Optimization of an Object–Oriented File System

Ling–Hua Chang
Department of Information Management
Kun Shan University, No.195, Kunda Rd., YongKang Dist., Tainan City 710–03, Taiwan (R.O.C.)
changlh@mail.ksu.edu.tw

Sanjiv Behl
Thomas Edison State College
101 W. State St, Trenton, NJ 08608–1176
sanbehl@yahoo.com

***Abstract*—Our research provides a unified and coherent presentation of the essential concepts and techniques of object–oriented file systems. It consolidates the results of research and development in the semantics and implementation of a full spectrum of information system facilities for object–oriented systems, including data modeling, querying, storage structures, composite objects and integration of a programming language. This approach presents a tool for building an object–oriented file system called object–oriented file system tool (or OOFS for short) for completing the development of a large object–oriented information system, and its associated applications development framework. First we present the technological objectives underlying the project. Then we present the process of developing the information system and detail its architecture and construction, concentrating on the areas in which object–oriented technology has had a significant role.**

Keywords-object serialization; object data modelling; object–oriented database; information system generator; web system generator

I. INTRODUCTION

A data model organizes data elements and standardizes how the data elements relate to one another. Object–oriented data modeling has achieved great popularity in recent years. The major factor contributing to its success is that object–oriented data modeling offers its users a high level of abstraction for the representation of information in a manner close to users' conceptual view of that information. We present a tool called OOFS to build an object–oriented file system. This can be used in the development of a large object–oriented information system, and its associated applications development framework. The primary focus for OOFS development is the implementation of a large object–oriented information system.

In our earlier work, we developed a customized software tool for automatically generating a complete Java program based on the values or parameters inputted by the user, called ISG [1] [2] [3]. ISG offers users interface screens for generating an information system and has six transformational functions – *building object–oriented file system (*OOFS*), linking to the next window, building data processing window, displaying data, previewing a designed window* and *printing data*. The advantage of ISG is that it uses object serialization mechanism to fill objects with data, which saves CPU execution time. The attributes of an object and its path need to be specified for ISG to translate it to Java code. Thus the program can store and retrieve data efficiently. It can also saves time in coding, debugging, testing and implementing an information system.

II. RELATED WORK

Some of the papers regarding how to store Java objects are "Reading Large Volumes of Java Objects from Database" [4], "A Framework for Object–Oriented Data Mining" [5], "A Composite Data Model in Object–Oriented Data Warehousing" [6], "Efficient object serialization in Java" [7], "Object Serialization Support for Object Oriented Java Processor" [8], etc.

In 2000, Raimund K. Ege [4] explores issues in his paper that arise when Java programs access objects stored in databases. They report on their experience with designing and implementing an approach that allows a Java program to pretend that all objects are in main memory, and relieving the Java program from most database housekeeping chores. The architecture is supported by APIs to an actual database: the API can map to an object–oriented database, a relational database via JDBC, or to files using object serialization.

In 2008, Linna Li et al. [5] proposed a system called Escher that is very suitable for describing knowledge for object–oriented data mining. Escher supports a variety of data types and can describe complex data. They also presented a framework for object–oriented data mining, where type information of data and semantic information of data model could be used to guide the data mining process. A specific data mining task, the frequent pattern discovery, is investigated under this framework.

In 1999, Wei–Chou Chen et al. [6] introduced a composite data model in which they proposed to store data in an object–oriented data warehouse. The data warehouse is an information provider that collects necessary data from individual source databases to support the analytical processing of decision–support functions. The data model forms new classes consisting of the attributes listed in the definitions of views and copies necessary class structure from the data source. The query performance of the data warehouse can thus be improved. The corresponding view creation and deletion algorithms were also proposed.

The authors in [7] state that object serialization is the ability to write the complete state of an object to an output stream, so that it can be recreated from the serialized representation at a later time. They also present a number of improvements to the serialization mechanism aimed at decreasing pickle sizes without visible degradation in the serialization performance. Through performance results, they show that it produces

(DOI Reference Number: 10.18293/SEKE2015-108)

pickles up to 50% smaller without degrading the serialization performance.

Another paper "Object Serialization Support for Object Oriented Java Processor" [8] introduces a functional unit which consists of a serialization and de–serialization unit along with the descriptors and pool to describe the stored serialized objects. This design can enhance the performance of Java based mobile devices which run applications that communicate with other similar applications very often. This design makes use of architectural features of processors.

