

Automating the Collection of Object Relational Database Metrics

Samer M. Suleiman

Department of Computer Science, Faculty of Computer
and Information Technology,
Jordan University of Science and Technology,
Irbid, Jordan

Qasem A. Al-Radaideh, Bilal A. Abulhuda, Izzat M.
AlSmadi

Department of Computer Information Systems, Faculty of
Information Technology and Computer Sciences,
Yarmouk University,
Irbid, Jordan

Abstract— The quality of software systems is the most important factor to consider when designing and using these systems. The quality of the database or the database management system is particularly important as it is the backbone for all types of systems that it holds their data. Many researches argued that software with high quality will lead to an effective and secure system. Software quality can be assessed by using software measurements or metrics. Typically, metrics have several problems such as: having no specific standards, sometimes they are hard to measure, while at the same time they are time and resource consuming. Metrics need also to be continuously updated. A possible solution to some of those problems is to automate the process of gathering and assessing those metrics. In this research the metrics that evaluate the complexity of Object Oriented Relational Database (ORDB) are composed of the object oriented metrics and relational database metrics. This research is based on common theoretical calculations and formulations of ORDB metrics proposed by database experts. A tool is developed that takes the ORDB schema as an input and then collects several database structural metrics. Based on those proposed and gathered metrics, a study is conducted and showed that such metrics' assessment can be very useful in assessing the database complexity.

Keywords- Object Oriented Relational Database; Metrics; Software Quality.

I. INTRODUCTION

The need to store and retrieve data efficiently relative to traditional file structure retrieving was the basic motivation to introduce the relational databases, which is basically the process of representing the data on the form of a collection of related relations (i.e. tables) [1].

Due to the increasing demand on more efficient techniques to store, retrieve and represent complex and huge data types such as images, a new data model is presented with the inspiration of object oriented programming languages. Object Oriented Database (OODB) emerged to meet these demands. OODB is the process of representing the data in a form of complex columns (i.e. objects) that contain attributes and operations to access them [1] [2] [3] [4].

Object Relational Databases (ORDB) have recently evolved for two reasons: the first is the limitation of the traditional relational databases against the increasing demands of the huge

applications for storage and fast retrieval of data. The second is the great complexity of the pure OODB [5] [2]. The integration between the relational and the object oriented methodologies could overcome some of the drawbacks that are known in the relational databases, as well as, enable developers to utilize the powerful features of the relational and object oriented databases such as simplicity and usability [5] [6].

Generally, an ORDB system has two main natures: (1) the dynamic nature which reflects the external quality of the system that can be collected from the system at runtime (i.e. dynamic or runtime metrics) (2) the static nature of the system which reflects the internal quality and that can be measured at the design time (i.e. static metrics) [7].

In this scope, metrics are tools to show indications that can help software management in several aspects. For example it can help facilitating the maintenance effort of the schema and hence improve the quality and reduce the complexity of the resulting schema [5]. Controlling the quality of the database system in the design phase may help in preventing the whole system from collapsing in the later phases (e.g. the implementation phase). It also saves the cost and time for the development process in general [8]. The assumption here is that these metrics are standardized and formulated in order to be measured as numbers, and thus facilitates the automation process.

The main metrics for ORDB are: Table Size (TS), Complexity of Weighted Methods (CWM), Cohesion Between Methods (CBM), Coupling Between Objects (CBO), Number of Inherited Properties (NIP), Referential Degree (RD), and Depth in the Relational Tree (DRT) [9] [5] [8] [10].

Several researchers deduced that the value of automation process comes from making the collection and the evaluation process for software and system metrics easier in comparison with manual techniques. In an example in this direction, Stojanovic and El-Emam [11] constructed an object oriented prediction model that can detect the faulty classes based on previous data. They described an open source tool for C++ source code that can calculate the object oriented metrics from the interface specifications at the design phase; these metrics are size, coupling, and inheritance.

There are several challenges facing the evaluation of the metrics. One of the main challenges is the ambiguity in the definitions and formulation of these metrics in addition to the nature of the process of collecting, processing and evaluating those metrics [9] [5] [6]. According to [8], most of the developed software metrics are programs' oriented and they are not dedicated to database systems.

Al-Ghamdi *et al.* [11] described three tools for the collection; analysis and evaluation of object oriented metrics. The tools are: (1) Brooks and Buells: which contains: a parser, a query engine and a model class hierarchy. (2) The second tool for analyzing C++, and (3) The third tool for gathering OO metrics. In addition, they built their own tool which collects and measures inheritance coupling in object oriented systems. They compared their tool with the three other tools in terms of differences and main features.

