An Efficient Information System Generator

Ling-Hua Chang[1] and Sanjiv Behl[2]

[1] Kun Shan University of Technology, Department of Information Management
No. 949, Da Wan Rd., Tainan City, Taiwan, R.O.C.
[2] Thomas Edison State College,
101 W. State St, Trenton, NJ 08608-1176
changlh@mail.ksu.edu.tw, sanbehl@yahoo.com

Abstract. We developed a new customized software tool for automatically generating a complete Java program based on the values or parameters inputted by the user. We call it an efficient Information System Generator or ISG for short. It is efficient in terms of the processor usage and the development time. We illustrate how it can be used by building a system for keeping track of student's scores that can be used by any faculty member who teaches multiple courses at a university or a college. It can also be used for generating e-commerce web sites.

Keywords: Java GUI Generator, prototype, Information System Generator, E-commerce Generator.

1 Introduction

We know that building a software system is a very time-consuming process and therefore we designed a customized software tool to help people generate any information (software) system instead of writing it themselves. We call this an Efficient Information System Generator or ISG for short. For example, we can use ISG to generate a conference system generator, which can be used to regenerate any conference system for any company or institute after they understand the context of the whole conference. ISG can also be used to generate e-commerce web sites, for instance the following website can be generated using ISG - http://www.36086789.com/category-2-b0.html?=1013yyyy .

The system designers need to provide the graphical user interface and input data to the ISG. The GUI interface includes how the input is obtained and the output is displayed, as well as the relationship between the two. Once this information is inputted, the ISG tool will convert it to the necessary Java code and database.

2 Related Work

Microsoft Excel [1] is a commercial spreadsheet application written and distributed by Microsoft for Windows and Mac OS X. It has a "*smart recalculation*" feature where when the cell data changes, only the data associated with it will be updated, and the user

J.-S. Pan, S.-M. Chen, N.T. Nguyen (Eds.): ACIIDS 2012, Part III, LNAI 7198, pp. 286–297, 2012.

can immediately see the changes resulting from the change in the cell data. It also has powerful graphics capabilities, so the users can see when the graphic data changes. We hope that in the future ISG would be as simple to use as Excel is today.

There are many Java generators currently on the market like JFlex[2], BOUML[3], Javadoc [4] and JAG[5]. JFlex is a lexical analyzer generator, BOUML generates the code based on the definition made at the artifact, class, operation, relation, attribute and extra member levels. Javadoc is the Java API Documentation and generates HTML pages of API documentation from Java source files. JAG (Java Application Generator) is an excellent tool and uses open source Apache Ant, a Java library and a command-line tool. The system design provides the database, object, window frame and files information, which can be generated to an information system that is built on the J2EE platform. However we think that it is too difficult to use for most small and medium enterprises.

UJECTOR [6] is a tool for creating executable code from UML Models. It uses UML class, sequence and activity diagrams for automated code generation. It generates structural code from the class diagram, and then adds behavioral aspects from the sequence and activity diagrams. In the rule based production systems for automatically generating Java code (or RPSAGJ) [7], the user writes the requirements in simple English and the designed system is able to extract associated information after compound analysis, which is then used to draw various UML diagrams as activity, sequence, class and uses cases diagrams. The designed system has a robust ability to create code automatically without external environment.

ISG is also a development tool for generating an information system which incorporates the advantages of Excel and JAG and also introduces a prototyping output design. We illustrate how our tool can be used to design and prototype the input and the output layout for the system users and programmers. UJECTOR and RPSAGJ are consistent with the UML models, however, the current version of ISG creates colorful, attractive and user-friendly screens which can be used by an information system or for generating any e-commerce web system. At present we are using ISG to develop a business information system for East Land International Company. The screens provided by UJECTOR and RPSAGJ are not very convenient or user-friendly for entering data as they are in ISG. We are looking into incorporating their features in the future versions of ISG.

3 ISG System Architecture

Since the *ISG i*nformation system is a Java GUI tool, the user must provide system analysis and system design information, including the panel design and its functionalities, the relationship between the panels and where the data storage takes place. ISG offers users interface screens which can be used for generating an information system and has seven transformation functions - for building a database, for building files, for linking to the next window, for building data processing window, for displaying data, for previewing a designed window and for printing data. We now discuss each transformation function individually.

3.1 Building a Database

ISG provides a function for building a database from a large collection of data that allows users to search for and extract the needed information.

