A DATA SOURCE DEFINITION TOOL FOR REPORTING TOOLS

Bachelor of Technology (Information Technology)

End of First Semester Project Report

Haoxiang Zhu

June, 2007

Supervisors:

Gareth Cronin (Industrial)
Development Team Leader
Kiwiplan Ltd.
New Zealand

Dr. Xinfeng Ye (Academic)
Department of Computer Science
University Of Auckland
New Zealand

ABSTRACT

The Kiwiplan GUI framework provides a CrystalReportsstyle report designer that uses Kiwiplan's flexible table system as its data source. Everyday users are able to create reports based on existing customizable tables within application user interfaces. However, power users in organizations often wish to report on custom data sources. Traditionally power user reporting tools allow the definition of an SQL query and then reports are built on the resultset from this query.

In this project, we are going to build a Data Source Definition Tool which allows the user to link together business objects in a SQLjoin style fashion to achieve custom data sources based on the business objects, rather than just a SQL query on the database backend.

Contents

1. INTRODUCTION ..... - 3 -

1.1. THE COMPANY ..... - 3 -

1.2. MOTIVATION ..... - 3 -

1.3. PROJECT GOAL ..... - 4 -

2. HIGH LEVEL SYSTEM OVERVIEW ..... - 5 -

3. INVESTIGATING IREPORT AND JASPERREPORTS ..... - 7 -

4. SYSTEM REQUIREMENTS SPECIFICATION ..... - 9 -

5. INTEGRATE IREPORT INTO THE SYSTEM ..... - 13 -

5.1 DATA SOURCE SERVICE V.S. DATA SOURCE ..... - 13 -

5.2 IDATASOURCESERVICEPROVIDER ..... - 14 -

5.3 USING JRDATASOURCE ..... - 15 -

5.4 USING IREPORTCONNECTION ..... - 16 -

5.5 WORKING WITH FLEXIBLETABLE IN KIWIPLAN FRAMEWORK ..... - 17 -

6. SUPPORT FOR SUB-REPORTS ..... - 19 -

6.1 MASTER DATA SOURCE ..... - 19 -

6.2 DATA SOURCE EXPRESSION ..... - 21 -

6.3 OBJECT LEVEL JOIN WHEN BUILDING SUB-REPORT ..... - 22 -

7. CONCLUSIONS ..... - 25 -

8. ACKNOWLEDGEMENT ..... - 26 -

9. REFERENCES ..... - 27 -

1. Introduction

This report documents the outcomes from the first half of my final year Bachelor of Technology project. In this section, I am going to briefly introduce the company which offers this project, followed by the motivation and the final goal of this project.

1.1. The company

Kiwiplan is a software development company that services the corrugating and packaging industries. Typical customers include firms that produce corrugated cardboard products such as boxes, display stands, and other packaging products.

Their product range covers the entire business process for a packaging firm, from order entry to shipping. The core products relate to controlling the plant and scheduling the corrugating machinery. These products communicate with the equipment on the factory floor to control production and collect data.

In the 1970's, Kiwiplan was starting business as a small corrugating firm. As their throughput increased they developed computer systems to help them keep up with demand. There was considerable interest in these computerised systems from other packaging companies, and the IT department grew and eventually separated from the box plant division.

Kiwiplan is now one of the world's leading software suppliers to the packaging industry. They have customers in 28 countries and four international offices. All research and development work is done in New Zealand.

1.2. Motivation

- Currently, Kiwiplan uses *DataVision* as the reporting tool. This type of reporting tool only supports reports to be built from a single data source. There is no way of combining data that obtained from different data stores (e.g. different databases) together to produce the desired reports. However, power-users in organizations often wish to report across multiple data sources, this is typically the case, for big organization such as Kiwiplan, where a report may require data stored at different database servers located in different counties (e.g. US office and NZ office).

- Sub-reports are widely used in many organizations. A sub-report is an entire report that is placed in the detail area of another report. Its main purpose is to display data from data sources linked in a set using one-to-many links at the same level. The existing reporting facility in Kiwiplan does not have the build-in support for creating a sub-report. Having the ability of producing sub-reports not only make the resulting report more visually understandable, it also saves time

and effort for technical staffs to prepare lots of complicated reports.

1.3. Project Goal

The goal of this project is to enhance the existing reporting facility in Kiwiplan, so that:

- Report user can use multiple data sources to create a single report.
- Report user can design the desired report using an easy-to-use tool.
- Sub-report should be supported.

A key difference between this reporting facility and the traditional reporting facility is that the tool works on-object-basis.

Traditional reporting tool usually has a connection to a back-end database, and the report is generated from the resultset of executing some query languages. In this tool, we do not have a back-end database connection. And all the data sources that the reporting tool works with are objects. An example of such objects can be a collection of JavaBean objects.

In the next section, a high level view of the proposed system is to be shown to get us familiar with the proposed system.

2. High Level System Overview

In this section, we are going to have a general structure of the proposed system, and also have some ideas of how the system can achieve the goals that I specified in the project goal.

There are a few very important components in this system, let us now look at them one by one:

- Raw Data Source: The raw data source is the place where the data that we use in the report is originated. This can typically be some relational databases.
- Custom Data Source: As I mentioned earlier, report users typically want some kind of custom data source(s) for the report, and therefore, the custom data source here, will be the data source(s) that the reporting tool makes use of. The custom data source in this case, can typically be a collection of java objects, since our proposed tool works on object level.
- Reporting Tool: A tool that allows the users to design the report using the supplied data source(s).
- Report: The result the system should produce. Note that the content of the report may come from different raw data source(s).

Having explained these important components, let us have a look at how our proposed system links them all together, and achieve our goal:

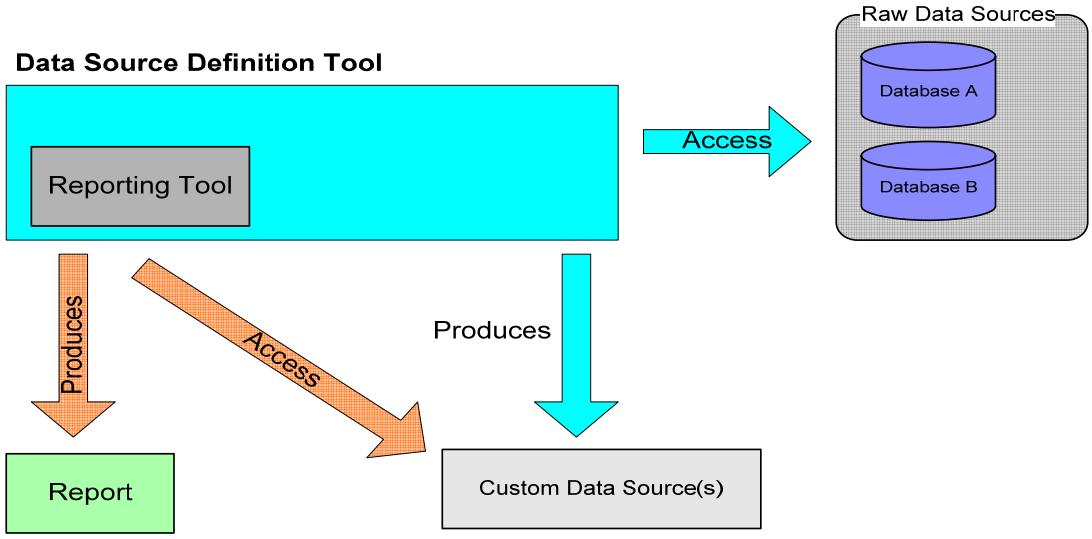


Figure.1 High level overview of data source definition tool

As we can see from Figure.1, the Data Source Definition Tool has linked all the important components together. This can be further explained by the following workflow:

1. The Data Source Definition Tool accesses some raw data sources.
2. The Data Source Definition Tool produces some custom data source(s). Note that there may be multiple custom data sources produced.

