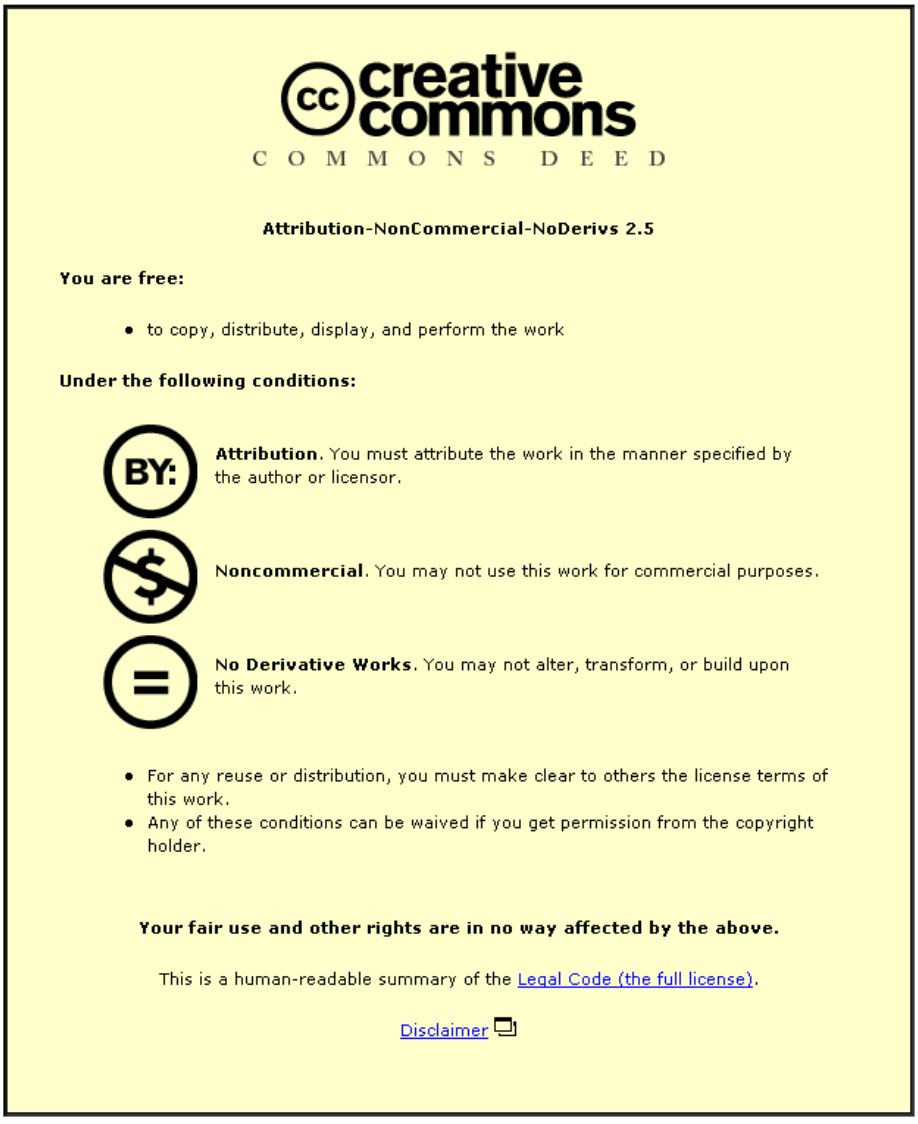

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

DRAWING AND MARKING GRAPH DIAGRAMS

Roger Stone
Department of Computer Science
Loughborough University
Loughborough, LE11 3TU
R.G.Stone@lboro.ac.uk

Firat Batmaz
Department of Computer Science
Loughborough University
Loughborough, LE11 3TU
F.Batmaz@lboro.ac.uk

Chris Hinde
Department of Computer Science
Loughborough University
Loughborough, LE11 3TU
C.J.Hinde@lboro.ac.uk

