Testing Web Database Applications

Yuetang Deng
Phyllis Frankl
Jiong Wang

Polytechnic
UNIVERSITY

Outline

- Introduction
- Techniques for testing Web Database applications
  - Example
  - Tool
- Preliminary experiment based on TPC-W benchmark
- Ongoing/future work

Motivation

- Databases play a central role in most web applications
- Previous work did not stress the database aspect
- Extend AGENDA (database application testing toolset) to test web database applications

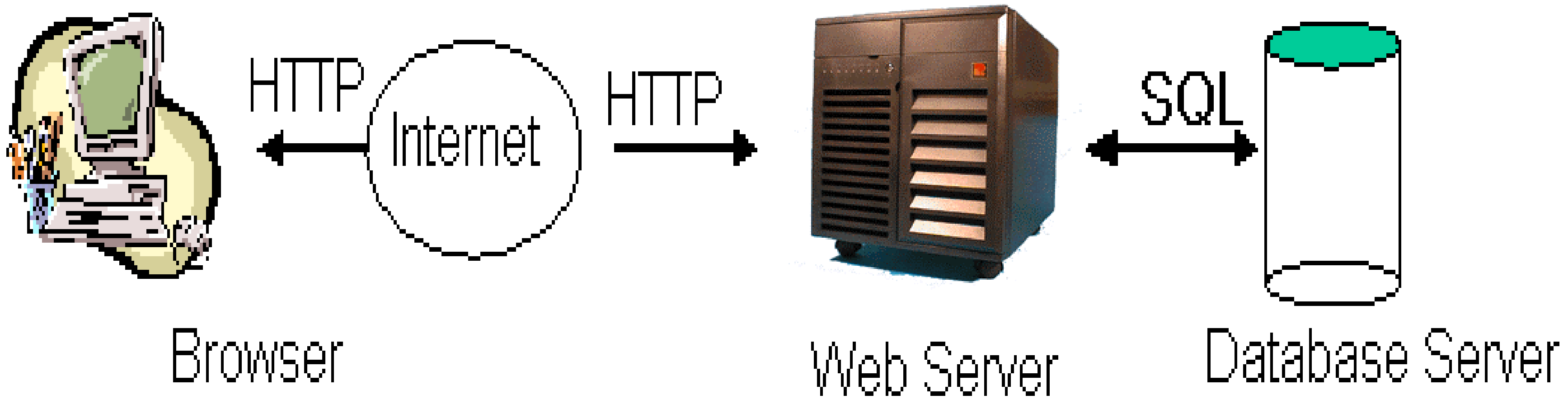
AGENDA tool set

- **Agenda Parser** extracts information from the user's schema, application and "sample value files"
- **State generation** consists of populating the user's DB State.
- **Input generation** consists of instantiating the input parameters with actual values.
- **State validation** consists of examining the change in DB state.
- **Output validation** consists of examining the result of executing the test case.

AGENDA System Overview

- Inputs
  - Input files
    - database schema
    - application source code
    - Sample values files
  - User interactively
    - selects test heuristics
    - provides info about expected behavior of test cases
- Data groups + heuristics $\longrightarrow$ test templates
- Outputs
  - Database state
  - Test cases
  - Validation of resulting DB state and output

Typical Web Application Configuration



New Challenges to Software Testing

- Dynamic, interactive
- Scalability
- Fault-tolerant
- Security

White-box testing for web application

- Pros
  - Better coverage of those aspects of the application that might not be “found” by a crawler
  - More appropriate input values for HTML forms
  - Better targeting of test effort, by using static analysis to determine that certain tests are necessary or unnecessary
- Cons
  - Must be targeted to particular source language
  - Cannot handle dynamic URLs and dynamic parameters

Web pages Types

- Static Page
  - The content is determined when page is created
- Dynamic Page
  - Data based page
    - Some of the content is derived from DB or files
  - Pages using client-side scripting (JavaScript/VB script)

Test Procedure

- Extract useful information from application source of URLs
- Construct an application graph, select "interesting" paths
- Generate inputs for URLs in the path, organize the test case as an XML file
- Automatically execute the test case
- Check the output HTML content and new database state

Outline

- Introduction
- Techniques for testing Web Database applications
  - Example
  - Tool
- Preliminary experiment based on TPC-W benchmark
- Ongoing/future work

