Sketching ER diagrams

Paul Schmieder, Beryl Plimmer, Gillian Dobbie

Department of Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand

{psch068@ec, beryl@cs, gill@cs}.auckland.ac.nz

Abstract

Hand-drawn diagrams are frequently used as the first visualization of a model. Converting these preliminary diagrams into a specific formal format is time consuming. Computer based sketch-tools can offer support during the informal sketching stage and automatic conversion to formal representations. Entity Relationship diagrams are particularly difficult to convert because of their characteristics such as cardinality notations. We extend the general diagram sketching tool InkKit with domain semantics to successfully recognize and automatically convert Entity Relationship diagrams. This approach takes advantage of sketching as the preferred initial design realization while minimizing the effort required to translate the initial design to a functional prototype.

Keywords: Sketch tools, system modelling, recognition algorithms

1 Introduction

During childhood expressing feelings, wishes and impressions by sketching or painting is a natural and familiar process. Keeping this familiar approach of illustrating ideas with a stylus rather than a keyboard or mouse in a computer environment enables a natural and informal way of interaction. Studies have been conducted to examine the benefits of computer-based sketch tools (Goel 1995; Plimmer and Apperley 2004). Although sketch tools are still not as intuitive as traditional sketching (pen and paper), input based on a digital stylus was recognized to be more genuine and friendly than other input devices (Bailey and Konstan 2003; Yeung 2007).

One function of sketch tools is to recognize and convert the sketches into symbolic expressions representing the user's intent. The demands on the recognition algorithms are high because different element types within one sketch might consist of similar looking shapes making them hard to distinguish. Additionally, the fact that sketches from different domains use different notations makes general approaches to successful recognition, interpretation and conversion a more challenging task.

One use of sketching is Entity Relationship diagrams (ER). Applied in database development processes ER's provide an easy solution to specify database drafts at a preliminary stage. Commonly used for designing formal databases including attributes and their relationships, ER's describe coherences between its components.

A software toolkit already recognizing and converting digital sketches is InkKit (Plimmer and Freeman 2007). It supports the development of additional diagram domains with its layered architecture. This makes InkKit an effective platform to extend. Furthermore, the distinction between the interpreter layer within a certain domain and its following output layer enables easy code integration to generate various database specific formats from the domain specific interpreter.

ER diagrams are an example of a range of diagrams for which sketch tool support has not been fully explored. For example, similar types of connectors are used in UML class, activity and object diagrams. Furthermore, due to the interconnectivity of the ER diagram's components, spatial and context error detection can be processed. A successful ER domain implementation, therefore, contributes to extensible sketch tools improving core recognition engines and techniques for domain specific error correction. To validate our implementation we applied student textbook exercises to InkKit's new ER domain from which the rate of precision, success and simplicity are measured.

Before ER diagrams are introduced in more detail in section three, related work is described in the next section. In section four, our approach is explained in detail and is followed by an evaluation of the new ER domain in section five. A discussion follows in section six and we finish with conclusions and future work.

2 Related Work

There is a variety of existing sketch tools which differ in some of their basic features such as recognition domains, recognition engine or the ability to process text. Many of these sketch tools are used to recognize diagrams related to Computer Science or Engineering such as User Interface diagrams (Landay and Myers 1996; Lin, Newman et al. 2000; Bailey, Konstan et al. 2001; Plimmer and Apperley 2003), UML class diagrams (Damm, Hansen et al. 2000; Hammond and Davis 2002) or circuit diagrams (Alvarado and Davis 2004).

The core component of every sketch tool is the recognition engine of which basically two different types exist: domain specific and generic. While specific recognition engines are designed for specific domains, generic recognition engines are capable of recognizing

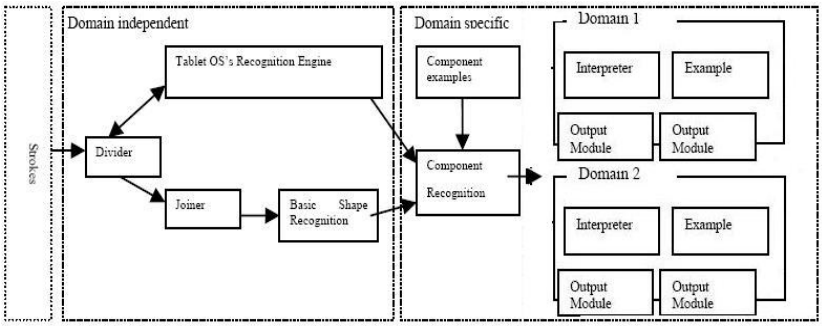

Figure 1. Architecture InkKit

different types of diagrams and support extensibility for domain specific rules.