3. The reporting tool uses the custom data source(s) produced by the tool.
4. A report is generated based on the custom data source(s).

We can also notice that the reporting tool is actually part of the Data Source Definition Tool. This means that in order to develop such a reporting facility, we either have to develop a reporting tool of our own, or we have to use one of the existing reporting tools that are publicly available, which can be easily embedded into our system. There are many reporting tools that are publicly available, such as DataVision, iReport, and BIRT (as an Eclipse plug-in). They all have very similar functionalities, but in this project, I have chosen iReport as our underlying reporting tool, due to the fact that it is better suited to this project. The following table shows a comparison between iReport and DataVision on some selected features (Note that new version of DataVision may have some more features, this comparison is done based on the DataVision version that Kiwiplan uses.):

Tools / ***Features***	**iReport**	**DataVision**
Drag-n-drop report design	√	√
Language	Java	Java
Custom Data Source	Very well supported	Supported, but limited
Embeddability	√	√
Sub report	√	No
Report Parameters	√	√
Complexity	Heavy-weighted	Relatively light-weighted
Databases	Any with JDBC driver defined	Any with JDBC driver defined
Report Engine	Jasperreports	Build-in

Table.1 comparison between iReport and DataVision

As we can see from table.1, although the major functionality of these two reporting tools are quite similar, iReport is a better option due to its sub-report and custom data source support, hence, I have chosen iReport as the reporting tool for this project.

It is also shown in the table that the report engine iReport uses is called JasperReports, in next section, we are going to look at the connection between iReport and JasperReports, and how they are combined together to produce an end-user report.

3. Investigating iReport and JasperReports

JasperReports and iReport are two widely used open source software developed by JasperSoft. They are purely written in Java, and their existences have made reporting in Java applications. In this section, we are going to look how JapserReports and iReport work.

JasperReports is an open-source Java class library designed to aid developers with the task of adding reporting capabilities to Java applications. Since it is not a standalone tool, it cannot be installed on its own. Instead, it is embedded into Java applications by including its library in the application's CLASSPATH. JasperReports is a Java class library, and is not meant for end users, but rather is targeted towards Java developers who need to add reporting capabilities to their applications.

JasperReports takes in a report design as an XML file (jrxml file), and compile into a jasper report file (jasper file). Through a JasperFillManager, a report print is produced for the end report users. The following diagram shows the work flow of how JasperReports work.

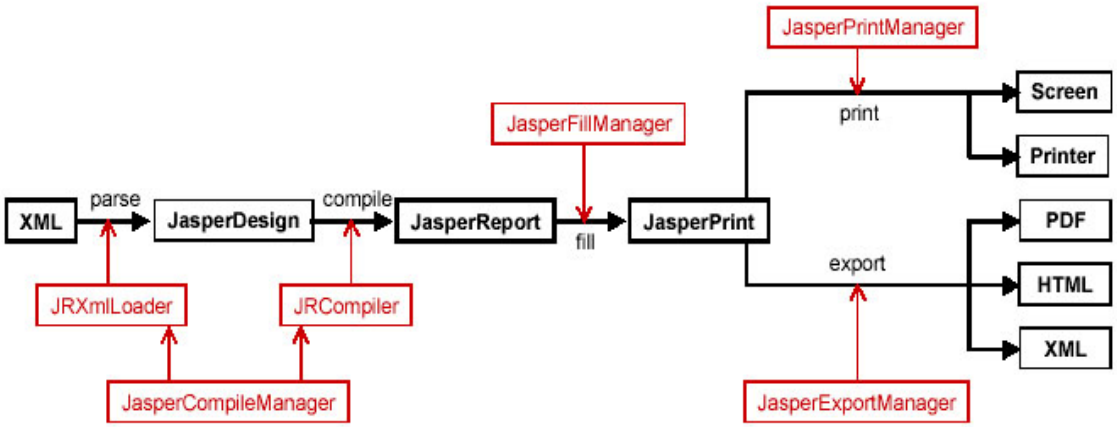


Figure.2 Work flow for JasperReports

(www.hisp.info/confluence/download/attachments/3330/seminar.ppt)

iReport, on the other hand, provides a front-end Graphical User Interface, for the end report users to define the design of a report, unlike JasperReports, iReport is targeted towards any report users, i.e. not necessary Java developers. The primary job of iReport is to produce the jrxml file for JasperReports to use. Hence, we can see iReport has JasperReports build-in as its report engine. The following diagram illustrates how iReport and JasperReports work together to produce a report for the end report users:

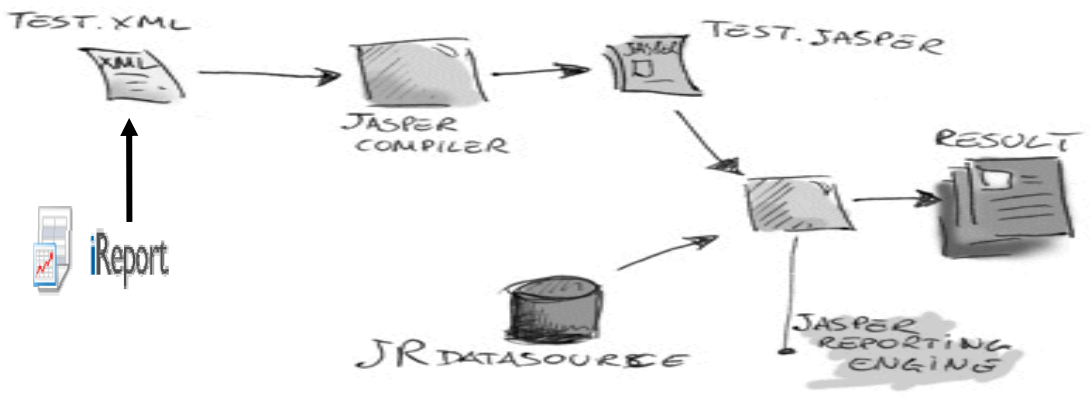


Figure.3 iReport and JasperReports
(http://ireport.sourceforge.net/cap3.html)

In Figure.3, report users define a report structure using iReport. The result that iReport provides is a jrxml file (in the diagram, it is shown as TEST.XML). Then this jrxml file is passed alone to JasperReports, it compiles the design of the report into a .jasper file (in the diagram above, it is shown as TEST.JASPER). This compiled report design is combined with a JRDataSource, to produce a final report (RESULT in the diagram), also known as a jasper-print. It is not hard to see that in JasperReports:

JasperPrint = Jasper file + JRDataSource

<<interface>> **JRDataSource**
+next() : bool
+getFieldValue(JRFeild field)() : object

Now, let us have a look at another very important concept in JasperReports, JRDataSource. JRDataSource is an interface provided by JasperReports, it is the data source used for JasperReports to produce a print. Therefore, any possible data sources need to implement this interface, to provide the compatibility to JasperReports.

Some examples of implementations of JRDataSource can be:

- JRResultSetDataSource – wraps a JDBC ResultSet object as the data source.
- JRXMLDataSource – wraps an XML document as the data source.
- JRTableModelDataSource – wraps a TableModel as the data source.

Through this interface, JasperReports has provided the users with the ability to define custom data sources. Having this interface, we can therefore implement any kind of data source of our own, and those custom data sources that defined by ourselves can be used by JasperReport as the data source to produce reports. This is one of the most important reasons that I chose iReport/JasperReports as the reporting tool in this project.

