A FRAMEWORK AND TOOLSET FOR THE DEVELOPMENT OF SOFTWARE TEACHING TOOLS

Richard Cooper
Computing Science
University of Glasgow
G12 8QQ
rich@dcs.gla.ac.uk

ABSTRACT

For the past few years we have been developing software which can be used by students learning about a variety of techniques of database use and internet programming. Examples include the use of ER modelling and normalisation for database design, the development of web services, the way in which HTTP and middleware programming underpins dynamic web sites and the use of XML programming. It has become clear that all of these applications use the same set of techniques - namely the mapping from one formulation to another, allowing the student to step through procedures, visualising the behaviour of complex process components and giving access to a hyper-linked glossary and a help system. This paper presents a proposal to isolate the techniques from the specific programs and to build a collection of abstract techniques which may be re-used in the construction of new programs, together with a framework and methodology for their use.

Keywords
Teaching Applications. Software Support Systems. Active Learning

1. INTRODUCTION

Introducing students to the range of concepts involved in software engineering is greatly facilitated by the availability of interactive software to allow the student to interact with the concepts. Software development revolves around the creation of successive descriptions of the software which range from highly abstract designs to precise implementations. A complete appreciation of the development process depends on an understanding both of the way the descriptions produce their effect and of the ways in which the descriptions are transformed into one another. Teaching is hampered because effects at the low level are not accessible to observation and the transformations must either be performed by the software engineer or are carried out automatically with no explanation. At the same time, many software systems are constructed as complex assemblages of components whose inter-relationship is not always easy for the student to understand.

As a consequence, attaining mastery over the software engineering process is hard to achieve. One solution is to involve the student in active learning (e.g. [1], [2]) by creating interactive software that gives the student access to that which is usually hidden, being irrelevant and indeed distracting to end users and software developers alike.

There are three particular uses for such software:

1. Allowing the student to observe the transformations being made facilitates understanding of those transformations. For instance, the process of transforming an ER diagram into an appropriate set of relational tables can be shown by program as a series of steps.
2. Providing interactive software which demonstrates what is happening at the lower levels to allow the student to try out concepts and techniques which can usually only be practiced off-line. For example, allowing the student to issue relational calculus or relational algebra queries and observe the results can give practical experience of these languages.
3. Providing a schematic view of a component-based system can show the processes that the system carries out and allow the student to query the state of the components at any time. For instance, a visualisation of the structure of user interaction with an internet site is used can be used to show the messages which are sent and to support querying the state of the browser, the server and the data source, as

Teaching, Learning & Assessment in Databases, Edinburgh 2004

well as HTTP objects such as cookies and request and response objects.

We have been developing a set of software tools to support the teaching of database and internet programming techniques, and these are further described in Section 2. In doing so, a common set of techniques has emerged. For instance, it is useful in showing the equivalence of two representations to display them both and to permit the student to select components of one and to see the equivalent component highlighted. Similarly, it is useful to permit the student to step through the execution of an executable description and observe the resulting effect on a data structure.

This paper identifies a set of such practices in Section 3 and proposes a set of Java-based abstractions which can be configured to produce further specific tools as Section 4. The paper ends with some closing observations.

2. SOFTWARE FOR TEACHING ABOUT DATABASE AND INTERNET PROGRAMMING CONCEPTS

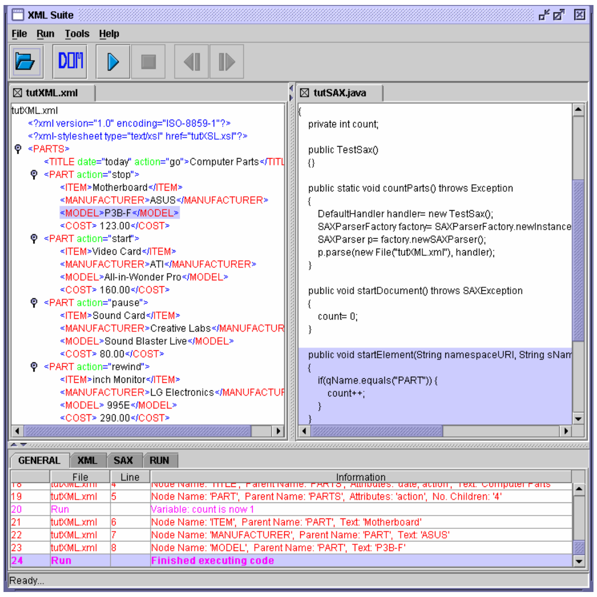
To illustrate the techniques which will be identified in the next section, a number of examples will be given. More examples are available in a previous paper [3], including the use of the HTTP protocol and server-side middleware to support dynamic web sites, the use of ER diagramming and normalisation to develop relational databases and the use of relational algebra and calculus to query relational databases. This section restricts itself to three further examples which each illustrate the use of different techniques.

2.1 Web Services

In order to demonstrate the development and use of web services, a program has been developed with following features:

- an animated diagram which shows how a web service is developed, deployed, located and bound into an application;
- staged development of a normal application, in which an application method calls a service method, into a web service in which the service method is now a web service and the application method is in an application which wishes to be bound to the service;
- visualisation of the code of both the application and the service which highlights the use of the SOAP communication protocol.

