Granularity on Persistent Data Flow Testing of Active Database Applications

Plinio S. Leitao-Junior\textsuperscript{1}, Plinio R. S. Vilela\textsuperscript{2}, Mario Jino\textsuperscript{3}, Joao C. Silva\textsuperscript{1}

\textsuperscript{1}Instituto de Informática – Universidade Federal de Goiás (UFG)  
Campus Samambaia, Caixa postal 131 – 74001-970 – Goiania – GO – Brazil

\textsuperscript{2}Universidade Metodista de Piracicaba (UNIMEP)  
Rodovia do Açucar, Km. 156 – 13400-911 – Piracicaba – SP – Brazil

\textsuperscript{3}Universidade Estadual de Campinas (UNICAMP)  
Avenida Albert Einstein, 400 – 13083-970 – Campinas – SP – Brazil

\{plinio,jcs\}@inf.ufg.br, prvilela@unimep.br, jino@dca.fee.unicamp.br

Abstract. Active databases have been traditionally used as an alternative to implement persistent data requirements of applications on several knowledge domains. Their principle is the activation of tasks with specific functionalities as a response to events. These reactive abilities are generally expressed with active rules defined within the database itself. We investigate the use of data flow-based testing to identify the presence of faults in active rules written in SQL. Our research is based on the precision of data flow analysis, also named as data flow granularity, aiming at comparing different granularities and preliminarily evaluating their fault-revealing effectiveness. This analysis has an important impact on the cost of database application testing. The results point to higher granularities do not improve the fault detecting ability.

1. Introduction

Data persistency is an attribute associated to the demand for non-volatile data. Applications that manipulate such data are different from conventional applications, since they incorporate persistent data in their execution input and output domains. In this context, database-enabled applications play an important role in the majority of modern organizations [Chays et al. 2000]. A database application is a program running in an environment containing one or more databases [Kapfhammer and Soffa 2003].

Considering that faults are inherently part of software production, the proposition of approaches to improve the quality of database applications is pertinent. Software testing is the most commonly used method of software quality assurance. Software testing techniques for conventional programs have been proposed, implemented and evaluated over the years, but relatively little effort has been dedicated to the development of systematic testing techniques aimed to the database application domain [Chan and Cheung 1999, Chays et al. 2000, Kapfhammer and Soffa 2003, Zhang et al. 2001].

The motivation for this work is to contribute to the improvement of the quality of SQL-based applications. An SQL-based application interacts with the database using SQL (Structured Query Language), the most used language by database application developers [Fortier 1999, Elmasri and Navathe 2006]. The use of relational databases has
increased in applications that manipulate persistent data and, in this context, SQL is the most widely accepted and adopted in relational database systems [Daou et al. 2001].

Active database systems monitor specific events and, as they occur, trigger appropriate responses at the appropriate time. When the triggering of events occurs, a condition is evaluated against the database state, and an action is activated if its true. From the application’s point of view, part of its functionality is expressed as rules, such that the control and data flows are transferred from the application to the active rules during execution. An application domain overview of active database systems and their implementation aspects are found in [Ceri et al. 2000, Ceri and Widom 1996].

The question associated to this research is the lack of testing techniques in the context of active rules written in SQL. This motivates new testing techniques based on the use of SQL, aiming to reveal faults not yet discovered.

Testing criteria for active rules written in SQL were proposed and analyzed in the context of data flow based structural testing [Leitao-Junior et al. 2008]. The criteria are an extension to the All-Uses criterion [Rapps and Weyuker 1985], and exploits persistent data flow relations associated to rule interaction. We carried out an empirical investigation aiming to evaluate the applicability of different data flow analysis precisions and to compare their fault detecting ability.

Section 2 deals with active rules written in SQL. Persistent data flow is discussed in Section 3. Section 4 introduces persistent data flow analysis precision. An empirical analysis is exploited in Section 5. Section 6 deals with related work. Section 7 concludes with directions for future work.

2. Active Rules in SQL

Active rules in SQL, called triggers, are activated by a database state transition. The source of events is limited to operations on the database structure, specifically a relation state transition. The rule condition consists of any SQL predicate, including sub-queries and user defined functions [Kulkarni et al. 1998]. Baralis et al. [Baralis et al. 1998] present the event-condition model for triggers. Three components make up the model: i) the event set: the set of data manipulation operations being monitored; ii) the condition: a predicate that references the current database state and the rule transition values; and iii) the action: a sequence of data manipulation operations. The transition values associated to a given active rule execution are transient data that are being inserted, updated or excluded by the event operation. Figure 1 presents the syntax for the creation of a trigger, according to Kulkarni et al. [Kulkarni et al. 1998].

