Malaysian Journal of Computer Science, Vol. 16 No. 1, June 2003, pp. 1-8

REPCOM: A CUSTOMISABLE REPORT GENERATOR COMPONENT SYSTEM

Chee Hoong Leong and Sai Peck Lee
Faculty of Computer Science & Information Technology
University of Malaya
50603 Kuala Lumpur
Malaysia
email: chee.hoong.leong@accenture.com
saipeck@um.edu.my

ABSTRACT

This paper introduces a XML-driven and component-based development approach to report generation with the purpose of promoting portability, flexibility and genericity. In this approach, report layout is specified using a user-defined XML elements together with queries that retrieve data from different databases. A report is output as a HTML document, which can be viewed using an Internet browser. This paper presents the approach using an example and discusses the usage of the XML-driven report schema and how the proposed reusable report engine works to output an HTML report format. In addition, the tool is implemented to support heterogeneous database models. Overall, this system reduces the implementation cost and improves the productivity of work.

Keywords: ***Report model, Report schema, Report generator, XML, Component-based development***

1.0 INTRODUCTION

It is undeniable that report generation is one of the most important tasks in many companies regardless of the size of the company. A good report generation mechanism can increase a company's productivity in terms of effort and time. This is more obvious in some startup companies, which normally use some in-house report generators. In common practice, a big company normally uses more than one report generator to cater for their reporting needs. The lack of a generic format of report model has the impact that reports generated in one report generator would very unlikely work on another report generator due to the proprietary format used by different vendors. An application is no longer considered an enterprise-level product if XML is not being used elsewhere [1].

In order to minimise the problems associated with report generators, the scope of this approach will focus on the following aspects:

▪ Reusable reporting components

A set of reusable components will be provided and used to develop the report generator. These components are created by using SUN's Java programming language after the domain analysis has been performed.

▪ XML-based report layout

This function allows the application developers to construct the report layout by using XML technology to meet the portability criteria.

In Section 2, the conceptual architecture for realising the XML-driven and component-based development approach to report generation will be described. Section 3 presents the report schema with an example of the usage through an application system. Section 4 presents the report model and report engine, which is the core component of the system. Section 5 describes the data mapping to reports. Finally, a conclusion is drawn in Section 6.

2.0 OVERVIEW

The proposed conceptual architecture for realising the XML-driven and component-based development approach to report generation consisting of various conceptual components is depicted in Fig. 1. The database contains dynamic data, which are changed from time to time, and therefore, the corresponding contents of a generated report will change accordingly whenever the report model is executed [2].

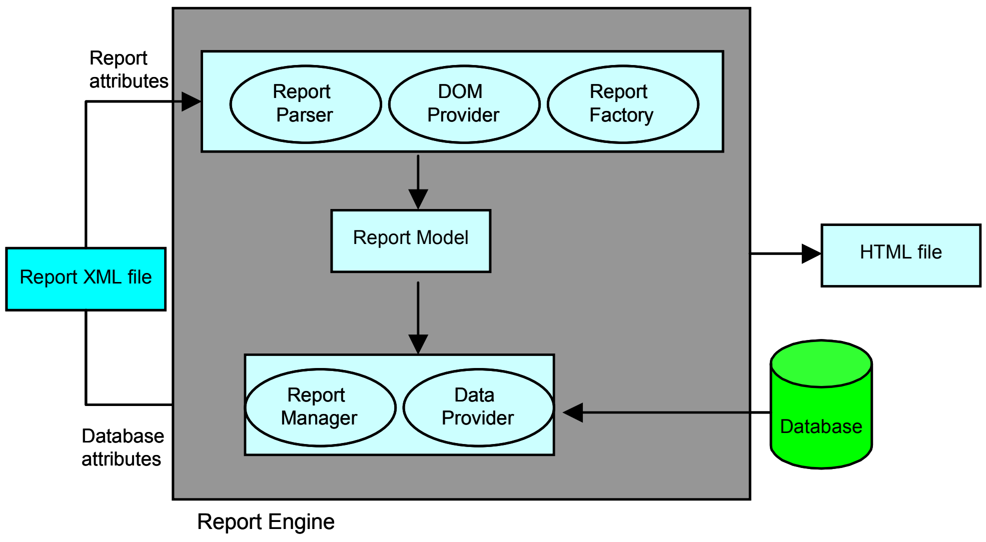


Fig. 1: A conceptual architecture

The report schema is in XML format. The report schema defines the possible layout of the report. Each report has its own report schema and report model. The report model is the representative of the report schema in Java format. It is derived from the report schema after the report engine has parsed it. The parsing process is done by three main components, namely report parser, DOM parser and lastly the report factory component. It takes the report schema as input and produces the report model as output. Only a successful parsing will generate the report model.

