AutoDBT: A Framework for Automatic Testing of Web Database Applications

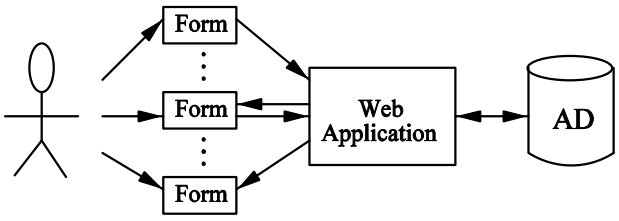
Lihua Ran, Curtis Dyreson, Anneliese Andrews
School of E.E. and Computer Science
Washington State University
USA

WISE 2004 - Brisbane

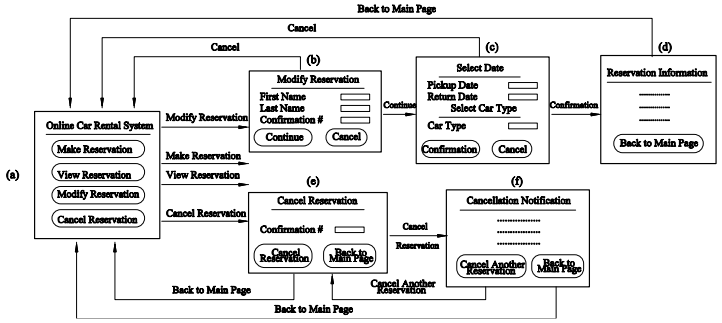
Outline

- Motivation
- Two approaches
- AutoDBT
- Implementation plan
- Related work
- Conclusion and future work

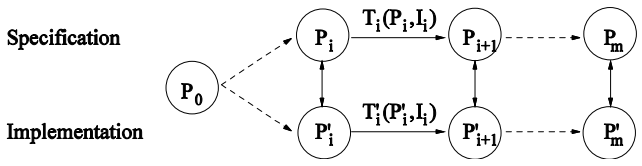
Architecture of Web Applications



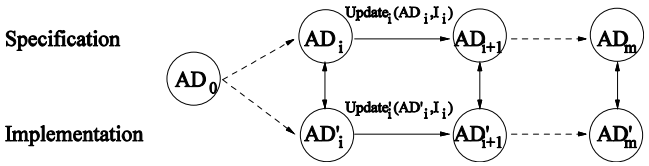
Example Web Application



Black Box Testing



Testing a Web Database Application



Additional Problems

- Evolving database state
  - Non-database: each test starts from same initial state
  - Database: database states differ for each test
- Modeling of database updates
- Correctly choosing test data
  - Non-database: manufactured as needed
  - Database: chosen from two dynamic changing sources (AD & SD)
- Data Dependency between transitions
- Correctness of each AD state
  - Non-database: expected page
  - Database: page + AD state

Outline

- Motivation
- Two approaches
- AutoDBT
- Implementation plan
- Related work
- Conclusion and future work

Two Complementary Approaches

- Dirty Test Suite
  - Generated in advance
  - Based on the initial database state
- Clean Test Case (AutoDBT)
  - Generated on the fly
  - Based on the most current database state

Dirty Test Suite Approach

- Why “dirty”
  - Too expensive to keep copies of AD0
  - Practical solution: run each test case on ADi
  - ADi <> AD0
  - The preset test case might be invalid based on ADi
- Solution
  - Filter query

Comparing Approaches

- Dirty test suite approach is more efficient
  - No overhead at run time
  - Reused to retest the same application
- Clean test case approach is guaranteed to make progress

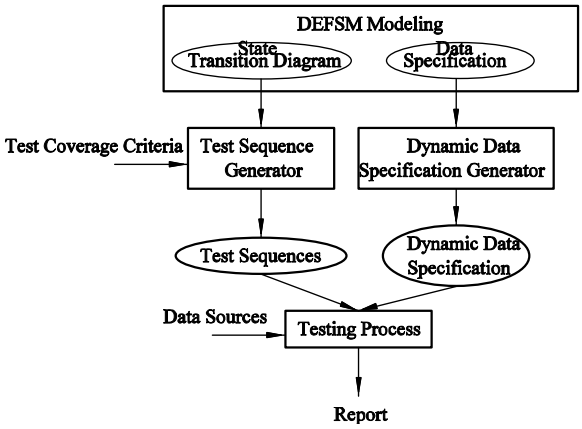
Outline

- Motivation
- Two approaches
- AutoDBT
- Implementation plan
- Related work
- Conclusion and future work