This application requires the staged animation of the service use, the staged development of code and the ability to identify one piece of code and highlight an equivalent section in another.



```
tutXML.xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <?xml-stylesheet type="text/xsl" href="tutXSL.xsl"?>
  <PARTS>
     <TITLE date="today" action="go">Computer Parts</TITL
     <PART action="stop">
        <ITEM>Motherboard</ITEM>
        <MANUFACTURER>ASUS</MANUFACTURER>
        <MODEL>P3B-F</MODEL>
        <COST> 123.00</COST>
     <PART action="start">
        <ITEM>Video Card</ITEM>
        <MANUFACTURER>ATI</MANUFACTURER>
        <MODEL>All-in-Wonder Pro</MODEL>
        <COST> 160.00</COST>
     <PART action="pause">
        <ITEM>Sound Card</ITEM>
        <MANUFACTURER>Creative Labs</MANUFACTUR
        <MODEL>Sound Blaster Live</MODEL>
        <COST> 80.00</COST>
     <PART action="rewind">
        <ITEM>inch Monitor</ITEM>
        <MANUFACTURER>LG Electronics</MANUFACTUR
        <MODEL> 995E</MODEL>
        <COST> 290.00</COST>
```

```
{
  private int count;

  public TestSax()
  {}

  public static void countParts() throws Exception
  {
    DefaultHandler handler= new TestSax();
    SAXParserFactory factory= SAXParserFactory.newInstance
    SAXParser p= factory.newSAXParser();
    p.parse(new File("tutXML.xml"), handler);
  }

  public void startDocument() throws SAXException
  {
    count= 0;
  }

  public void startElement(String namespaceURI, String sNam
  {
    if(qName.equals("PART")) {
      count++;
    }
  }
```

	File	Line	Information
18	tutXML.xml	4	Node Name: 'TITLE', Parent Name: 'PARTS', Attributes: 'date, action', Text: 'Computer Parts'
19	tutXML.xml	5	Node Name: 'PART', Parent Name: 'PARTS', Attributes: 'action', No. Children: '4'
20	Run		Variable: count is now 1
21	tutXML.xml	6	Node Name: 'ITEM', Parent Name: 'PART', Text: 'Motherboard'
22	tutXML.xml	7	Node Name: 'MANUFACTURER', Parent Name: 'PART', Text: 'ASUS'
23	tutXML.xml	8	Node Name: 'MODEL', Parent Name: 'PART', Text: 'P3B-F'
24	**Run**		**Finished executing code**

Figure 1. XML Programming Tutor

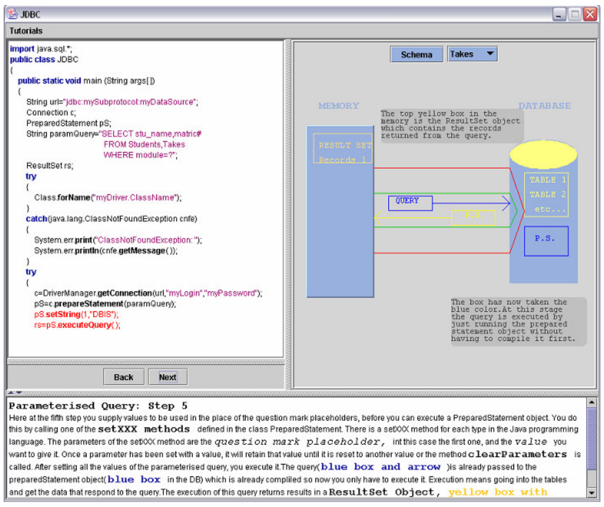
2.2 XML Programming

Several programming systems have been proposed for managing XML programs including DOM, SAX and XSLT. To demonstrate these, an application (Shown in Figure 1) has been developed which displays an XML file and a program file. The student can step through the program and in doing so, the current XML entity is displayed and a commentary is displayed in a panel at the bottom of the screen.

This program depends on the ability to highlight fragments of a piece of text (shown in purple in the figure) and to show successive program statements as they are executed (e.g. the SAX statements in the panel on the left. The program is designed to work with a variety of XML files and programs and must capture the equivalence of data and program statements acting on the data. This program also has a feedback panel shown at the bottom describing what is happening in each step. The steps are controlled by the cassette player style buttons in the toolbar.

2.3 JDBC Programming

This application shows how a Java program uses JDBC to interact with relational data. The program works by displaying the database in one panel and the program in another. As the successive steps unfold, program statements are added and a graphical representation of the connection between the code and the database appears which illustrates the various program components. The diagram is queryable by selecting a component in order to display its state.



```
import java.sql.*;
public class JDBC
{
   public static void main (String args[])
   {
      String url="jdbc:mySubprotocol:myDataSource";
      Connection c;
      PreparedStatement pS;
      String paramQuery="SELECT stu_name,matric#
                         FROM Students,Takes
                         WHERE module=?";
      ResultSet rs;
      try
      {
         Class.forName("myDriver.ClassName");
      }
      catch(java.lang.ClassNotFoundException cnfe)
      {
         System.err.print("ClassNotFoundException: ");
         System.err.println(cnfe.getMessage());
      }
      try
      {
         c=DriverManager.getConnection(url,"myLogin","myPassword");
         pS=c.prepareStatement(paramQuery);
         pS.setString(1,"DBIS");
         rs=pS.executeQuery();
```

The top yellow box in the memory is the ResultSet object which contains the records returned from the query.

The box has now taken the blue color.At this stage the query is executed by just running the prepared statement object without having to compile it first.