The production <trigger action time> – Figure 1 – determines whether the rule will be triggered before or after the event operation acts on the database. The production <trigger event> defines the type of state changing operations that cause the rule to be triggered. The clause FOR EACH { ROW | STATEMENT } specifies whether the rule is triggered for each event occurrence or for a set of event occurrences. The production <search condition> does not include state changing operations, differently from the production <triggered SQL statement>; the dispatch between triggers, not necessarily distinct, can occur only from the action rule execution, not from the condition rule evaluation. If the execution of the action rule causes the event of another rule, the execution of the former is interrupted and the control is given to the latter.
CREATE TRIGGER Reorder
AFTER UPDATE OF PartOnHand ON Inventory
FOR EACH ROW
DECLARE NUMBER X;
BEGIN
SELECT COUNT(*) INTO X
FROM PendingOrders
WHERE Part = :New.Part;
IF X = 0 THEN
  INSERT INTO PendingOrders
  VALUES (:New.Part, :New.OrderQuantity, SYSDATE);
END IF;
END;

3. Persistent Data Flow

The data flow in manipulation commands is characterized by: definition, use and use-definition. The first is the result of the execution of the insert command; the second corresponds to the execution of the select command; and the last one is the result of the execution of the update and delete commands. Data flow of implicit operations are also considered; for example, the opening of a cursor is a persistent data use.

The ddef notation represents persistent data definitions: ddef(i) = \{variable v — v is a database variable defined in node i\}. The definition of a persistent variable is a
persistent definition. The persistent use may affect the control flow, due to the occurrence of exception conditions. Persistent uses occur in the output edges of SQL nodes, due to the potential exceptions. The \textit{duse} notation represents the persistent data use: \textit{duse}(i,j)\text{=\{}variable \textit{v} \text{— \textit{v} is a database variable used in the edge (i,j)}\text{\}}. The use of a persistent variable is a persistent use.

A persistent data flow association, persistent association, ddef-duse-association, or simply \textit{ddua}, is a triple \([i, (j, k), v]\), such that: \(v \in \text{ddef}(i)\); \(v \in \text{duse}(j, k)\); and there is a definition-free path with respect to (w.r.t.) \(v\) from \(i\) to \((j, k)\). Alternatively, if \(\text{ddu}(v,i)\text{=\{}edges \(j,k)\text{—}v \in \text{ddef}(i), v \in \text{duse}(j,k)\text{ and there is a definition-free path w.r.t.} v\text{ from }i\text{ to } (j,k)\text{\}}\), a persistent association is represented by the triple \([i, (j, k), v]\), such that \((j, k) \in \text{ddu}(v,i)\).

4. Granularity on Persistent Data Flow

To define the data flow associations created from the database usage we should decide on a level of granularity of the database variables in which we can trace their definition and their later use [Daou et al. 2001]. The granularity, also called \textit{data flow analysis precision}, determines the rigor of data flow analysis and defines coverage levels for persistent data flow relations. According to Kapfhammer and Soffa [Kapfhammer and Soffa 2003] the granularity levels are: database, relation, attribute, tuple, and attribute value.

At granularity level relation persistent variable are mapped to database relations, no matter which tuples or attributes had been affected by manipulation operations. This approach easy the analysis and reduce the number of variables, since it considers each relation as a single variable. Nevertheless establishes a conservative approach, since every variable definition followed by a variable use is a data flow association, regardless of which data they are manipulating.

At granularity level attribute the definition occurrences and uses of relation attributes are explored, with no concerns of which tuples have been manipulated. It is more precise then the relation level, since it differentiates the attributes at each relation. Data flow associations can be characterized when the intersection of the two sets of manipulated columns, the definition occurrences and the use occurrence, are not empty. An advantage of this approach is that the number of columns in a relation is fixed and their occurrences of definitions and uses are statically identified. Daou et al. [Daou et al. 2001] set forth a strategy to the regression testing of database applications and use the granularity at the column level to determine the database modules affected by modifications in the database component definitions.