In this approach, it is assumed that different databases use queries expressed only in SQL query language which is widely used in multiple database models such as Oracle, DB2, Informix and SyBase. Database queries imposed on a specific database schema and database are specified in the report schema. The SQL query statements defined in the report XML file will be used by the Data Provider component to retrieve the data from the database to fill the report. In fact, the Data Provider component helps the application developers to develop different database query implementations. The report manager is used to interpret the report model and generate the report in HTML format, which can be presented via an Internet browser.

3.0 REPORT SCHEMA

The modelling of a report starts with the definition of a report schema, which captures the layout structure of the report. A good report schema will help in further processing of the report model as well as generating the final report using the proposed approach described in Section 1.0. It is important to note that one report will have only one report schema and one report model respectively. Specifying a report with this approach is a matter of putting in the heading of the report and titles for various rows and columns with the data contents left to be specified with the help of the SQL query language.

The requirement analysis of **RepCom** is documented in a UML use case diagram as shown in Fig. 2. The following is the possible main use cases for **RepCom**:

- Create Report
- Upload report
- Delete report
- Retrieve report
- View report

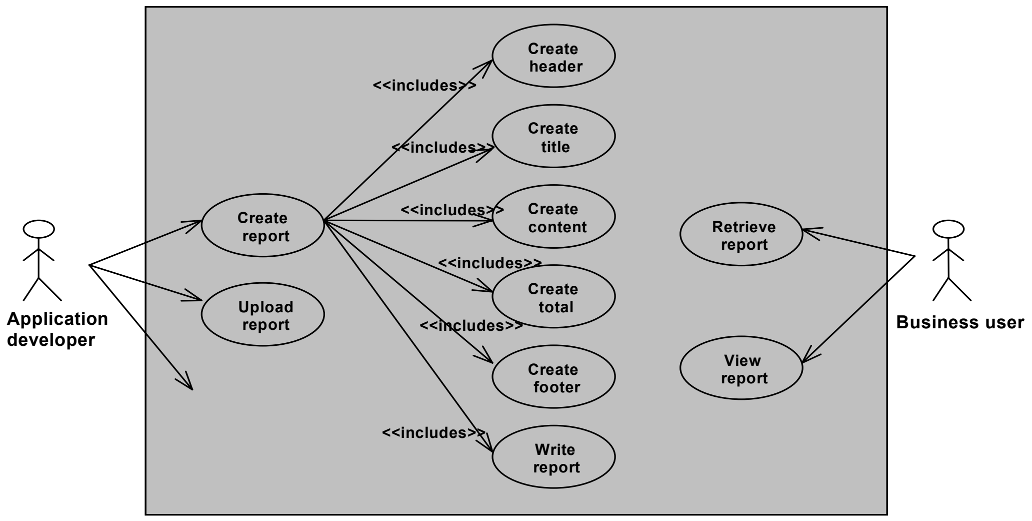


Fig. 2: RepCom Use Cases

As depicted in Fig. 2, the application developer is responsible to create report according to business user requirement. Fig. 3 shows a traceability between the Create Report use case model and the analysis model using a trace dashed line. In the analysis model, the report schema is the boundary entity that acts as an interface for the application developer to specify the user requirements. The Create Report use case starts by defining the heading of the report of the report schema, and followed by the title, the content of the report and the footer of the report. The process ends when the report schema are parsed and transformed into a report model by the report engine. The example of the partial report schema for a student report is shown as below:

```
<JavaXML:Report id="ReportFormat3">