Scotto *et al.* [12] mentioned that there is no standardization for software measure and metrics where they suggested using an intermediate abstraction layer to handle the frequent changes on the extraction process for these metrics. They used an automated tool to collect several web metrics. They recommend to separate the two primary activities for any tool that is supposed to measure the metrics, these two activities are the process of extraction and storing the information from the source code, and the second one is the process of analyzing these information and get some conclusion from it.

AL-Shanag and Mustafa [13] proposed and built a tool to facilitate the maintenance and understanding effort for C# source code. The assumption is that software maintenance process can benefit indirectly from software metrics through predicting complexity and possible areas of problems in the software code. Such metrics can help the developer and maintenance software engineers in understanding the source code of programs especially those that have no or little documentation. The proposed tool collects several code elements such as: interfaces, classes, member data and methods from the source code. The tool can also collect the following code metrics: Weighted methods per Class, Depth of inheritance tree, Lines of code, Number of public methods, and Data access metric.

As for this research, the main objective is to build a tool that can collect and evaluate ORDB metrics that will enable designers to calibrate and tune the database schemas to increase usability, maintainability and quality for schemas. The automation of this process becomes essential to overcome the complexity of the evaluation process. Based on the most common definitions of these metrics and units, this research automates the collection of these metrics and realizes them by units scale based on some formal definitions for these metrics. For testing purposes, the proposed automation tool assumes that the input schema is syntactically correct with respect to a standard MS SQL database (particularly MS SQL 2003).

II. RESEARCH BACKGROUND

The challenge in OORD metrics is to find suitable definitions and formulas in order to measure these metrics. These measures are assumed to facilitate the process of

controlling the quality of the schema which will enhance the overall performance of the associated information systems [9].

The evaluation of the quality of the database schema must be validated formally. These metrics have to be validated formally through both theoretical and practical approaches. Piattini *et al.* [8] stated that the concentration is on the practical approaches through developing practical experiments. They developed an experiment to validate the ORDB metrics in order to ensure the benefits of these metrics. They proof the formality and validity of these metrics by repeating the same experiments twice in CRIM center in Canada and University of Castilla-La Mancha in Spain and get similar results. These results showed that Table Size (TS) and Depth in the Relational Tree (DRT) can be used as indicators for the maintainability of database tables.

In ORDB, the table consists of two types of columns: the first is the Standard Column (SC) which is defined as integer or dynamic string data types, and the other one is the Complex Column (CC) which is the User Defined Type (UDT) column. According to this categorization, the metrics also are classified as table related metrics. Those can be applied to a table which includes: TS, DRT, RD, and Percentage of Complex Columns of tables (PCC), Number of Involved Classes (NIC), and Number of shared classes (NSC). Other metrics are applied to schema which includes the DRT, RD, PCC, NIC, and NSC [8].

ORDB schema requires an extra metric due to additional capabilities which come from the object oriented features in order to ensure its internal quality which will be reflected on its external quality in terms of understandability, usability and reliability [9].

A. Metrics of ORDB

Justus and Iyakutti [9] defined and formulated the metrics of ORDB based on three schemas. These metrics are:

- *Table Size (TS)*: It represents the summation of the size of both the simple columns (SC) which includes the traditional attributes data types, such as integer and varchar, and the complex columns (CC) which represents the User Defined Types (UDTs). The larger the value of this metric leads to a higher maintenance cost [9] [8]. It is calculated as follows:

$$\mu(\text{TS}) = \sum_{i=0}^n \text{SC}_i + \sum_{j=0}^m \text{CC}_j \quad (1)$$

Where μ is a metric function, SC represents the Simple Columns and n represents their numbers, CC is complex columns and m's are their numbers.

- *Complexity of Weighted Methods (CWM)*: It represents the summation of the whole complexities for each weighted method in the table [9] [14]. It is calculated as follows:

$$\mu(\text{CWM}) = \sum_{i=0}^m C_i \quad (2)$$

Where C_i is the complexity of method i .

- *Cohesion Between Methods (CBM)*: It measures the connectivity between two or more methods and it is measured as the proportion between the similar used attributes in the methods of the class to the total number of attributes. High cohesion is desired which indicates that we are grouping together related methods. Low cohesion

should have a negative impact on maintenance [9] [7] [14] [3]. It is calculated as follows.

$$\mu(A) = \frac{I_1 \cap I_2 \cap I_3 \cap \dots \cap I_n}{V_s} \quad (3)$$

Where I is the instance of the attributes for the class, and V is the total number of variables.

- **Coupling Between Objects (CBO):** It represents the dependency or the connectivity between two methods that exist in two different classes. While high cohesion is desired, high coupling is not as it complicates the design and will complicate maintenance effort, update and reuse [9][7]. It is calculated as follows:

$$\text{Count}_{cp} = \sum_{i=0}^n I_i + \sum_{j=0}^m \text{Min}v_j \quad (4)$$

Where $Min v_j$ is the number of methods called and the average number of arguments involved in each invocation.