4. System Requirements Specification

Requirement Engineering has been one of the most important components throughout Software Development Life Cycle (SDLC). In order to detail the requirements of the project, I have had quite a number of meetings with the development leader within Kiwiplan. And the resulting software requirement of this project is summarized using the following use case diagram:

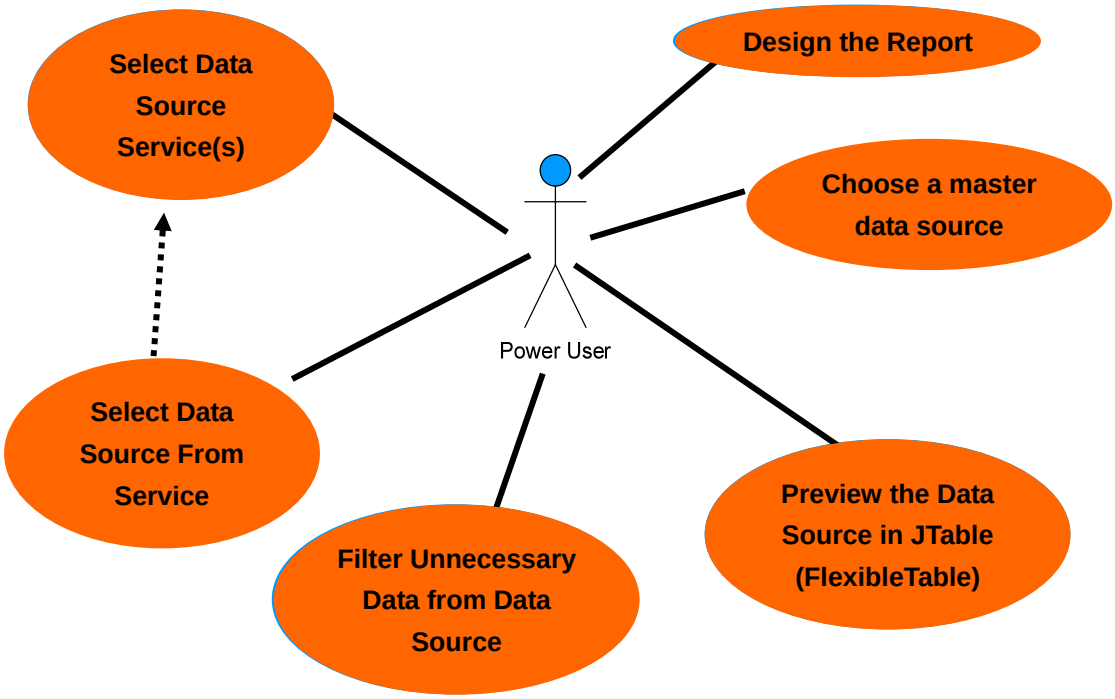


Figure.4 Use Case Diagram

Now, let us have a look at each of the use case in more detail:

- **Select Data Source Service(s):** As previously mentioned, the user is able to design the report across multiple data sources. Therefore, the user needs to be able to select different data source services which provide accesses to different data sources. Typically, a data source service provides access to more than one data sources.

- **Select Data Source From Service:** When the users have chosen the data source service, they should be able to select the corresponding data source from the particular service.

- **Filter Unnecessary Data:** The users might not be interested in all of the data from a particular data source. They want to filter out un-necessary data. For example, select the order records during a particular period of time.

- **Preview Data Source:** After the user has chosen the data source, they should be

able to preview the selected data source(s). The data source(s) are typically displayed in a JTable under the current implementation. And in Kiwiplan, we are going to using a FlexibleTable (table implementation under Kiwiplan GUI framework) to display the data source(s).

- **Choose Master Data Source:** The functionality is primarily used for the sub-report support of the tool. A master data source is the data source used for the master report. When the user has chosen more than one data sources from the services, they should (they are required) choose a master data source for iReport to use, and any other data sources are treated as detail data sources used for sub-report. This feature will be discussed in more detail later this report.

- **Design Report:** Of course, the users need to be able to design the report structure.

After the use cases have been decided. The User Interface has been developed according to those use cases. The following figure is a screen-shot of the current implementation of the user interface (Note that this user interface design might change later):

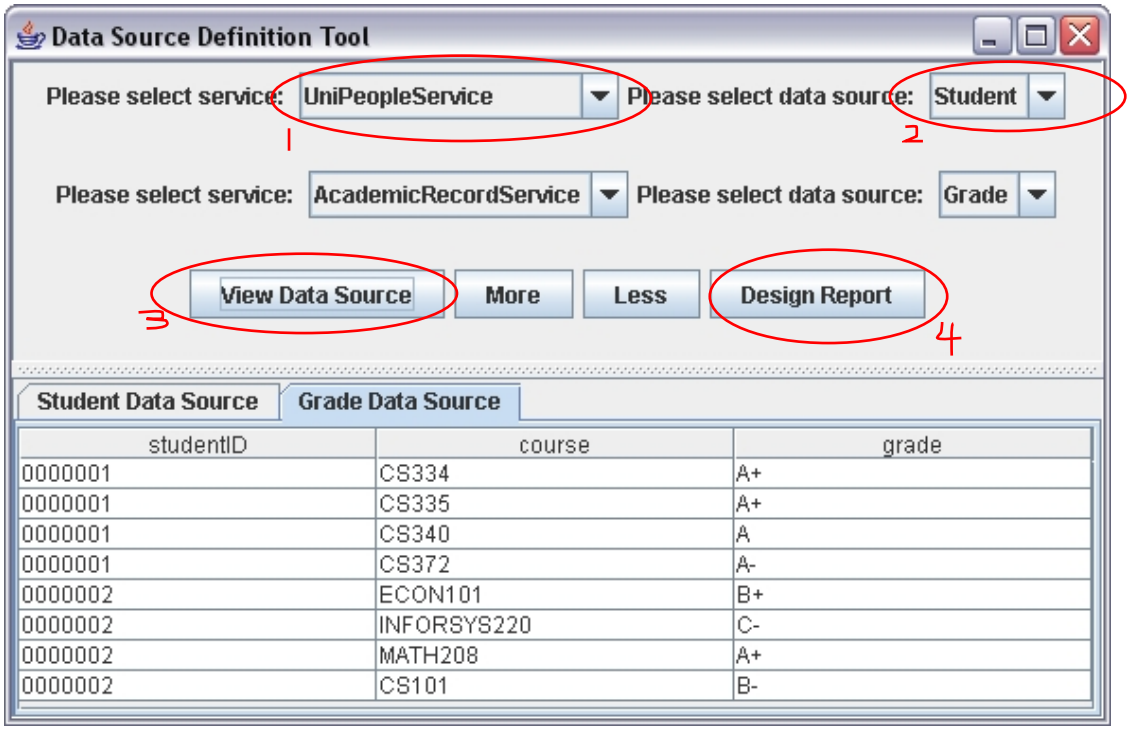


Figure.5 User Interface

In the above User Interface design:

- The component that is labeled "1" corresponds to the **Select Data Source Service** use case where user is able to choose among different data sources. For example, if there are four data source services available, each of the dropdown lists will have the four services available for users to choose from. By default, the

system shows one pair of "select service" option and "select data source" option, the user can ask to choose from more/less data source services/sources by clicking the "More"/ "Less" buttons. In this example shown in Figure.5, the user has clicked the "More" button once, and has chosen a service called "UniPeopleService" which provides the people service.

- The component that is labeled "2" corresponds to the "**Select Data Source From Service**" use case where the user has chosen the "Student" data source.

- The component that is labeled "3" corresponds to the "**Preview Data Source"**, where the user is able to preview the selected data source(s) in JTable, in the example above, as the user has chosen two data sources from two different services, there are two data sources table displayed in a tabbed pane.