Parameterised Query: Step 5

Here at the fifth step you supply values to be used in the place of the question mark placeholders, before you can execute a PreparedStatement object. You do this by calling one of the `setXXX methods` defined in the class PreparedStatement. There is a setXXX method for each type in the Java programming language. The parameters of the setXXX method are the *question mark placeholder,* int this case the first one, and the *value* you want to give it. Once a parameter has been set with a value, it will retain that value until it is reset to another value or the method `clearParameters` is called. After setting all the values of the parameterised query, you execute it.The query(`blue box and arrow`)is already passed to the preparedStatement object(`blue box` in the DB) which is already compliled so now you only have to execute it. Execution means going into the tables and get the data that respond to the query.The execution of this query returns results in a `ResultSet Object,` `yellow box with`

Figure 2. JDBC Programming Tutor

This application demonstrates the gradual development of a program, a queryable diagram, a feedback panel and staged execution.

Of course, each of these would also benefit from an integrating help and glossary system, which is at best available in a rudimentary form at present.

3. A SET OF COMPONENTS FOR DEVELOPING TEACHING TOOLS

In this section, a number of techniques are identified and isolated, so that they can be re-implemented in an abstract and configurable form. In this section, it is assumed that there is an underlying data structure and program which must make use of it. The section starts with the static components and then moves on to the dynamic components.

3.1 The Components Required

Interactive Text Panels. A program or textual data representation is a piece of structured text. Thus a Java program or the description of a database is a text document in which individual fragments such as statements or table names need to be identified clearly, so that they can be selected and visualised in different ways. Fortunately Java has the Swing text framework to assist with the visualisation of such documents.

Interactive Diagrams. Similarly, a diagrammatic representation is required for database design or the display of networks. In the same way as the texts mentioned above, it will be necessary to allow the user to select diagram components and for the teaching tool to highlight them. This means we need software which matches each kind of icon and line with software which manages the appearance and which reacts to selection.

Correspondence Structures. The process of illustrating the relationship between two representations must be supported by an underlying structure in which the fragments of one representation are linked to the equivalent fragments of others.

Staged Execution. In order to permit the student to explore the effects of a dynamic process, it is useful for that process to be executed in stages. There are two ways in which the execution could be controlled – manually or automatically. The component here will permit either the program designer or the student to select between the two. If automatic execution is selected, there will be a facility for controlling the speed. If manual execution is chosen, a panel of forward and back buttons, similar to those found in a cassette recorder, appear for use. The buttons are tied to a collection of do and undo actions. Specific versions will also be available - one of which displays the whole of the executed code from the outset (as in the XML program) or adds to it to demonstrate progress (as in the JDBC example).

Feedback Panel. A feedback panel is just an example of the structured text panel, to which text can be added.

Hyperlinked Glossaries. As well as a help system, these programs benefit greatly by the addition of a hyperlinked glossary. Each important concept of the technique should be added to the glossary so that any time an icon is added to a diagram or text is added to a text panel, if it illustrates a glossary entry, it will be made into a hyperlink. The module which builds up structured texts manages this.

3.2 Interfaces for a Java Framework

All of the programs which can be developed using our proposed system work by controlling a set of document descriptions visible through a number of interacting windows. Each document consists of fragments which may be selected, highlighted, hidden or revealed, etc. Much of the implem-entation of the following designs is simplified by the existence of the wide range of Java packages, notably Swing and the Swing text framework. The Java Teaching Tool Framework consists of an interface structure described in this.

Programs are built by implementing the following interfaces (standard implementations will be provided as is usual with Java):

Fragment. A fragment is a component of a document which has a distinctive existence – i.e. a fragment can be selected, have the style of its presentation changed, can be hidden or revealed, or can be associated with a set of other actions. There are sub-interfaces for textual fragments and iconic fragments, the latter being the elements of a diagram. Fragments have a kind (text or icon in the first instance) and a type. The type indicates the role the fragment plays in the document – e.g. a textual fragment may be a keyword, while an icon may represent an entity type in an ER diagram. Fragments may be nested – e.g. lexemes in a program can be grouped into lines, expressions, methods and so on, while an entity type and all of its attributes may be a useful grouping.

Document. A document is a collection of fragments. It supports methods to add and remove fragments and also operations over all fragments of a particular type. There are specialised forms of *Document* which are hypertext documents (appearing in a browser window) and feedback documents (essentially documents which can be replaced by a new piece of text, or to which text can be appended).

Parser. A parser manages a document ensuring that it accords to the syntax of a formal language. There are two kinds of parser. A static parser takes a file in the formal language and turns it into a document, while an interactive parser permits the user to enter and edit the document. A diagrammatic static parser permits the input of a completed diagram from a textual description, while a textual static parse creates a textual document with fragments identified for code statements, keywords, constants and variables, together with any other fragment types specific to the application – e.g. for SQL, table and column names. A diagrammatic interactive parser permits the input and editing of the diagram components in a drawing window, while a textual interactive parser provides a command line interface for a language (for instance, SQL). Interactive parsers provide a method to validate the entry – syntax direct control of document entry has not been included.

Correspondence. A correspondence links the fragments of two or more documents, so that mappings can be revealed. Correspondences support operations which permit the selection of one fragment to cause an action on a corresponding fragment, such as altering its appearance. Among the methods provided are search operations for corresponding elements, methods which highlight corresponding fragments and so on.

Staged Execution. This is a process which consists of a series of actions which are controlled by a panel with forward and back buttons. Each action makes some change to the windows on display.

Action. An action is the description of a change made to a document or which bring up or remove a frame or a panel. Actions can be part of staged executions or be tied to user actions on fragments or through the use of menus and toolbars.