Java is one of the stable object oriented programming languages which is widely used. In the papers mentioned above, the proposed methods do an efficient serialization in object oriented Java processor or applications. Many of the major projects in industry are developed using Java. A suggestion has also been made to enhance the Java serialization package to reflect this hardware enhancement on the overall performance.

III. OOFS System Architecture

Let's focus on aspects of OOFS that are particularly appropriate for its use in an e–Business system.

A. *OOFS Standard Model*

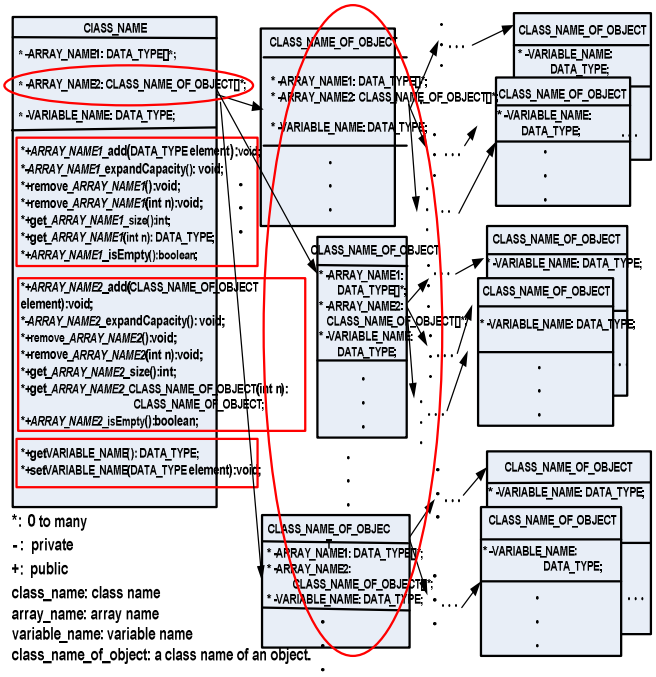

Figure 1. A file structure of OOFS standard model.

OOFS stores data using object streams rather than regular streams because Java has *persistence* in object–oriented circles [9], which means the object layout on disk will be exactly like the object layout in memory. So when an object (the first created object) is saved to disk, the memory addresses of objects that are created subsequently are stored in their associated array and are stored to the disk automatically. Since OOFS introduces this mechanism we design an OOFS standard data model in Fig. 1 to help users to design their file systems.

Fig. 1 shows a file structure of OOFS standard model. Since OOFS builds a file as an object stream file and uses arrays to manage their associated objects of the same class, a subsequent class is created whose objects are stored in an array of the previous class. Therefore if there are a number of object-oriented arrays in a class diagram, a number of subsequent class diagrams are shown as in Fig. 1 (see two oval marks).

A class diagram with an OOFS design has a class name at the top, its attributes are in the middle and methods which the class can execute are at the bottom. Attributes of OOFS contain variables and arrays.

A variable in OOFS is defined as VARIABLE_NAME: DATA_TYPE. A variable named VARIABLE_NAME can reserve memory locations to store a value. Data type DATA_TYPE of the variable decides what can be stored in the reserved memory. There are 9 data types supported by OOFS including byte, short, int, long, float, double, boolean, char and String.

OOFS provides two types of arrays are ARRAY_NAME1:DATA_TYPE[]* and ARRAY_NAME2: CLASS_NAME_OF_OBJECT[]* (See the first oval mark in Fig. 1). []* means the number of dimensional array. Currently OOFS only works for arrays that have up to three dimensions.

ARRAY_NAME1:DATA_TYPE[]* stores a fixed–size sequential collection of elements of the same data type (DATA_TYPE mentioned has 9 data types) and the array is named ARRAY_NAME1.

ARRAY_NAME2:CLASS_NAME_OF_OBJECT[]* describes array ARRAY_NAME2 created uses defined the constructor of the class named CLASS_NAME_OF_OBJECT which is used to access objects.

For a variable VARIABLE_NAME: DATA_TYPE, OOFS offers two methods to manage variable VARIABLE_NAME and whose methods are getVARIABLE_NAME() and setVARIABLE_NAME(DATA_TYPE element). getVARIABLE_NAME() returns the element stored in this variable VARIABLE_NAME and setVARAIBLE_NAME(DATA_TYPE element) sets the element stored in this variable VARIABLE_NAME (See the third rectangular mark in Fig. 1).