3.2 Building Files

ISG can load the Java Swing components or the data objects onto the memory and then translate it into Java programs for an information system. For example, consider a system for keeping track of the student's scores in a course. Each teacher may teach 4 or 5 courses in a semester and needs to keep track of the student's scores for all the assignments and tests in a course. Therefore this system needs to provide the following functions – to enable the entry of courses, the percentage rate of scores, a description of the test scores, students' names, homework or quiz scores, midterm scores and final scores; and to show the score list and the students' final grades. The data is stored using

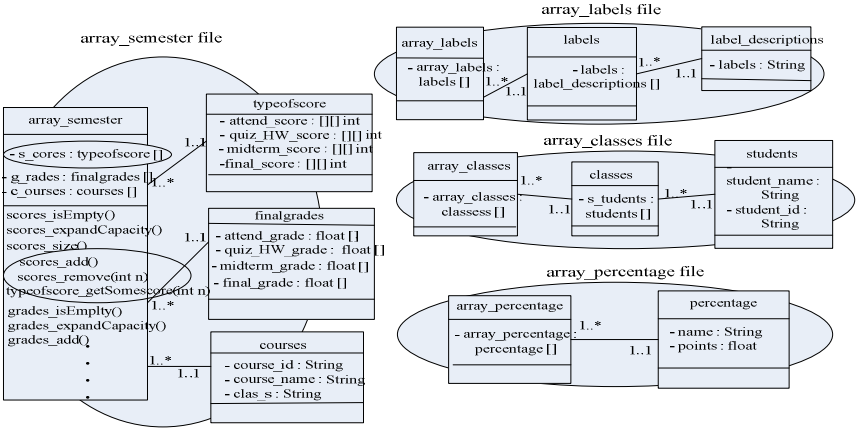

Fig. 1. Class diagram for the file structure of the Student Scores system

object streams rather than regular streams. Because Java has *persistence* in object-oriented circles [8], that means we can let the object layout on disk to be exactly like the object layout in memory. So we save an object (the first created object) to disk and objects that are created subsequently, whose memory addresses are stored in each array, are managed and stored to the disk automatically. ISG file system introduces this mechanism. For using this mechanism, we designed the file structure of the system as shown in Fig 1. There are 4 object stream files in the system, each of which is shown as an oval in the figure and labeled at the top of it – array_semester file, array_labels file, array_classes file and array_percentage file. There are 3 arrays in the array_semester file, which are shown below the top label in the array_semester rectangle, for storing scores, grades and courses. Array s_cores is used to store students' scores for each class in this semester (shown in a separate rectangle), array

g_rades is used to store the student's final grades for each class, and array c_ourses is used to store the course number and name for each course. An element of array s_cores is an object of class typeofscore that contains four different arrays viz. attend_score, quiz_HW_score, midterm_score and final_score array. They are used to store the student's scores for the various components that make up the final score or grade. Array_percentage file stores information on the type of test like the midterm, final, or homework-quiz, and the percentage weight of each in the final grade. Array_labels file is used to describe the details of each test type such as chapter 1 homework assignment or quiz on March 22 and those scores are counted in the quiz and homework scores. Array_class file stores the student names and student ids.

Consider the array_semester file for illustrating how the object streams are stored in a file. ISG creates an object. For example, the array_semester is an object type. Related to it are objects typeofscore, finalgrades and courses which are stored in s_cores, g_rades and c_ourses arrays respectively. Method add() is used to add an object to its associated object array. For example, scores_add() adds a typeofscore object to the s_cores array. Method remove() removes an object from its associated object array. For example, scores_remove(int n) removes a typeofscore object from the array s_cores. Method *typeofscore_getSomescore*(int n) is used to get a typeofscore object from the array s_cores. Thus the file structure of Student scores system, with the arrays and the methods provided makes it convenient to retrieve the data elements in any array.