Style. A component will be available to manage style. The font of textual fragments, colours for distinguishing kinds of data or for highlighting items. As shown rather garishly in Figures 2 and 3), line styles and icons for the diagram and overall layout will be managed by a controlling module. Java's *Attribute Set* forms the basis of this.

Panels and Frames. A document appears in a frame or a panel inside a frame. Designing the structure of these is the first step to creating the application.

Framework. A framework is a collection of panels and frames which constitute the interface to the application. By separating off a framework interface, it will be possible to reuse previous frameworks for new applications.

4. A PROGRAMMING METHODOLOGY FOR DEVELOPING TEACHING TOOLS

This section describes a methodology which can be described for developing a teaching tool.

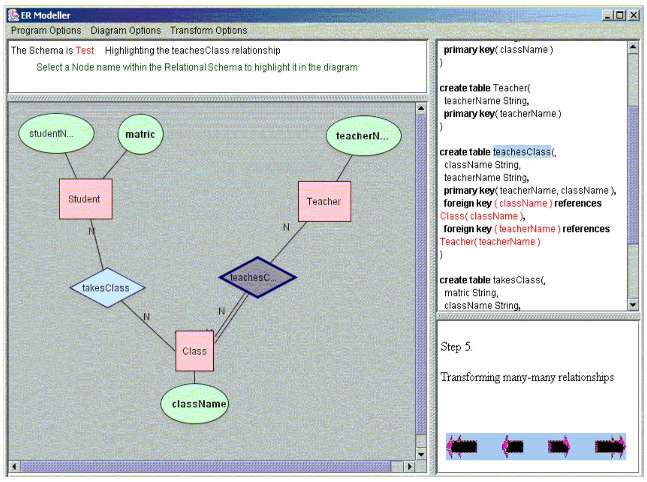
4.1 A Programming Methodology

The development of an interactive piece of teaching software can now be described as a series of implementation tasks.

1. Firstly, an overall interface must be set up in terms of a framework, i.e. a series of panels, either as separate frames or grouped within a single frame.
2. The use of each panel must be identified. It may be a browser, feedback panel or represent one of the document types being used in the application.
3. A parser for each language involved must be developed as implementations of the Parser interface. These will allow documents to be read in or create the interactive editing panels required.
4. Correspondences between each of the documents on display must be established in order to support the interaction.
5. The set of actions to be supported on the documents must be created.
6. The set of menus and toolbars must be specified as calls to these actions.
7. Staged execution as a series of calls to actions must similarly be set up.

4.2 Example

The example developed here is the creation of a program to illustrate how ER diagrams are used to design relational databases. The application permits ER diagrams to be constructed in one window, while another window is available for the display of the equivalent SQL *create table* statements. The application provides menu options to verify the structural correctness of the diagram and then permits the SQL to be created one phase at a time – entity types first, attributes next and so on [3]. The example requires five panels: a diagram drawing panel; a panel containing the SQL statements to create the database; a feedback window; a staged execution panel; and a hypertext browser. The placement of these must be decided – for instance, the feedback might be placed at the top of a window which also contains the ER diagram panel and the SQL panel on the left and right and the staged execution panel at the bottom. The hypertext browser could be a separate window which appears on demand. The main window is shown as Figure 3.



The Schema is Test   Highlighting the teachesClass relationship

Select a Node name within the Relational Schema to highlight it in the diagram

```
 primary key( className )
)

create table Teacher(
 teacherName String,
 primary key( teacherName )
)

create table teachesClass(,
 className String,
 teacherName String,
 primary key( teacherName, className ),
 foreign key ( className ) references
Class( className ),
 foreign key ( teacherName ) references
Teacher( teacherName )
)

create table takesClass(,
 matric String,
 className String,
```

Step 5.

Transforming many-many relationships

Figure 3. An Example Application

The first step is to construct the framework with the four panels shown and the browser (not shown). The document classes for SQL statements and ER diagrams are created together with a correspondence between the two. Then an interactive diagram parser is created which manages the creation and editing of ER diagrams. The feedback panel is shown as it is used when the correspondence is being demonstrated to locate and highlight the SQL component of an ER fragment. The verification option calls the validation method of the parser. The transformation option brings up the phased execution panel shown bottom left. As the phases are requested, a description of what is happening is produced inside the panel. During the highlighting option, at any time, a diagram component or an SQL variable can be selected. Further information is provided by permitting the selection of a diagram component or an SQL keyword to result in explanatory hypertext appearing in the browser panel.

5. Conclusions

The paper has presented a plan for a toolset and accompanying methodology for developing interacting software teaching tools. The tools allow for the highlighted display of structured texts and diagrams, for the highlighting of correspondences, for the staged execution of processes and for the integration of help and glossary systems.

Work on development of the toolset has just started, but it seems likely that starting to create a new application given the framework described would greatly accelerate the development of applications such as these.

Acknowledgements

Thanks are due to James Macrae and a large number of students who have worked on versions of these tools. The author would also like to thank Karen Renaud for commenting on a draft of this paper.

References

[1] Modell, H.I. & Michael J.A. (1993). Promoting active learning in the life science classroom. New York: New York Academy of Sciences

[2] Kolb, D., Experiential learning: experience as the source of learning and development. London. Prentice Hall (1984).

[3] Cooper, R.L. and Macrae, J. Software Systems to Support the Teaching of the Use of Relational Database Systems, Teaching, Learning and Assessment of Databases, O'Reilly, James and Jackson (eds), 2003.