ABSTRACT

The marking of graph diagrams (that is to say diagrams that are composed of nodes, possibly joined by edges) is tedious if the diagrams are presented on paper. If the key content of the diagrams is available in electronic form then the marking can be much more efficient. This is achieved because the tutor only has to mark each different diagram element once and this mark is transmitted to all diagrams that contain the element. This benefit to the tutor is obtained by requiring the students to use a diagram drawing program of some kind. However using such an editor can simplify the process for the students by allowing them to concentrate more on the problem and less on its graphical representation. The students can also be rewarded for going to this extra effort by receiving a much more detailed, personalised commentary on their work than would have been possible before, given the same amount of tutor time. We present the evolution of a drag-and-drop diagram editor specialised for the area of ER diagrams and an associated marking system with a simple but effective feedback mechanism. Some results from initial trials are presented along with some ideas for improvement and extension.

Keywords

Marking, Diagrams, ER diagrams, Graphs, UML, Drag-and-drop.

1. INTRODUCTION

Teachers tend to complain about the business of marking, especially when the number of students in the class is high. If it is possible an experienced tutor will arrange the questions so that the answers are easier to mark. Even so, it is still necessary to process each student's work individually. Now consider the subset of the marking problem where the student has drawn a diagram. If it is a representation of a physical system (e.g. apparatus in a Chemistry experiment) then it will be quite easy to check (Is the bunsen burner under the flask? Is the flask connected to the condenser? etc.) Apart from perhaps a mirror image, all correct student diagrams should be recognisably the same. However in more abstract diagrams, for example graphs (meaning diagrams with nodes connected by edges) there is no single correct place for any particular node. It is just that certain nodes must be included, and they must be connected to specific other nodes in specific ways. Thus a graph of 5 nodes if drawn with nodes in 'domino' spot formation, can be presented in 5!=120 ways. In other words all the graphs from a class of 100 students could look very different.

However it is not the topology which is important in these diagrams. As stated before it is the nodes, their labelling and the edges which connect them to other nodes which is important. The visual layout is of no help in recognising the 'correctness' and it is very tedious to check each graph node-by-node and edge-by-edge against a master solution.

This marking process is beginning to sound very mechanical and should be amenable to transferring to a computer. The concept is that the computer should process all the students' diagrams and present the work to the tutor, not on a student-by-student basis, but on a component-by-component basis. Every node and every connection that appears in all the diagrams can be marked independently by the tutor.

How much work has the tutor done? Instead of marking every node and every edge for every student the tutor has only had to make decisions once for every different node and every different edge. This should be a significant reduction in effort. What is more, if this exercise is repeated with a different class, the tutor should only have to mark any new nodes or edges that have been generated by the new class - hopefully only a trivial amount of work.

The obvious problem is how the computer is going to be able to 'read' the student's diagrams. Our solution is to give the students a smart editor which allows them to construct the diagram on screen and save their work so that it can be submitted to the tutor in electronic form. Then the biggest obstacle to computer-based marking is overcome.

But what do we mean by a 'smart' editor? Ideally we are looking for an editor that the students would use as a matter of choice, because it benefits them in some way (other than being the only route to getting their work marked). From the student point of view the editor should be quick, easy and satisfying to use with a minimal learning curve. Another is that the editor should be attractive to use in comparison with 'pencil and paper'. From the tutor point of view it would be particularly good if the editor actually contributed to the learning process in some way and of course the editor should deliver data about the student diagrams in a form that is convenient to 'mark'.

The immediate problem faced by the authors is that of marking Entity-Relationship (ER) diagrams [1] and so the rest of this paper will use ER diagrams as the example context although it is claimed that the method can be applied to any other graph-based work. The idea was to provide a highly interactive web page that could be blended in with other teaching material - a Javascript 'gadget' in the sense of Stone [2]. The web page would contain a diagram editor, purpose made for this situation, though in the light of current trends the Javascript technology needs to be updated to AJAX (Advanced Javascript And Xml).