- **Number of Inherited Properties (NIP):** It represents the number of the properties that have been inherited by the child and thus its value determines high coupling complexity as a trade off from its reuse [9]. It is calculated as follows:

$$\mu(\text{NIP}) = \sum_{i=0}^n I_i + \sum_{j=0}^m M_j \quad (5)$$

Where $M_j = [\text{No. of types} * \text{CWM}] + [\text{No. of methods} * \text{CBO}]$.

- **Referential Degree (RD):** It represents the number of foreign and reverse refernce keys in the database schema [9] [15] [8]. It is calculated as follows:

$$\mu(\text{RD}) = \sum_{i=0}^n \text{FK} + \sum_{j=0}^m \text{Rvref} \quad (6)$$

Where FK is the foreign key and $Rvref$ is the inverse reference.

- **Depth in the Relational Tree (DRT):** It represents the longest referential path between the tables in or of a database schema [9] [10]. It is calculated as follows:

$$\mu(\text{DRT}) = \sum_{i=0}^n d_i \quad (7)$$

Where d is the distance between the related tables.

B. Metrics Units

In an attempt to unitize the ORDB metrics, Justus S and K Iyakutti [9] proposed to formulate and calibrate some of these metrics. This is because there is a great need for a standard scale for these ORDB metrics to be correlated with software code metrics. Justus and Iyakutti [9] proposed some units for this purpose based on some experimental studies. The units are:

- **Column Complexity (clm) unit for the TS metric.** Figure 1 shows the relation between TS and the cost in term of *clm* unit.

If the value is in one of the scaled ranges then we can conclude its cost and complexity. For example if the TS value is 50 *clm*, this means that the maintenance table cost is between optimal and low scale.

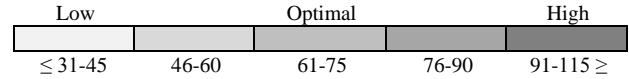


Figure 1: The Table Size unit scale.

- **Number of interactions per variables set (intr/vs) unit.** It is used to measure the cohesion metric as calculated in equation (3).
- **Number of messages imported or exported per interaction (msgs/intr) unit.** It is used to measure the coupling metric as shown in equation (4).

For both *intr/vs* and *msgs/intr*, Figure 2 shows the relation between these two measures against reusability and maintainability where high *intr/vs* indicates a high class reusability. However, the situation with *msgs/intr* is the opposite which means that when *msgs/intr* is high, this indicates a less class reusability.

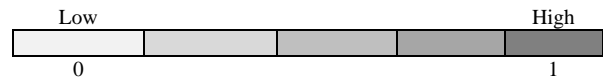


Figure 2: The COM and CBO Units Scale.

- The *laxity* unit is used to measure the reusability metric (NIP). The reusability denotes the usage of the same class-type another time in another class or type. Figure 3 shows the relationship between the reusability of the class-type and this unit. The higher the value of *laxity* the more the probability for the class-type to be reused.

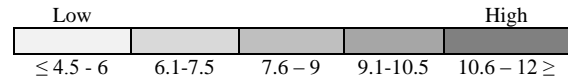


Figure 3: The NIP laxity unit scale.

III. THE DEVELOPED TOOL

The goal of developing this tool is to automatically collect and evaluate ORDB metrics. The proposed tool should enable designers to calibrate and tune the database schemas to increase usability, maintainability and quality for schemas. The proposed tool consists of three main modules: The tokenizer, the lexical analyzer, and the metrics calculator. The lexical analyzer is considered to be the main module of the tool. The architecture of the developed tool is illustrated in Figure 4 and the basic data model for the tool is illustrated in Figure 5.