- The "**Filter Data Source**" use case has not been implemented under the current implementation, but it will be implemented in the final version the system. The plan is to make use of the Filter System inside Kiwiplan framework, therefore, the implementation is not considered to be particularly difficult.

- The component that is labeled "4" corresponds to the "**Design Report**" use case. However, if more than one data sources are chosen, a dialog will pop up prompting for the master data source (corresponds to "**Choose Master Data Source"** use case). The dialog is shown below:

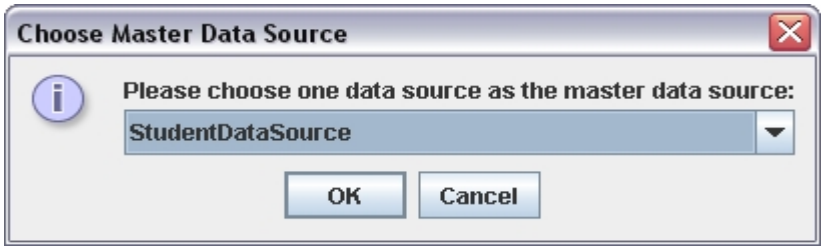


Figure.6 Selection of master data source dialog

After the master data source has been chosen, the user interface for iReport will display allowing the user to design the report using the selected data source(s).

The following is a brief class diagram for the user interface design:

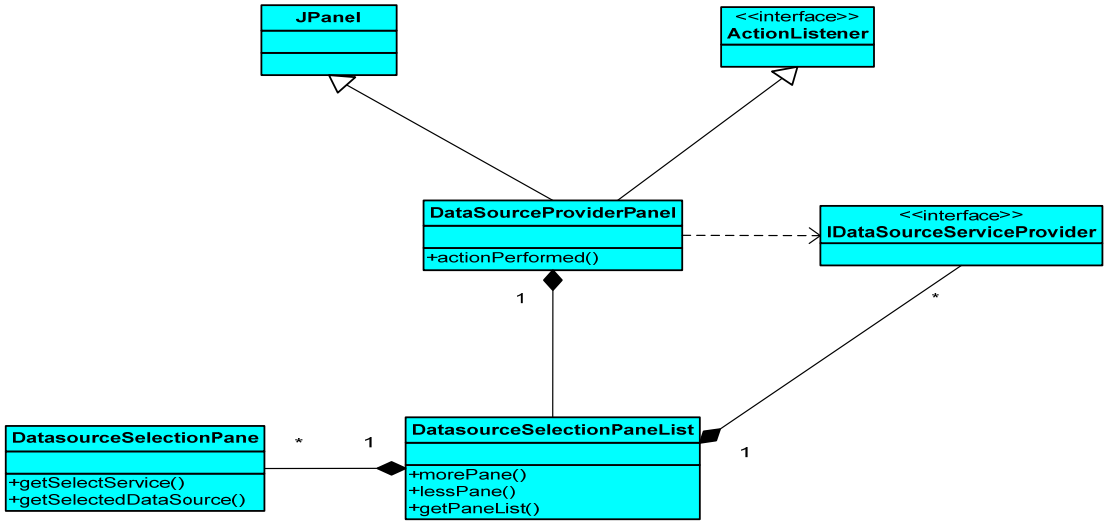


Figure.7 class diagram for user interface design

5. Integrate iReport into the System

Up until this point, we have had the data source(s) ready for the iReport to use. In this section, we are going to see how iReport uses our custom data source(s) produced by the Data Source Definition Tool, and how those data source(s) are passed along to iReport.

5.1 Data Source Service V.s. Data Source

Before I introduce how iReport is integrated into the system, we have to clarify two very important terms that we used: **Data Source Service** and **Data Source**, first, let us have a look at the diagram:

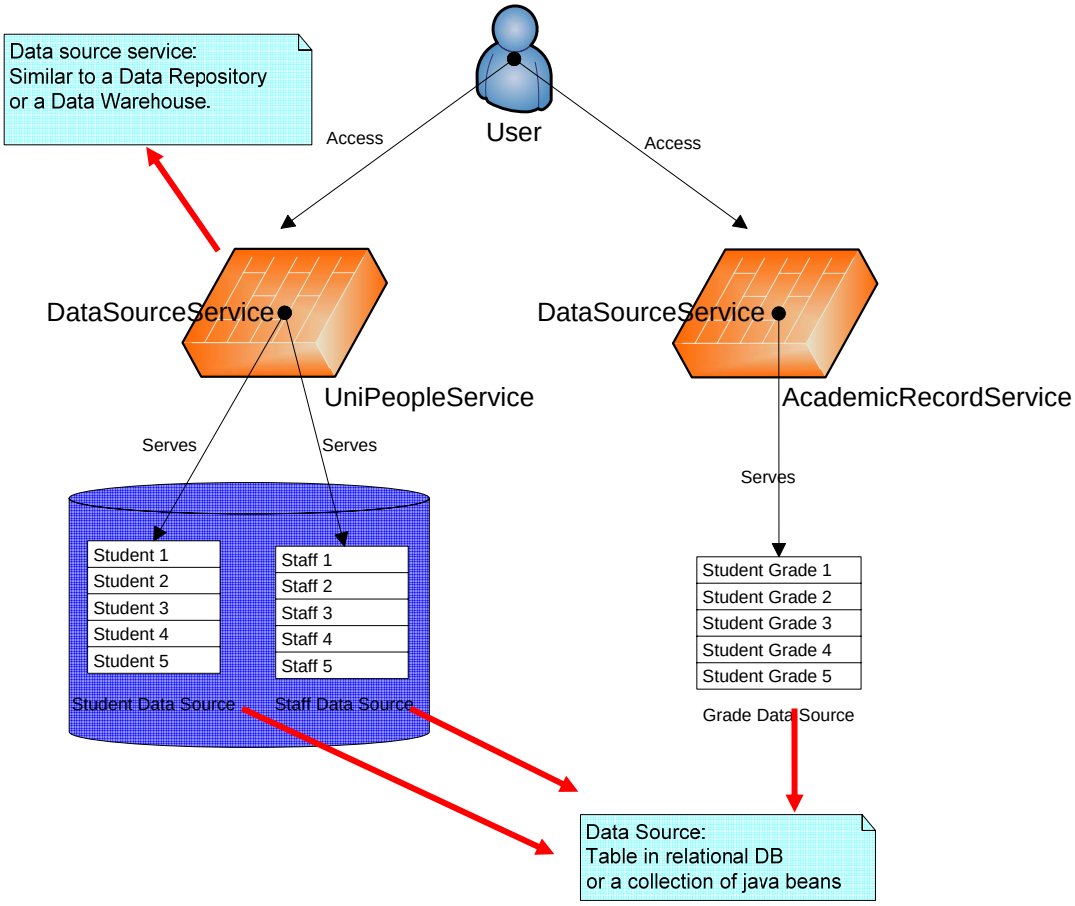


Figure.8 Data Source Service V.s. Data Source

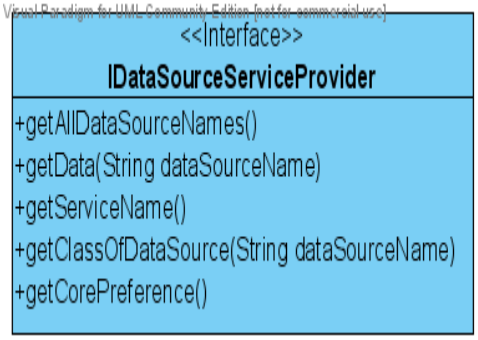
A **data source service** is a **service access layer** that provides accesses to the data sources it serves. In the above diagram, there are two data source services, UniPeopleService and AcademicRecordService, the UniPeopleService provides two data sources: Student and Staff data sources. And the AcademicRecordService provides only one data source StudentGrade data source. A data source service is similar to a data repository or a data warehouse, in the sense that it stores all the data

sources corresponding to that particular service.