2. BUILDING A SMART EDITOR

2.1 Existing Editors

There are any number of traditional diagram editors on offer. Many of these can be used to create ER diagrams - see Wikipedia for a list [3]. The general trend is towards producing VLEs with Web-based interfaces and the choice of techniques and technologies for drawing a diagram in the web environment is more restricted if standalone applications are ignored. However this still allows the use of embedded technologies like Flash and Java (notably JGraph [4]) and more native technologies like images with click maps (e.g. Webdot [5]) and, possibly the newest of all, Scalable Vector Graphics (SVG) as a client-side technology.

Whether standalone or built for the web these editors are mostly built on a standard pattern of having a selection of drawing tools (ovals, rectangles, arrows lines, etc.). So the process is one of clicking a shape tool, clicking on the drawing area to place the shape, typing a word or words as a label for the shape and finally connecting this newest shape to other shapes. There are two principal disadvantages with this approach. One is that the interaction keeps bringing the user's focus onto the drawing tool itself and the other is that the end result is only a picture. What is required is a machine readable description of the diagram preferably rendered in the semantic terms of the application area (e.g. entity, attribute, relationship) rather than in the semantics of the drawing tool (rounded shape, rectangular shape, arrow, etc.).

The KERMIT system [6] and ERM-VLE [7] have similar pedagogical goals to ours. However we do not wish to be restricted to a command-line interface and passive graphics (ERM-VLE) and we do not wish to give advice as the student is drawing the diagram (KERMIT). The Exerciser project [8] and the Theseus component of CourseMarker [9] are tools which contain the ability to draw ER diagrams but do not have the intimate relationship between scenario and diagram that we seek. An early prototype of our own, designed primarily to try out our tutor marking ideas rather than the students' drawing, was reported in Batmaz [10].

The exercise of drawing an ER diagram is to take a written, textual scenario and deduce the entities, attributes and relationships from the text and finish by drawing these elements as a diagram according to certain conventions. Thus a scenario extract like

> A hospital may administer several wards. Each hospital has a name and address. ...

would produce two entities (*hospital* and *ward*), two attributes associated with the entity hospital (*name* and *address*) and one relationship between the entities hospital and ward (*administer*). From a pedagogical point of view we want the students to pay attention to the scenario section-by-section, sentence-by-sentence and word-by-word and to tease out the information that it offers.

2.2 Building with AJAX

Before the technology mix referred to as AJAX came into common use a web page author was conditioned to think in terms of an interaction or intervention by the user as causing a whole page refresh or at least a whole frame refresh. Thus it comes as a surprise and a release to find that using AJAX technology the designer is

ISSN: 1473-7507

encouraged to design an initial, possibly quite sparse, document which is treated in a very dynamic way. The document is viewed as a tree having as its root an HTML node. All the nodes on the tree are then regarded as possible sites for addition or replacement, including any newly added nodes. When a replacement is made the browser is required to render the change dynamically in full view of the user and most browsers do this remarkably well. JQuery [11] capitalises on this by allowing the addressing of a replacement node to follow the well established CSS selector notation and allowing the replacement node to come from marked up text or be copied from an existing structure and either supplied locally via Javascript, read from an external xml document or delivered as the result of a server-side script.

2.3 Editor – First Version

Thus we embarked on the building of a special purpose graph editor. The first design for the interface contained the ideas that the scenario would not be just a static piece of text but that the noun phrases that might later become entities and attributes could be highlighted. To this end the scenario was given HTML markup. Each section is marked as a paragraph (<p>). Each sentence is identified by a with class attribute *sentence*. Each noun phrase is identified by a with class attribute *nPhrase*.

```
<p class='section'>
   <span class='sentence'>
      The  <span class='nPhrase' title='Hospital'>Hospital</span>
      may administer several <span class='nPhrase' title='Ward'>wards</span>.
   </span>
   <span class='sentence'>
      Each <span class='nPhrase' title='Hospital'>Hospital</span>
      has a <span class='nPhrase' title='Name'>name</span>
      and <span class='nPhrase' title='Address'>address</span>.
   </span>
</p>
```

Figure 1: Scenario mark-up

Thus the first version of our editor was able to process the scenario, extract the noun-phrases and present them on a menu. By selecting from the menu the user was able to highlight any chosen noun-phrase together with all its repetitions.