For an array ARRAY_NAME1: DATA_TYPE[]*, OOFS offers 7 methods to manage this array and which are ARRAY_NAME1_add(DATA_TYPE *element*), ARRAY_NAME1_expandCapacity(), remove_ARRAY_NAME1(), remove_ARRAY_NAME1(int *n*), get_ARRAY_NAME1_size():int, get_ARRAY_NAME1(int *n*):DATA_TYPE, ARRAY_NAME1_isEmpty() (see the first rectangular mark in Fig. 1).

For an array **ARRAY_NAME2: CLASS_NAME_OF_OBJECT[]*,** OOFS also offers 7 methods to manage the array and which are ARRAY_NAME2_add(CLASS_NAME_OF_OBJECT *element*), ARRAY_NAME2_expandCapacity(), remove_ARRAY_NAME2(), remove_ARRAY_NAME2(int *n*), get_ARRAY_NAME2_size():int, get_ARRAY_NAME2_CLASS_NAME_OF_OBJECT(int *n*): CLASS_NAME_OF_OBJECT, ARRAY_NAME2_isEmpty() [10] (see the second rectangular mark in Fig. 1). We take array

ARRAY_NAME2 for example and describe how these methods manage array ARRAY_NAME2.

ARRAY_NAME2_add(CLASS_NAME_OF_OBJECT *element*) adds the specified object *element* whose class name CLASS_NAME_OF_OBJECT to array ARRAY_NAME2.

ARRAY_NAME2_expandCapacity () creates a new array to store the contents of array ARRAY_NAME2 with twice the capacity of the old one.

remove_ARRAY_NAME2() removes all of objects from the array.

remove_ARRAY_NAME2(int *n*) operation consists of making sure the array is not empty and removes the specified object from the array using index *n*.

get_ARRAY_NAME2_size():int returns the number of objects in the array.

get_ARRAY_NAME2_CLASS_NAME_OF_OBJECT(int *n*): CLASS_NAME_OF_OBJECT returns an object whose class name CLASS_NAME_OF_OBJECT using index *n*.

The following uses as an example the e-Business system of the Eastland Company to describe these 7 methods in detail.

B. Applying OOFS Standard Model on Eastland e–Business System

Since OOFS is to build the file system of an information system and *for linking to the next window* and *for building data processing window* of ISG are to generate graphic user interface screens which are to input data and then store data. Now take Eastland e–Business for example to describe how we implement the file system of Eastland e–Business.

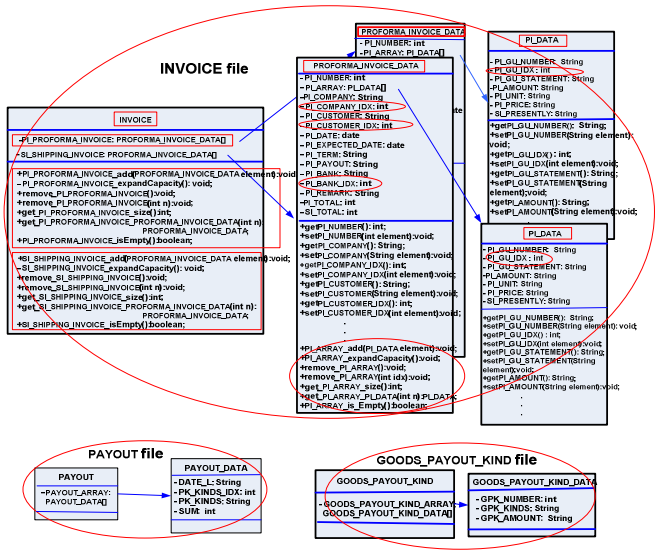

Figure 2. Class diagrams of six groups of classes from Eastland e–Business.

Consider Eastland e–Business that computes the monthly shipping amount, generates reports on their monthly earnings, profit, orders, etc. Let's focus on building the file system of Eastland e–Business and there are 17 groups of classes (classes that are related to each other through composition) in this system such as classes for foreign bank data, local bank data, invoice (including pro forma invoice, shipping notice), company data, product description, vendor purchase orders, products, single inventory statistics, goods expenses, types of expenses, monthly earning, profit etc. Then use OOFS standard model in Fig. 1 to illustrate the file structure of these 17 groups of classes. Now we document these designs using OOFS standard model to examine three of these groups, for invoice, goods expenses and goods shown in Fig. 2. It shows class diagrams of file INVOICE, file PAYOUT, file GOODS_PAYOUT_KIND. Each of files has its associated classes such as class INVOICE, class PROFORMA_INVOICE_DATA and class PI_DATA in file INVOICE, class PAYOUT and class PAYOUT_DATA in file PAYOUT, class GOODS_PAYOUT_KIND and class GOODS_PAYOUT_KIND_DATA in file GOODS_PAYOUT_KIND.