 <JavaXML:Header fontsize="5" width="120"  line="ABOVE" linetype="-">
     <JavaXML:Component type="text" width="120" align="CENTER" >First Header First
line</JavaXML:Component>
   </JavaXML:Header>
  <JavaXML:Header width="120" linetype="-" line="BELOW" fontsize="3">
     <JavaXML:Component type="text" width="80">Second Header First line</JavaXML:Component>
    <JavaXML:Component type="pageno" width="40">Page No : ###</JavaXML:Component>
  </JavaXML:Header>
  <JavaXML:Title width="80" linetype="=" line="BELOW" fontsize="2" align="CENTER">
    <JavaXML:Component type="text" id="blank" width="30" align="LEFT"></JavaXML:Component>
    <JavaXML:Component type="vtotal" id="BIT" width="20" align="LEFT">BIT</JavaXML:Component>
    <JavaXML:Component type="vtotal" id="BEC" width="20" align="LEFT">BEC</JavaXML:Component>
  </JavaXML:Title>
  <JavaXML:Body>
    <JavaXML:Data>
            <JavaXML:Query id="query1">
            <JavaXML:SQL>
SELECT DISTINCT region,",","," FROM table_student WHERE Region = 'WP'
</JavaXML:SQL>
<JavaXML:SQL>SELECT DISTINCT ' ', 'Number of Student for year 2000',
(SELECT COUNT(stream) FROM table_student WHERE stream = 'BIT' And Region = 'WP' And semester like
'%1999/2000'), (SELECT COUNT(stream) FROM table_student WHERE stream = 'BEC' And Region = 'WP' And
semester like '%1999/2000')  from table_student WHERE region = 'WP'
</JavaXML:SQL>
<JavaXML:SQL>SELECT DISTINCT ' ','Number of Student for year 2001',
(SELECT COUNT(stream) FROM table_student WHERE stream = 'BIT' And Region = 'WP' And semester like
'%2000/2001'), (SELECT COUNT(stream) FROM table_student WHERE stream = 'BEC' And Region = 'WP' And
semester like '%2000/2001')  from table_student WHERE region = 'WP'
```

```
</JavaXML:SQL>
</JavaXML:Query>
<JavaXML:Query id="query2">
<JavaXML:SQL>
SELECT DISTINCT region,",","," FROM table_student WHERE Region = 'JOHOR'
</JavaXML:SQL>
<JavaXML:SQL>SELECT DISTINCT ' ','Number of Student for year 2001',
(SELECT COUNT(stream) FROM table_student WHERE stream = 'BIT' And Region = 'JOHOR' And semester like
'%2000/2001'), (SELECT COUNT(stream) FROM table_student WHERE stream = 'BEC' And Region = 'JOHOR'
And semester like '%2000/2001') from table_student WHERE region = 'JOHOR'
</JavaXML:SQL>
</JavaXML:Query>
    </JavaXML:Data>
  </JavaXML:Body>
  <JavaXML:Footer fontsize="2">
    <JavaXML:Component type="text" width="115" align="CENTER">End of Report</JavaXML:Component>
  </JavaXML:Footer>
</JavaXML:Report>
```