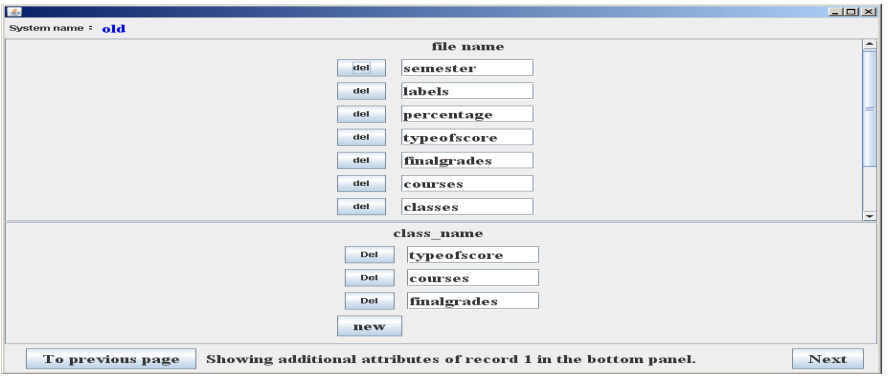

Fig. 2. Building each file of the system

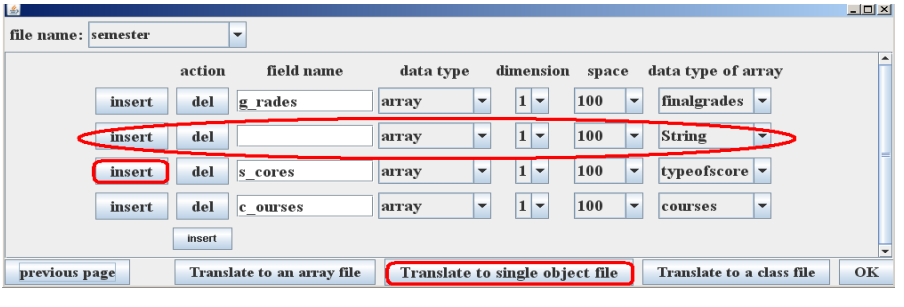

Fig. 3. Building each field of every class

We now discuss how ISG builds the file system of the student scores system. Fig 2 shows a shot of the window that is used to create all the files and their objects for the system, for the classes and files shown in Fig 1 viz. array_semester, typeofscore, finalgrades, courses, array_labels, labels, label_descriptions, array_classes, classes, students, array_percentage and percentage. Fig 3 is used to set the attributes of each class. In the screen shot shown, the attributes of class *semester* are being set, which from Fig 1 are g_rades, s_cores and c_ourses. The figure shows that each of these attributes is a one dimensional array of the same size, and the data type of the arrays is finalgrades, typeofscore and courses respectively. When the user presses the *Translate to a single object file* button, array_semester.java program file will be created. Thus the file structure shown in Fig 1 and the screen shots shown in Fig 2 and 3 can be used to generate the Java classes that have the add, remove, etc methods which can be used to manage an array.

3.3 Linking to the Next Window and Building a Data Processing Window

There are two types of windows provided in the ISG - a link to the next window (Fig 4) and building a data processing window (Fig 5). Fig 4 shows the homepage of the student's scores system and has 5 buttons and one text field. The name of the directory where the data is saved is entered into the textfield by the user. The figure shows the directory as "Fall of 2011", so all the data will be saved in a directory by that name. Clicking on any of the buttons will take you to the corresponding window. Fig 5 shows the records being entered by entering the course id and the course name in the respective fields. In addition to that, *object serialization* mechanism [8] is used to fill objects with data, which saves CPU execution time. These ideas were used to design the following four different kinds of windows to manage data:

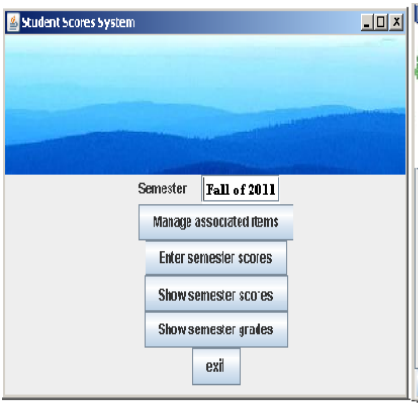

Fig. 4. A link to next window

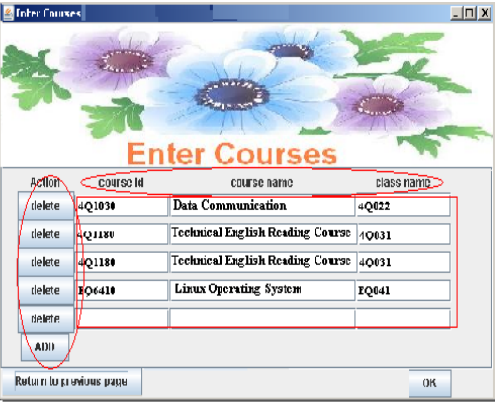

Fig. 5. A basic data processing window

1. *Basic data processing window* (Fig 5) allows for the data of the same type to be added, modified and deleted. Can use the add button to add a new record, and the delete button to delete the corresponding record. Pressing the OK button saves the data to an object stream file.
2. *Basic data processing window plus more data attributes* (Fig 6) has all the features of the *basic data processing window* and shows more data attributes in a panel at the square bottom. When the number of data attributes is more than can be shown in the basic data processing window, then the additional attributes of a record can be shown in the bottom panel. In the figure shown, the message at the bottom of the window says "showing additional attributes of record 2 in the bottom panel", which means that the focus is on an attribute of record 2 and the additional attributes of that record are being shown in the bottom panel.
3. *Special data processing window* (Fig 3) is obtained by adding insert buttons next to the delete buttons in the *basic data processing window*. Pressing an insert button displays an empty line of fields on top of the line whose insert button was pressed, and can be used to enter a new record.
4. *Special data processing window plus more data attributes* - when you add a panel at the bottom of the special data processing window for entering more attributes, then you get this window.

Figures 6, 7 and 8 show the attributes that need to be set in order for the ISG to translate it to a Java GUI program. For example, consider the window shown in Fig 5, which lets a teacher enter the courses in a semester. Before ISG translates it, the user needs to enter the window size and location on the screen, the fonts for displaying text and images, etc. This information is needed by the Java layout manager for creating the window. Fig 6 displays the window where the users can enter each window's name, frame title, frame size, window location, window type and where the data was read from and where the data will be written to. The oval mark in the figure shows the name of the window to be *course*, frame title to be *Enter courses*, frame size to be 500 by 500, location to be (0,0) and type of window to be *basic data processing window*.

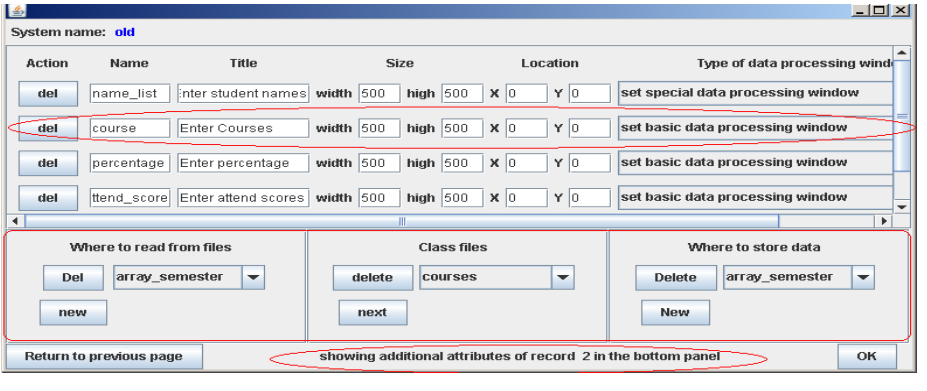

Fig. 6. All the data processing windows of students' score system will be set

At the bottom of the Fig 6, there are three columns; *where to read from files*, *class files* and *where to store data*. The *where to read from files* specifies the files the ISG needs to read from. After users update *Enter Courses* data, the data will be written back to the files specified by *where to store data*. *Class files* tells ISG where each data attribute such as course id, course name and class name, is located. In Fig 8, the rectangle on the right hand side shows the class diagram of the stored file array_semester. Because ISG builds each file structure as an object stream file and array will manage objects of the same class, therefore in the class diagram, it shows