Silk (Landay and Myers 1996) was one of the first interactive sketch tools, its recognition engine was specifically designed to recognize user interfaces. It was followed by Knight (Damm, Hansen et al. 2000) and Tahuti (Hammond and Davis 2002) which recognize UML class diagrams, DEMAIS (Bailey, Konstan et al. 2001), Freeform (Plimmer and Apperley 2003) and SketchiXML(Coyette, Vanderdonckt et al. 2007) enable UI design recognition.

Whereas these tools are all based on a recognition engine dedicated to particular domains, Lank's framework (Lank 2003), SketchREAD (Alvarado and Davis 2004) and InkKit (Plimmer and Freeman 2007) use a generic recognition engine which can be extended to processed sketches from different domains. However, the level of complexity to add a domain differs. While it is very complex and code intense to add a domain in Lank's framework it is not in InkKit or SketchREAD. A different general approach is proposed by Costagliola et al. (2004) which operates on grammar productions describing the spatial relations between diagram elements. An important feature of sketch tools is text recognition. While Lank's framework (Lank 2003) and InkKit (Plimmer and Freeman 2007) support text and drawing recognition SketchREAD (Alvarado and Davis 2004) does not support it all and Tahuti (Hammond and Davis 2002) only partially.

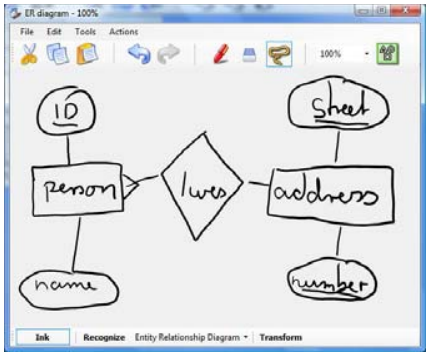

Figure 2. ER diagram

In this project we extend InkKit to recognize ER diagrams (Figure 2) as an exemplar of a class of diagrams with similar notations. The closest diagram type which has been addressed by other projects (Hammond and Davis 2002; Chen, Grundy et al. 2003) is the UML class diagram. However, ER diagrams are different and provide new challenges such as the complex relations between components.

Furthermore, in contrast to UML, ER models are not standardized. Originally proposed by Chen (Peter Pin-Shan 1976) many different notations have since been introduced: one frequently used notation is the SSADM (Weaver 1998).

Beside the ER domain InkKit already successfully recognizes other domains such as UI's, undirected and directed graphs, organization charts and UML class diagrams. The particular challenge with ER diagrams is the multiple types of connectors.

2.1 InkKit

Consisting of two major user interfaces, portfolio manager and sketch pages, InkKit supports a well tested robust and intuitive interaction (Plimmer, Tang et al. 2006). The user can ink on the sketch page, manipulate their sketch with resize, copy, cut and paste actions. At the same time all sketches are shown as thumbnails on the portfolio manager and can be linked with each other. To preserve the hand drawn appearance no beautification of the sketch is processed (Bailey and Konstan 2003; Yeung 2007).

InkKit's overall architecture is presented in Figure 1. Before the recognition is processed the sketched strokes (the group of single lines drawn on a computer) are divided into either text strokes or drawing strokes. Then the text strokes are grouped into words and recognized whereas the drawing strokes belonging to one shape are internally joined together, to be considered as one stroke by the shape recognizer (Plimmer and Freeman 2007).