The granularity level tuple is used when a refined data flow analysis is necessary to determine the affected tuples in each data manipulation operation. Generally, statically determine which tuples are affected by a data operation is not possible, due to the complexity that can be reached by the line selection predicates [Vaduva 1999] and the current database state. In a relational database a tuple, in general, represents a real world entity, an association among entities or some particular aspect of an entity. So use the line granularity level means reach the real world entities manipulated by database operations.

The database and attribute value granularity levels represent two ends of data flow analysis precision. The former considers the whole database as a single variable and the
definition and use occurrences are statically determined. The latter is the level of greater dataflow analysis precision and represents a composition of tuple and attribute, and is also not decidable by static analysis.

The coverage of persistent data flow association is affected by the granularity of the data flow analysis. The coverage of a ddua at the $\beta$ granularity is reached when the intersection of sets $ddef$ and $duse$ established at level $\beta$ is not null and at least one definition free path with respect to the persistent variable is executed.

Therefore, in the context of persistent data flow based testing, testing requirements are pairs $<\text{criterion}, \text{granularity}>$ which coverage analysis consider exercised paths and manipulated persistent data along these paths. The pair $<\text{criterion}, \text{granularity}>$ established a new testing requirement, since it requires the analysis of the input domains that are appropriate to the coverage of a particular criterion on each granularity level. To reach coverage at more precise granularity levels implies a greater effort during the testing activities, since the input domain is reduced. Moreover, it is in general not possible to statically determine if a particular $<\text{criterion}, \text{granularity}>$ will be covered by the test.

5. Empirical Analysis

The experiment investigated the applicability of testing requirement $<\text{criterion}, \text{granularity}>$, and preliminarily analyze its fault-revealing effectiveness at different data flow analysis precisions.

The coverage of pair $<\text{criterion}, \text{granularity}>$ demands dynamic analysis of persistent data along exercised paths, since in general the static determination of defined and used database entities due to manipulation command execution is an open question. The effectiveness of coverage criteria depends not only on the selected paths but also on the test data for those paths [Clarke et al. 1989].

Consider that the application of a test case set produces the tuple $<\Pi, \Gamma>$ such that: $\Pi$ is the resulting path set, $\Pi = \{\pi_1, \ldots, \pi_p\}$, $p > 0$; and $\Gamma$ is persistent data defined and used along each path of $\Pi$; $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$, such that $\gamma_k \in \Gamma$ is persistent data defined and used along path $\pi_k \in \Pi$.

We will consider a testing requirement effective if the application of an adequate test data is capable of revealing at least half of the software faults in the subject programs. In addition, we consider a testing requirement to be applicable if the number of test cases required in a real application is substantially less than that expected by their theoretical complexity, so that the use of the criteria is possible in a practical manner.

5.1. Experiment Design

The experiment applied a family of adequacy testing criteria, called Interaction Between Active Rules based Criteria, defined in [Leitao-Junior et al. 2008]. The complexity of testing requirements can be used to derive their application cost; it is defined as the number of required test cases in the worst case, even so in practice this situation is unlikely to occur. Although Interaction Between Active Rules based Criteria are an extension to All-Uses criterion, in which the worst case scenario is a function of control flow structures, their complexities are indexed by manipulation operations that possess both definition and use of persistent data, such that SQL command $\text{update}$. 
A four-rule set of SQL rules, named as $R_x$, which has 74 interaction associations, was applied to the Oracle database management system. The Oracle system has been frequently used by the academic community [Daou et al. 2001]. This approach could be extended to other systems, such as SQLServer, DB2, SyBase, MySQL, and Informix.

Versions of the rule set $R_x$ were built by seeding one fault for each version. The manipulation fault type list introduced in [Leitao-Junior et al. 2005] was used to derive 26 faulty versions of $R_x$, aiming at testing them by applying Interaction Between Active Rules based Criteria. The set of manipulation commands in $R_x$ in which their execution can cause persistent data flow of interaction associations was targeted during the seeding of faults. The SQL manipulation command and the fault type were selected randomly for each faulty version in order to avoid bias in fault seeding.