Student Online Registration Appl

- Student
  - Register courses
  - View grade
- Faculty
  - Set grades
- Schema:
  - USER (<u>Id</u>, Name, Passwd, Type)
  - COURSE (<u>CrsCode,</u> Credit, MaxEnrollment)
  - TRANSCRIPT (<u>StudId, Code</u>, Grade)

Sample Integrity Constraints

- Enforcement of the enrollment limit
  - the number of students registered for a course must not exceed the maximum enrollment for that course
- A valid letter grade cannot be changed to an Incomplete
- A student cannot be registered for more than 12 credits

Challenges to detect the fault…

- Identify a path which corresponds to student registration scenario
- Populate the database, so that at least one student has registered several courses
- Generate inputs which corresponds that this student is registering one more course
- Keep track on how the database state is changed (ensure nothing else is changed)
- Check which constraints are affected and validate the affected constraints

```
public class UserValidate extends HttpServlet {
  private String id, password;
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws Exception {
    AgendaDbBean con = new AgendaDbBean();
    response.setContentType("text/html");
    PrintWriter out =response.getWriter();
    out = response.getWriter();
    try {
      id= request.getParameter(“id");
      password = request.getParameter("passwd");
      out.println("<html> <head> <title>  CourseWork System </title> ("</head> <body ");
    String sql="select type from user where id="+id+ " and passwd="+password;
      ResultSet  rs = con.execSQL(sql) ;
      if (rs.next()) {
        int type = rs.getInt("type");
        if (type ==1 ) {
           out.println("<A HREF =“view_grade"> View my grades <br></A>");
           out.println("<A HREF =“register"> Register courses <br></A>");}
        else if (type ==2) {
           out.println("<A HREF =“set_grade">Set grades </A>");}     }  }
     catch(Exception e) {out.println("Exception in database operation"); }
        out.println("<A HREF =“home.html"> Return to main page </A>");}
     out.println("</body> </html>");
     out.close(); }  }
```

Possible output page

```
<html>
  <head>
     <title> CourseWork System </title>
  </head>
  <body>
    <A HREF="view_grade">View grades </A>     <BR>
    <A HREF="register">Register courses </A>     <BR>
    <A HREF="home.html">Return to main page </A>
  </body>
</html>
```

Sample data values and groups

Name:

Deng
David
Phyllis
Gleb
Eric
Wang

Id:
--choice_name: STUDENT
111
252
334
121
013
311

--choice_name: FACULTY
888
887

Studid:
--choice_name: HIGH
--choice_prob: 80
311
013

--choice_name: LOW
--choice_prob: 20
112
252
334

Sample output of State Generator

Table user

Id	Name	Passwd	Type
311	Deng	deng123	1
112	Eric	eric123	1
887	Phyllis	pf123	2
252	Wang	w123	1
888	Gleb	g12	2