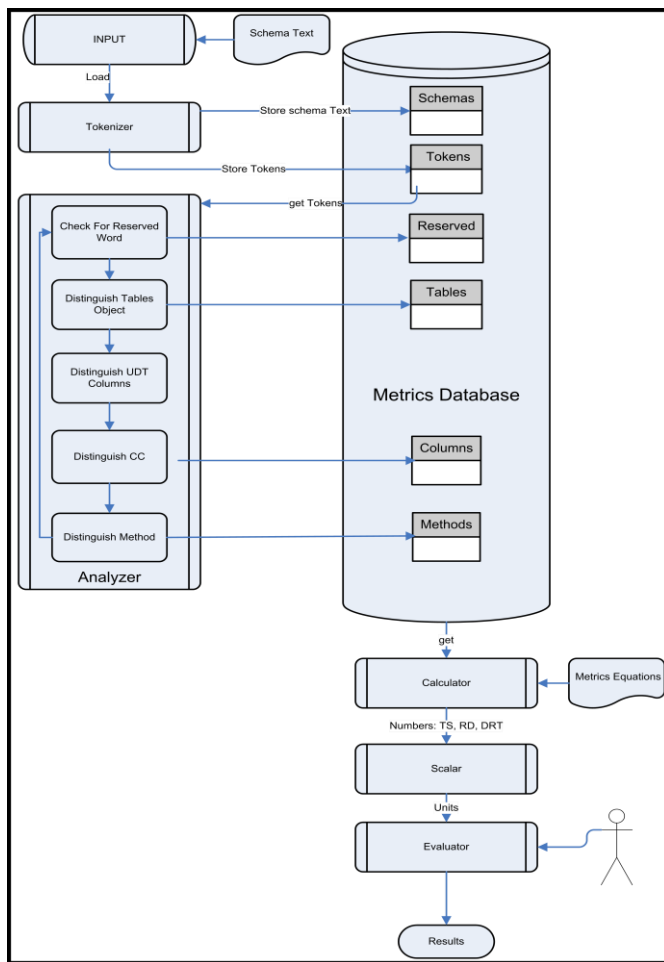


Figure 4: The Lexical Analyzer Architecture

A. Tokenizer

The tokenizer aims to facilitate the evaluation process by reading the stream of the text file (i.e. the schema) and tokenize it by recognizing each word individually. In this phase, there is no need to build the relationships between these tokens where the analyzer will handle this job. Tokens will be stored in the database and tagged in order to be used later. Some of these tokens will not be stored in the database for efficiency reasons and because it will not produce any relevant information to the process such as punctuation marks and other language related symbols.

The tokenizer also stores the schema as an entry to the database and links it with other objects and artifacts, in order to create a dataset and accomplish related statistics to obtain a design trend for each dataset. It is worth to mention here that some metrics could be calculated directly such as Lines of

Code (LOC). As a summary for this part: the input for the tokenizer is a schema text, and the output is a set of tokens.

B. Analyzer

The analyzer is the main part of the tool. It actually accomplishes a great percentage of the evaluation process for the schema. It acts as the bridge between the other parts of the tool. The analyzer starts by reading the tokens from the tokenizer and loads them onto the computer memory in order to be processed and classified into the basic objects for the evaluations. These objects are: the tables, the complex columns and their methods. The challenge here is to reserve the relational structure of the database that will be used subsequently.

The analyzer will check each token against a set of constraints. It will check if the token is an SQL reserved word and if this word is related to the measures or not. For example the table name which can be known if it appears after the two reserved words: “create table” or at least the word, table, hence will be stored in the database and connect all the following distinguished tokens with this entry until the next table name appears.

The tool will store all the information that the metrics’ formulas mentioned earlier may need to be collected in efficient and smooth manners. The design of the tool is based on separating the analyzer from the rest of other modules. This may help for any future changes on the metrics equation and guarantee that these changes will not affect the analyses process.

The analyzer distributes the tokens in different connected relational tables and ensures that all the tokens have been stored in the exact place. The equation metrics can easily be applied on these values that we can get from the tables. To summarize: the analyzer inputs are stored tokens, the output from the analyzer categorizes tokens are stored in related tables.

C. The Calculator Engine

The calculator engine module uses the information stored in tables to calculate the different metrics presented in section 2. The results are saved in the database associated with each schema for later possible revisions. The calculator module has two sub-modules: the scalar and the evaluator. The scalar role is to map the result of each equation against the scales presented in section II.

The scalar gives meaning to these numbers by assigning a suitable unit to each one of them. The Evaluator evaluates the overall quality of the schema based on the quantized numbers that have been already obtained from the previous units.