A FRAMEWORK AND TOOLSET FOR THE DEVELOPMENT OF SOFTWARE TEACHING TOOLS

Richard Cooper
Computing Science
University of Glasgow
G12 8QQ
rich@dcs.gla.ac.uk

ABSTRACT

For the past few years we have been developing software which can be used by students learning about a variety of techniques of database use and internet programming. Examples include the use of ER modelling and normalisation for database design, the development of web services, the way in which HTTP and middleware programming underpins dynamic web sites and the use of XML programming. It has become clear that all of these applications use the same set of techniques - namely the mapping from one formulation to another, allowing the student to step through procedures, visualising the behaviour of complex process components and giving access to a hyper-linked glossary and a help system. This paper presents a proposal to isolate the techniques from the specific programs and to build a collection of abstract techniques which may be re-used in the construction of new programs, together with a framework and methodology for their use.

Keywords
Teaching Applications. Software Support Systems. Active Learning

1. INTRODUCTION

Introducing students to the range of concepts involved in software engineering is greatly facilitated by the availability of interactive software to allow the student to interact with the concepts. Software development revolves around the creation of successive descriptions of the software which range from highly abstract designs to precise implementations. A complete appreciation of the development process depends on an understanding both of the way the descriptions produce their effect and of the ways in which the descriptions are transformed into one another. Teaching is hampered because effects at the low level are not accessible to observation and the transformations must either be performed by the software engineer or are carried out automatically with no explanation. At the same time, many software systems are constructed as complex assemblages of components whose inter-relationship is not always easy for the student to understand.

As a consequence, attaining mastery over the software engineering process is hard to achieve. One solution is to involve the student in active learning (e.g. [1], [2]) by creating interactive software that gives the student access to that which is usually hidden, being irrelevant and indeed distracting to end users and software developers alike.

There are three particular uses for such software:

1. Allowing the student to observe the transformations being made facilitates understanding of those transformations. For instance, the process of transforming an ER diagram into an appropriate set of relational tables can be shown by program as a series of steps.
2. Providing interactive software which demonstrates what is happening at the lower levels to allow the student to try out concepts and techniques which can usually only be practiced off-line. For example, allowing the student to issue relational calculus or relational algebra queries and observe the results can give practical experience of these languages.
3. Providing a schematic view of a component-based system can show the processes that the system carries out and allow the student to query the state of the components at any time. For instance, a visualisation of the structure of user interaction with an internet site is used can be used to show the messages which are sent and to support querying the state of the browser, the server and the data source, as

Teaching, Learning & Assessment in Databases, Edinburgh 2004

well as HTTP objects such as cookies and request and response objects.

We have been developing a set of software tools to support the teaching of database and internet programming techniques, and these are further described in Section 2. In doing so, a common set of techniques has emerged. For instance, it is useful in showing the equivalence of two representations to display them both and to permit the student to select components of one and to see the equivalent component highlighted. Similarly, it is useful to permit the student to step through the execution of an executable description and observe the resulting effect on a data structure.

This paper identifies a set of such practices in Section 3 and proposes a set of Java-based abstractions which can be configured to produce further specific tools as Section 4. The paper ends with some closing observations.

2. Software for Teaching about Database and Internet Programming Concepts

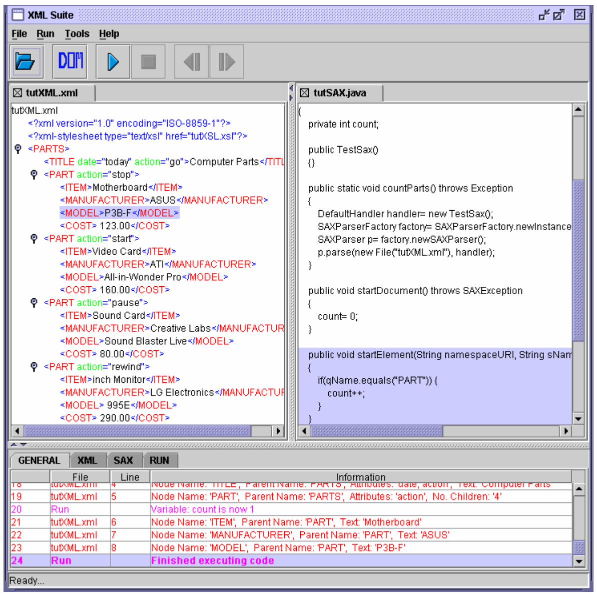
To illustrate the techniques which will be identified in the next section, a number of examples will be given. More examples are available in a previous paper [3], including the use of the HTTP protocol and server-side middleware to support dynamic web sites, the use of ER diagramming and normalisation to develop relational databases and the use of relational algebra and calculus to query relational databases. This section restricts itself to three further examples which each illustrate the use of different techniques.

2.1 Web Services

In order to demonstrate the development and use of web services, a program has been developed with following features:

- an animated diagram which shows how a web service is developed, deployed, located and bound into an application;
- staged development of a normal application, in which an application method calls a service method, into a web service in which the service method is now a web service and the application method is in an application which wishes to be bound to the service;
- visualisation of the code of both the application and the service which highlights the use of the SOAP communication protocol.