Table course

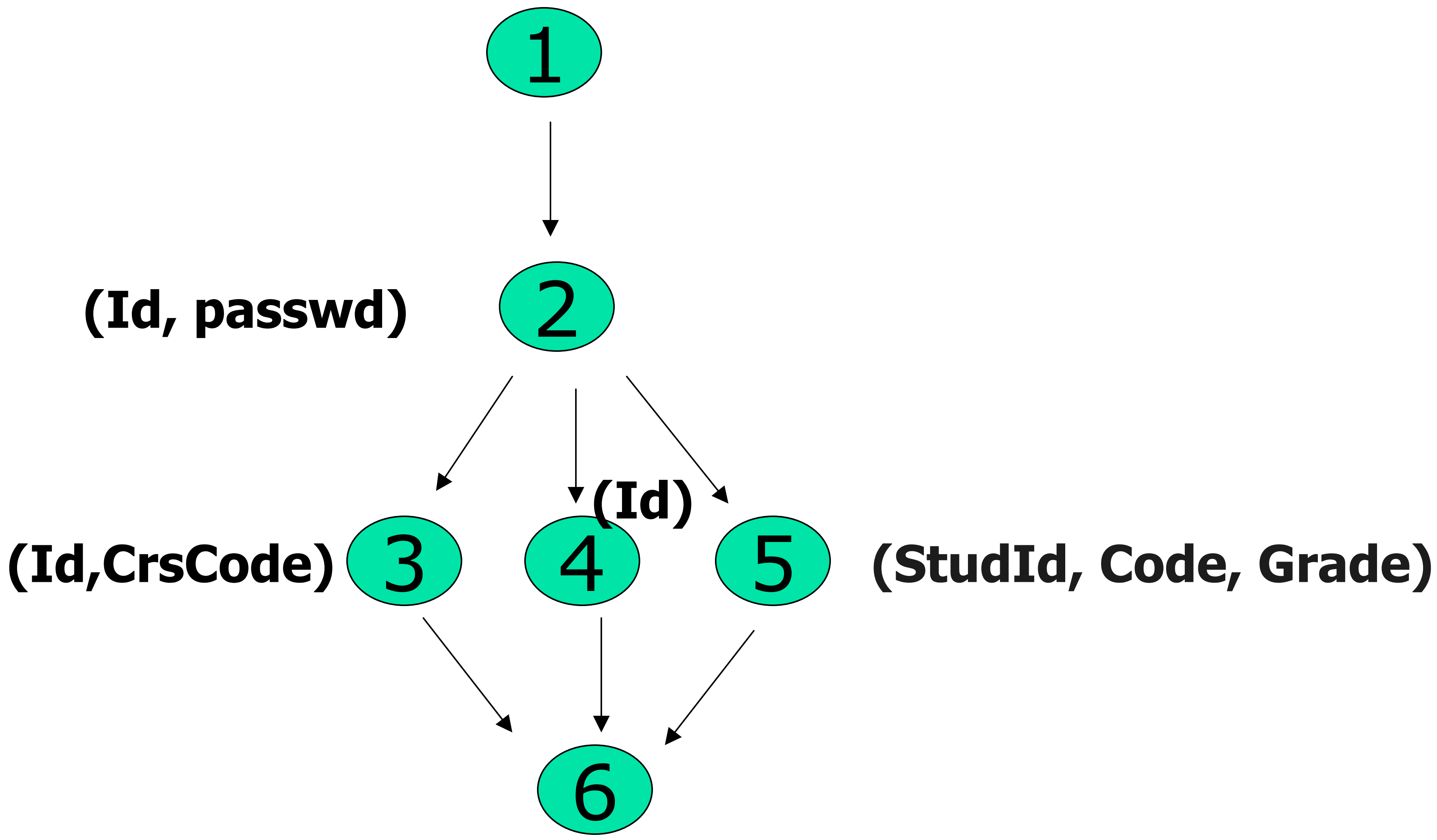
Crscode	Credit	Maxenrollment	ProfID
CS912	4	60	887
CS608	3	50	887
EL501	3	120	888
CS905	5	30	888
CS110	3	60	888

Table transcript

Sudid	Code	Grade
311	CS912	
252	CS905	C
311	EL501	A
311	CS608	
112	CS110	B

$$\begin{tikzcd}
 & 1 \arrow[d, shift left] & \\
 & 2 \arrow[u, shift left] \arrow[dl] \arrow[d] \arrow[dr] & \\
3 \arrow[uur] \arrow[dr] & 4 \arrow[uu, bend right] \arrow[d] & 5 \arrow[uul] \arrow[dl] \\
 & 6 &
\end{tikzcd}$$

id	URL	type
1	Home.html	static
2	userValidate	data-based
3	Register	data-based
4	ViewGrade	data-based
5	UpdateGrade	data-based
6	Result	data-based


1
(Id, passwd)
2
(Id)
(Id,CrsCode)
3
4
5
(StudId, Code, Grade)
6

Input

- **Path 1 (1,2,3,6): register course**
  - **Test case 1: (311, deng123, CS905)**
  - **Test case 2: (252, w123, CS530)**
- **Path 2 (1,2,4,6): view grade**
- **Path 3 (1,2,5,6): set grade**
  - **Test case 3: (888, phyllis123, 311,CS608, B)**
  - **Test case 4: (311, deng123, 311 CS912, A)**

Sample output of State Generator

Table user

Id	Name	Passwd	Type
311	Deng	deng123	1
112	eric	eric123	1
887	phyllis	pf123	2
252	Wang	w123	1
888	gleb	g12	2

Table course

Crscode	Credit	Maxenrollment
CS912	4	60
CS608	3	50
EL501	3	120
CS905	5	30
CS110	3	60

Table transcript

Sudid	Code	Grade
311	CS912	
252	CS905	C
311	EL501	A
311	CS608	
112	CS110	B

How does the State Validator work?