A **data source** is the actual objects have the data information stored. Typical examples of data sources can be a table in a relational databases or a collection of java objects. In the above diagram, all the data sources are provided as a list of simple JavaBean objects.

5.2 IDataSourceServiceProvider

Now, let us have a look at how the data source service is implemented. As we noticed from Figure.7, there is an Interface called *IDataSourceServiceProvider*. In the use case diagram (shown in Figure.4), the user is able to select from different data source services. This interface provides such compatibility for any class that provides this kind of service. The class diagram of this interface is shown on the left:



For any classes that implements this interface, they should specify how each of the data sources that this service provide is obtained, for example, from a JDBC resultset, or from a collection JavaBean objects in the getData() method. A sample implementation of this interface is shown in Figure.9, which provides the data sources shown in Figure.8.

```
public List getData(String dataSourceName) {
  List list = new ArrayList();
  if (dataSourceName.toLowerCase().equals("student")) {
    Student s1 = new Student("0000001","James", "Bond");
    Student s2 = new Student("0000002","Bill", "Gates");
    list.add(s1);
    list.add(s2);
  }
  else if (dataSourceName.toLowerCase().equals("staff")) {
    Staff st1 = new Staff("4545674","Abc","Def","Computer Science");
    Staff st2 = new Staff("8745374","Kkk","Hhh","Computer Science");
    Staff st3 = new Staff("3524364","Loo","Ccc","Economics");
    list.add(st1);
    list.add(st2);
    list.add(st3);
  }
  return list;
}
```

Figure.9 Sample implementation of getData method for IDataSourceServiceProvider

In this implementation, the data sources are generated from a List of JavaBean objects.

5.3 Using JRDataSource

By having the data source services, we have the ability to enable users to choose data sources from different services. However, after these selected data sources are in place, we need to make use of these custom data sources in iReport. As I mention in Section.3 of this report, iReport make use of custom defined data sources through an interface called "*JRDataSource*".

In our case, when the user has selected the desired data sources, they are displayed in a JTable. Therefore, we can make use of a pre-defined data source called *JRTableModelDataSource* to warp the table model as the data source for iReport to use.

Typically, any kind of *JRDataSource* is provided through a corresponding JRDataSourceProvider, which is another interface provided by JasperReport. In my implementation, I have implemented a *TableModelDataSourceProvider* to provide the *JRTableModelDataSource***.**

The detail of the structure is shown in the following class diagram:

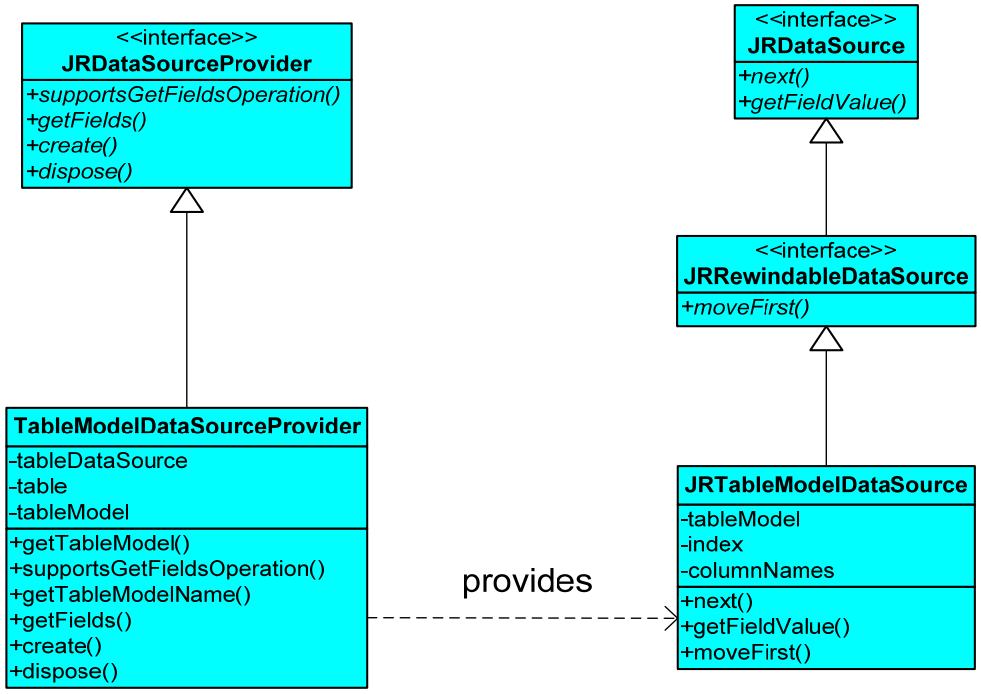


Figure.10 class diagram for JRDataSource and its provider

- The *JRRewindable* interface is an extension to *JRDataSource*, any class that implements this interface is a data source that can move back to its very first element. *JRTableModelDataSource* is such a class.

- *TableModelDataSourceProvider* provides the *JRDataSource* through the *create*() method. The following code segment shows the constructor and the create

method of *TableModelDataSourceProvider*:

```
/* constructor */
 public TableModelDataSourceProvider(TableModel tableModel){
    this.tableModel = tableModel;
    this.tableDataSource = new
                          JRTableModelDataSource(this.tableModel);
  }
/* create a JRDataSource */
public JRDataSource create(JasperReport report) throws JRException{
    return new JRTableModelDataSource(tableModel);
  }
```

Figure.11 code segment showing constructor and create method from TableModelDataSourceProvider

5.4 Using IReportConnection

Using *JRDataSourceProvider* provides the *JRDataSource* for iReport to use, however, we still have to find out a way to pass the *JRDataSource* from our Data Source Definition Tool to iReport.

iReport has provided class called "*IReportConnection*" which enables the custom connections to iReport. Typically, each *IReportConnection* (and its sub classes) warps a *JRDataSourceProvider* instance in to, so that when this connection gets connected to iReport, the *JRDataSource* is provided through the provider that is wrapped inside this connection. In this project, I have implemented a *TableDataSourceConnection* which wraps a *TableModelDataSource* in it as the provider of *JRTableModelDataSource*. The following class diagram shows how iReport uses *IReportConnection* to produce data source through the provider:

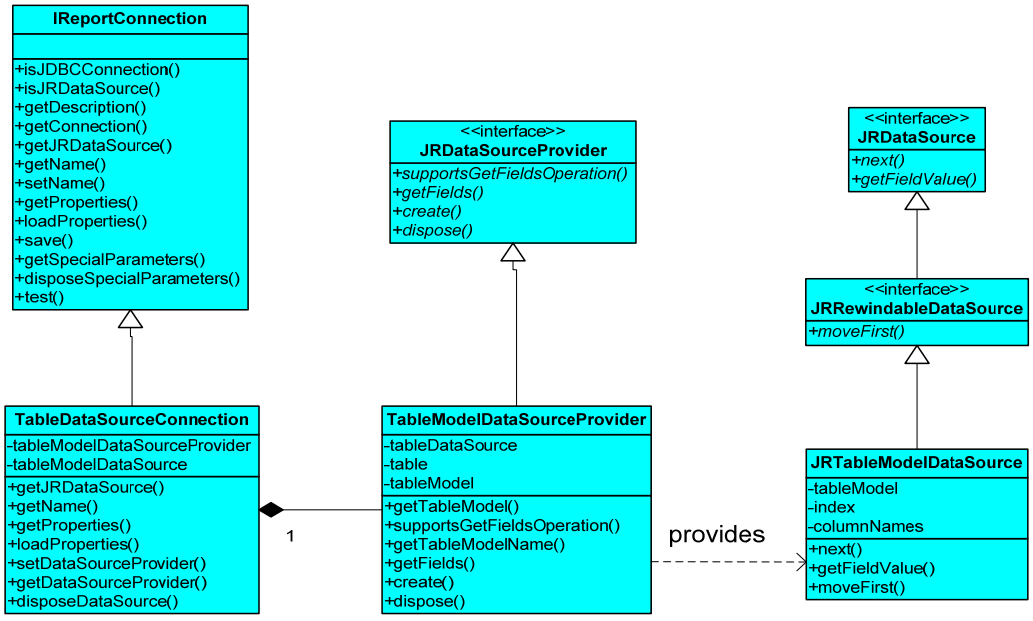


Figure.12 class diagram: connects to iReport

When the users choose more than one data sources from the Data Source Definition Tool, each of those data sources will be created into a *JRDataSourceProvider* and wrapped into an *IReportConnection*, i.e. every data source is passed alone to iReport as a separate connection. The following code segment shows how iReport is launched with the pre-defined custom data source(s) that user has chosen using the Data Source Definition Tool:

```
public static void launchIReportWithDefaultConnection(…){
  … …
     Vector connList = new Vector();
     for (Iterator iterator = tableModels.iterator();
                                  iterator.hasNext();) {
     TableModel dataViewTableModel = (TableModel) iterator.next();
     TableModelDataSourceProvider provider = new
                 TableModelDataSourceProvider(dataViewTableModel);
     IReportConnection conn = new
                            TableDataSourceConnection(provider);
     connList.add(conn);
    … …
}
frame.setConnections(connList);
... …
```

Figure.13 code segment: launch iReport using custom data sources

5.5 Working with FlexibleTable in Kiwiplan Framework

In Kiwiplan, instead of displaying data using a *JTable*, we use a *FlexibleTable* to display the data source(s). The *FlexibleTable* is similar to *JTable* but with more sophisticated functionalities, e.g. the table headers are defined from a preference bundle XML file. The following screen shot is an example of using the Data Source Definition Tool with *FlexibleTable*:

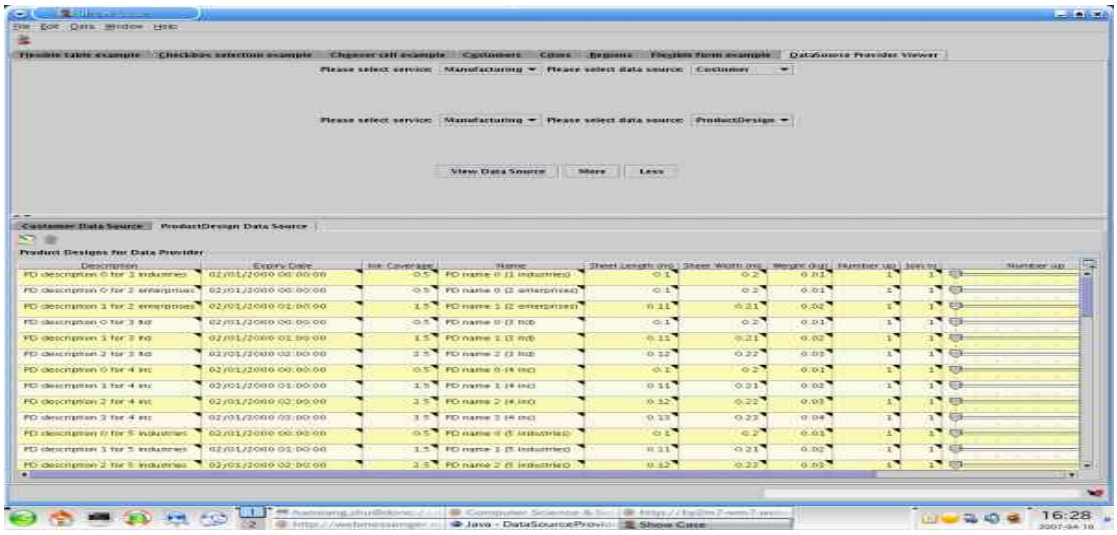


Figure.14 Data Source Definition Tool with *FlexibleTable*

The implementation using *FlexibleTable* is not fully completed yet. However, the concepts of using the *FlexibleTable* are exactly the same as using *JTable*. The following class diagram shows the design using *FlexibleTable*:

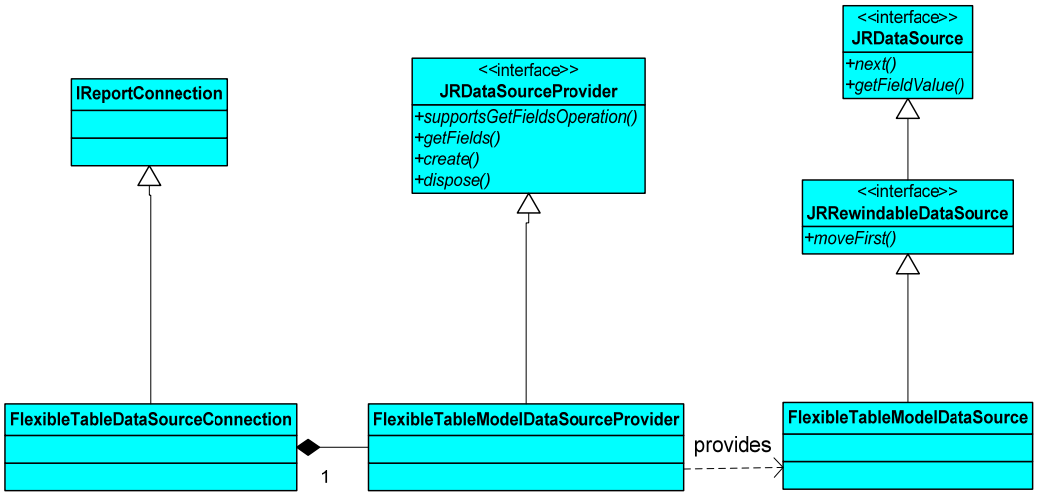


Figure.15 class diagram: connects to iReport using FlxibleTable

The second half of this project will be focusing on the implementation using *FlexibleTable*.

6. Support for Sub-reports

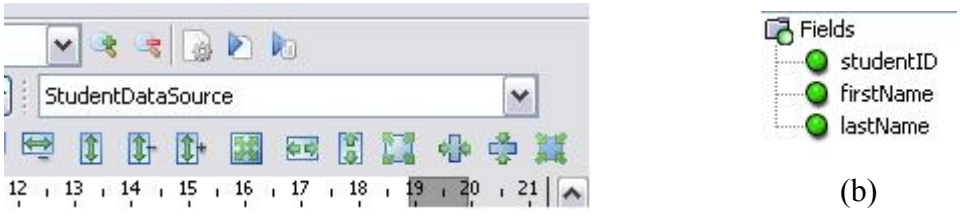
Nowadays, sub reports have been widely used in many organizations, therefore, it is necessary to allow the Data Source Definition Tool to have the ability to build sub-reports. A sub report usually uses multiple data sources that have some kind of relationship. In this section, we are going to see how the Data Source Definition Tool supports the user in creating sub reports using the custom data sources that it provides.

6.1 Master Data Source

As described in Section 4, when the user chooses more than one data sources, a dialog will pop up asking the user to choose one of the selected data sources as the master data source to iReport.