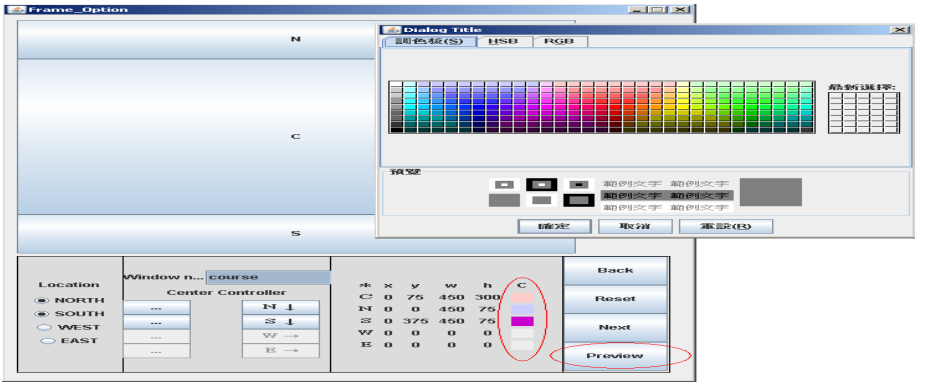

Fig. 7. Selecting a panel to set its' attributes

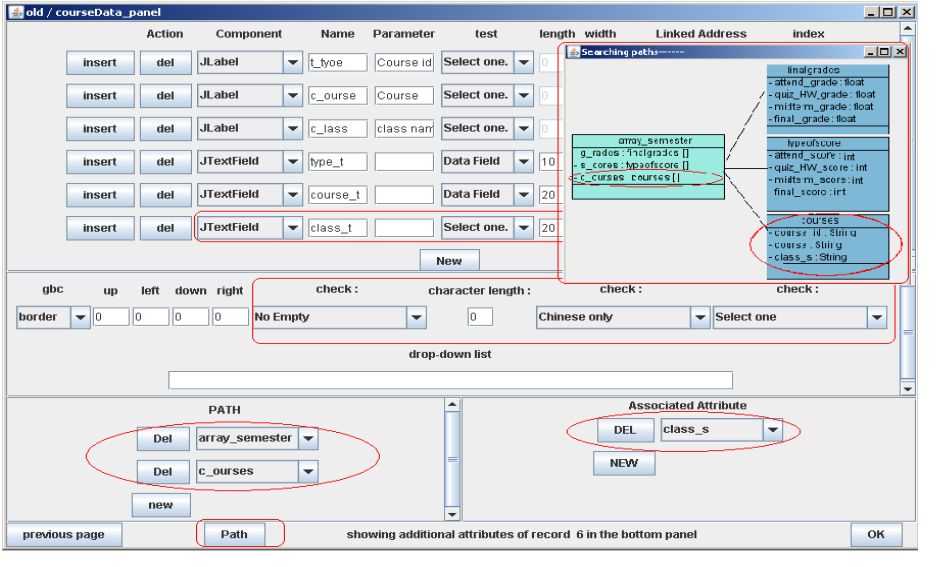

Fig. 8. Set Swing components of *data processing window* course

that every object stored in the file will be stored in its associated array. Therefore the objects created by class finalgrades are stored in array g_rades, the objects created by class typeofscore are stored in array s_cores and the objects created by class coures are stored in array c_ourses because the array_semester file stores student scores, final grades and courses. In the *Enter Courses* window (Fig 5), the course id, course name and class name entered by the users are stored in an object of class *courses.* This class and its' attributes course_id, course and class_s are shown in the rectangle on the right hand side of Fig 8. Therefore in Fig 6, class *courses* should be selected in *class files* column, in the *where to read from files* column semester should be selected and in the *where to store data* column semester also should be selected. Fig 7 is used to select panels located on the window - north, south, east or west panel. The center panel is always there but the other four panels are selectable. ISG offers a color space for users to set a background color for each chosen panel (see area marked with an oval with a 'c' on the top of Fig 7). Fig 8 is used to set the Swing components of the *Enter courses* window (Fig 5). The labels at the top are implemented by using JLabels components, and the textfields for entering the attributes of a record are implemented by using JTextfield components. See the oval at the top in Fig 5, there are 3 labels in it viz. course id, course name and class name.

For the labels, we set the names and parameters of the JLabel components (t_type and course id for the first one, and so on). For the text fields, we set the names and sizes of the JTextfield components (type_t and 10 for the first one, and so on). We can also specify constraints on the data that can be entered into the text fields. For example, we can specify that the text field for entering the class name has to be not empty, and only chinese values can be entered in it. This can be accomplished by setting the parameters shown inside the rectangle shown near the middle of the figure. *Associated Attribute* (bottom right, Fig 8) is used to specify where in the file each data will be stored. Since ISG file structure is an object stream file, each entry data will be stored in an object of some class. In Fig 5, when users enter course id, course name and class name, they actually type in the following text fields viz. type_t, course_t and class_t separately. The data they enter is encapsulated into an object of class *courses* with attributes course_id, course and class_s. Therefore the value of associated attribute course_id is set to what is entered in textfield type_t, course is set to value entered in textfield course_t, and class_s is set to value entered in textfield class_t. The Associated Attribute rectangle only shows class_t. ISG knows how to get and set these attributes since they are entered by the user, but doesn't know where to start searching from until it finds the value of the needed attribute class_s. That information can be specified in the Path rectangle (bottom left of the figure). The Fig 8 specifies the path of the course_id and course or class_s.