- Log the changes in the application tables
  - Create a log table for each application table
  - Create trigger/rule to log the changes in the application tables
- Tester specifies the constraint visa XML file or GUI
  - Precondition: studid = ‘311’
  - Postcondition sum(credit) <12
- Create a temporary table and generate SQL constraints and apply it to the temporary table
  - Transcript_temp (studid, credit_sum)
  - Constraint: check credit_sum<12
  - SELECT stud_id sum(credit) FROM transcript where studid =‘311’;

Outline

- Introduction
- Techniques for testing Web Database applications
  - Example
  - → Tool
- Preliminary experiment based on TPC-W benchmark
- Ongoing/future work

Information Extracted

- URL type (static/data based)
- HTTP request type (GET/POST)
- URL links
- FORM Information (fields, form submission)
- Parameter information
  - Name
  - Type

```
public class UserValidate extends HttpServlet {
  private String id, password;
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws Exception {
    AgendaDbBean con = new AgendaDbBean();
    response.setContentType("text/html");
    PrintWriter out =response.getWriter();
    out = response.getWriter();
    try {
      id= request.getParameter(“id");
      password = request.getParameter("passwd");
      out.println("<html> <head> <title>  CourseWork System </title> ("</head> <body ");
    String sql="select type from user where id="+id+ " and passwd="+password;
      ResultSet  rs = con.execSQL(sql) ;
      if (rs.next()) {
        int type = rs.getInt("type");
        if (type ==1 ) {
           out.println("<A HREF =“view_grade"> View my grades <br></A>");
           out.println("<A HREF =“register"> Register courses <br></A>");}
        else if (type ==2) {
           out.println("<A HREF =“set_grade">Set grades </A>");}     }  }
     catch(Exception e) {out.println("Exception in database operation"); }
        out.println("<A HREF =“index.html"> Return to main page </A>");}
     out.println("</body> </html>");
     out.close(); }  }
```

Web application graph and simplification

- Application graph
  - Node = URL
  - Edge =URL links
- Simplify
  - Remove external link
  - Remove edge (static URL links) as long as the graph doesn’t become disconnected

Path Selection

- Cyclomatic Complexity: V(G)= E-V+2
- Independent paths through the CFG: the minimal # paths that can, in linear combination, generate all possible paths through the application.
- Generate paths based on Cyclomatic Complexity measurement

URL Parameters types

- Type A: name-value pairs are input fields in HTML FORM
- Type B: name-value pairs are passed from the previous page
- Type C: name-value pairs are generated in the application

Explore various situations

- View grade, input parameter: id/password
- Case 1: invalid id and invalid password
- Case 2: valid id and invalid password
- Case 3: valid id and password, no course
- Case 4: valid id and password, one course
- Case 5: valid id and password, multiple courses

XML file as Test Case

- A Test Case is a sequence of pages to be visited plus inputs for pages containing forms
- XML: A standard, self-describing data interface