Taking file INVOICE for example (see the first oval mark in Fig. 2), there are two arrays in object of class INVOICE; array PI_PROFORMA_INVOICE is to store pro forma invoice and array SI_SHIPPING_INVOICE is to store shipping notice and both store objects of class PROFORMA_INVOICE_DATA. Class PROFORMA_INVOICE_DATA includes attributes – PI_NUMBER, PI_COMPANY, PI_CUSTOMER, etc. and an array PI_ARRAY which stores objects of class PI_DATA. Class PI_DATA includes attributes – PI_GU_NUMBER, PI_GU_IDX, PI_GU_STATEMENT, PI_AMOUNT, PI_UNIT, PI_PRICE and PI_PRESENTLY.

Another issue to consider is how an array manages its associated objects. Consider arrays PI_PROFORMA_INVOICE and SI_SHIPPING_INVOICE (see four rectangular marks at class INVOICE in Fig. 2) for illustrating how the object streams are stored in a file. Take array PI_PROFORMA_INVOICE for example and array PI_PROFORMA_INVOICE is used to manage objects of class PROFORMA_INVOICE_DATA and seven methods enclosed in the third rectangular mark are generated by OOFS that are methods PI_PROFORMA_INVOICE_add(PROFORMA_INVOICE_DATA element), remove_PI_PROFORMA_INVOICE( ), remove_PI_PROFORMA_INVOICE(int *n*), get_PI_PROFORMA_INVOICE_size(), PI_PROFORMA_INVOICE_isEmpty(), get_PI_PROFORMA_INVOICE_PROFORMA_INVOICE_DATA(int *n*) and PI_PROFORMA_INVOICE_expandCapacity().

If get_PI_PROFORMA_INVOICE_PROFORMA_INVOICE_DATA(int *n*) returns an object of class PROFORMA_INVOICE_DATA named A_PROFORMA_INVOICE1 and using this object to call method getPI_NUMBER() gets value of attribute PI_NUMBER. Therefore ISG can translate getting value of PI_NUMBER with A_PROFORMA_INVOICE1 into a command statement A_PROFORMA_INVOICE1.getPI_NUMBER(). Another command statement A_PROFORMA_INVOICE1.setPI_NUMBER(n) is to set a value *n* to attribute PI_NUMBER. When array PI_PROFORMA_INVOICE is created (means object of class INVOICE is created and array PI_PROFORMA_INVOICE in it), it is allocated a specific number of cells into which elements can be stored. Since we use a fixed–size data structure, at some point the array may become full. So method PI_PROFORMA_INVOICE_expandCapacity() will be called automatically to double the size of array PI_PROFORMA_INVOICE. Thus the file structure of

Eastland e–Business, with arrays and methods provided makes it convenient and efficient to retrieve data elements in any array. How users use OOFS to set parameters and translate to Java programs is discussed in [1].

C. How ISG Retrieves data from OOFS File Structure

If users specify a path for retrieving PI_GU_STATEMENT data as INVOICE/PI_PROFORMA_INVOICE/PI_ARRAY/ PI_GU_STATEMENT, then ISG uses this path to translate a Java program for getting PI_GU_STATEMENT. *INVOICE/PI_PROFORMA_INVOICE/PI_ARRAY/PI_PRICE* means that there are two arrays – PI_PROFORMA_INVOICE and PI_ARRAY and there are objects stored in their associated arrays.

The path indicates that an object of class INVOICE contains a PI_PROFORMA_INVOICE array. Each element of PI_PROFORMA_INVOICE array is an object of class PROFORMA_INVOICE_DATA. Each of these objects contains a PI_ARRAY array. Each element of PI_ARRAY array is an object of class PI_DATA. Each of these objects contains an attribute called PI_GU_STATEMENT. Therefore ISG knows how to get attribute PI_GU_STATEMENT by following steps using the above path specified. The command statements can be seen in [2].