We mentioned earlier that ISG builds object stream files using Java persistence which means that the object layout on a disk is exactly like the object layout in memory. Therefore we save an object to disk and then the memory addresses of the objects that are created subsequently are stored in each array. For example, ISG creates an object of class array_labels and then stores many memory addresses of objects of class labels in array array_labels. Later, there are many objects of class label_descriptions are created and then store their memory addresses in array labels.

However if the file system is complicated, it would not be easy for the users to remember it. Therefore ISG provides a class diagram of stored files to help users to set

the PATH and Associated Attribute. For our student's courses system example, the class diagram of array_semester file will be displayed (see the right top rectangle in Fig 8) when the Path button is pressed. From the class diagram, we know course id, course name and class name are stored in an object of class courses and its associated attribute is course_id, course and class_s. Therefore these objects will be stored in an array c_ourses and that is one of arrays stored in an object of class array_semester. PATH (in the left bottom of Fig 8) is used for ISG to translate how to retrieve attribute course_id, attribute course and attribute class_s easily. The class diagram shows only class array_semester and class course needed in this window. Therefore we know the object array_semester is stored in file and in this object contains 3 arrays. There is only array courses what we need and which store objects of class courses with values of course_id, course and class_s. Therefore we need to get object array_semester first and then each object with course_id, course and class_s will be stored in array c_ourses. Therefore, users will set path to array_semester first and then c_ourses next (see in Fig 8). ISG got these values of path and then it is every easy for ISG to translate it into a data processing window course.java and also includes where and how to input or output these data. The advantage of this idea is the file structure introduced is very useful for ISG to retrieve every attribute of an object with users helping to offer these parameters.

3.4 Displaying and Printing the Data

There are three sets of data to be displayed for the Student scores system viz. quiz or homework_scores, midterm scores and final scores. Query data is displayed in a table and can be sorted. Users can press the print button to print the table.

3.5 Previewing a Designed Window

Pressing the next button in figure 7 will translate all the Swing components of the already set *data processing window course* (see Fig 8) to Java GUI programs and then pressing the preview button will compile and execute these programs and then show the designed *data processing window* on the screen (see Fig 5). The users can go back and forth through the windows by pressing the next and previous page buttons. When the whole system is done, it can be shown to the users to give them an idea of what the system might be like or how it will work. If they have any new ideas or suggestions or if this prototype is rejected, the feedback can be used to modify it. This cycle can be repeated until an acceptable or desired system is created. Can press the reset button to go back to the original values or the original state and can start over again.

4 Experimental Results and Analysis

4.1 Improving CPU Efficiency

The CPU time is one of the most important resources and determines how fast a program executes. Therefore we used a Linked queue [9] data structure to manage the GUI, which lets us easily create a window by adding, deleting, or modifying the Swing

components to the window. Linked queue (see figures 9, 10) is a data structure that uses object reference variables to create links between objects and is large enough to hold the numeric address of an object. It is a dynamic data structure because its size grows and shrinks as needed to accommodate the number of elements stored.

For example, consider the four classes of students we discussed earlier. We use an array to represent the four school classes and then another array *classes* to represent students of each class. Because an array is a convenient data structure for storing objects and each array element implementations is efficient, we only allocate enough space per element for the object reference variable. If managed properly, by using an appropriate initial capacity and then expanding it as needed, this additional space is not a problem. Therefore we use an array to store a collection.