This application requires the staged animation of the service use, the staged development of code and the ability to identify one piece of code and highlight an equivalent section in another.



```
tutXML.xml
    <?xml version="1.0" encoding="ISO-8859-1"?>
    <?xml-stylesheet type="text/xsl" href="tutXSL.xsl"?>
    <PARTS>
        <TITLE date="today" action="go">Computer Parts</TITL
        <PART action="stop">
            <ITEM>Motherboard</ITEM>
            <MANUFACTURER>ASUS</MANUFACTURER>
            <MODEL>P3B-F</MODEL>
            <COST> 123.00</COST>
        <PART action="start">
            <ITEM>Video Card</ITEM>
            <MANUFACTURER>ATI</MANUFACTURER>
            <MODEL>All-in-Wonder Pro</MODEL>
            <COST> 160.00</COST>
        <PART action="pause">
            <ITEM>Sound Card</ITEM>
            <MANUFACTURER>Creative Labs</MANUFACTUR
            <MODEL>Sound Blaster Live</MODEL>
            <COST> 80.00</COST>
        <PART action="rewind">
            <ITEM>inch Monitor</ITEM>
            <MANUFACTURER>LG Electronics</MANUFACTUF
            <MODEL> 995E</MODEL>
            <COST> 290.00</COST>
```

```
{
  private int count;

  public TestSax()
  {}

  public static void countParts() throws Exception
  {
    DefaultHandler handler= new TestSax();
    SAXParserFactory factory= SAXParserFactory.newInstance
    SAXParser p= factory.newSAXParser();
    p.parse(new File("tutXML.xml"), handler);
  }

  public void startDocument() throws SAXException
  {
    count= 0;
  }

  public void startElement(String namespaceURI, String sNam
  {
    if(qName.equals("PART")) {
      count++;
    }
  }
```

```
    File         Line  Information
18  tutXML.xml   4     Node Name: 'TITLE', Parent Name: 'PARTS', Attributes: 'date, action', Text: 'Computer Parts'
19  tutXML.xml   5     Node Name: 'PART', Parent Name: 'PARTS', Attributes: 'action', No. Children: '4'
20  Run                Variable: count is now 1
21  tutXML.xml   6     Node Name: 'ITEM', Parent Name: 'PART', Text: 'Motherboard'
22  tutXML.xml   7     Node Name: 'MANUFACTURER', Parent Name: 'PART', Text: 'ASUS'
23  tutXML.xml   8     Node Name: 'MODEL', Parent Name: 'PART', Text: 'P3B-F'
24  Run                Finished executing code
Ready...
```

Figure 1. XML Programming Tutor

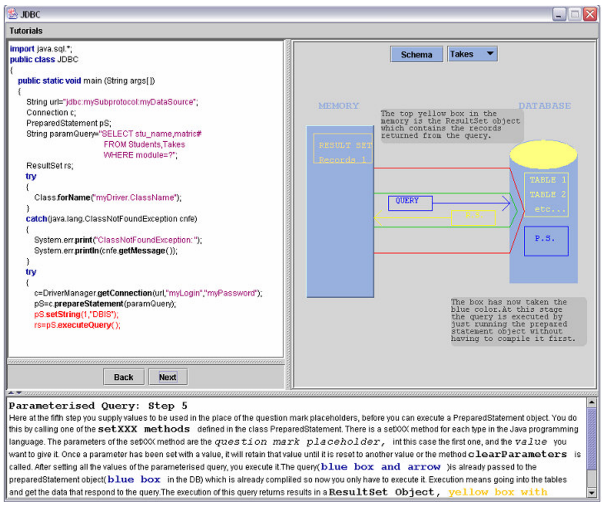
2.2 XML Programming

Several programming systems have been proposed for managing XML programs including DOM, SAX and XSLT. To demonstrate these, an application (Shown in Figure 1) has been developed which displays an XML file and a program file. The student can step through the program and in doing so, the current XML entity is displayed and a commentary is displayed in a panel at the bottom of the screen.

This program depends on the ability to highlight fragments of a piece of text (shown in purple in the figure) and to show successive program statements as they are executed (e.g. the SAX statements in the panel on the left. The program is designed to work with a variety of XML files and programs and must capture the equivalence of data and program statements acting on the data. This program also has a feedback panel shown at the bottom describing what is happening in each step. The steps are controlled by the cassette player style buttons in the toolbar.

2.3 JDBC Programming

This application shows how a Java program uses JDBC to interact with relational data. The program works by displaying the database in one panel and the program in another. As the successive steps unfold, program statements are added and a graphical representation of the connection between the code and the database appears which illustrates the various program components. The diagram is queryable by selecting a component in order to display its state.



```
import java.sql.*;
public class JDBC
{
   public static void main (String args[])
   {
      String url="jdbc:mySubprotocol:myDataSource";
      Connection c;
      PreparedStatement pS;
      String paramQuery="SELECT stu_name,matric#
                           FROM Students,Takes
                           WHERE module=?";
      ResultSet rs;
      try
      {
         Class.forName("myDriver.ClassName");
      }
      catch(java.lang.ClassNotFoundException cnfe)
      {
         System.err.print("ClassNotFoundException: ");
         System.err.println(cnfe.getMessage());
      }
      try
      {
         c=DriverManager.getConnection(url,"myLogin","myPassword");
         pS=c.prepareStatement(paramQuery);
         pS.setString(1,"DBIS");
         rs=pS.executeQuery();
```

The top yellow box in the memory is the ResultSet object which contains the records returned from the query.

The box has now taken the blue color.At this stage the query is executed by just running the prepared statement object without having to compile it first.