The joining is necessary to apply Rubine's single stroke algorithm (Rubine 1991) to recognize shapes within a sketch. To increase the recognition accuracy basic shapes are recognized first. This is possible because complex shapes can be defined as a sum of basic shapes. For example Figure 3 shows a connector with an optional cardinality constraint which is a composition of the basic shapes line and circle.

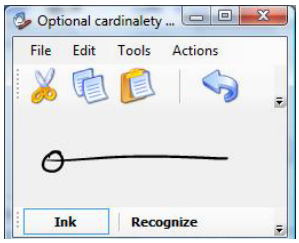

Figure 3. A connector consisting of the basic shapes line and circle

Instead of recognizing this complex shape as a whole, the circle and the line are recognized as single shapes and combined later in the recognition process. InkKit has an extendable set of basic shapes such as rectangles, circles, lines or triangles.

Up to this point the recognition process is domain independent: all sketched strokes have been classified as writing or drawing and further recognized as a word or basic shape. The next step is to group the single basic shapes together to the most likely components which are predefined complex shapes belonging to a specific domain.

The predefined shape sketches are stored in a domain library and can be edited or extended by the user. The matching is processed using three sets of features. First, based on spatial features between elements (enclosing, enclosed, near or intersecting, relative position and their orientation) the probability of relationships between components is computed. Next, based on the already calculated probabilities, the sketched element's shape and its spatial position, the likelihood for this element being part of a predefined component of the domain library is determined.

As a result of the first two steps a complete set of combinations of possible basic shapes is built, upon which probability tables are computed. The table for each component includes the combination's chance of being a certain predefined complex component. This step also considers features such as the shape's bounding box properties. After bringing all calculated features for possible combinations together an overall probability per combination is determined. The final assignment of a combination to its domain specific component is then processed starting with the combination which has the highest probability followed by all other still possible combinations ordered by descending probabilities.

To integrate a new type of diagram a domain specific interpreter and sketched examples of all components defining the domain are necessary. Within the interpreter the components have to be defined and a data model of these components has to be build which presents the components themselves and their relations to each other. In addition error correction routines can be part of this model. Upon the generated component model additional output modules which produce data in a specific format can be implemented.

All explained steps are implemented in single layers which communicate through interfaces with each other. This design enables an easy modification of every layer which is why new technologies can be adopted, implemented and tested within the appropriate layer without the need to alter the other recognition steps.

3 Entity Relationship Diagram

First introduced by Chen (Peter Pin-Shan 1976) in 1976 the Entity Relationship model offers a way to present structured data in an abstract manner. Based on the ER model the diagram is a visual representation of given data. ER diagrams are used as a formal representation of different entities within a database and their relations.

In contrast to UML models ER models are not standardized. There are several different notations that have been proposed such as Weaver (1998) or Barker (1990). The components used in ER diagrams are entities, relationships, attributes and their relations with each other symbolized through connectors including cardinality constraints.

Entities represent discrete objects which exist in the real world and are different from each other whereby relationships describe the associations among these entities. Attributes can be owned by both, entities and relationships. They can be seen as properties describing values of their owners.

To distinguish different given entities unique attributes called keys are used to identify them. That means that two identical entities with equal attributes are not allowed. However, it is possible for an entity not to have a unique attribute, a so called weak entity. To be able to distinguish it, the weak entity must have another identifying entity which owns and identifies it. While a normal attribute is symbolized by a circle surrounding its name, key attributes are additionally underlined.

To describe possible relations between entities cardinality constraints are used. They specify upper and lower bounds. One-to-one, one-to-many, many-to-one and many-to-many are commonly used constraints.

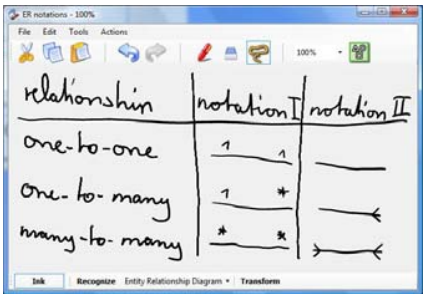

Figure 4. Different notation styles for cardinality constraints

As mentioned before there is no universal standard defining valid notations of ER diagrams. The most common differences between several currently used notations are:

Chen (Peter Pin-Shan 1976) used written cardinalities such as N/M or 1 whereas Barker (Barker 1990) used symbols called 'crowsfoot' to represent the relations between dissimilar elements (Figure 4).