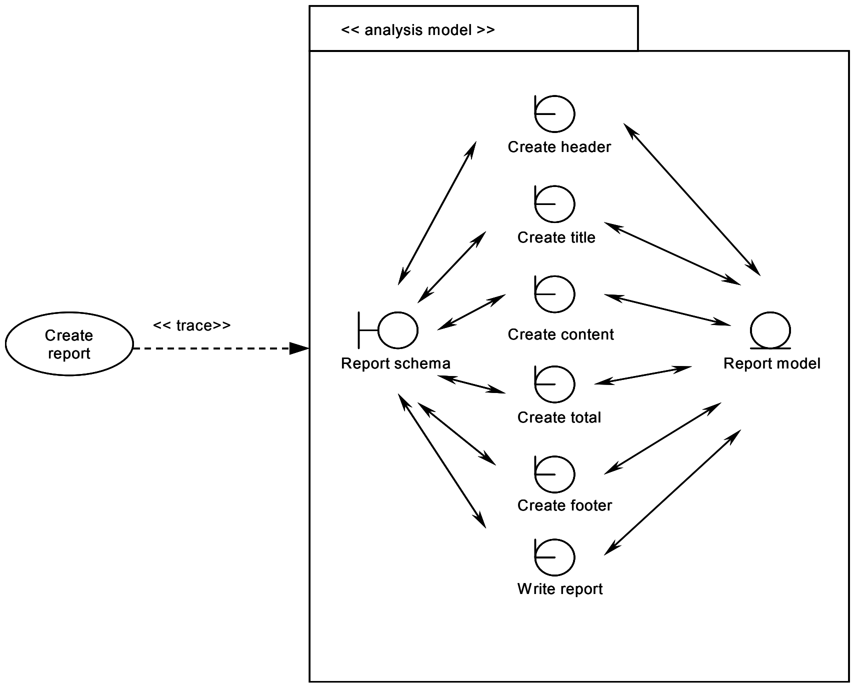


Fig. 3: Traceability between Create report use case model and analysis model

4.0 TRANSFORMATION OF REPORT SCHEMA TO REPORTS

Before a report model is created, the report factory component has to be invoked to transform the report schema to the report model as depicted in Fig. 1. After the transformation, the report model will be made up by a set of reusable report components [3] as shown in Fig. 4.

4.1 Report Model

In the report model, a report object (RepoObj) is a generic object or a container object for other objects like head object (HeadObj), title object (TitleObj), amount object (TotalObj), data object (DataObj) or foot object (FootObj) as shown in Fig. 4. The header report object and footer report object have only one head object or foot object respectively, whereas, the data report object consists of the combination of title object and data object.

A report object is essential in a report model to become a major object holder and this is taken care by the report manager component, which will be discussed in the next section. A report object can have a one-to-one relationship or one-to-many relationship with other objects like title object and data object. It is therefore possible to construct a report object to have more than one title and data object depending on the complexity of the report. Another key component in the report model is the component object (CompObj). A component object contains the information like alignment (left, right, justify), text value, length of the text, etc and it is the inner-most object in a report model.

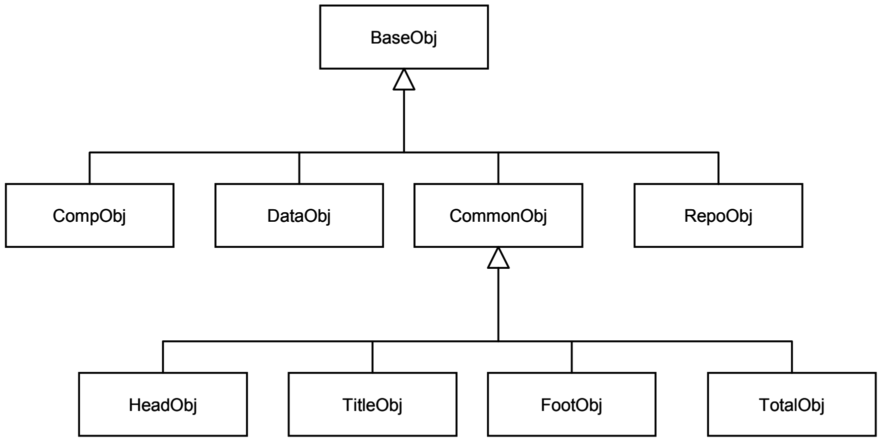


Fig. 4: Components in report model

4.2 Report Engine

The report engine is the main component of this approach. In general, the report engine is made up of several sub components such as the report parser component, a third-party DOM parser component, report factory component, report manager as well as data provider component and a set of reusable report components. Fig. 5 is the class diagram that shows partially the report engine components.

