Abstract—In this paper, we propose an active, autonomic and composite data dictionary system for cataloguing and managing the structures and descriptions of both database and application objects including system configuration data. The system stores the structures and descriptions of tables, views and stored procedures, and detects any modifications on these Relational Database Management System objects using data definition language (DDL) triggers, and reflects these changes to an internal data dictionary. Non-database objects residing in the application layer and system configuration data such as application menu items, module inventory can also be detected by reflection mechanisms and included in the data dictionary. Through a user interface, the structures and descriptions of these catalogue objects are accessed, reported and modified. Moreover, the dependencies between these objects and the relations between application and database layers are fully presented to the users. By enabling a two-way synchronization, changes made by the users from this program are fully reflected to the Relational Database Management Systems, and vice versa. This system expedites obtaining information about database and application objects, their relations and dependencies and produces an autonomic documentation, and as a result it is observed that it improves the performance of software team members and shortens the software development life cycle.

Keywords—data dictionary, relational database management system, software development life cycle, trigger.

I. INTRODUCTION

KNOWING and obtaining rapid information about your system is very important at all stages of the software development life cycle (SDLC) [1], [2]. In the process of developing complex systems, many team members including software architects, analysts, developers and testers take roles building or consuming the components of the system. For instance, one team member can create a table for a project and then the data in this table can be used by a team member in a different profile or in a different project group. In this case, all team members have a need for understanding the table and knowing the description of its columns and reference information in order to use the data accurately. Therefore, the member who created the table has to describe the table and its details to other team members. Instead of verbally describing every time the information is needed, there should be a system that carries out this duty. The system that handles this duty is called Data Dictionary, which is a structure that stores metadata, i.e., data about data, and used for information resource management [3]-[5]. There are two types of data dictionary. A passive data dictionary that needs to be updated manually and an active data dictionary that is automatically updated according to the system changes. Passive data dictionary systems are mainly used for documentation and require extra operation to register meta-data, whereas active data dictionary systems are more powerful in implementing control mechanisms but all transactions must go through the system [5].

The basic requirement about data dictionary systems is to require minimum human interaction. Transferring objects, their details and descriptions to a simple passive data dictionary when a change occurs, can be seen as a solution for sharing information between team members. But in common practice, there is a problem about this one way approach. Since the data dictionary is a stand-alone system, lacking integration with development tools, every modification to an object (e.g. changing column data type, adding custom field description etc.) has to be reflected to the data dictionary by the users manually. Instead, with an autonomic, two-way active and composite data dictionary model that is integrated with development tools, a change at any point is reflected to all software development environments. Minimizing the human interaction by an autonomic way, the operational risks are also by-passed. In addition, this autonomic mechanism provides an online, synched data dictionary and allows querying data for audit purposes.

Moreover, an autonomic data dictionary system assures easy management and online documentation of applications running on various platforms and Relational Database Management Systems (RDBMS) [6], [7]. Establishing an autonomic data dictionary using off-the-shelf features of an RDBMS, does not fulfil the data cataloguing requirements of an Enterprise System (ES) [8]. Customization is necessary according to the requirements of ESs, and a data dictionary system for an ES should be able to address issues like cataloguing authentication and authorization. Besides, a data dictionary system which contains only database objects is not adequate for an ES. Application layer objects such as web services, complex types, assemblies, etc. are also expected to be included in the data dictionary. This kind of implementation shortens software development life cycles and allows performing healthier impact analysis under the concern of Change Management, which is one of the core processes of the Information Technology Infrastructure Library (ITIL) [9].

M. Gündebahar and M. C. Kuş Khalilov are with the Kuwait Turkish Participation Bank Inc., Büyükdere Cad. No: 121 Esentepe Istanbul, Turkey (phone: 00905333854899, 00905332244879; fax: 009021223704357; e-mail: mucahitg@gmail.com, mervecankus@gmail.com).
Optimized Change Management process leads to satisfying business goals and minimizing costly project disruptions [10]. System users are now able to obtain rapid information about the relations of a stored procedure which takes place in an application class, or know which classes call the base class that uses the stored procedure, or which end-users are affected by the changes to that specific stored procedure.

In this paper, we present an Active and Composite Data Dictionary System which is designed for storing, viewing, searching data about database objects like tables, views, stored procedures, application objects like assemblies, classes, methods, relations between these objects and system configuration data.

II. RELATED WORK

In the literature, advantages of using a data dictionary are presented in several studies. Absolute need of a system as a data dictionary is indicated in [11]. It is stated that data dictionaries increase programmer productivity, simplify access and changes to a system in [12]. It is shown in [13] that audit operations are carried out easier with data dictionary systems. In [14], ways of including data about business processes in a data dictionary system are presented. Fundamentals of integrated dictionary systems are given in [15] and it is stated that use of a dictionary system can assist in data inventory management, cost control and improving resiliency to changing requirements. In [16], a methodology for the design of a passive data dictionary is presented with the help of E-R diagrams [17], [18], Data Flow Diagrams [19], [20] and CASE tools [21]. All these studies are initiative in this area and they are implemented by using former technologies. Most of the previous studies implement passive data dictionaries, whereas we present an autonomic, composite and two way active data dictionary. Data dictionary presented in this paper is implemented and integrated with modern development tools and RDBMSs. Moreover, data about application layer objects and system configuration are included.