Parameterised Query: Step 5
Here at the fifth step you supply values to be used in the place of the question mark placeholders, before you can execute a PreparedStatement object. You do this by calling one of the **setXXX methods** defined in the class PreparedStatement. There is a setXXX method for each type in the Java programming language. The parameters of the setXXX method are the *question mark placeholder,* int this case the first one, and the *value* you want to give it. Once a parameter has been set with a value, it will retain that value until it is reset to another value or the method **clearParameters** is called. After setting all the values of the parameterised query, you execute it.The query(**blue box and arrow**)is already passed to the preparedStatement object(**blue box** in the DB) which is already compliled so now you only have to execute it. Execution means going into the tables and get the data that respond to the query.The execution of this query returns results in a **ResultSet Object,** yellow box with

Figure 2. JDBC Programming Tutor

This application demonstrates the gradual development of a program, a queryable diagram, a feedback panel and staged execution.

Of course, each of these would also benefit from an integrating help and glossary system, which is at best available in a rudimentary form at present.

3. A SET OF COMPONENTS FOR DEVELOPING TEACHING TOOLS

In this section, a number of techniques are identified and isolated, so that they can be re-implemented in an abstract and configurable form. In this section, it is assumed that there is an underlying data structure and program which must make use of it. The section starts with the static components and then moves on to the dynamic components.

3.1 The Components Required

Interactive Text Panels. A program or textual data representation is a piece of structured text. Thus a Java program or the description of a database is a text document in which individual fragments such as statements or table names need to be identified clearly, so that they can be selected and visualised in different ways. Fortunately Java has the Swing text framework to assist with the visualisation of such documents.

Interactive Diagrams. Similarly, a diagrammatic representation is required for database design or the display of networks. In the same way as the texts mentioned above, it will be necessary to allow the user to select diagram components and for the teaching tool to highlight them. This means we need software which matches each kind of icon and line with software which manages the appearance and which reacts to selection.

Correspondence Structures. The process of illustrating the relationship between two representations must be supported by an underlying structure in which the fragments of one representation are linked to the equivalent fragments of others.

Staged Execution. In order to permit the student to explore the effects of a dynamic process, it is useful for that process to be executed in stages. There are two ways in which the execution could be controlled – manually or automatically. The component here will permit either the program designer or the student to select between the two. If automatic execution is selected, there will be a facility for controlling the speed. If manual execution is chosen, a panel of forward and back buttons, similar to those found in a cassette recorder, appear for use. The buttons are tied to a collection of do and undo actions. Specific versions will also be available - one of which displays the whole of the executed code from the outset (as in the XML program) or adds to it to demonstrate progress (as in the JDBC example).

Feedback Panel. A feedback panel is just an example of the structured text panel, to which text can be added.

Hyperlinked Glossaries. As well as a help system, these programs benefit greatly by the addition of a hyperlinked glossary. Each important concept of the technique should be added to the glossary so that any time an icon is added to a diagram or text is added to a text panel, if it illustrates a glossary entry, it will be made into a hyperlink. The module which builds up structured texts manages this.

3.2 Interfaces for a Java Framework

All of the programs which can be developed using our proposed system work by controlling a set of document descriptions visible through a number of interacting windows. Each document consists of fragments which may be selected, highlighted, hidden or revealed, etc. Much of the implem-entation of the following designs is simplified by the existence of the wide range of Java packages, notably Swing and the Swing text framework. The Java Teaching Tool Framework consists of an interface structure described in this.

Programs are built by implementing the following interfaces (standard implementations will be provided as is usual with Java):

Fragment. A fragment is a component of a document which has a distinctive existence – i.e. a fragment can be selected, have the style of its presentation changed, can be hidden or revealed, or can be associated with a set of other actions. There are sub-interfaces for textual fragments and iconic fragments, the latter being the elements of a diagram. Fragments have a kind (text or icon in the first instance) and a type. The type indicates the role the fragment plays in the document – e.g. a textual fragment may be a keyword, while an icon may represent an entity type in an ER diagram. Fragments may be nested – e.g. lexemes in a program can be grouped into lines, expressions, methods and so on, while an entity type and all of its attributes may be a useful grouping.

Document. A document is a collection of fragments. It supports methods to add and remove fragments and also operations over all fragments of a particular type. There are specialised forms of *Document* which are hypertext documents (appearing in a browser window) and feedback documents (essentially documents which can be replaced by a new piece of text, or to which text can be appended).

Parser. A parser manages a document ensuring that it accords to the syntax of a formal language. There are two kinds of parser. A static parser takes a file in the formal language and turns it into a document, while an interactive parser permits the user to enter and edit the document. A diagrammatic static parser permits the input of a completed diagram from a textual description, while a textual static parse creates a textual document with fragments identified for code statements, keywords, constants and variables, together with any other fragment types specific to the application – e.g. for SQL, table and column names. A diagrammatic interactive parser permits the input and editing of the diagram components in a drawing window, while a textual interactive parser provides a command line interface for a language (for instance, SQL). Interactive parsers provide a method to validate the entry – syntax direct control of document entry has not been included.

Correspondence. A correspondence links the fragments of two or more documents, so that mappings can be revealed. Correspondences support operations which permit the selection of one fragment to cause an action on a corresponding fragment, such as altering its appearance. Among the methods provided are search operations for corresponding elements, methods which highlight corresponding fragments and so on.

Staged Execution. This is a process which consists of a series of actions which are controlled by a panel with forward and back buttons. Each action makes some change to the windows on display.

Action. An action is the description of a change made to a document or which bring up or remove a frame or a panel. Actions can be part of staged executions or be tied to user actions on fragments or through the use of menus and toolbars.