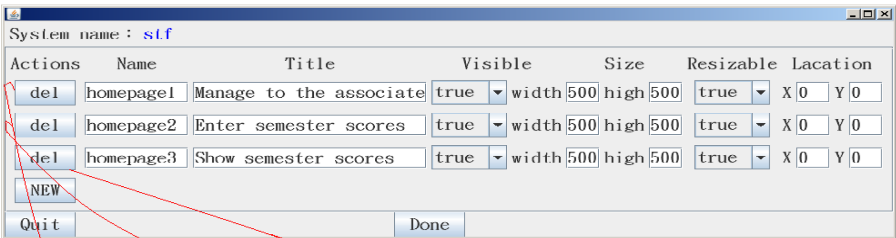

Fig. 9. Set the attributes of a *link to next window* for figure 5

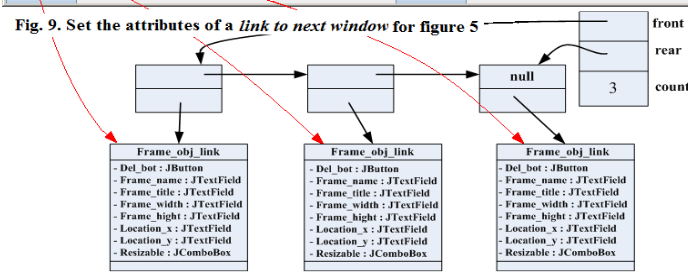

Fig. 10. Using Linked Queue to store Swing Compenents

4.2 ISG has a Quick Development Process Time

ISG helps to develop a prototype quickly which allows the objectives to be tested and developed even further. ISG uses the concept of a spiral model [10] in which feedback from the earlier prototype is used to further refine it. Thus ISG is a user-friendly and efficient program generating software tool which can be used to generate Java programs.

ISG can be used to generate a prototype quickly which can then be shown to the system designers and users. When they see this prototype and start using the system, they might find some functions that does not meet their needs, or certain functionality that is missing. That information can be used by the developers to revise the system to meet users' requirements. Since changes can be done quickly by using our ISG, it can help users understand what they need and what the information system should be like, thus resulting in an end product which would be much better than what it would be otherwise. Since ISG can save time in writing, debugging and testing the program, it would also lower the cost of producing software written in Java.

5 Conclusions and Future Work

We showed how our efficient information system generator was used to develop a system for keeping track of student's scores by generating a total of 56 Java GUI programs. ISG is also very efficient since, for instance, it translated 300 lines of code in the system we illustrated in just 32 milliseconds.

We are currently developing an I-Conference generator that can generate JSP code for a conference system. A conference is a meeting of people who "confer" about a topic. When you need to develop a conference system for your department, school, institute or company for example, you can use ISG to help you implement it after you do the analysis and design. I-Conference generator is a GUI interface for translating the input provided by the user to JSP or HTML programs. The user provides the name of each web page and their corresponding documents or information resources on the Web. After the Web pages have been created, the user needs to specify what the data (in the text fields of the Web pages) is about and where to store it (such as in a database table or a file). The generator will then convert it to the appropriate JSP code, which can be stored on a server and the conference can be installed on a Web site. The GUI interfaces can be completely generated by ISG regardless of how many windows are needed.

Our ISG is not 100% ready at the moment for generating all of the GUI windows for an I-Conference. We still need to add more functions to the *data processing windows*. Since the four types of data processing windows we discussed use a fixed-length record format, we need to change that as records are rarely of the same length. Therefore we need to add different length records format for building the *data processing windows*.

We are in collaboration with East Land International Company Limited for developing a business information system for them. It is an international business company that exports glass containers and health food. We will be developing an information system for them that would involve computing the monthly shipping amount, generating reports on their monthly earnings, profit, orders, etc.

References

1. Excel (2010), `http://office.microsoft.com/en-us/excel/`
2. JFlex- The Fast Scanner Generator for Java, `http://jflex.de/`
3. BOUML- a free UML 2 tool box, `http://bouml.free.fr/doc/javagenerator.html`
4. javadoc - Java API Documentation Generator, `http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/javadoc.html`
5. JAG- Java Application Generator, `http://jag.sourceforge.net/`
6. Usman, M., Nadeem, A., Kim, T.-H.: UJECTOR: A tool for Executable Code Generation from UML Models. IEEE Advanced Software Engineering & Its Applications, 165–170 (2008)

7. Bajwa, I.S., Siddique, M.I., Choudhary, M.A.: Rule based Production Systems for Automatic Code Generation in Java. IEEE Digital Information Management, 300–305 (2006)
8. Horstmann, C.S., Cornell, G.: Core Java Volume I–Fundamentals, 8th edn. Sun Microsystems Press, Prentice Hall (2008)
9. Lewis, J., Chase, J.: Java Software Structures designing and using data structures. Pearson Education Inc. (2005)
10. Whitten, J.L., Bentley, L.D., Dittman, K.C.: System analysis design methods. McGraw-Hill (2004)