Differences are also found within the connector drawing methods. While Chen uses single lines for optional and double lines for a mandatory participation in

a relationship, Weaver (Weaver 1998) proposes the use of a broken line for optional and a solid line for a mandatory participation.

Another common difference is whether relationships can have attributes. While Chen's model facilitates this, Weaver (Weaver 1998) only allows entities to have attributes.

The main difference visually is the representation of connectors including cardinalities. Purchase et al. (2004) showed that crowsfoot symbols (Barker 1990; Weaver 1998) increases the speed of identification when compared to written characters (Peter Pin-Shan 1976) such as M, N, 0 or 1. Furthermore, Purchase et al. (2004) compared Chen's and Weaver's cardinality notation and found an overall preference of Weaver's SSADM notation especially in the case of the cardinality representation. The use of crowsfoot symbols instead of written characters was easier to understand.

From a machine recognition point of view the use of a crowsfoot is easier to process. This is mainly because of recognition problems with single letters or numbers. The first challenge is to successfully determine these as text instead of drawing and the second one is to recognize the correct character or digit. In this process the letters 'O' and 'l' get mixed-up with the digits '0' and '1', and 'M' and 'N' are often mistaken too.

Current student textbooks such as Abraham, Henry et al. (2005) and Elmasri and Navathe (2006) propose notations which use Weaver's double lines for mandatory participation, allow relationships having attributes and propose the use of either Chen's or Barker's cardinality constraint notation.

4 Our Approach

Our ER extension in InkKit is designed for initial sketches therefore the ER's scope of services is not fully exploited. ER features which are currently addressed by InkKit are: entities, binary relationships, attributes and connectors with cardinality constraints.

Crowsfoot notation was chosen because of the human and machine recognition advantages over the written cardinality notation. The different types of connector ends specify the cardinality. The lack of a symbol at a connector's end sets the lower and upper bound to 1 whereas a crowsfoot sets the upper bound to many (Figure 4). In the case that no lower bound exists, a circle representing an optional constraint is used at the corresponding end of the connector. A relation between two components can be sketched by using a diamond as an alternative to many-to-many connector.

Furthermore, InkKit's chosen notation is designed to be as close as possible to modern student textbooks for example we use diamonds to represent relationships instead of writing them directly on the connector. To guarantee the best possible recognition rate an easily recognized notation was prioritized over staying precisely with a particular textbook notations.

A complete list of ER components recognized by InkKit is shown in Figure 5.

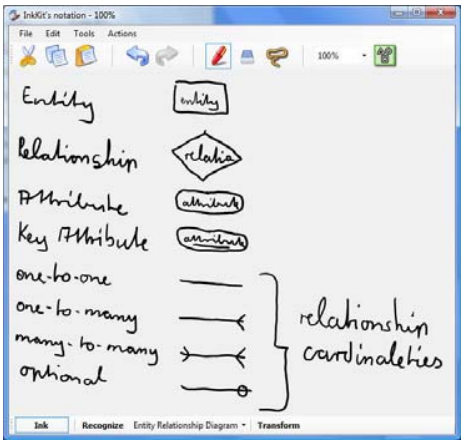

Figure 5. InkKit's notation set

We first integrated an initial interpreter and added new domain specific sketches of all ER component via InkKit's library interface. This first implementation clarified the challenges the recognition engine must overcome. The main difference between the ER domain and the already implemented domains is that different ER components have very similar shapes; for example the three different types of connectors.

In section 4.1 this problem and its solution is explained in more detailed. Afterwards the interpreter was implemented in detail including a data model which among other things corrects recognition errors. The implementation is described in section 4.2. Finally, two output modules have been written; one generates text and the other Microsoft Office Access files. These are explained in more detail in section 4.3.