Choosing more than one data sources will inform the Data Source Definition Tool that the current user tries to create a sub report using iReport. Typically, sub-report uses more than one data sources when the report is being filled with data, however, in iReport, there can be only one active connection at a time, this means any data sources that are used by the sub reports will therefore needs to be recreated at run time. Thus, when the user has selected the master data source, this data source will be passed alone to iReport as an active connection. The following are the event flows after the user has chosen a master data source:

1. The user chooses one of the data sources as the master data source.
2. This data source is wrapped into an *IReportConnection,* and set to be the active connection, which is the connection that the master report uses.
3. The *TableModel* of any non-master data sources are stored into a parameter list for master report, and this parameter list is being passed alone to iReport as the parameters of the master report.



(a) (b)

Figure.16 (a) active connection and (b) fields from active data source

Figure.16 (a) shows that when user choose student as the master data source in the Data Source Definition Tool, the student data source has been the data source that the active connection uses for the master report. Figure.16 (b) shows the available fields from the current active connection. In this example, there are studentID, firstName, and lastName from student data source.

Figure.17 parameters for master report

In iReport, each report has a list of parameters. Each of the parameters is a JRParameter object.

Figure.17 shows the parameter list for the master report. Notice at the bottom of the list, there is a parameter name: *SUBREPORT_GRADEDATASOURCE_TABLEMODEL*. This means that the user has previously chosen two data sources from the Data Source Definition Tool and the grade data source is not chosen as the master data source. Therefore, the table model of this data source is stored in the parameter list of the master report for further recreation of this data source.

A *JRParameter* is constructed using the name of the parameter and the corresponding class type of that particular parameter. The actual value (or reference address) of each parameter in the list are stored in the first JRParameter: *REPORT_PARAMETERS_MAP* which is itself a *JRParameter*. This is a special parameter of type *HashMap*. Before the report is getting filled, this hash map is iterated through, and each value is assigned to the corresponding parameter if there is a match, otherwise, that parameter will have a null value. The follow diagram illustrates the structure of the parameter list:

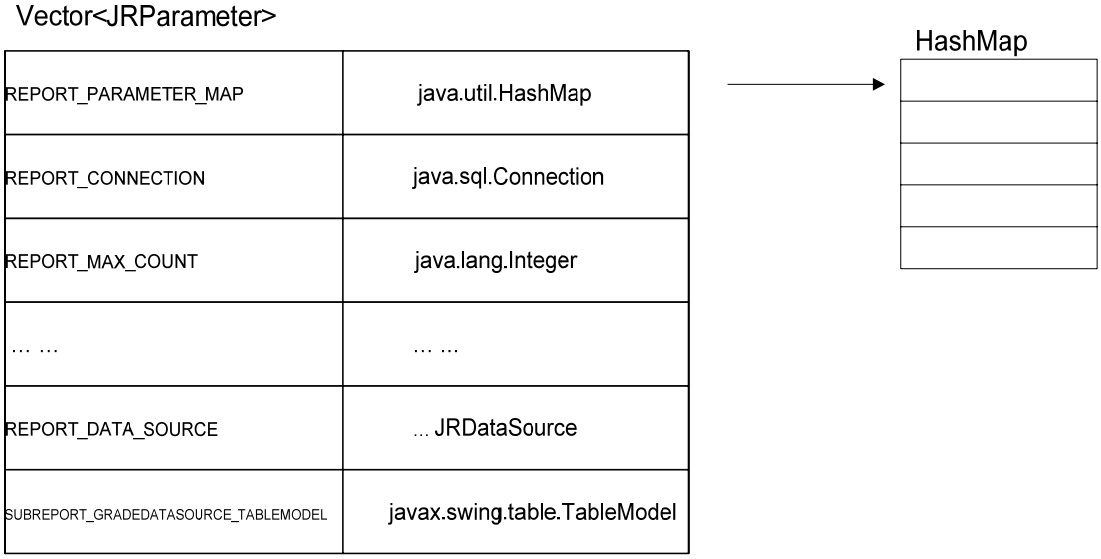


Figure.18 JRparameter list structure for master report

The sample of the above HashMap is shown in Figure.19

HashMap

REPORT_PARAMETER_MAP	java.util.HashMap@3f23a
REPORT_CONNECTION	null
REPORT_MAX_COUNT	15
… …	… …
REPORT_DATA_SOURCE	… JRDataSource@4a3d2
SUBREPORT_GRADEDATASOURCE_TABLEMODEL	javax.swing.table.TableModel@3ea2c

Figure.19 HashMap storing the actual value of *JRParameters*

6.2 Data Source Expression

Since iReport can only have one active connection at a time, therefore, when we are filling the sub-report with data, we need to create the data source for sub report at run time. The recreation of sub report data source is done through a feature called **Data Source Expression** provided in iReport. The following diagram is a screen-shot from iReport when the user is setting up the connection for sub-report, typically, we choose use data source expression.

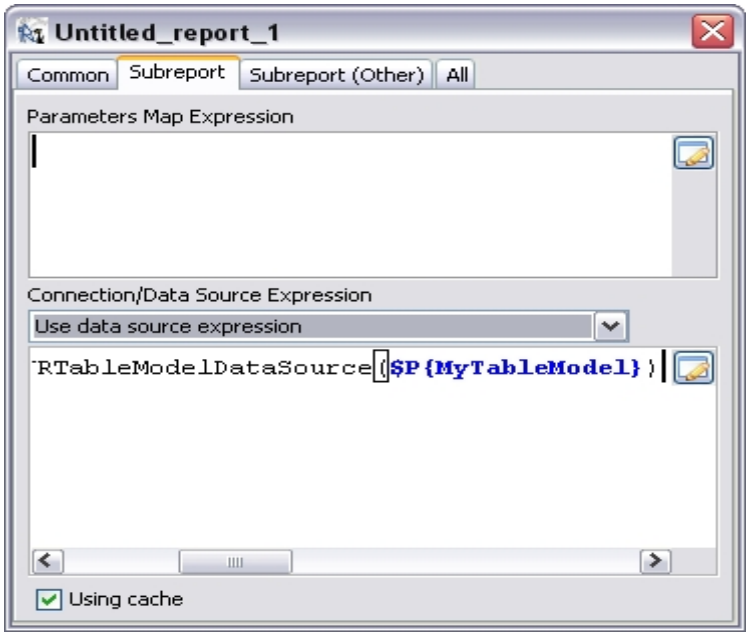


Figure.20 Set up connection for sub report

As we previously discussed, iReport uses *JRDataSource* when building the report, therefore, we need to create a *JRDataSource* object in order to fill the sub report. This *JRDataSource* is recreated every time the report is filled using the table model for that particular data source. The table model has been saved as a parameter of the master report that this sub report belongs to. The following screen-shot is the data source expression editor, where a new *TableModelDataSource* is created using the table model:

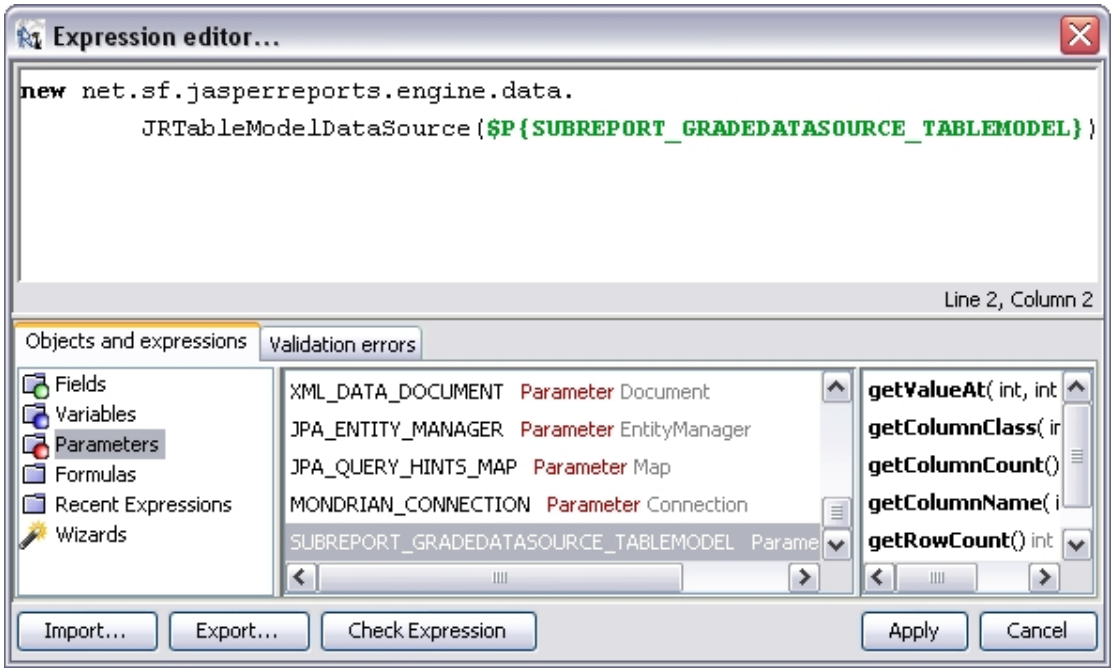


Figure.21Data Source Expression Editor

In this example, we are building a *JRTableModelDataSource* using the table model passed as the parameter from the master report parameter list (shown in Figure.17).

After this data source expression is in place, every time the report is filled, the master report is filled with the data source in the active connection, and the sub-report is filled with this newly created *JRDataSource*.

6.3 Object Level Join when building Sub-report

When we build a sub-report, typically, there are some kind of relationship between the master report and the detail report. This is similar to a foreign constraint in relational databases. However, we use JRDataSource as our custom data source in creating reports; we therefore need to achieve the same type of join in object level.

Let us look at the following example, image we have two data sources displayed in *JTable* as follow:

Student Data Source

studentID	firstName	lastName
0000001	James	Bond
0000002	Bill	Gates

Grade Data Source

studentID	course	grade
0000001	CS334	A+
0000001	CS335	A+
0000001	CS340	A
0000001	CS372	A-
0000002	ECON101	B+
0000002	INFORSYS220	C-
0000002	MATH208	A+
0000002	CS101	B-

And we want to build a report as shown in the diagram below:

0000001 James Bond

studentID	course	grade
0000001	CS334	A+
0000001	CS335	A+
0000001	CS340	A
0000001	CS372	A-

0000002 Bill Gates

studentID	course	grade
0000002	ECON101	B+
0000002	INFORSYS220	C-
0000002	MATH208	A+
0000002	CS101	B-

Figure.22 Sample sub report

As we can see in the above report, the student information is the master report and the corresponding student grade information for that particular student is displayed in the sub report. In order to achieve this, we have to do a join on the student ID between these two data sources when filling the sub report. If we were doing this in a relational database, the following SQL statement will apply:

```
select GradeTable.studentID, GradeTable.course, GradeTable.grade
from GradeTable, StudentTable
where StudentTable.studentID = GradeTable.studentID
```

Figure.22 SQL for join between tables

This kind of join can be achieved in iReport by defining a filter expression on the sub report data source, the following diagram shows how such expression is defined:

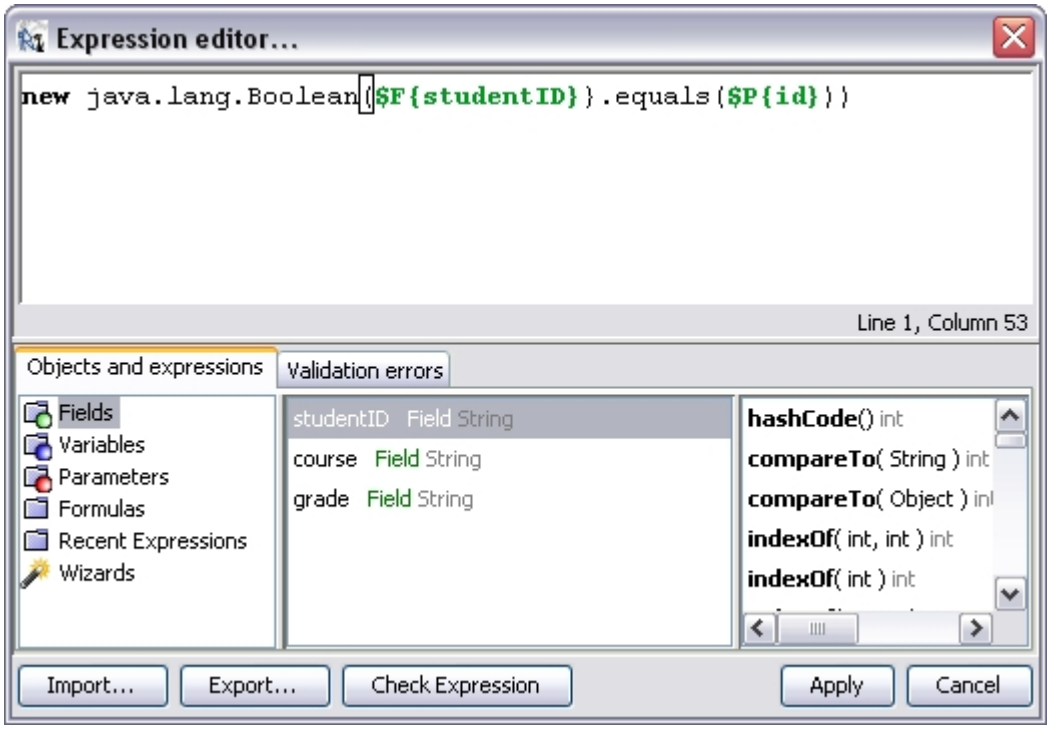


Figure.23 Filter Expression

In the above filter expression, $F{studentID} represents the studentID in the Grade data source. $P{id} is the studentID from the student data source (the master data source) that we have to previously passed from the master report as a sub report parameter.

7. Conclusions

During the first half of the project, I have completed the tasks that outlined in the initial project plan. The development of the prototype version of the Data Source Definition Tool (using JTable) will be very useful when I convert the system into FlexibleTable version in the next semester.

I have now gained relatively good understanding of how iReport/JasperReport work, this must be a big advantage in the next a few iterations of this project.

In the next half of the project, I am planning to focus on the following issues:

- The parameter passing seems to be very complicated for users to be able to use easily, and therefore it is necessary to simply the process by making it more "automatic".
- How the report definition efficiently so that the data sources can be retrieved back again if we want to re-run the report.
- Passing queries through to restrict the amount of data being retrieved

8. Acknowledgement

I wish to extend my great thanks to my supervisors: Gareth from Kiwiplan and Dr. Xinfeng Ye, for their continuous support throughout the first half of this project.

I would also like to thank Dr. S. Manoharan, the BTech coordinator, for providing lots of useful information on the BTech programme.

Many thanks to all the experienced programmers from Kiwiplan, for giving me lots of interesting and positive suggestions/advices, and for helping me fixing all kinds of small bugs in the code.

9. References

1. JasperReport Homepage, "***JasperReports Documentation"***, Available from: http://jasperforge.org/sf/wiki/do/viewPage/projects.jasperreports/wiki/HomePage. Accessed Apr 2007

2. Elizabeth Montalbano, "***Eclipse Developers To Get Open-Source Reporting Tool. (Business Intelligence and Reporting Tool)***", Computer Reseller News Sept 13, 2004 p37.

3. iReport Home, ***"iReport Tutorial"***, Available from: http://ireport.sourceforge.net/tutorial1.html. Accessed Apr 2007