All faulty versions were tested at granularities relation, attribute, tuple, and attribute value, in order to observe whether higher data flow analysis precisions improve the fault-revealing ability. Test data were generated for each faulty version, aiming at exercising all interaction associations at each granularity. The triggering commands were elaborated to cover all event rule operations of $R_x$. The generation of testing database was based on [Ostrand and Balcer 1988, Chays et al. 2000]: basically, the tester provides a list of values that are conceptually different for each database attribute, according to its domain, which are combined to derive insertion commands to database relations; the commands in which execution does not satisfy any database integrity constraints are discarded.

5.2. Experiment Application

All rules of $R_x$ were enabled for triggering before the application of each test case; this establishes the real operation of $R_x$. The execution of the faulty rule was not isolated, and the triggering between rules could occur in a chain.

The experiment application was supported by a tool, called ADAPT-TOOL – Active Database APplication Testing TOOL for active rules written in SQL – that was developed aiming at the automation of Interaction Between Active Rules based Criteria. The tool builds a database testing using relational model structures, and focuses on the following functions: test data generation, control of rule set faulty versions, application (and re-application) of test cases, testing oracle, and evaluation of test case sets by granularity.

5.3. Analysis of the Results

Table 1 shows the summary of 632 applications of test cases. The data table relate the number of test cases, presenting values of interaction association coverage per granularity, and of raised exceptions. The columns are labeled (I), (II), and (III); lines are labeled (a) to (j). Columns (I) and (II) distinguish fault-revealing and non fault-revealing test cases, respectively; column (III) refers to all test cases, sum of columns (I) and (II). Line (a) refers to test cases that did not exercise the faulty node. Test cases that exercised the faulty node but covered no interaction association are summarized in line (b). Lines (c), (d), (e), and (f) synthesize test cases related to interaction association coverage at granularities relation, attribute, tuple, and attribute value, respectively. To understand how the values in lines (c) to (f) were computed, consider the cell (I : d), column (I) and line (d); it states the number of fault-revealing test cases to the granularity tuple,
those test cases in which \textit{tuple} were the highest granularity obtained at all interaction associations covered by them. Raised exceptions are exploited in lines \((i)\) to \((j)\). Line \((i)\) refers to user exceptions – those raised by the programmer using the command \textit{raise}. Line \((j)\) refers to persistent manipulation exceptions – those raised due to manipulation command execution and not handled by the programmer. Line \((h)\) refers to test cases applied with no exception.

<table>
<thead>
<tr>
<th>Test cases</th>
<th>(I) Fault-revealing</th>
<th>(II) Non fault-revealing</th>
<th>(III) All</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Fault not exercised</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>(b) No coverage</td>
<td>20</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>(c) Relation coverage</td>
<td>213</td>
<td>89</td>
<td>302</td>
</tr>
<tr>
<td>(d) Tuple coverage</td>
<td>46</td>
<td>20</td>
<td>66</td>
</tr>
<tr>
<td>(e) Attribute coverage</td>
<td>81</td>
<td>37</td>
<td>118</td>
</tr>
<tr>
<td>(f) Attribute value coverage</td>
<td>64</td>
<td>52</td>
<td>116</td>
</tr>
<tr>
<td>(g) all</td>
<td>424</td>
<td>208</td>
<td>632</td>
</tr>
<tr>
<td>(h) No exception</td>
<td>281</td>
<td>198</td>
<td>479</td>
</tr>
<tr>
<td>(i) User exception</td>
<td>112</td>
<td>10</td>
<td>122</td>
</tr>
<tr>
<td>(h) System exception</td>
<td>31</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>(j) All</td>
<td>424</td>
<td>208</td>
<td>632</td>
</tr>
</tbody>
</table>

The application of the adequate test data was capable of revealing all of the faults, and the number of test cases was substantially less than that expected by the theoretical complexity of each rule set faulty version. To evaluate the applicability of the data flow analysis, a measure was used, denoted by the number of fault-revealing test cases of adequate test set, stated by Equation 1.

\[
\frac{(I : c) + (I : d) + (I : e) + (I : f)}{(III : c) + (III : d) + (III : e) + (III : f)}
\]  

Equation 1 was used by all data flow criteria, using any model, deriving adequate test sets that include at least one fault-revealing test case. The value 0.6711 resulted from the expression above, denoting that \(2/3\) of the adequate test sets are fault-revealing. Computing the effectiveness per granularity, the values 0.7053, 0.6970, 0.6864, and 0.5517 were obtained for granularities \textit{relation}, \textit{attribute}, \textit{tuple}, and \textit{attribute value}, respectively, showing that the effectiveness reaches higher values for the lower data flow analysis precision.