III. IMPLEMENTATION

The active, autonomic and composite data dictionary system includes data store, trigger mechanisms, autonomic search and view interfaces and maintenance jobs. Since data dictionary is an operational system, it has its own database, maintenance plan and integration services.

A. Data Store

Object, ObjectDetail, Responsibility, Relation and ActivityLog tables are responsible for storing catalogue metadata and form the core elements of the Data Dictionary System. Table relations are shown in Fig. 1.

Object table stores information about tables, views, stored procedures, functions, and user defined types in database layer of application. Object table also stores application objects like assemblies, services, types, interfaces and classes in application layer. Beside these, Object table stores system configuration data like application menu items and configuration parameters. ObjectDetail table stores description of the objects and detail information about each object like table-view columns, stored procedure parameters, methods in application layer, parameters and comments of methods, types of the classes and application menu sub-items.

Responsibilities are defined by objects and responsibility levels are read, change and owner. Members in a project group can only access the objects under their responsibility and change their descriptions and properties. They cannot access the objects which they are not accountable for. Therefore, a full access like change and a limited access like read are necessary. Owner responsibility type is used for contacting the members who own the object in case of a problematic situation occurring for that specific object or a notification that has to be done about that object. For instance, if an object and its details do not have descriptions, a notification is sent to the owners of the object by the data dictionary system. Responsibility table provides a basis for carrying out these duties using references to Object table.

Relation table stores all relations between objects. It contains relations between application methods, relations between methods and database objects like stored procedures, relations between stored procedures and tables, etc. Type of the relation is also a distinguished property of the table to form and track the relations.

All items in the data store, Object, ObjectDetail, Responsibility, Relation and ActivityLog tables also store information like user name, host name, system date synchronized with the ActivityLog table which allow us to handle activity deterministically.

Additionally, some control blocks are implemented to serve reusability and enable standardization tasks at the time of insertion or update of objects, in accordance with the needs of the enterprise system. This kind of control has great importance since it provides integrity and quality of the system. These control blocks are implemented as triggers of tables. They are used for cases like checking naming conventions like prefix and suffix, checking length of object names, preventing usage of data types which are not standard.
B. Trigger Mechanisms

Basically, the purpose of using trigger mechanisms is to reflect changes in the system to the data dictionary with online or minimum delay. The details of these mechanisms which are online and have transactional integrity are discussed in detail.

1) Database Layer: Data Definition Language (DDL) [22] triggers of RDBMSs are used to track the object and object detail changes. In modern database management systems, an event is raised in the time of create, alter and drop operations of objects. This event instance contains information about the object and the change details. For instance, when a table name is changed, DDL trigger detects this change, and issues which object has changed, what its old name was and what its new name is. Similarly, when a description of a column of a table is changed in the RDBMS, the column name, the table name, old description and new description data can be retrieved by DDL trigger. When alteration of a stored procedure is detected, the parameters of stored procedure are updated in the data dictionary using event instance data. These create, alter, drop, and rename events and description changes are reflected to the data dictionary system online in an autonomic way. These triggers are also the data source of ActivityLog, since they can also detect the user and date of the change.

2) Object/Application Layer: The contents of the objects in the application layer can be reached by reflection mechanisms for the modern software languages. Reflection is a method that allows inspection of classes, interfaces, fields and methods at runtime [23], [24]. Reflection mechanism is activated by a directory listener in file system and with the help of reflection, when new objects are deployed to the system they are added to the data dictionary by inserting them to Object and ObjectDetail tables. Assemblies, classes, methods and their relations are extracted and stored.

Class and method comments are retrieved daily by automatically building projects in the software development tool and programatically analysing the XML documentation file produced after the build event. Extracted comments are then stored in data store of data dictionary by mapping to the corresponding object. The post-build XML documentation file also contains information of method parameters and return types. This information is also extracted and stored in data store of data dictionary system.

Finally, database operations which take place in methods are detected and the names of the database objects are matched in the Object table and relation between method and database object is inserted to the Relation table. This step forms the bridge between application and database layers.

C. System Configuration Data

Other components of application layer like naming conventions, application user roles, application menu and application inventory are also included in the data dictionary. Alternatively, this kind of system configuration data can also be used by joining to application databases. Since we want our composite data dictionary to run automatically, this data is preferred to take place in the database of composite data dictionary. Since these data are open to change, daily one-way synchronization from application databases to data dictionary is included in the maintenance plan.