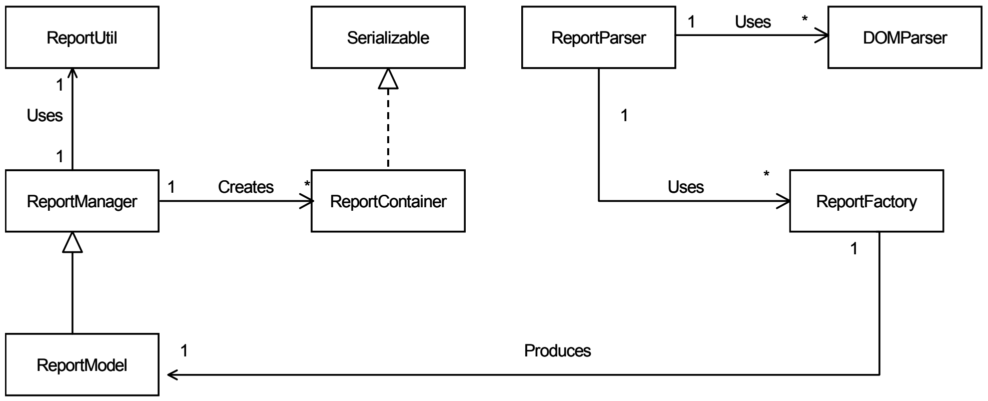


Fig. 5: Class diagram for the Report Engine

The report parser component is built on top of the DOM parser. The DOM parser gives a tree structure as output and allows the application to process the "tree" object by accessing the data at each node of the tree. A set of interfaces and classes that define and implement the DOM is specified in the report parser component. The report parser component parses the XML document and prepares the tree structure for the report factory component. A document object will be created after the parsing. The document object is obtained in the report parser component and passed to the report factory to process. A set of methods is defined in the report factory component to process the XML document object. Each node in the "tree" object is processed accordingly to build the report model.

The transformation from the report schema to the report model is done by the report factory component. It begins by mapping the specific XML node into the respective report components in the report model. For instance, the *JavaXML:Title* node will be transformed into the title object, *JavaXML:Data* node will be the data object and *JavaXML:Footer* node will be the foot object. The header attributes defined within the *JavaXML:Header* node such as text, date, time and page number are the component objects in the report header as depicted in Fig. 6.

As part of the report engine, the report manager is perceived as an interpreter of the report model. The interpretation of a report model is a matter of converting the report objects into a more presentable format. In this case, the report model is converted into HTML format for output display to the user. Each object of the report model is restructured and added the HTML attributes such as font size, font type, alignment, etc. For example, the *JavaXML:Header* when converted to head object is justified to fit in the default report length with an underline to separate from the content.

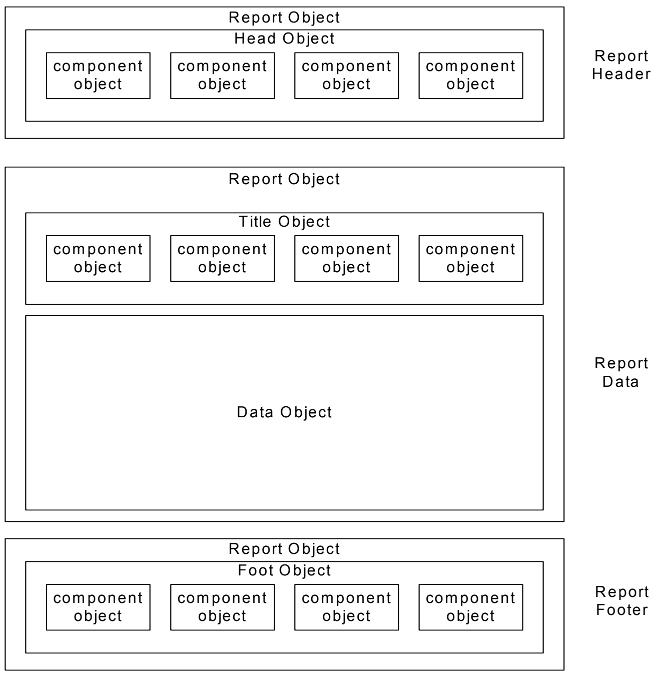
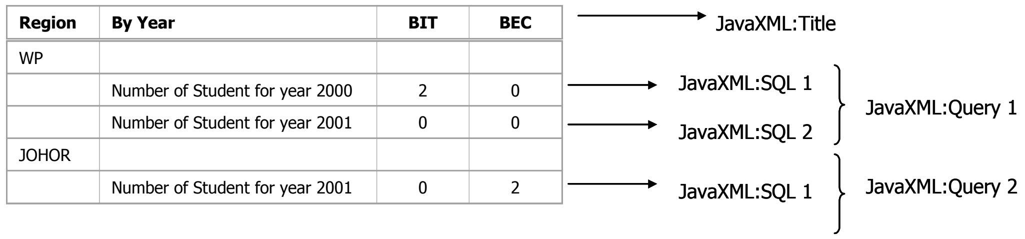


Fig. 6: A logical report model

5.0 MAPPING DATA TO REPORT MODEL

The SQL query language forms the contents of the report [4]. In other words, in order to map the data retrieved from the database to a report, SQL query language is widely used in this approach. Data can be retrieved either directly from a particular column or specified table views that the application developers need to define or from a particular SQL function. Fig. 7 shows a mapping between the data from the database to the report objects defined in the report model. Tuples of *Number of student* are extracted from the database. Data will be restructured and mapped to the query node (JavaXML:Query) under the data node respectively.

