Generalised Diagram Revision Tools with Automatic Marking

Pete Thomas
Centre for Research in Computing
The Open University
Milton Keynes, UK
+44 (0) 1908 652695
p.g.thomas@open.ac.uk

Kevin Waugh
Centre for Research in Computing
The Open University
Milton Keynes, UK
+44 (0) 1908 652695
w.g.waugh@open.ac.uk

Neil Smith
Centre for Research in Computing
The Open University
Milton Keynes, UK
+44 (0) 1908 652695
n.smith@open.ac.uk

ABSTRACT
In this paper, we describe an approach to the generalization of revision tools for modelling with diagrams. We have implemented two revision tools that automatically mark and provide feedback on students’ attempts at constructing diagrammatic models of given scenarios in different domains. The similarities between these tools and diagrams in other domains suggest that it might be possible to generalise both the marking algorithm and the drawing editor in such a way that new revision tools could be easily generated for new domains. This paper briefly describes the existing revision tools and our approach to automatic marking and discusses how we are approaching the generalization of our work for application in other domains.

Categories and Subject Descriptors
K.3.2 [Computer and Information Systems Education]: computer science education.

General Terms
Design, Experimentation.

Keywords
Diagrams, automatic marking, feedback, revision tools.

1. INTRODUCTION
Over the past few years we have been developing revision tools for teaching and learning graph-based diagram modelling skills [8]. These tools are designed to provide the student with a series of graded questions which require the students to draw a diagram which models the scenario given in the question. The tools incorporate a diagram marking engine that compares a student-generated diagram with one or more model solutions and grades the student’s attempt [10,11]. The tools also provide graphical as well as textual feedback at a variety of levels of detail that the student can use to improve their answers. The intention has been that the tools should be used as formative revision aids.

To-date we have built two such tools: one for entity-relational diagrams and one for UML sequence diagrams [10]; both are used in courses and have been well received by students [9]. We have also experimented with biological flow diagrams.

This work has been a spin-off from our more general interest in understanding diagrams and in particular imprecise diagrams (such as diagrams that contain errors as is often the case with student productions) [7]. But, our longer term aim is to generalise this work so that the automatic marker can be applied to a wider set of domains. One consequence of this is the need to produce revision tools applicable to different domains or for the same domain but with different notations (the classic example is of entity-relationship diagrams for which notations abound). Moreover, there is a need to produce authoring tools for the creation of questions, model solutions and mark schemes that can feed into the revision tools.

Ideally, we would have one single revision tool and one authoring tool which could be easily tailored to specific domains. Our work to-date suggests that it might be possible to generalise both the marking engine and the drawing editor. There are other examples of generalising diagram editors. For example, the CIDER project [5] takes a description of a diagramming domain in the form of a formal syntax and generates an editor for drawing diagrams in that domain. To-date they have applied their technique to finite state automata. This paper describes our early work at designing such software.

2. REVISION TOOLS FOR DIAGRAMS
Our original revision tool is designed for learning the modelling skills associated with entity-relationship diagrams (ERDs). Figure 1 shows the user interface of this tool. There are three main panes. The left-hand pane contains the question (in textual form); the top, right-hand pane is used for drawing diagrams and the bottom, right-hand pane is used for displaying the model solution.

The tool bar towards the top of the window contains a variety of tools for building and editing ERDs. There are also tools for marking a diagram (Mark button) and viewing the model solution (Show/Hide button) although students are encouraged not to use it until they have exhausted all the other teaching elements built into the revision tool. Once a diagram has been drawn, the student can ask for it to be marked and will be given a grade and some elementary feedback. Further feedback, in
diagrammatic form, can be requested in which the student’s diagram is compared with the specimen solution and errors identified.

Figure 1. The ERD Revision Tool

If this feedback is insufficient or the student is lacking in confidence about the answer they have produced, the student can examine the model solution by clicking on its elements and receiving an explanation of why that element exists and how it relates to the question scenario.

An important aspect of the ERD revision tool is that the diagram editor will allow the user to create both syntactic and semantic errors. The editor does not attempt to impose a correct syntax. This enables the tool to provide a wider range of feedback and can reinforce the notation when it is applied incorrectly. It is the analysis of imprecise diagrams that distinguishes this tool from other tools.

At its simplest, an ERD is basically a set of boxes with relationships between them. The relationships are either explicit in the diagram, being represented by lines joining boxes, or implicit, being represented by the spatial juxtaposition of entities (used to denote subtyping between entities). In some notations, attributes of entities are drawn using other shapes that are connected to the entities by lines and constitute another type of relationship in the diagram.