1. Read an object named *THE_INVOICE* from file INVOICE, since PI_PROFROMAN_INVOICE is an array stored in the *THE_INVOICE* object.
2. Next ISG uses for loop to retrieve each object of class PROFORMA_INVOICE_DATA from the PI_PROFORMA_INVOICE array which is temporarily stored in AN_INVOICE object.
3. There is an array PI_ARRAY in object AN_INVOICE and ISG use this object to get each object of class PI_DATA stored in array PI_ARRAY. ISG also uses for loop to get each object of class PI_DATA as step 2 does.
4. From the retrieved objects of class PI_DATA, we can get the value of PI_GU_STATEMENT by using each of them.

IV. Experimental Results and Analysis

We analyze the efficiency obtained in terms of the OOFS CPU time using the two tools (ISG and DWL).

A. Analyzing Efficiency of OOFS CPU Time Using ISG

ISG generated an e–Business system for the Eastland Company by generating a total of 74 Java programs. There were 15 user interface screens for entering the shipping cost, monthly expenses, foreign manufacture's data, local manufacture's data, foreign bank data, local bank data, packaging data, about products, about the company, foreign customer shipments etc. The average time it took for translating these GUI screens was 14.46 milliseconds (not including I/O time) which is pretty good since ISG moves the parameters to the memory and uses fast access methods. For storing these parameters there are two files—HF file and FRAME file. An HF file is for storing data architecture such as field name, data type, array dimension, size of array and data type of array. A FRAME file is for storing graphic interface screen component and data path. Since OOFS moves everything to memory at once, it reduces the amount of time it takes to read a FRAME file (it took 288.93 milliseconds to move data). It took just 22.8 milliseconds for reading the *HF* file. So the total time for translating a screen was 326.19 milliseconds.

B. Analyzing Efficiency of OOFS CPU Time Using DWL

We used DWL [11] to implement an E–commerce web–based system that we call E–POLEMONG for POLEMONG Plastic Company. POLEMONG Plastic Company manufactures eleven types of products viz. telecommunication parts, auto parts, sports equipment, daily supplies, appliances, aquarium supplies, etc. E–POLEMONG displays six items viz. News, Products, About POLEMONG Company, Investor Information, Product Quality and Contact us. We translated 22 web pages using DWL for E–POLEMONG such as News and eleven different types of products like About POLEMONG, Product Quality, Investor Information, Contact Us, etc. We also applied OOFS to retrieve data and then to translate it to web-pages. The average translation time for these web pages was 545.5 milliseconds, with the shortest time being 330 milliseconds and the longest time being around 700 milliseconds.

ISG's and DWL's experimental results show that OOFS is scalable. Since our extents are implemented as one object in one segment, as the number of objects in the extension increased, the size of the extent object increased and that space is big enough to keep all the class extension objects needed for a transaction.

C. Another Advantage of OOFS – Join Approach

Note that ISG uses arrays and methods to create an information system that is convenient and efficient for retrieving data elements in any array. Another characteristic of this OOFS standard model is when the drop–down lists of a GUI screen has data items from other files, ISG can combine data from these files into one file. This mechanism is similar to SQL joins in a database. Since an index is an integer pointer (into an array), using this index can identify elements of the array. Therefore we can use the selected indices from a drop–down list to retrieve its associated data elements from the source file. This mechanism for drop–down lists combines data elements from two or more files into a file which is a Joined Approach. Therefore a Joined Approach can eliminate duplication of information when objects may have one–to–many relationships.

Figure 3: The Medical Supplier page of E–POLEMONG.

D. *How to Update Web Pages Quickly as the Company Produces New Products Frequently*

Figure 3 shows the Medical Supplier webpage of E–POLEMONG. It shows six types of hospital handrail image pictures. We describe the steps below on how to implement an application that translates it into a new webpage whenever a new product is introduced.

1. Execute *building data processing window* of ISG to generate an interface screen to input data – product name, its image file name, and data is stored into a file called MEDICAL_SUPPLIERFILE.
2. Use DWL to generate an HTML program called MEDICAL_SUPPLIER.html which shows a webpage similar to the one shown in Fig. 3. We wrote a translator to update Fig. 3 which calls PrintWriter to print formatted representations of objects to a text-output stream such as MEDICAL_SUPPLIER.html. There is a for loop in the translator which can be used to make changes to the RIGHTCOLUMN section in Fig. 3. It gets the number of hospital handrails from the MEDICAL_SUPPLIERFILE file. The command statements can be referenced from [11].