It should be noted that the number of result sets returned by a query object in the data node must be the same with the number of component objects in the title node. A wrong reference for data node to the title node will cause runtime exception. The mapping is a straightforward process as long as the XML rules and constraints are followed and used correctly. The report displays the result according to the order of the query element in the report schema.



Region	By Year	BIT	BEC
WP			
	Number of Student for year 2000	2	0
	Number of Student for year 2001	0	0
JOHOR			
	Number of Student for year 2001	0	2

Fig. 7: Mapping data to report model

6.0 CONCLUSION

RepCom is designed to help users generate web reports with minimum understanding of programming knowledge. In this approach, a set of user-defined XML tags are developed to allow web reports of different complexities and sizes to be unified for different database models. With this approach, data are retrieved from the database using SQL queries and reports are specified independently as HTML documents. The report structure is defined using the XML language, which provides a lot more flexibility and simplicity. However, application developers have to follow the strict conformity of the XML specifications used in this approach to develop reports.

The benefits of this approach are simplicity, genericity and independence. The approach is simple because the user-defined XML-based report schema is very easy to understand and does not require much expertise in any programming languages. This simplicity speeds up the development time and minimise effort to report generation that indirectly leads to the reduction of cost to build reports. Due to the use of HTML format, this approach also provides some degree of genericity compared to other formats. The approach is said to be independent because it does not bind to any database models. Each individual report can be specified to retrieve data from any database system. This is not usually the case with other approaches.

Component-based development (CBD) is also an important approach of **Repcom**. The use of CBD provides rapid development and ease of integration of components. For example, with this approach, reports can be turned into different output formats by customising the existing components. In addition, the possibility of architectural mismatch [5] can be avoided with the CBD approach provided information such as API, exception handling, design patterns, etc are well defined.

In a nutshell, this approach promotes two important technologies (XML and Java) to offer the ease of generating reports and the flexibility of retrieving data from any database systems.

REFERENCES

[1] Brett McLaughlin (2000), *Java and XML*, pp. 16-19.

[2] Daniel Chan (1998), "A Document-Driven Approach to Database Report Generation". *Proceeding of the Ninth International Workshop on Database and Expert Systems Applications*, Le Chesnay, France.

[3] Ivar Jacobson (1997), *Software Reuse Architecture Process and Organization for Business Success.* Addison-Wesley, pp. 38.

[4] Oracle Corporation, U.S.A. (1995), *Building Reports Manuals*, 2.5 Edition.

[5] David Garlan, Robert Allen, John Ockerbloom, "*Architectural Mismatch or Why It's Hard to Build Systems Out of Existing Parts*", 1995, pp. 179-185.

BIOGRAPHY

Sai Peck Lee is an Associate Professor at Faculty of Computer Science and Information Technology, University of Malaya. She obtained her Ph.D. Degree in Computer Science from University of Pantheon-Sorbonne (Paris I) in 1994 and her Master of Computer Science from University of Malaya in 1990, and her Diploma d'Etudes Approfondies (D.E.A) in Computer Science from University of Pierre et Marie Curie (Paris VI) in 1991. She has published a number of research papers in several computer science journals as well as in local and international conferences. Her research interests include Software Engineering, Object-Oriented Methodologies, Software Reuse, Information System and Database Engineering. She is a member of IEEE Computer Society.

Chee Hoong Leong is currently a student in the Master of Software Engineering at Faculty of Computer Science and Information Technology, University of Malaya. He obtained his Bachelor Degree of Information Technology from University of Northern Malaysia in June 1992. His research interest is Component-Based Development, Internet Computing and Reporting Tools.