4.1 Recognition engine extensions

Connectors including several different kinds of ends (any combination of a plain end, a circle and a crowsfoot) have to be successfully recognized. Until the ER domain extension was implemented one type of connector per domain was standard: the directed graph domain has directed connectors whereas the graph domain only has undirected ones. The fundamental problem is that all connector types are based on at least one shaft. The way that the InkKit recognition process builds a graph of all likely combinations (note this includes single and multiple stroke combinations) and calculates the similarity to the predefined sketched examples in the domain library causes a problem.

Let us assume a one-to-many relationship is sketched with two strokes, InkKit builds a graph consisting of three combinations, one combination which includes both strokes and one each for one stroke. By calculating the similarities between each combination and the predefined sketched examples it is very likely that the combination which only contains the 'shaft' stroke has a higher similarity with its predefined example than the multi stroke combination. To cope with this problem InkKit calculates features taking spatial relationships within multiple stroke combinations into account (such as enclosing, enclosed, near or intersecting). Based on the relationship a graph weighting is processed supporting multi stroke combinations. Connector shafts and their

crowsfoot symbols typically intersect but connectors can also intersect with the components they link to. Therefore the tree weighting factor based on intersections cannot be strong otherwise all intersecting components are heavily supported although they might not belong together.

To cope with this problem a new feature called 'over' has been introduced. It calculates the amount of ink on both sides of the intersection point of the strokes. If both amounts of ink are similar the strokes are over each other. In Figure 6 the connector's crowsfoot is divided into two similar sized halves (a) whereas in (b) the relationship component is divided into two different sized halves.

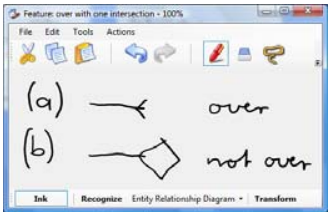

Figure 6. Feature: over with one intersection

If one stroke divides another one into two halves through multiple intersections the algorithm compares the amount of ink on both sides of the first stroke (Figure 7).

Before we introduced our new feature the recognition of connectors frequently failed. The extension to the recognition engine is resulted in considerably less failures. To what extend the overall recognition rate has been improved cannot be exactly measured in a reasonable way because the rate depends on factors such as the sketched components and their spatial relations. Whereas the new feature has no direct impact on multi stroke components which do not match it, the recognition rate of the feature matches has been significantly increased. An informal test of connectors showed that without the implemented 'over' feature, 4 out of 10 connectors were falsely recognized but with the error rate fell to less than 1 in 10.

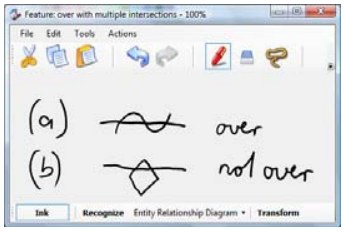

Figure 7. Feature: over with multiple intersections

4.2 Interpreter data model

The ER domain specific interpreter defines the existing component types; entity, attribute, relationship, connector and text. In InkKit's domain library the sketched ER component examples (Figure 5) are assigned to one of the component types defined in the interpreter. After the recognition process is finished the interpreter gets a list of recognized shapes whereby every single shape is assigned to one interpreter component type. Based on this list the interpreter builds a data model which is handed over to the output modules. The data model is built on classes describing the entity, attribute and relationship components. The build starts with assigning the attributes to the corresponding entities and continues with assigning attributes to their relationships. Finally the entities are assigned to the appropriate relationships. Before the data model is built the recognition results is tested for errors. The detection is divided into two parts; a spatial and context error detection.

The spatial error detection tests the component's relations based on simple grammar rule productions (Costagliola, Deufemia et al. 2004) particularly designed for ER diagrams. Thereby any pair of components (entity, relationship and attribute) has to be linked with a connector. However, two connectors in a consecutive spatial order are an invalid sequence of elements. If an invalid and not reconstructable arrangement is detected the entire sketch recognition result is discarded. The interpreter cannot rerun the recognition process so the user has to manually classify the incorrectly recognized components and restart the recognition.

The context error detection is based on certain rules; entities, relationships and attributes must enclose text whereas optional cardinalities do not include text. In the case when an attribute without text is detected which is also intersecting a connector it is reclassified as an optional cardinality constraint. In contrast, an optional cardinality constraint enclosing text is reclassified as an attribute.

