Demonstration of AGENDA Tool Set for Testing Relational Database Applications

Yuetang Deng
David Chays
Polytechnic University *
6 Metrotech Center
Brooklyn, NY 11201
{ytdeng,dchays}@cis.poly.edu

Abstract

Database systems play an important role in nearly every modern organization, yet relatively little research effort has focused on how to test them. AGENDA, A (test) GENerator for Database Applications, is a research prototype tool set for testing DB application programs. In testing such applications, the states of the database before and after execution play an important role, along with the user's input and system output. AGENDA components populate the database, generate inputs, and check aspects of the correctness of output and new DB state.

Keywords

database, software testing, test data

1. Introduction

A database application program can be viewed as an attempt to implement a function, just like programs developed using traditional paradigms. However, considered in this way, the input and output spaces include the database states as well as the explicit input and output parameters of the application. This has substantial impact on the notion of what a test case is, how to generate test cases, and how to check the results produced by running the test cases. Furthermore, DB application programs are usually written in a semi-declarative language, such as SQL, or a combination of an imperative language and a declarative language, such as a C program with embedded SQL, rather than using a purely imperative language. Most existing program-based software testing techniques are designed explicitly for imperative languages, and therefore are not directly applicable to the DB application programs of interest here.

* Supported in part by NSF Grants CCR-9870270 and CCR-9988354, AT&T Labs, and the New York State Office of Science, Technology, and Academic Research.

2. AGENDA Tool Set

Our goal is to assist the database application developer or tester in a usable, useful way. AGENDA consists of five interacting components that operate with guidance from the tester. Design and implementation details of each component are provided in [2]. Our approach leverages the fact that the database schema is described formally in the Data Definition Language of SQL, in order to ensure that the data we generate satisfies the integrity constraints, and allows the user to provide additional information to guide the generation. We want to automate the testing process as much as possible, but not at the expense of usability. We don't want to burden users with having to describe their data and/or applications in yet another language, as is required by existing approaches.

AGENDA generates data that are known to satisfy the constraints and then populates the DB with it. Furthermore, the data is selected in such a way as to include situations that the tester believes are likely to expose faults in the application or are likely to occur in practice, to assure that such scenarios are correctly treated. In order to insure that the data are consistent, AGENDA takes advantage of the database schema, which describes the domains, the relations, and the constraints the database designer has explicitly specified. This information is expressed in a formal language, SQL's Data Definition Language (DDL), which makes it possible to automate much of the testing process.

To address the issues detailed in [1], we have developed a tool set, AGENDA, to help test database applications. The AGENDA architecture is shown in Figure 1. AGENDA

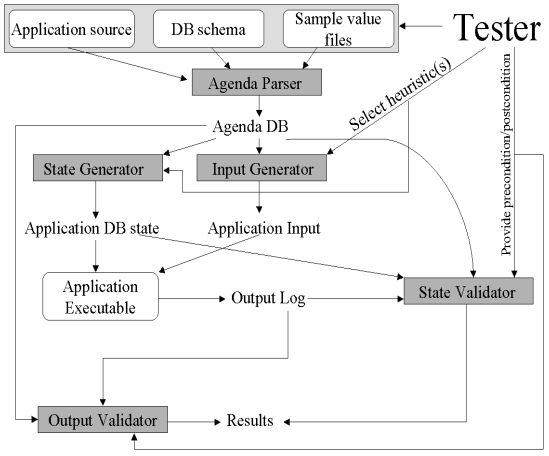

Figure 1. Architecture of the AGENDA tool set

takes as input the database schema of the database on which the application runs; the application source code; and "annotation files", containing some suggested values for attributes. The user interactively selects test heuristics and provides information about expected behavior of test cases. Using this information, AGENDA populates the database, generates inputs to the application, executes the application on those inputs and checks some aspects of correctness of the resulting database state and the application output.

The approach is loosely based on the Category-Partition method [3]: the user supplies suggested values for attributes, partitioned into groups, which we call *data groups*. This data is provided in the annotation files. The tool then produces meaningful combinations of these values in order to fill database tables and provide input parameters to the application program. Data groups are used to distinguish values that are expected to result in different application behavior, e.g. different categories of employees.

Using these data groups and guided by heuristics selected by the tester, AGENDA produces a collection of *test templates* representing abstract test cases. The tester then provides information about the expected behavior of the application on tests represented by each template.

In order to control the explosion in the number of test templates and to force the generation of particular kinds of templates, the tester selects heuristics. Available heuristics include one to favor "boundary values", heuristics to force the inclusion of NULL values where doing so is not precluded by not-NULL constraints, heuristics to force the inclusion of duplicate values where so doing is not precluded by uniqueness constraints, and heuristics to force the inclusion of values from all data groups.

Finally, AGENDA instantiates the templates with specific test values, executes the test cases and checks that the outputs and new database state are consistent with the expected behavior indicated by the tester.

The first component (*Agenda Parser*) extracts relevant information from the application's database schema, the application queries, and tester-supplied annotation files, and makes this information available to the other four components.

The second component (*State Generator*) uses the database schema along with information from the tester's annotation files, and populates the database tables with data satisfying the integrity constraints. Heuristics, listed above and detailed in [2], are used to guide the generation of both the application DB state and inputs.

The third component (*Input Generator*) generates input data to be supplied to the application. The data are created by using information that is generated by the Agenda Parser and State Generator components, along with information derived from parsing the SQL statements in the application program and information that is useful for checking the test results.

The fourth component (*State Validator*) investigates how the state of the application DB changes during execution of a test. The State Validator automatically logs the changes in the application tables and semi-automatically checks the state change.

The fifth component (*Output Validator*) is similar to the State Validator tool. When the application query is a `select` statement, it is handled by the Output Validator; otherwise, it is handled by the State Validator. The Output Validator captures the application's outputs and checks them against the query preconditions and postconditions that have been generated by the tool or supplied by the tester.

We believe that database application testing contains three levels: the query level, the transaction level, and the transaction concurrency level [2]. Currently, our system operates on the query level and the transaction level. Ongoing work involves adding more features to our existing system, and extending our tool set to the transaction concurrency level.

References

[1] D. Chays, S. Dan, P. G. Frankl, F. I. Vokolos, and E. J. Weyuker. A framework for testing database applications. *Proceedings of the 2000 International Symposium on Software Testing and Analysis*, pages 147–157, Aug. 2000.

[2] D. Chays, Y. Deng, P. G. Frankl, S. Dan, F. I. V. olos, and E. J. Weyuker. Agenda: A test generator for relational database applications. *Journal of Software Testing,Verification and Reliability*, to appear.

[3] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Commun. ACM*, 31(6):676–686, June 1988.