A hospital may administer several wards. Each hospital has a name and address.

Figure 2: Noun-phrase highlighting

After selecting a noun-phrase the user was able to create an element on the ER diagram by selecting an appropriate tool (add_entity, add_attribute). Each editing action was recorded in a database. Each record consisted of the name of the user, the scenario being considered, the editing action, parameters and a sequence number.

Editing action	Parameters
Create entity	Name of entity
Create attribute	Name of attribute and name of related entity
Create relationship	Name of relationship, names of the entities it connects (2), the arities of the connection (2) and the number of the scenario sentence from which the relationship was deduced.

Figure 3: Database record detail

By storing every action (including deletions), a description is kept not only of the finished diagram, but also of the sequence of steps by which the user had arrived at the end product. The growing diagram was rendered on screen by using Graphviz/WebDot. This was used principally to avoid us having to write our own layout algorithm. This version was trialled with an MSc class with some success. The class were able to build their ER diagrams quite rapidly after a very quick visual demonstration of how to use the editor. Their diagrams were stored and the marking was achieved and the results returned. However there was a feeling that the interface was still too clumsy and did not focus as much on the scenario as we had intended.

2.4 Editor – Version 2

The second version featured a drag-and-drop interaction. The idea was that the students should point directly at words in the scenario and turn them into entities and attributes by dragging them to specific targets. Thus

noun-phrases in the scenario were made draggable and the building tools made into drop targets. This was achieved simply and in a cross platform fashion by using the UI facilities of JQuery. Now the user could drag a noun-phrase directly from the scenario and drop it onto (say) the entity builder tool and the new entity element would be added to the diagram.

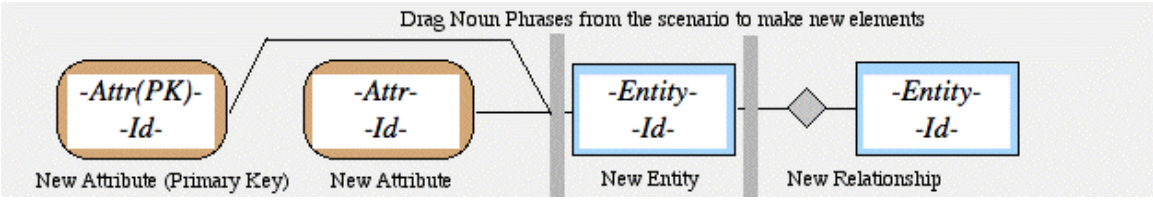

Figure 4: Screenshot of the editor tools, version 2

This version was well received when tried with a large first year class. Almost all of them were able to build their own diagrams easily after a very quick visual demonstration of how to use the editor. The impression was that this editor allowed the diagrams to be built considerably faster, that the interface did not get in the way and that the questions we were being asked were more about the decisions about how to interpret the scenario than how to work the interface.

The relationship tool in this version was improved. We required six items to be recorded. These items are the two entities that are involved in the relationship, the two arities of the connection (1-to-1, 1-to-many, etc), the name of the relationship and the sentence of the scenario that justified the creation of the relationship. In version 1 the name of the relationship had to be typed in by the user. In the version 2 editor the sentence of the scenario was deduced from the position of the two entities. The non noun-phrase words of that sentence were then offered via a menu so that one or more could be selected to form the name of the relationship. This meant that there was now no typing at all required during the diagram construction process.