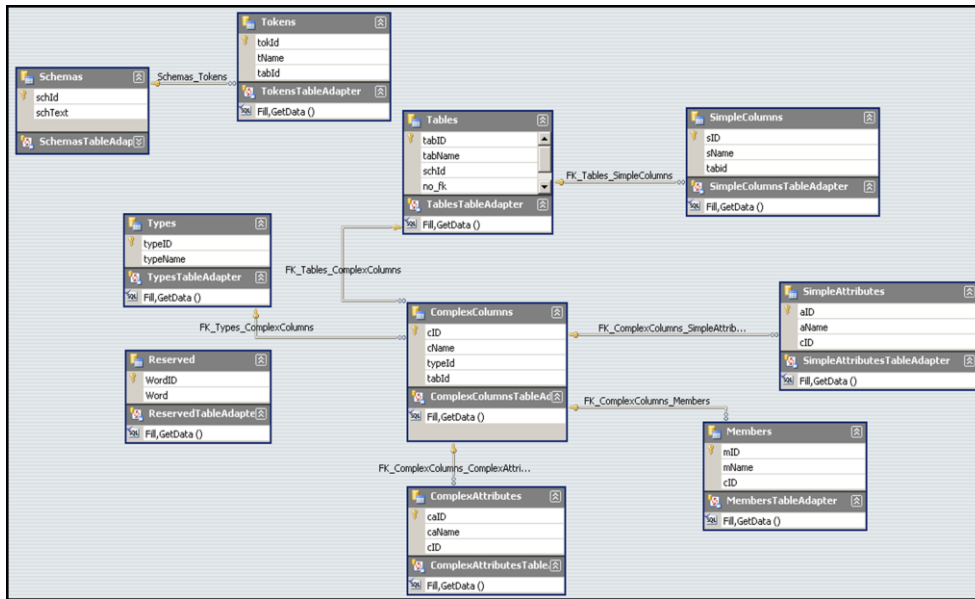


Figure 5: The Tool Data Model

IV. EXPERIMENTS AND EVALUATION

In order to evaluate the tool we will provide a complete example that shows how the tool is used to accomplish the tasks. Figure 6 represents a small part of a schema definition adopted from [9] that contains several tables. Table 1 represents the analysis for the table’s tokens. Figure 7 shows a sample script for creating a “person_t” type for the same schema definition.

```
CREATE TABLE Tab_Staff
(
    emp_no varchar(4),
    person_info person_t,
    do_joining date,
    working_department department_t,
    work_for varchar(7) FK Tab_Student (roll_no) inverse ref );
CREATE TABLE Tab_Student
(
    roll_no varchar(7),
    person_info person_t,
    in_department department_t,
    ward_for varchar(4) );
```

Figure 6: A Sample Schema Definition Script (Justus and Iyakutti, 2008)

As mentioned earlier, the main input for the tool is the schema script (written in SQL 2003 standards) and it is assumed to have object-oriented and relational features that are syntactically correct. The schema is read by the tool as a stream of words or tokens. The main two segments are: the schema definition segment and the implementation segment. They are separated using a special tag or flag.

TABLE 1: TABLES OF THE SAMPLE IN FIGURE 6

tabID	tabName	schID	No_fk	No_rev
1	Tab_Staff	1	1	1
2	Tab_Student	1	0	0

```
CREATE TYPE person_t
(
    Name varchar(20),
    Gender varchar(1),
    Birth_date date,
    Address_info address_t,
    MEMBER FUNCTION set_values () RETURN person_t,
    MEMBER PROCEDURE print_person () );
```

Figure 7: A Sample Schema definition for UDT

The tool analyzes this UDT table and relates each member to this new defined type:

- Simple attributes: those represent the standard types just like the simple columns each recognized one is stored in “SimpleAttributes” table and relates to its entry in the CC table. Table 2 represents the record Instances for the “SimpleAttributes” table after the tool reads the script.

TABLE 2. SIMPLE ATTRIBUTES INSTANCES

aID	aName	typeID
1	Name	1
2	Gender	1
3	Birth_date	1

- Complex Attributes: the other possible member items for the UDT is the complex column which is either stored on the “Types” table or need to be treated as we did with the same item member in table type. It can be related to one of the CC entries. Table 3 represents the record Instances for the “ComplexAttributes” table after the tool reads the script.

TABLE 3. COMPLEX ATTRIBUTES INSTANCES

caID	caName	typeID
1	Address_info	1

- Member methods. These members are recognized by the keywords PROCEDURE and FUNCTION, so once the tool has read one of these tokens it will consider the next one as a member function and insert it to the “Members” table and relates it to its entry in the CC table. Table 4 represents the record Instances for the “ComplexAttributes” table after the tool reads the script.

TABLE 4: MEMBERS INSTANCES

mID	mName	typeID
1	set_values	1
2	print_person	1

After this analysis is accomplished, all the necessary information for the schema definition are already stored in the database tables. The following subsection describes the calculation for the Tables Size, RD and DRT metrics, by using the stored data.

A. Calculations

The following sub sections present the calculations process for Table size metrics, Referential Degree, and Depth of Referential Tree

The Calculation of TS

In order to calculate the table size, the following information has to be retrieved from relevant tables:

- “Tables”: choose every record which represents a unique table, and retrieve the “tabid”, in order to be used to retrieve the related records from the other tables.
- “SimpleColumns”: the tool retrieves the related simple columns records from the table for a specific “tabid” value through the “FK_Tables_SimpleColumns” relation (see Figure 8). The number of the retrieved records is stored temporarily in order to be used later.
- “ComplexColumns” the tool retrieves the related complex columns records from the table for a specific “tabid” value through the “FK_Tables_ComplexColumns” relation. The number of the retrieved records is stored temporarily in order to be used later.
- The tool checks some other tables which are: “SimpleAttributes”, “ComplexAttributes”, and “Members”. It retrieves all the related records for the type of the complex columns and stores the count sequentially.
- The retrieving process is making use of the following relation between related tables:
FK_Types_ComplexColumns,
FK_ComplexColumns_SimpleAttributes,
FK_ComplexColumns_Members
FK_ComplexColumns_ComplexAttributes.