4.3 Output modules

Upon the data model being handed over by the interpreter, output modules generate files in specific formats which can be used by other tools. We have implemented two output modules; one generates files in Microsoft Office Access format whereas the other generates plain text files.

The mapping from the turned over data into the data specific format is based on a compact mapping methodology introduced by Draheim and Weber (2005). The principle is not to map every relation into a table; this prevents unnecessary inflation of the generated code. The decision whether to map a relation into a table is made in respect to the given cardinalities. However, if a relationship in the form of a diamond is sketched it is mapped into a table regardless of the given cardinality.

Microsoft Office Access is a relational database management system. Due to its user friendly User Interface and other features software developers almost regardless of their skill level can use it to build functional prototypes.

5 Evaluation

To evaluate InkKit's implemented ER plug-in an exercise from a current student textbook (Abraham, Henry et al. 2005) is used. The assignment (page 256, 6.1) is to "Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to many number of recorded accidents." This exercise includes every implemented ER

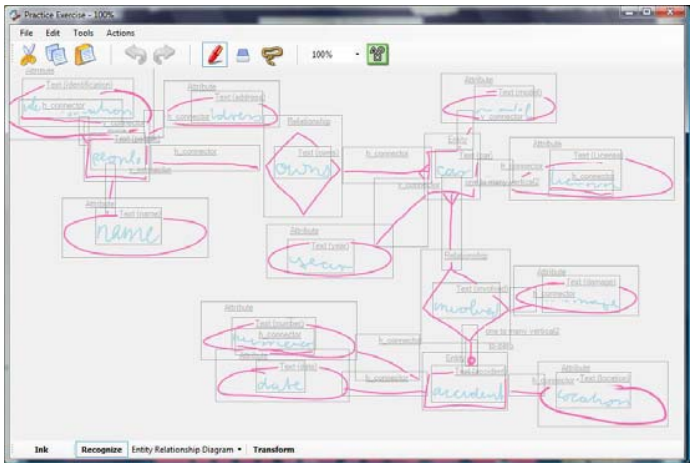

Figure 8. The successfully recognized sketched solution of the practice exercise

component and several different cardinality constraints such as zero-to-many, one-to-one and one-to-many. Additionally, the solution includes binary relationships owning attributes. The sketched solution is shown in Figure 8, plain text in Figure 9 and the corresponding Microsoft Office Access file in Figure 10.

6 Discussion

Sketched diagrams are often used as the first step for designing systems. Different kinds of diagrams are already recognized by InkKit and other sketch tools. However, recognizing ER diagrams using a generic recognition engine carries new challenges. For example, finding features to distinguish relationship cardinality constraints from other sketched shapes as well as from each other is difficult to realize with a generic recognition engine because it cannot be specialized towards particular domain components. Being a central part of ER diagrams makes the successful differentiation of cardinality constraints essential. Similar notations are also found in other types of diagrams such as UML class and activity diagrams. This was a motivating factor for the decision to add ER diagrams to InkKit's scope of services.

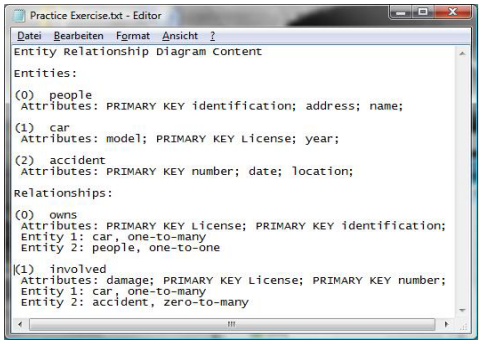

Figure 9. The generated plain text output solution of the practice exercise.

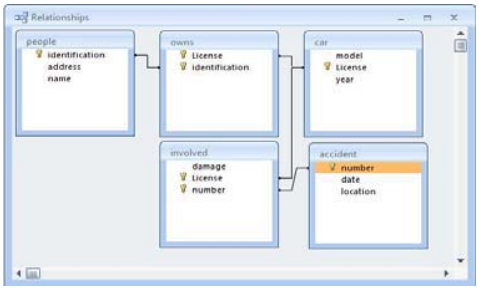

Figure 10. The generated Microsoft Office Access output solution of the practice exercise.