D. Autonomic Search and View Interfaces

Interfaces of the data dictionary application are integrated with the software tools that development teams use. For example, dictionary data is retrieved in the software development tool with the help of add-ins by function key combinations and the data gets launched in a gadget style pop-up. If a stored procedure call takes place in a class, the developer can access the information about the tables that stored procedure has relations without using a database management tool. Likewise, in database layer, class design and dependencies are viewed as an interface included in the RDBMS, the user can see which classes use which tables, since data dictionary system includes data about class-stored procedure relations and stored procedure-table relations.

There is also a stand-alone interface called Dependency Watcher, which is included in the data dictionary application. In this interface, the method calls between classes, database stored procedure calls and the tables that stored procedures use are plotted, graphically presented to the user. The objects are displayed as nodes, and the relations are shown between these nodes. The description of each object is shown as tooltip of each node. Moreover, parameters are included for the stored procedures and class methods, column names are included for the tables in the tooltips.

A search interface is implemented to enable users to search keywords in the whole data dictionary data. Users can view the detail of the search results. For example, if search result is a table name, then user can view descriptions of it and its columns. If the search result is an application method name, dependencies of the method are shown in the Dependency Watcher interface. Search contains application file names, assembly names, class names, class comments, method names, method parameters, method return types, method comments, database object names, database object detail names, database object descriptions and database object detail descriptions.

E. Maintenance

Requirements of a typical system like database backup, high availability, performance, security and schedule concerns must be met for the data dictionary. Therefore, this system has daily, weekly and monthly maintenance jobs such as getting backups, rebuilding indexes, updating statistics, checking the data integrity, archiving as typical operational systems. Besides, synchronization of system configuration data with the data dictionary is included in the maintenance plan.

F. Audit and Analytic Queries

The change events and access information including dates about the objects are also stored in the activity log of the data dictionary system. Analytical queries in different dimensions (i.e. the application that has changed the most, the least volatile components, the user who has the maximum number of change events, etc.) can be can be extracted from the composite data dictionary. Additionally, the application
components which will have a change ahead can be detected by analysing the date information of the change data. Similarly, change information including user name, object names, date are also used for audit purposes.

Basically, every object is a component of a business application. However, the modules can be used by different development groups. At this point, changes on the objects which are not standard can be detected and this data can be enlightening about problems of planning and designing modules and development groups.

IV. EXPERIMENTS AND RESULTS

The data dictionary system does not have a direct aim of improving code metrics by decreasing total lines of code, class and method complexity of the projects. However, it significantly decreases the amount of duration that is needed to implement a project from beginning to end. Therefore, experiments are aimed to analyse the influence of data dictionary system by measuring the duration spent at each step of SDLC.

10 similar software projects are implemented with the same business scenario by three project teams with equal competence levels. First team did not use any data dictionary system. Second team used a passive data dictionary which is not integrated with an RDMS or a development tool, where the user intentionally adds the descriptions of database objects to a passive data dictionary. Third team used the proposed data dictionary system that is presented in this paper. The durations elapsed at each step for these 10 projects are measured according to the 3 approaches and average durations and differences are calculated. Results are given in Tables I-IV. Durations elapsed till the end of each SDLC step for three approaches are shown in Fig. 2.

Proposed data dictionary system increases the effectiveness of and speeds up SDLC by 17.86% compared to not using a data dictionary. Passive data dictionary system speeds up SDLC 7.03% compared to not using a data dictionary. Proposed data dictionary also speeds up SDLC 10.11% compared to passive data dictionary. As seen from the results, using proposed data dictionary significantly decreases the SDLC overall duration and gives better performance than not using a data dictionary or using a passive data dictionary.

By using proposed data dictionary, most acceleration is obtained at the development step, whereas the least acceleration is obtained at the analysis step. The benefits of data dictionary are utilized especially at the development step with integration, standardization and autonomic capabilities.

V. CONCLUSION

This study proposes an active and composite data dictionary system including data about application and
database objects. The goal is to develop an autonomic data dictionary that reduces human intervention to store, manage and access metadata of the data model of an enterprise system. The developed data dictionary system is integrated with software development tools and database management systems. It is experimentally shown that, when used, this data dictionary framework substantially increases the performance of developers and accelerates the software development life cycle. To sum up, the factors that provide this speed up are as follows. The data dictionary system provides quick access to data about the system. It is a self-managing and autonomic data dictionary system, therefore extra time to gather data about the system is no longer needed. In addition, documentation can be performed by using these data automatically. It also provides users to see dependencies between objects visually, therefore impact analysis can be performed in an easier way. Awareness about the system is increased by using a common, self-populated, centralized approach.

ACKNOWLEDGMENT

This work is developed as a research and development project under the supervision of Associate Professor Fatih Alagöz. The authors would like to thank their colleague Emre Özer for valuable discussions.

REFERENCES