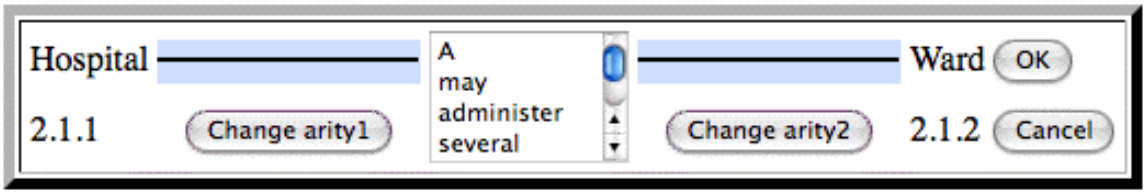

Figure 5: Screenshot of the attribute editor, version 2

A potential collaborator at another university had said that they worked with UML-like notation for ER diagrams and requested that as an alternative notation. As a response to that suggestion the interface now includes a 3-way switch to list the elements of the diagram in tables or show them drawn in ER or UML style.

2.5 Editor – Version 3

It was noticed that the re-drawing of the diagram from scratch after every touch of the editor was causing a distraction as the diagram would quite often be drastically rearranged by the layout algorithm after an apparently innocuous addition. This irritation together with the recent introduction of UML-style diagrams suggested a third version of the interface in which the diagram is not drawn as a graphic but simulated using HTML tables. In this version noun-phrases are dropped directly onto the 'diagram' without any intervening tools.

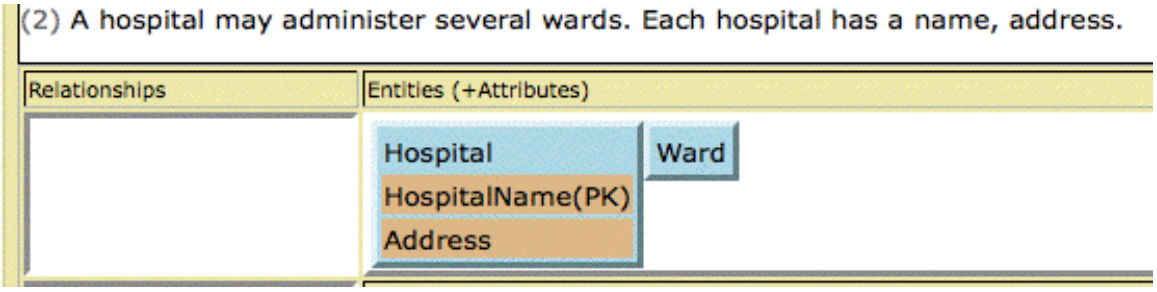

Figure 6: Screenshot of part of the editor interface, version 3

A noun-phrase dragged from the scenario and dropped into the space below the heading 'Entities' builds an entity. A noun-phrase dropped onto an existing entity builds an attribute for that entity. A noun-phrase dropped into the space below the heading 'Relationships' starts off the building of a relationship.

ISSN: 1473-7507

As regards implementation, the initial version of the page is almost empty. The scenario is loaded dynamically and any elements of the diagram from a previous session are created dynamically. The table data (<td>) node below the heading 'Entities (+Attributes)' has the attribute *id='entities'*. This allows the table data node to be made a droppable target in JQuery by writing *$("#entities").droppable()*. Each new entity can be introduced as a table with attribute *class="entity"* so that the most recently added entity can be addressed as *$("#entities .entity:last")* i.e. the *last* node with class *entity* which is a descendant of the node with id set to *entities*. In this way the browser page that the user sees is being managed dynamically with very compact code.

This interface seems to focus attention fully on the (ER) task at hand. The interface is unusual for a diagram drawing tool in that it has no obvious 'tools'. It has been noted that it works well using a touch screen and we anticipate some experiments to see if finger touching of the words in the scenario and also finger dragging serve to bring the user into even closer and beneficial contact with the words of the scenario. Looking further ahead work with multi-touch screens is planned which will include experiments to discover whether multi-touch gestures such as pinch and stretch can be utilised naturally for more advanced work (e.g. merging and splitting entities).