The result of this activity is a number that represents the size of the specific type that is mentioned in the table definition. The calculation is based on equation (1) presented in section 2 and the extracted size is stored on the “size” field of the “Types” table. Table 5 illustrates these calculations.

TABLE 5: UDT INSTANCES WITH SIZES

typeID	typeName	Size
1	person_t	14
2	department_t	16
3	address_t	7

By summation of all the retrieved values from these tables, the tool can calculate the size for each table (i.e. number of records from the “SimpleColumns” table + size of the existing UDT type(number of records from “SimpleAttributes” table + number of records from “ComplexColumns” table + number of records from the “Members” table). This is illustrated in Table 6.

TABLE 6: TABLE SIZES

tabID	tabName	schID	No_fk	Size
1	Tab_Staff	1	1	22
2	Tab_Student	1	0	23

To understand these numbers the tool will check them against the proposed scale as in Figure 1. This indicates that tables in this schema fall in the first scale ($\leq 31-45$) and they have low complexity level. Comparing this automatic measurement process with the manual calculations presented by Justus and Iyakutti [9] for TS metric for the same sample schema, the tool shows the same results.

The Calculation of RD metric

The calculation of this metric depends on equation (6) presented in section 2. The tool looks for the existence of the keywords “FK” which stands for foreign key and for “inverse ref” which stands for inverse reference that means the relation is bidirectional between the two subject tables.

The tool counts the frequency for FK token in the schema definition and stores the summation on the “no_fk” field for each table as illustrated in Table 1. The same process is applied on the inverse reference to get the number that represents their counts. In order to calculate the value for this metric, the tool adds the two summations (i.e. sum of FK’s and sum of inverse ref’s) for each table. Thus we can conclude the complexity of the schema according to the value for this metric. The higher the value of RD means the higher the level of complexity for the schema. Comparing results with [9] for this metric for the same sample schema, the tool gets the same results as they had.

The Calculation of DRT metric

The calculation of this metric depends on equation (7), in which the tool stores the referential path between tables by analyzing the foreign key constraint. It stores the id for the referencing and referenced table. It compares the frequency of the same “tabid” value in both columns and then counts this frequency to get the depth of the referential tree.

The tool gets the length of DRT as a single number that represents the number of tables associated with this relation. Each time it counters the FK keyword, it stores both tables name in the “Tree” table. The first table name after FK

keyword is stored in the “DetailTab” field where as the current processing table is stored in “MasterTable” field. Table 7 illustrates these findings:

TABLE 7: DRT FOR THE SAMPLE SCHEMA

MasterTable	DetailTab
Tab_Student	Tab_Staff
Tab_Staff	Exam

The tool then performs an SQL Statement that counts the existence frequency of each table in both columns for each record in the “Tree” table.

B. Tool Evaluation

Table 8 summarizes the results of this research and the output for the developed tool compared with Justus and Iyakutti [9] manual work to calculate the table size. The results in Table 8 show that the tool gets the same results in addition to storing each object in a relational manner in order to be retrieved when they are required. Justus and Iyakutti [9] calculated neither the “Exam” table size nor the “subject-t” type size. Our tool calculates them and compared with the results of manual calculations which are the same.

The same comparison is made between the two approaches with respect to the remaining calculated metrics; the Referential degree (RD) and DRT. The result is illustrated in Table 9, which shows that the automation process facilitates the evaluations for these metrics since the manual approach requires human memorization and tracing.

Figure 8 presents a screen shot of the analysis process that appears for the “Get Tokens” button. All the analyzed tokens appear in the targeted grid view.

V. CONCLUSION AND FUTURE WORK

The evaluation process for the software system quality in general is said to be complex especially for the ORDB metrics. This is due to different aspects such as the lack of formal definitions and standard evaluation. In addition to the lack of such tools that are capable to perform this evaluation.

However, one key point that was investigated in this research is the need to separate the analysis process from the evaluation and calculations processes regarding to these

metrics. This separation makes the two main processes independent from one another and handles, partially, the problem of changing the metrics evaluation or definition.

The developed tool performs the analysis process on the schema definition, separates different ORDB artifacts and stores them separately. These artifacts include: tables, their simple columns, their new defined types for the complex columns and the objects created for the complex columns which includes the simple attributes and the members of these objects (i.e. procedures or functions).