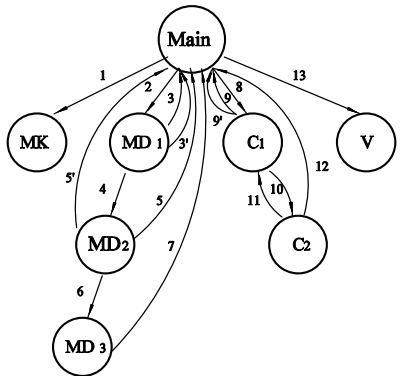
AutoDBT's Framework

- DEFSM Modeling
  - State Transition Diagram
  - Data Specification
- Test Sequence and Dynamic Data Specification Generation
- Testing Process

AutoDBT's Framework



Step1: State Transition Diagram



Step 2: Data Specification

- Input Specification
  - Data Sources
    - Application Database (AD): existing data
    - Synthetic Database (SD): new data
  - Data Flow (Data Inheritance)
- Update Specification
- Specification Language: Prolog
  - Well-defined declarative semantics
  - Evaluation model: Top-down, tuple-at-a-time.

Data Specification

- Assumption
  - AD/SD schema are known
  - Same schemas for AD and SD (can be relaxed)
- Populate AD/SD
  - Gray et al.

An Example Application Database (AD)

Customer		
FirstName	LastName	CustomerID
john	smith	c0001
mike	green	c0002
rick	reed	c0003
kate	brown	c0004

Reserves	
CustomerID	Confirmation#
c0001	0001
c0002	0002
c0003	0003
c0004	0004
c0004	0005

Reservation			
Confirmation#	PickupDate	ReturnDate	CarType
0001	10/01/03	10/03/03	economy
0002	10/02/03	10/05/03	compact
0003	10/15/10	10/23/03	full size
0004	11/03/03	11/30/03	minivan
0005	11/03/03	11/30/03	full size

Available_Car_Type	
CarType	CarNumber
economy	50
compact	40
full size	60
minivan	45
luxury	20
convertible	30

An Example Synthetic Database (SD)

Customer		
FirstName	LastName	CustomerID
franklin	bond	c0006
alicia	wong	c0007

Reserves	
CustomerID	Confirmation#
c0006	0006
c0007	0007

Reservation			
Confirmation#	PickupDate	ReturnDate	CarType
0006	12/04/03	12/10/03	luxury
0007	12/06/03	12/08/03	convertible

Input Specification Definition

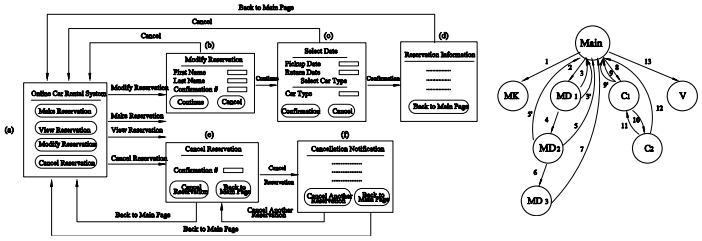
- A button name or a Prolog rule of the following form followed by a button name:

$$input_i(V_{i^1}, \ldots, V_{i^n}) \text{ :- } Predicate_1, \ldots, Predicate_m.$$

Input Specification

Transition	Input Data Flow Specification
2	Button(Modify Reservation).
3	$input_3(Fn, Ln, C\#)$:- AD_Customer(Fn, Ln, Cid), AD_Reserves ($Cid, C\#$). Button(Cancel).
3'	$input_{3'}(Fn, Ln, C\#)$:- SD_Customer(Fn, Ln, Cid), SD_Reserves ($Cid, C\#$). Button(Cancel).
4	$input_4(Fn, Ln, C\#)$:- AD_Customer(Fn, Ln, Cid), AD_Reserves($Cid, C\#$). Button(Continue).
5	$input_5(Pd, Rd, Ct)$:- AD_Reservation(_, Pd, Rd, Ct). Button(Cancel).
5'	$input_{5'}(Pd, Rd, Ct)$:- SD_Reservation(_, Pd, Rd, Ct). Button(Cancel).
6	$input_6(Pd, Rd, Ct)$:- SD_Reservation(_, Pd, Rd, Ct). Button(Confirmation).
7	Button(Back to Main Page).
8	Button(Cancel Reservation).
9	$input_9(C\#)$:- AD_Reservation($C\#$, _, _, _). Button(Back to Main Page).
9'	$input_{9'}(C\#)$:- SD_Reservation($C\#$, _, _, _). Button(Back to Main Page).
10	$input_{10}(C\#)$:- AD_Reservation($C\#$, _, _, _). Button(Back to Main Page).
11	Button(Cancel Another Reservation).
12	Button(Back to Main Page).