3. Diagram Marking Systems

There is not much reported work in the area of diagram marking. Higgins and Bligh [9] have developed an extensible system in the context of the CourseMarker CBA system which has been used for ER Diagram marking. Thomas *et al.* have a system [12] which has also been used for ER diagram marking.

There are various decisions to be made in designing any machine-based marking system. Is it for revision/support or formative or summative assessment? Is it to be fully automatic, partially automatic (with tutor completion) or just there to assist the tutor in the whole process? Is it to be applied to a single student on demand or applied to a whole class when invoked by the tutor? All these issues are also relevant for a diagram marking system but additional issues arise. For example how much help should the diagram editor offer the student? When students draw diagrams using an editor program they may be prevented from making certain mistakes (e.g. using completely inappropriate shapes) because menus or toolboxes only offer a limited choice. Another issue is whether the editor or the student should take responsibility for the layout of the diagram. Existing systems only seek out and mark components of a diagram and do not contain the ability to comment on the layout of the diagram. An important design decision is whether the students are allowed free labelling of diagram components. We constrain the students to choose words from the scenario while other systems [e.g. 8] allow free text entry and may then have to use a combination of synonym, stemming and similar algorithms to allow a student's answer to match a tutor solution.

3.1 Building the Marking System

The basic necessity is to mark every node and every edge of the diagram separately. The topology of the diagram is crucial but not the layout. There was no desire to give a numerical mark but merely to indicate good and bad parts of the diagram and add comments. The central idea was to colour elements of the diagram with traffic light colours using red to convey *wrong*, amber to convey *acceptable* and green to convey *correct*. The tutor's comment for an element is made visible by hovering the mouse over the element. The text *wrong*, *acceptable*, *correct* is shown with the comment to assist any student who may not be able to distinguish the colours.

3.2 The Tutor Diagram

If the tutor draws a master version of the diagram for a particular scenario then this can be used as a first cut at marking. Any element on a student diagram that matches an element on the tutor diagram can be coloured green and given an appropriate automatic comment.

3.3 Tutor Marking

The tutor can then be directed to unmarked elements on student diagrams, firstly to entities, then attributes, then relationships. After all the remaining entities have been marked by the tutor, the system can assist by automatically marking as wrong any attributes and relationships that involve entities that have themselves been marked wrong. The tutor can then continue and mark all remaining attributes and relationships. The tutor is then in a position to release the marks. For the student this means that they see another option (checkbox) on their screens which allows them to view the marked version of their diagram. Using this they can view their diagram now coloured in a combination of red, amber and green. This method of conveying the tutor's marking decisions was extremely well received by the students and led to lively, positive discussion of the principles involved with interpreting the scenarios which was very beneficial.