Style. A component will be available to manage style. The font of textual fragments, colours for distinguishing kinds of data or for highlighting items. As shown rather garishly in Figures 2 and 3), line styles and icons for the diagram and overall layout will be managed by a controlling module. Java's *Attribute Set* forms the basis of this.

Panels and Frames. A document appears in a frame or a panel inside a frame. Designing the structure of these is the first step to creating the application.

Framework. A framework is a collection of panels and frames which constitute the interface to the application. By separating off a framework interface, it will be possible to reuse previous frameworks for new applications.

4. A PROGRAMMING METHODOLOGY FOR DEVELOPING TEACHING TOOLS

This section describes a methodology which can be described for developing a teaching tool.

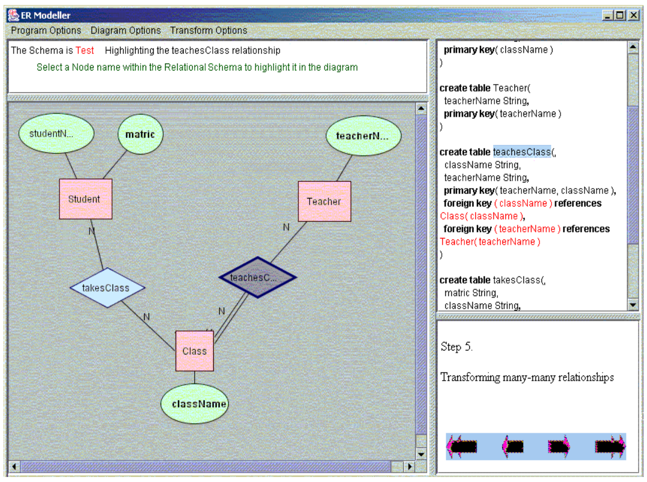
4.1 A Programming Methodology

The development of an interactive piece of teaching software can now be described as a series of implementation tasks.

1. Firstly, an overall interface must be set up in terms of a framework, i.e. a series of panels, either as separate frames or grouped within a single frame.
2. The use of each panel must be identified. It may be a browser, feedback panel or represent one of the document types being used in the application.
3. A parser for each language involved must be developed as implementations of the Parser interface. These will allow documents to be read in or create the interactive editing panels required.
4. Correspondences between each of the documents on display must be established in order to support the interaction.
5. The set of actions to be supported on the documents must be created.
6. The set of menus and toolbars must be specified as calls to these actions.
7. Staged execution as a series of calls to actions must similarly be set up.

4.2 Example

The example developed here is the creation of a program to illustrate how ER diagrams are used to design relational databases. The application permits ER diagrams to be constructed in one window, while another window is available for the display of the equivalent SQL *create table* statements. The application provides menu options to verify the structural correctness of the diagram and then permits the SQL to be created one phase at a time – entity types first, attributes next and so on [3]. The example requires five panels: a diagram drawing panel; a panel containing the SQL statements to create the database; a feedback window; a staged execution panel; and a hypertext browser. The placement of these must be decided – for instance, the feedback might be placed at the top of a window which also contains the ER diagram panel and the SQL panel on the left and right and the staged execution panel at the bottom. The hypertext browser could be a separate window which appears on demand. The main window is shown as Figure 3.



The Schema is Test   Highlighting the teachesClass relationship

Select a Node name within the Relational Schema to highlight it in the diagram

```
 primary key( className )
)

create table Teacher(
 teacherName String,
 primary key( teacherName )
)

create table teachesClass(,
 className String,
 teacherName String,
 primary key( teacherName, className ),
 foreign key ( className ) references
Class( className ),
 foreign key ( teacherName ) references
Teacher( teacherName )
)

create table takesClass(,
 matric String,
 className String,
```

Step 5.

Transforming many-many relationships

Figure 3. An Example Application

The first step is to construct the framework with the four panels shown and the browser (not shown). The document classes for SQL statements and ER diagrams are created together with a correspondence between the two. Then an interactive diagram parser is created which manages the creation and editing of ER diagrams. The feedback panel is shown as it is used when the correspondence is being demonstrated to locate and highlight the SQL component of an ER fragment. The verification option calls the validation method of the parser. The transformation option brings up the phased execution panel shown bottom left. As the phases are requested, a description of what is happening is produced inside the panel. During the highlighting option, at any time, a diagram component or an SQL variable can be selected. Further information is provided by permitting the selection of a diagram component or an SQL keyword to result in explanatory hypertext appearing in the browser panel.

5. Conclusions

The paper has presented a plan for a toolset and accompanying methodology for developing interacting software teaching tools. The tools allow for the highlighted display of structured texts and diagrams, for the highlighting of correspondences, for the staged execution of processes and for the integration of help and glossary systems.

Work on development of the toolset has just started, but it seems likely that starting to create a new application given the framework described would greatly accelerate the development of applications such as these.

Acknowledgements

Thanks are due to James Macrae and a large number of students who have worked on versions of these tools. The author would also like to thank Karen Renaud for commenting on a draft of this paper.

References

[1] Modell, H.I. & Michael J.A. (1993). Promoting active learning in the life science classroom. New York: New York Academy of Sciences

[2] Kolb, D., Experiential learning: experience as the source of learning and development. London. Prentice Hall (1984).

[3] Cooper, R.L. and Macrae, J. Software Systems to Support the Teaching of the Use of Relational Database Systems, Teaching, Learning and Assessment of Databases, O'Reilly, James and Jackson (eds), 2003.