We have explored ER diagrams including their different notations and limitations. One part of the ER notation set where two different common versions exist is the cardinality constraints (Peter Pin-Shan 1976; Barker 1990). There is a written and a symbolic notation style (Figure 4) specifying the relations between entity components. The advantage of the symbolic notation for humans is the ease of comprehension. Additionally, the machine recognition rate of symbols is significantly higher than for single letters and digits. However, the use of symbols limits the granularity of relationship constraints; e.g. it is only possible to express nothing, one and many whereas the use of letters and digits enables an infinitely more specific expression (Purchase, Welland et al. 2004). Nevertheless, being designed for preliminary sketches the symbolic notation is sufficient for InkKit's ER plug-in.

Furthermore, not all ER components are considered in the ER plug-in (such as derivation, composite or multi-valued attributes and non-binary relationship sets) because the ER plug-in is intended to cover preliminary sketches. Another element which is not part of our implementation is the total participation notation, a double lined connection to indicate a mandatory participation of an entity in a relationship. We defined a connector without any crowsfoot symbols as a one-to-one cardinality constraint implying a total participation which makes the double line approach unnecessary. To model a voluntary participation we implemented the optional cardinality constraint component. This carried another

recognition challenge because the already existing component attribute has a very similar shape; an ellipse in comparison to a circle. The fact that attributes always enclose text and the optional cardinality constraint does not, is used to correct this false recognition in the plug in. However, this solution is not absolutely satisfying because it cures the problem after it occurs instead of preventing it from happening.

Another decision we made was how severe a false recognition has to be to abandon the complete recognized sketch result. We found that errors like mixing up attributes and optional cardinality constraints are not critical. Whereas if two consecutive connectors or any pair of components (entity, relationship and attribute) are detected the entire recognition result is discarded, since those arrangements are not allowed and if it is not possible to reconstruct the right components the entire result is compromised.

Another choice was the mapping method of the recognized data model into the format specific data. Concerning relationships there are two different mapping methods. The first one is a straight forward so called 'direct mapping' method where basically every relation is mapped into a table (Draheim and Weber 2005). The second one, called compact mapping, does not necessarily do that. Based on the given cardinality information the compact model merges relations. For example many-to-many relations are always mapped into a table whereas one-to-many and one-to-one are not. The reason to implement the compact mapping model is to dispose of unnecessary and redundant information. There is no need to generate a table for a one-to-many because referencing a foreign key from the 'many' table in the 'one' table is sufficient information.

Implementing an ER domain into InkKit with a scope of services covering preliminary sketches enabled an evaluation of this new combination. The new findings could be adopted in other domains implemented in generic sketch tools to increase the recognition rate of multi-stroke connectors. Additionally, the decisions made about ER's different notations regarding easiness of sketching and recognition provide a basis for future sketch tool domain implementations such as UML collaboration and object diagrams or ones with similar notations.

The effort required to extend InkKit is significantly less than that of developing a domain specific sketch tool. Depending on the scope of services the interpreter covers, an implementation varies from 150 lines of code (InkKit's Organization Chart Interpreter) to 880 (InkKit's ER Interpreter). The same applies for the output generators. While the simplest version consists of 130 lines of code (InkKit's graph text output module) the most complex one has over 350 (InkKit's ER Microsoft Office Access output module).

7 Conclusion and Future Work

As an example of diagrams with complex connectors the ER diagram has been implemented into InkKit. The interconnectivity between ER components enabled for the first time in InkKit the use of a spatial and context error correction model. We increased the recognition rate by introducing new features into InkKit's core recognizer.

We explored the combination of sketching and ER diagrams in detail and decided, based on our findings, which set of notations an ER sketching tool should have. By using the symbolic cardinality notation rather than the written one, a user and recognition friendly approach has been chosen. Furthermore, InkKit provides a powerful way to automatically generate preliminary ER diagram sketches into a specific data format. The evaluation has demonstrated the scope of services of InkKit's new ER domain including the successful recognition and generation of plain text and Microsoft Office Access code.