ISSN: 1473-7507

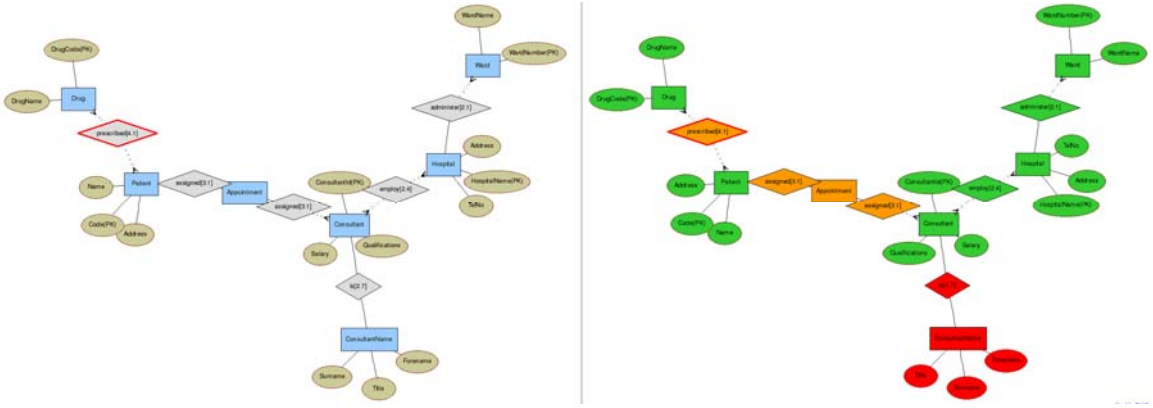

Figure 7: Screenshots of a student diagram, before and after marking

3.4 Quantity of Marking

Traditionally the marking of student diagrams would require the tutor to mark every element of every diagram separately. This method requires the tutor to mark every *different* element that appears across all the student diagrams. This produces a saving whenever two or more students have the same element on their diagram irrespective of whether or not it is correct. Some attempt to quantify this for the experience so far is given in a later section.

3.5 Repeated Use of Scenarios

When another class attempts a scenario that has already been marked then the existing marking is initially hidden from the students. The tutor will typically advertise when the marking is to take place and encourage the students to complete their work for this deadline. The tutor will then do what supplementary marking is required and notify the students that they can view the marked versions of their diagrams. The expectation is that after the first use of a scenario any subsequent use will only generate a trivial amount of extra marking.

4. Results and Experience

So far the system has been trialled with two scenarios by an MSc class of 20 and a first year class of 200 students. This represents 5 separate sessions since the first year class had to be split into four groups of 50 students. At the beginning of a 50min practical session a short demonstration of the system was given showing how to make one entity, one attribute and one relationship. The students were then asked to create ER diagrams for the first scenario (hitherto unseen) and if possible go on to do the same for the second scenario. By the end of the session most students had finished the first scenario and some had finished both.

The marking produced the efficiency gain in tutor time as predicted. For the first scenario which all students attempted, the students provided 10992 elements in total (including deletions) reducing to 5356 elements with deleted elements removed. Of these, 708 distinct elements (25 entities, 89 attributes and 594 relationships) required marking. Of the distinct elements, 468 (5, 46 & 417 respectively) were marked automatically by reference to the tutor solution, leaving 240 (20, 43 & 177) to be marked by the tutor. Leaving aside the tutor time needed to prepare the tutor solution, this suggests that the marking time has been reduced to 240/5356 i.e. less than 5% of its original duration.

A beneficial side effect of computer-based marking is that precise reports can be produced. For any chosen scenario the system produces a list of every distinct element, how it was marked and the number of students whose diagrams included that element. This reveals for example how many students made the 'same mistake'. So for example it was clear that something in the way the scenario was worded caused 50 students to wrongly identify "Consultant Name" as an entity and go on to make related mistakes with attributes. The next 'worst' mistake was only made by 18 students. The report also contains aggregate marks e.g. entities (green 82%, amber 7%, red 11%) attributes (green 82%, amber 10%, red 8%) and relationships (green 34%, amber 59%, red 7%) showing that it is the precise identification of relationships (amber 59%) that caused the most problems. The report also contains a breakdown of deleted items. This gives an overview of the backtracking by the students in the production of their final diagram.

The trials also caused us to think hard about the model for marking. Much VLE work goes into providing instant feedback to individual students from closed systems via multiple-choice questions, etc. This ER

diagram area is open-ended and thus requires individual marking, but to maintain efficiency of marking a tutor can only mark infrequently. So the ideally student effort should come in 'waves'.

In feedback sessions the students were able to ask questions about their marked diagrams. It was the elements marked in amber that caused the most discussion, but on reflection that was because the meaning of the amber colouring had not been properly explained to the class in advance. From a simple survey (6 questions about the editor and the associated marking feedback) the results from 70 returns show that the students were favourably disposed to the editor (particularly the drag-and-drop part) and they liked the coloured feedback. Only one student used the 'most dissatisfied' answer option.