The coverage at higher granularities does not improve the fault detection ability. The coverage at granularity \textit{relation} was enough to reveal the presence of all faults. In some versions of \(R_x\), the coverage at granularities \textit{tuple} and \textit{attribute value} was non fault-revealing. Two scenarios were observed: \(i)\) two state changing commands characterize the interaction association, such as \textit{update-update}, so that the execution of the second one fix the error due to execution of the first one; and \(ii)\) the triggering between rules leads to failure elimination due to the sequential execution of state changing commands.
The raised exceptions were traced in order to help the oracle role. From fault-revealing test cases, 33.7% (143 out of 424) raised exceptions, the majority resulted from `raise` command executions. Note that user exceptions implement behavior according to the software specification and they should be used to decide on the presence of faults. The instrumentation of exception situations required special mechanisms, since exception occurrences usually can cause the loss of database transactions, eliminating the instrumentation data.

The demonstrated effectiveness is an initial empirical evidence. Additional studies are required since several factors are involved in the investigation, such as, test data generation strategy, active rules used in the experiment, and fault types and active rules selected faults.

5.4. Threats to Validity

Although the seeded faults were not collected from real software projects, they are representative of real fault types, according to Leitao et al. [Leitao-Junior et al. 2005]. Even then the replication of this study with real fault data is desirable.

A threat is related to the fact that the experiment uses only one set of rules, with several different versions. To compensate this approach the set has several interaction points, which makes its behavior very complex at run time. Such interaction points, specially the triggering between rules, result in some cases on the execution of more than a thousand nodes, increasing the chances of unexpected or incorrect behavior.

Another limitation is related to the artificial seeding of faults, that are unique in each version of the rule set, in spite of the random selection of manipulation occurrences and of fault types. In practice it is observed that faults do not occur in isolation, defective programs will in general have more than one fault at the same time. We have to point out that a secondary goal of this empirical investigation is to evaluate different data flow granularities to detect the presence of faults. The isolation of faults allow a finer level of control when analyzing the influence of each granularity in the detection of each fault.

6. Related Work

In the active rule context, Vaduva [Vaduva 1999] establishes an approach based on rules sequences, independently of the existence of triggering between rules, in order to reveal inadequate rule interactions or improper sequences of rule execution. That work is based on SAMOS, an object oriented database system. The goal of each test case is the execution of a specific rule sequence from a rule set. SQL is not used and the structural elements of the rules, such as control and data flow, are not considered in the coverage of a particular sequence.

Kapfhammer and Soffa [Kapfhammer and Soffa 2003] define testing criteria based on data flow analysis for database applications exploiting several granularity levels. The list of data flow associations is dependent on the database initial state: the size of the database determines the number of required elements, since associations are defined for each possible element of the database. The results of their empirical investigation indicate that the number of required elements varies with the precision of the data flow analysis. The authors based their empirical investigation on a couple of Java programs accessing a MySQL database. There is a tendency for a high number of required elements when
tuple and attribute value granularities are used, even with a reduced database. The work does not investigate the fault-revealing aspect of the proposed criteria.


7. Conclusions

This work investigates the applicability of persistent data flow strategies in testing of active rules written in SQL, representing a valuable resource to the quality assurance activities related to the development of active database applications. A family of adequacy criteria was used, called Interaction Between Rules based Criteria, to promotes the use of a systematic approach to testing and eventually contributes to the dissemination of the use of active rules databases.

The results of the empirical investigation, supported by the tool ADAPT-TOOL, show that the criteria are capable of revealing faults in manipulation commands, in addition to their applicability at several granularities. The effectiveness was 2/3 of adequate data set, and reached higher values for the lower data flow analysis precision. The coverage of interaction associations at higher granularities does not improve the fault detecting ability; fault-revealing and non fault-revealing scenarios were identified at every granularity level.

The realization of new empirical studies is desirable, since several factors are involved in the results of the study, such as: data flow generation techniques; set of active rules utilized; test data generated; faults and types of faults seeded in the active rules. Additional sets of active rules could be used to include samples of real rules to reduce threats to the study’s validity.

References