More detailed evaluations about which components real preliminary sketches consist of, the chosen notation and its user acceptance are necessary. Additionally, to examine the efficiency and easiness an evaluation of InkKit's new ER domain against widget based tools is needed. There is potential for implementations of other similar domains based on this work.

8 References

Abraham, S., F. K. Henry, et al. (2005). Database System Concepts, McGraw-Hill, Inc.

Alvarado, C. and R. Davis (2004). SketchREAD: a multi-domain sketch recognition engine. Proceedings of the 17th annual ACM symposium on User interface software and technology, Santa Fe, NM, USA, ACM Press.

Bailey, B. P. and J. A. Konstan (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. CHI 2003, Ft Lauderdale, ACM.

Bailey, B. P., J. A. Konstan, et al. (2001). DEMAIS: Designing Multimedia Applications with Interactive Storyboards. ACM Multimedia.

Barker, R. (1990). CASE Method Tasks & Deliverables, Addison Wesley.

Chen, Q., J. Grundy, et al. (2003). An E-whiteboard application to support early design-stage sketching of UML diagrams. Human Centric Computer Languages and Environments, Auckland, NZ, IEEE.

Costagliola, G., V. Deufemia, et al. (2004). A Parsing Technique for Sketch Recognition Systems. Visual Languages and Human Centric Computing, 2004 IEEE Symposium on.

Coyette, A., J. Vanderdonckt, et al. (2007). SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping Berlin / Heidelberg, Springer 160-176.

Damm, C. H., K. M. Hansen, et al. (2000). Tool support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard. Chi 2000, ACM.

Draheim, D. and G. Weber (2005). Form-Oriented Analysis, Springer Berlin Heidelberg.

Elmasri, R. and S. B. Navathe (2006). Fundamentals of Database Systems (5th Edition), Addison-Wesley Longman Publishing Co., Inc.

Goel, V. (1995). Sketches of thought. Cambridge, Massachusetts, The MIT Press.

Hammond, T. and R. Davis (2002). Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. 2002 AAAI Spring Symposium on Sketch Understanding.

Landay, J. and B. Myers (1996). Sketching storyboards to illustrate interface behaviors. CHI '96, Vancouver, BC Canada, ACM.

Lank, E. H. (2003). A Retargetable Framework for Interactive Diagram Recognition. Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 1, IEEE Computer Society.

Lin, J., M. W. Newman, et al. (2000). Denim: Finding a tighter fit between tools and practice for web design. Chi 2000, ACM.

Peter Pin-Shan, C. (1976). "The entity-relationship model - toward a unified view of data." ACM Trans. Database Syst. 1(1): 9-36.

Plimmer, B. and I. Freeman (2007). A Toolkit Approach to Sketched Diagram Recognition. HCI, Lancaster, UK, eWiC.

Plimmer, B., G. Tang, et al. (2006). Sketch Tool Usability: Allowing the user to disengage. HCI London, ACM.

Plimmer, B. E. and M. Apperley (2003). Freeform: A Tool for Sketching Form Designs. BHCI, Bath.

Plimmer, B. E. and M. Apperley (2003). Software for Students to Sketch Interface Designs. Interact, Zurich.

Plimmer, B. E. and M. Apperley (2004). INTERACTING with sketched interface designs: an evaluation study. SigChi 2004, Vienna, ACM.

Purchase, H. C., R. Welland, et al. (2004). "Comprehension of diagram syntax: an empirical study of entity relationship notations." Int. J. Hum.-Comput. Stud. 61(2): 187-203.

Rubine, D. (1991). Specifying gestures by example. Proceedings of Siggraph '91, ACM.

Weaver, P. L. (1998). Practical SSADM Version 4, Trans-Atlantic Publications.

Yeung, L. W. S. (2007). Exploring beautification and the effects of designs' level of formality on the design performance during the early stages of the design process Department of Psychology. Auckland, University of Auckland. MSc.