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <test>
    <step index="0">
       <url>http://localhost:8080/coursework/home.html</url>
       <method>get</method>
    </step>
    <step index="2">
       <url>http://localhost:8080/coursework /UserValidate</url>
      <method>get</method>
      <parameter>
            <name>ID</name>         <value>311</value>
       </parameter>
       <parameter>
            <name>PASSWD</name>     <value>deng123</value>
       </parameter>
   </step>
    <step index="3">
       <url>http://localhost:8080/coursework /view_grade </url>
       <method>get</method>
       <parameter>
              <name>ID</name>     <value>311</value>
       </parameter>
    </step>
    <step index="4">
       <url>http://localhost:8080/coursework /result </url>
       <method>get</method>
    </step>
 </test>
```

Automatic Execution

- Two approaches
  - Use a standard browser or web crawler
  - Implement the exploration using HTTP library, DOM interface and JavaScript interpreters.
- XML parser + HttpClient
  - Web service

Outline

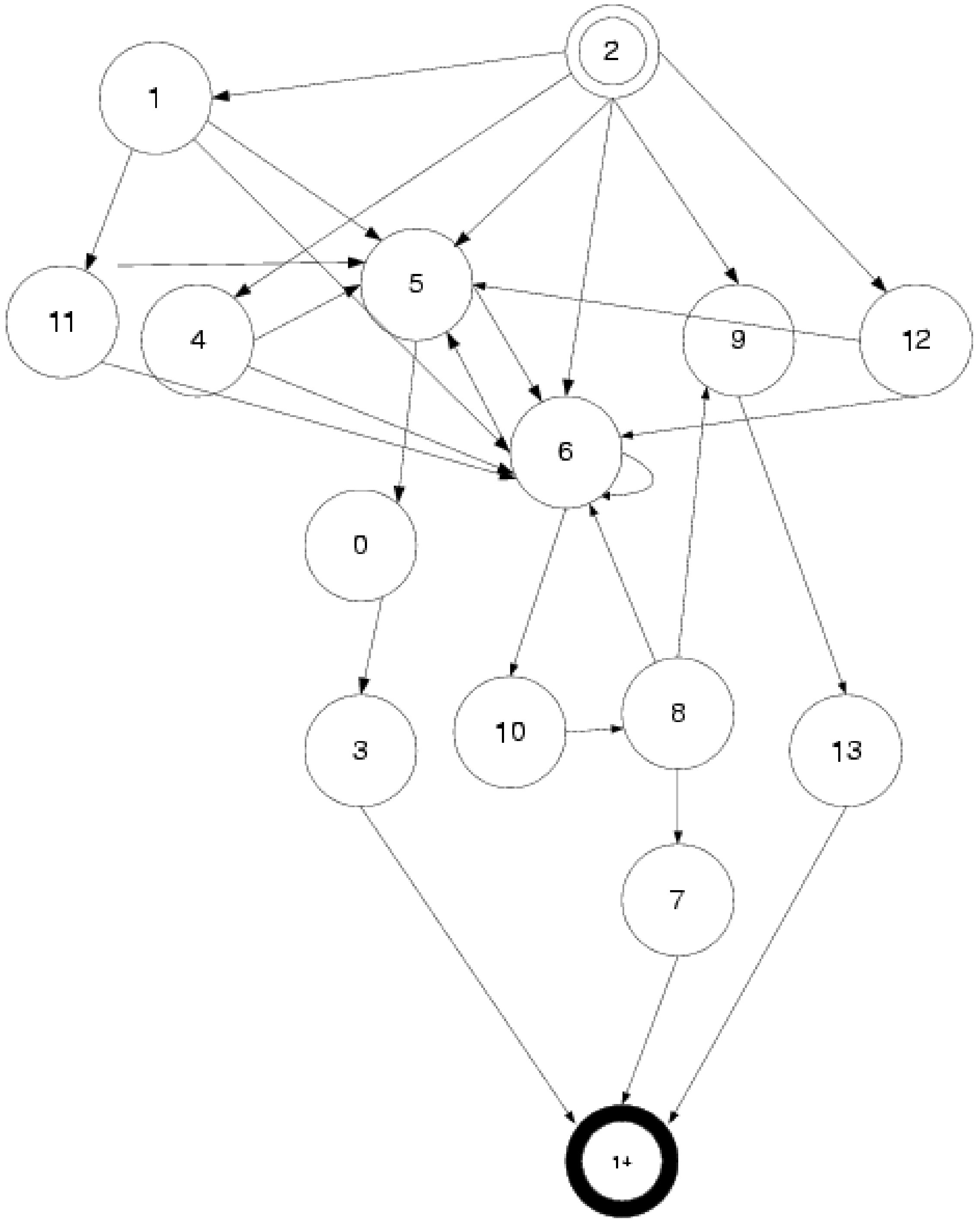
- Introduction
- Techniques for testing Web Database applications
  - Example
  - Tool
- → Preliminary experiment based on TPC-W benchmark
- Ongoing/future work

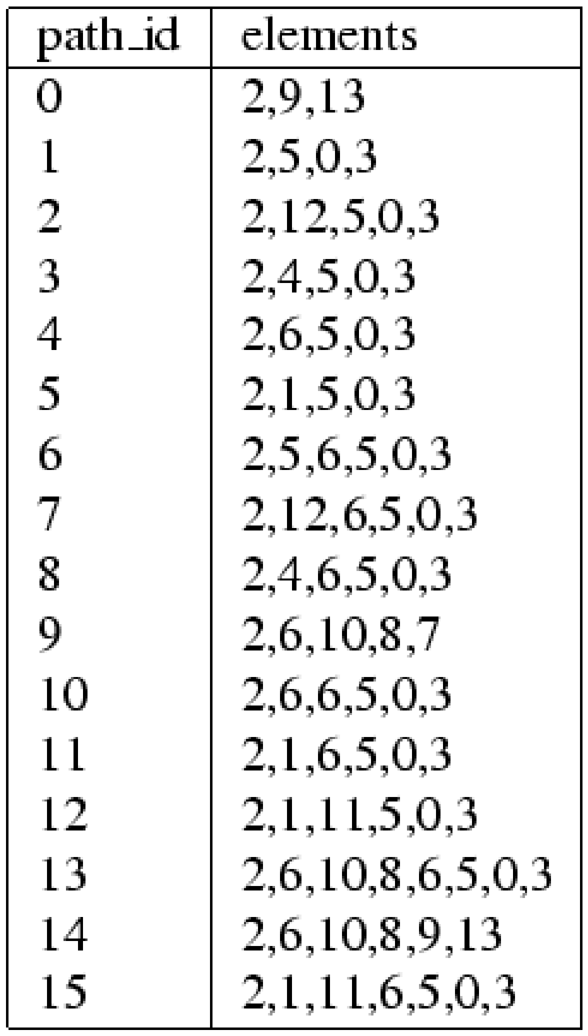
TPC-W E-commerce benchmark

- online bookstore
- 8 tables: customers, address, order, order_line, credit_info, items, author, country
- 14 web interactions

id	web interaction	type
0	tpcw_admin_request_Servlet	data-based
1	tpcw_search_request_Servlet	data-based
2	tpcw_home_interaction	static
3	tpcw_admin_response_Servlet	data-based
4	tpcw_best_sellers_Servlet	data-based
5	tpcw_product_detail_Servlet	data-based
6	tpcw_shopping_cart_interaction	data-based
7	tpcw_buy_confirm_Servlet	data-based
8	tpcw_buy_request_Servlet	data-based
9	tpcw_order_inquiry_Servlet	data-based
10	tpcw_customer_registration_Servlet	data-based
11	tpcw_execute_search	static
12	tpcw_new_products_Servlet	data-based
13	tpcw_order_display_Servlet	data-based

Figure 1. TPCW web interactions


2
1
5
11
4
9
12
6
0
3