When one examines UML sequence diagrams (SDs), there is a great deal of similarity to ERDs in that there are entities (now called objects) which are related by arrowed lines (representing the sending and receiving of messages between objects). Sequence diagrams also have activations and different types of message. In other words, a sequence diagram has a small set of types of things (objects and activations) related in different ways (indicated by different types of arrows and lines). However, sequence diagrams, as their name suggests, have one very different feature from ERDs: the messages and their associated activations are in sequences (indicating the passage of time).

Sequences are represented on a diagram by the relative position of activations and messages, with time conventionally progressing down a diagram from top to bottom. Figure 2 shows the user interface for the sequence diagram revision tool. It bears a deliberately strong resemblance to the ERD revision tool.

However, the SD revision tool has much more latitude in what can be drawn and has a syntax checking mechanism for student use.

3. DIAGRAM MARKING

The similarities between ERDs and sequence diagrams has encouraged us to look at other types of diagram. We have examined biological flow diagrams, UML class diagrams and influence diagrams and found considerable similarities, sufficient to convince us that our approach to diagram marking, in which we analyse a diagram into its minimal meaningful units (MMUs) and then aggregate these into more complex meaningful units (MUs) [7,11] before finding similar structures in two diagrams to determine their overall similarity, is a viable approach in diverse domains. In general, it is the relationships between the entities/objects in the diagrams which form the basis of most MUs. The approach compares all MUs in one diagram with all MUs in the other diagram and similarities are determined for each comparison. An optimisation step determines the most likely match.

Marking schemes are specified in terms of MUs and marks awarded on the basis of the degree of similarity between MUs. Experimental evidence has provided positive evidence that this
approach determines marks which match well with those from expert markers [9].

Figure 2. The SD revision tool.

The advantage of this approach is that the algorithm can match ‘broken’ or erroneous MUs in the student diagram with MUs in the model solution to determine the most likely match. Having determined matching MUs, the differences between the student-produced MU and the matching model solution MU are the basis of feedback.

4. DIAGRAM EDITORS

Our approach, in which we deliberately allow students to make errors in their diagrams without the drawing tool imposing constraints, means that the drawing editor can be quite simple and can be generalised for several domains. Once again, taking relationships as the basis of MMUs, provided the MMUs for each domain have been identified, it is possible to provide a specification tool that outputs a specification for a drawing editor which, when input to the generalised drawing tool produces a specialised editor for that domain.

As a first step, we have developed a specification tool which allows the author to construct a specification of a drawing tool for their own domain. The diagram specification tool has two functions: to provide data to drive the marking engine (various weights and thresholds required by the algorithm) and data to drive the diagram editor. Of particular interest is the relationship specifier: see Figure 3.

Relationships are viewed as having adornments (in much the same way that entities and objects can have attributes). An adornment is an icon or item of text which is placed on, or adorns, a relationship.

Adornments are normally positioned at either end of a relationship or in the middle, and can appear either above or below the line. The crosses in Figure 4 show the 10 positions supported by the specifier. In addition, it is possible to specify the format of the line itself (solid, dashed etc.), important in sequence diagrams, for example.

Figure 3 Specifying a relationship

There are four types of adornment. A string adornment is simply a text string which is matched with another string adornment on the basis of string edit distance. A label adornment is also a text string but is used to label (or name) either the relationship as a whole or a role that the relationship can take part in. There are substantial semantics associated with matching labels as these play a central role in identifying and distinguishing relationships [6]. A range adornment, of the kind that appears in UML class
diagrams, specifies a cardinality (multiplicity) of a relationship. Currently, the matching is performed on ranges as though they were simple strings. An icon adornment is a small picture denoting some attribute often associated with cardinality, for example, a crows foot in an ERD specifies the notion of ‘many’.

The specifier has been successfully used to generate different versions of the ERD revision tool (versions supporting different notations).

5. CONCLUSIONS AND FUTURE WORK
In this paper we have discussed only a small part of the research into diagram understanding. To-date we have a framework for the kinds of diagrams we believe our approach is applicable to: essentially relaxed graph-based diagrams. We have working versions of two revision tools and a mechanism for easily generating versions of these tools to support alternative notations.

Our next step is to formalise the work using constraint based grammars which will be used to drive the diagram editor specifier.

6. REFERENCES