We wish to make our software available to other universities and to this end we have been awarded a grant by the Higher Education Academy [13].

We hope to apply the ideas presented here to other diagramming systems. However there is a difference between moving to an alternative notation for capturing scenarios (which is relatively simple) and moving to a different diagramming system which has a totally different purpose. In section 2.1 it was pointed out that our style of editor deviates from the traditional pattern of providing a toolbox of all the different shapes required together with some provision for connecting the shapes. In the most recent version the editor must recognise the significance of different drop positions within the diagram in order to build different diagram components. Thus for a new diagramming system we would be committed to looking for a way to build a similar editor that understood the rules of the new system and minimised the effort required to create the diagram. So potentially the editor would have to be re-written for each new diagramming system, although the technology and the concepts would be transferred across.

5. CONCLUSION

The interfaces provided by our most recent editors have been found to be quick to learn and quick to use. However speed in itself is not the most important feature. We believe the most important feature is to allow and encourage the user to focus fully and continuously on the scenario and the deductions that can be made from it. The students have readily accepted the concept of using the editor as a way of submitting their work for scrutiny by the tutor in return for more extensive and personalised feedback than would have been possible otherwise. The marking has provided the tutor with the hoped-for benefits. Now that a basic system has been implemented where the components of the graph are directly referenced to the scenario, the next stage is to allow the user to create new elements by splitting and joining existing elements and thus producing elements which are only indirectly referenced to the original scenario.

6. REFERENCES

[1] Chen, P.P., The Entity-Relationship Model - Toward a Unified View of Data, *ACM Transactions on Database Systems* **1** (1), 9-36, (1976).

[2] Stone, R.G., Jasper - a Javascript Proof Editor, *Italics,* **6** (1), 2007.

[3] Wikipedia, ER Diagramming tools, list provided by Wikipedia, http://en.wikipedia.org/wiki/Entity-relationship_model [Accessed 10/6/09]

[4] JGraph, Java Graph Visualization and Layout, http://www.jgraph.com/ [Accessed 10/6/09]

[5] Graphviz, Open source graph drawing software, http://www.graphviz.org and particularly WebDot, a viewer for use in HTML documents http://www.graphviz.org/webdot/ [Accessed 10/6/09]

[6] Suraweera, P. & Mitrovic, A., KERMIT: A Constraint-Based Tutor for Database Modeling, *Proceedings of Intelligent Tutoring Systems : 6th International Conference*, ITS 2002, Biarritz, France and San Sebastian, Spain, June 2-7, 2002, [also in LNCS 2363, 377-387]

[7] Hall, L. & Gordon, A., A virtual learning environment for entity relationship modelling, *ACM SIGCSE Bulletin*, 30 (1), 345 - 349, (1998).

[8] Exerciser, http://mcs.open.ac.uk/Diagrams/html/exerciser.html [Accessed 10/6/09]

[9] Higgins, C.A., and Bligh B., (2006) Formative computer based assessment in diagram based domains, ACM SIGCSE Bulletin, Volume 38, Issue 3, Sept 2006, pp98-102

[10] Batmaz, F. & Hinde, C.J., A Web-Based Semi-Automatic Assessment Tool For Conceptual Database Diagram, *Proceedings of the sixth IASTED International Conference Web-Based Education*, Chamonix, France, 427-432 (2007).

[11] JQuery, The "write less, do more" *Javascript Library,* http://jquery.com/ with its user-interface library http://jqueryui.com/ [Accessed 10/6/09]

[12] Thomas, P. G., Smith, N., & Waugh, K. (2008), Automatically assessing graph-based diagrams. 'Learning, Media and Technology, Volume 33 Issue 3' pp 249-267

[13] Hinde, C.J., *A Web-Based Semi-Automatic Assessment Tool For Conceptual Database Model, HEA development* fund, 2008/09, http://www.ics.heacademy.ac.uk/projects/development-fund/fund_details.php?id=125 [Accessed 10/6/09]