The automation process facilitates metrics gathering and evaluation and gives the designers and developers more capabilities and perspectives to ensure the quality of the ORDB systems. It performs the same equations proposed by [9] for TS, DRT, and RD metrics and gets the same results in terms of accuracy compared with manual metrics calculation while improving the performance through calculating those metrics automatically.

The tool is adaptable to the changes of the metrics equations since it separates the analysis process from the evaluation process and this may be useful for the standardization effort by tuning only the evaluation parts of this tool. The analysis may not have to be changed or it may require a little modification. This can help in the continuous evolution of metrics’ formulas’ construction and assessment.

In future, the automation process should be extended to include the remaining proposed metrics such as: (COM) and (CWM). It should be also extended to include more metrics that will be further investigated through looking at different database systems. Another future issue to deal with is that some proposed metrics have no formal scale that enables the designers to conclude the quality of the schema in terms of its complexity and maintainability, and thus future work may define a formal scale for these kinds of metrics.

It is recommended to extend the automation process to include different schemas written in different databases and formats such as: Oracle and MySQL. Once these modifications are implemented, there is a need to build a dataset to test them and calibrate their results. The tool may be extended to have the ability to obtain the schema from an existing database and perform the same process as it did with the text schema format.

TABLE 8: COMPARISON BETWEEN MANUAL AND AUTOMATE TABLE SIZE CALCULATIONS

Objects		Type	Manual	The Tool					
Tab_Staff		Table	23	23=3+14+6					
	working_department	Department_t	6	Select count(*) from simplecolumns	3				
				person_info, working_department	Person_info	Working_department			
	person_info	Person_t	14	Select count(*) from simpleAttributes	3	2			
				Select count(*) from complexattributes	7	0			
				Select count(*) from members;	4	4			
	do_joining	Simple Column	1	Total					
work_for	Simple Column	1							
emp_no	Simple Column	1							
				14					
				6					
Tab_Student		Table	22	22=2+14+6					
	person_info	person_t	14	Select count(*) from simplecolumns	2				
	in_department	department_t	6		person_info	in_department			
	ward_for	Simple Column	1	Select count(*) from simpleAttributes	3	2			
	roll_no	Simple Column	1	Select count(*) from complexattributes	7	0			
				Select count(*) from members;	4	4			
				Total					
				14					
				6					
Exam		Table		13=3+4+6					
				Select count(*) from simplecolumns	3				
					Subject_info	for_department			
	Subject_info	subject_t		Select count(*) from simpleAttributes	4	2			
	for_department	department_t		Select count(*) from complexattributes	0	0			
	Exam_code	Simple Column		Select count(*) from members;	0	4			
Exam_date	Simple Column		Total						
Staff_charge	Simple Column					4			
				6					
department_t		UDT	6	6					
	Name	Simple Column	1	Select count(*) from simpleAttributes	2				
	HOD	Simple Column	1	Select count(*) from complexattributes	0				
	set_values	MEMBER	2	Select count(*) from members;	4				
	print_object	MEMBER	2	Total					
				6					
Address_t		UDT	7	7					
	Door_no	Simple Column	1	Select count(*) from simpleAttributes	5				
	Area_name	Simple Column	1	Select count(*) from complexattributes	0				
	Zone_name	Simple Column	1	Select count(*) from members;	2				
	City	Simple Column	1	Total					
	Pin_code	Simple Column	1				7		
	set_values	MEMBER	2						
Person_t		UDT	14	14=3+7+4					
						Address_info			
	Name	Simple Column	1	Select count(*) from simpleAttributes	3	5			
	Gender	Simple Column	1	Select count(*) from complexattributes	1	0			
	Birth_date	Simple Column	1	Select count(*) from members;	4	2			
	Address_info	Address_t	7	Total					
	set_values	MEMBER	2				7		
print_person	MEMBER	2	7						
				14					

TABLE 9: A COMPARISON BETWEEN MANUAL AND AUTOMATE RD AND DRT CALCULATIONS

Table Name	Foreign Key Statement	Manual		DRT	Automation		DRT
		RD	inverse ref		RD	inverse ref	
Tab_Staff	work_forvarchar (7) FK Tab_Student (roll_no) inverse ref	1	1	Memorize master, detail	Foreign Key inverse ref	counterFK + counterINVREF=2	Store Master, Detail
Tab_Student		0	0			0	
Exam	Staff_chargevarchar (4) FK Tab_Staff (emp_no) inverse ref	1	1	Memorize master, detail	Foreign Key inverse ref	counterFK + counterINVREF=2	Store Master, Detail
				Trace for detail for each memorized master			Select count(*) where master=detail from tree=1

