Regier and M. Zheng. Attention to endpoints: A cross-linguistic constraint on spatial meaning. Cognitive Science, 31:705-719, August 2007.