10
8
13
7
14



path_id	elements
0	2,9,13
1	2,5,0,3
2	2,12,5,0,3
3	2,4,5,0,3
4	2,6,5,0,3
5	2,1,5,0,3
6	2,5,6,5,0,3
7	2,12,6,5,0,3
8	2,4,6,5,0,3
9	2,6,10,8,7
10	2,6,6,5,0,3
11	2,1,6,5,0,3
12	2,1,11,5,0,3
13	2,6,10,8,6,5,0,3
14	2,6,10,8,9,13
15	2,1,11,6,5,0,3

Figure 2. Path Generation

- Path 0: Order inquiry
  - Home-interaction
  - Order-inquiry
  - Order-display
- Path 3: Administration Task
  - Home-interaction
  - Best seller
  - Admin-request
  - Admin-confirm
- Path 9: Purchase books
  - Home-interaction
  - Shopping-cart
  - Customer-register
  - Bug-request
  - Buy-confirm

path	URLs	input	XML	execution
0	2,9,13	1449	292	665
1	2,5,0,3	1765	457	835
2	2,12,5,0,3	2039	501	904
3	2,4,5,0,3	2050	451	969
4	2,6,5,0,3	1990	506	1047
5	2,1,5,0,3	1693	461	917
6	2,5,6,5,0,3	1925	550	1188
7	2,12,6,5,0,3	2256	564	1141
8	2,4,6,5,0,3	2190	516	1130
9	2,6,10,8,7	5154	575	1592
10	2,6,6,5,0,3	2100	534	1262
11	2,1,6,5,0,3	1910	512	1126
12	2,1,11,5,0,3	2206	469	1090
13	2,6,10,8,6,5,0,3	3136	748	1790
14	2,6,10,8,9,13	2227	640	1256
15	2,1,11,6,5,0,3	2387	584	1243
avg		2280	523	1135

Figure 1. Tool Performance(ms)

Related work

- Auto-execution(fill forms)
  - M. Benedikt et al , Veriweb,model checker, www2002
- Model
  - J. Offufft et al., Modelling FSMs/dynamic aspect
  - Ricca and Tonella, (Model based on UML, ICSE2001)
- Capture and replay
  - Elbaum et al , sessions data, ICSE2003

Ongoing / Future work

- Testing functionality: does the application produce correct results on (legitimate) user inputs?
  - Refine static analysis
  - Hybrid techniques: static analysis + crawler based approach
- Testing security: is the application vulnerable to attacks by malicious users?

Further information

- AGENDA System was demonstrated at ASE03/ICSE03
- "An AGENDA for testing relational database applications", *Journal of Software Testing, Verification and Reliability, Mar 2004*.
- "Testing Database Transaction Concurrency", *International Conference on Automated Software Engineering 2003* .
- "Testing Database Transaction Consistency", *CIS Technical Report 2004,* Polytech University

Questions?