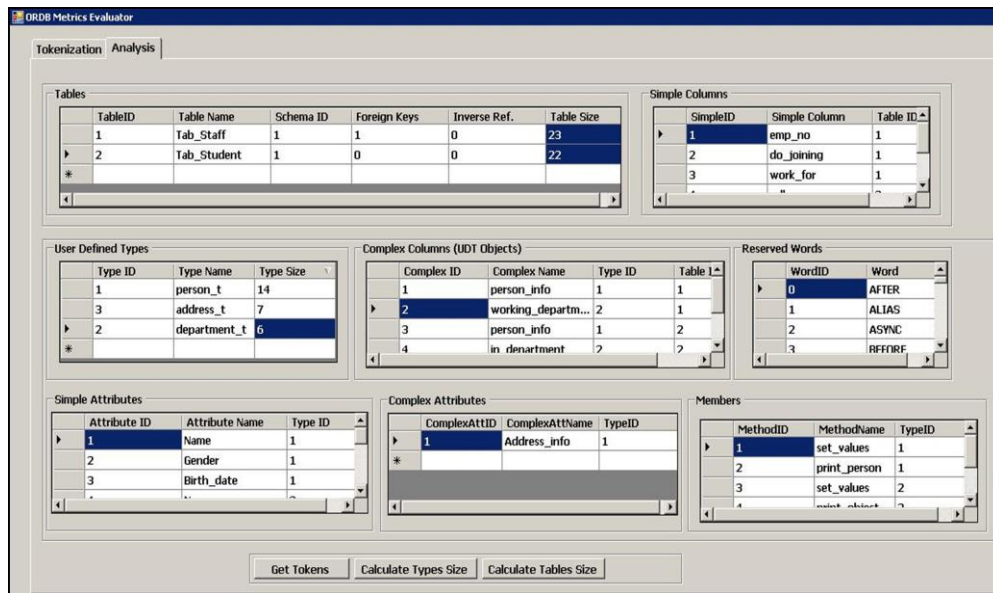


Figure 8: Screen Shot of Results

REFERENCES

- [1] Elmasri R. and Navathe S., "Fundamental of Database Systems", 5th edition, 2007.
- [2] Behm A., Geppert A., and Dittrich K., "On The Migration Of Relational Schemas and Data to Object-Oriented Database Systems" In Proc. of the 5th International Conference on Re-Technologies in Information Systems, Klagenfurt, Austria, December, 1997.
- [3] Vaishnavi V., Puroo S. and Liegle J. "Object-oriented product metrics: A generic framework", Information Sciences 177 (2007) pp. 587–606.
- [4] Van Zyl P. and Boake A., "Comparing the Performance of Object Databases and ORM Tools" in Proceedings of the 2006 annual research conference of SAICSIT, 2006.
- [5] Baroni A.L., Coral C., Brito e Abreu F., and Piattini M., "Object-Relational Database Metrics Formalization", Proceedings of the Sixth International Conference on Quality Software (QSIC'06), 2006.
- [6] Justus S. and Iyakutti K., "Measurement Units and Scales for Object-Relational Database Metrics", First International Conference on Emerging Trends in Engineering and Technology, 2007(a).
- [7] Justus S. and Iyakutti K., "Assessing the Object-level behavioral complexity in Object Relational Databases", International Conference on Software – Science, Technology and Engineering, 2007(b).
- [8] Piattini M., Genero M., Coral C. and ALARCOS G., "Data Model Metrics" In Handbook of Software Engineering and Knowledge Engineering: Emerging Technologies, World Scientific, 2002.
- [9] Justus S. and Iyakutti K., "A Formal Suite of Object Relational Database Metrics", International Journal of Information Technology 4:4, pp. 251-260, 2008.
- [10] Stojanovic M. and El Emam K. "ES1: A Tool for Collecting Object-Oriented Design Metrics", 2001.
- [11] Al-Ghamdi J., Elish M. and Ahmed M., "A tool for measuring inheritance coupling in object-oriented systems", Information Sciences—Informatics and Computer Science: An International Journal, v.140 n.3, p.217-227, 2002.
- [12] Scotto M., Sillitti A., Succi G., and Vernazz T., "A Relational Approach to Software Metrics", 2004.
- [13] Al-Shanag F. and Mustafa S., "Clustering Data Retrieved from C# Source Code To Support Software Maintenance", Master graduation project, Department of computer and information system, Yarmouk university, 2009.
- [14] Marinescu R., "Using Object-Oriented Metrics for Automatic Design Flaws Detection in Large Scale Systems", Object-Oriented Technology (ECOOP'98 Workshop Reader), LNCS 1543, 1998
- [15] Mao C., "DBViewer: A Tool for Visualizing and Measuring Dependency Relations between Tables in Database," wscs, vol. 4, pp.13-17, World Congress on Software Engineering, 2009.