V. Conclusions and Future Work

An information system was developed using ISG for Eastland that involves computing the monthly shipping amount, generating reports on their monthly earnings, profit, orders etc. We had already established the usefulness of ISG in our earlier work.

It is convenient and easy to use DWL to generate a web–based system for any company or business. The web–based system enables customers to better understand the company and its products, which would result in increased sales. We illustrated this for a company in the paper. In the future, we hope to use this tool to develop customized web–based systems for other small and medium sized businesses. Both ISG and DWL can save time in writing, debugging and testing a program.

We established the usefulness of DWL in our earlier work which reduced the cost of producing software written in HTML. For example, using DWL we developed a customized web–based system for GREATYO sunglasses for sports and kids [11] and for the 7th Ubiquitous–Home Conference UHC2013 [3]. ISG has been used to generate an e–Business system for Eastland International Company [2]. Our research also offered a tool called W–Revised for creating customized websites. Because companies introduce new products frequently and the web pages of its site need to be updated frequently, it can be done conveniently using W–Revised generated by ISG and DWL [11].

Websites are often hacked by hackers seeking to compromise the corporate network. Also programs are sometimes downloaded without the users consent or knowledge when they visit a web site (drive–by download). As a result, industry is paying increased attention to the security of the web applications themselves in addition to the security of the underlying computer network and operating systems. In order to keep a website clean and secure, we are trying to find a good solution to defend the websites. Therefore we hope that in the near future our software tools might be included in business applications as software as a service (SaaS).

ACKNOWLEDGMENT

Eastland e–Business System was developed in collaboration with Eastland International Company for implementing an e–Business System. E–POLEMONG was developed in collaboration with POLEMONG Plastic Company for developing a web–based system for their business. We would like to thank them for giving us this opportunity to work with them and test our tools for generating the information and the web–based systems.

REFERENCES

[1] Ling–Hua Chang, Sanjiv Behl, "An Efficient Information System Generator," 4th Asian Conference on Intelligent Information and Database Systems, pp. 286–297, 2012.

[2] Ling–Hua Chang, Sanjiv Behl, Tung–Ho Shieh, "Amazing Use of ISG for Implementing Information Systems," 2014 International Conference on Information Science, Electronics and Electrical Engineering, (iseee2014), pp. 1980–1985, April 26–28, 2014.

[3] Ling–Hua Chang, Tung–Ho Shieh, Sanjiv Behl, "Amazing of Using ISG on Implement a Web–Based System," 14th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'13), pp. 44–49, 2013.

[4] Raimund K. Ege, "Reading Large Volumes of Java Objects from Database," TOOLS '00 Proceedings of the Technology of Object–Oriented Languages and Systems (TOOLS 34'00), pp. 117–124 , Aug. 2000.

[5] Linna Li, Bingru Yang, Faguo Zhou, "A Framework for Object–Oriented Data Mining," the 5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2008), pp. 60–64 , Oct. 2008.

[6] Wei–Chou Chen, Tzung–Pei Hong and Wei–Yang Lin, "A Composite Data Model in Object–Oriented Data Warehousing," TOOLS '99 Proceedings of the 31st International Conference on Technology of Object–Oriented Language and Systems, pp. 400–405, 1999.

[7] L. Opyrchal, A. Prakash, "Efficient object serialization in Java," Proceedings of 19th IEEE International Conference on Distributed Computing Systems Workshops on Electronic Commerce and Web–based Applications., pp. 96–101, 31 May 1999–04 Jun 1999.

[8] Joe Cheri Ross, Dr. Priya Chandran, "Object Serialization Support for Object Oriented Java Processor," IEEE Transactions of Information Technology, pp. 1–6, vol. 3 Aug. 2008.

[9] C.S. Horstmann, G. Cornell, Core Java Volume I–Fundamentals, 8th ed. Sun Microsystems Press, Prentice Hall, New Jersey, 2008.

[10] John Lewis, Joseph Chase,, "Java Software Structures designing and using data structures," Pearson Education Inc., 2005.

[11] Ling–Hua Chang, Sanjiv Behl, Tung–Ho Shieh, "W–Revised: an Amazing Tool for Creating Customized Websites", 2014 IEEE, DOI 10.1109/ICSAI.2014.7009333, pp.465–470, 2014.