OCRS



Input Specification

Transition	Input Data Flow Specification
2	Button(Modify Reservation).
3	$input_3(Fn, Ln, C\#)$:- AD_Customer(Fn, Ln, Cid), AD_Reserves($Cid, C\#$). Button(Cancel).
3′	$input_{3'}(Fn, Ln, C\#)$:- SD_Customer(Fn, Ln, Cid), SD_Reserves($Cid, C\#$). Button(Cancel).
4	$input_4(Fn, Ln, C\#)$:- AD_Customer(Fn, Ln, Cid), AD_Reserves($Cid, C\#$). Button(Continue).

Update Specification Definition

- The update specification is one or more Prolog rules of the following two forms.

delete_AD_Relation (A_1, …, A_n) :- Predicate$_1$, …, Predicate$_m$.
insert_AD_Relation (A_1, …, A_n) :- Predicate$_1$, …, Predicate$_m$.

Update Specification Example

Transition	Update Specification
6	delete_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_4(\_, \_, C\#)$, AD_Reservation($C\#, Pd, Rd, Ct$). insert_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_4(\_, \_, C\#)$, $input_6(Pd, Rd, Ct)$.
10	delete_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_{10}(C\#)$, AD_Reservation($C\#, Pd, Rd, Ct$). delete_AD_Reserves($Cid, C\#$) :- $input_{10}(C\#)$, AD_Reserves($Cid, C\#$).

Step 3: Test Sequence Generation

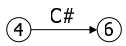
- Test Sequence Generation
  - A test sequence: a sequence of transitions
  - Apply graph theory algorithm on the state transition diagram

Step 4: Dynamic Data Generation

- Test case
  - An instantiation of the input parameters for the entire test sequence
  - Generated based on a sequence of dynamic changing AD states
- Dynamic data specification
  - Combines the input and update specification
  - Two Components:
    - Data Inheritance Graph (DIG)
    - Dynamic Input Specification (DIS)

Data Inheritance Graph (DIG)

- Model data inheritance
- Generated based on data specification
- Example:



Transition	Update Specification
6	delete_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_4(\_, \_, C\#)$, AD_Reservation($C\#, Pd, Rd, Ct$). insert_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_4(\_, \_, C\#)$, $input_6(Pd, Rd, Ct)$.
10	delete_AD_Reservation($C\#, Pd, Rd, Ct$) :- $input_{10}(C\#)$, AD_Reservation($C\#, Pd, Rd, Ct$). delete_AD_Reserves($Cid, C\#$) :- $input_{10}(C\#)$, AD_Reserves($Cid, C\#$).

Dynamic Input Specification (DIS)

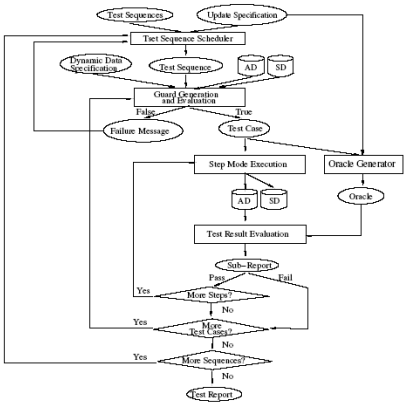
- Specify new versions of the AD/SD
- Automatically merging the input and update specifications
- Example

$$
\begin{aligned}
&input_6(Pd, Rd, Ct, \mathbf{C\#}, AD_{in}, AD_{out}, SD_{in}, SD_{out}) :- \\
&\quad \text{SD\_Reservation}(\_, Pd, Rd, Ct, SD_{in}), \\
&\quad \text{AD\_Reservation}(C\#, Pd', Rd', Ct', AD_{in}), \\
&\quad delete\_AD(\text{Reservation}(C\#, Pd', Rd', Ct'), AD_{in}, AD_t), \\
&\quad insert\_AD(\text{Reservation}(C\#, Pd, Rd, Ct), AD_t, AD_{out}), \\
&\quad delete\_SD(\text{Reservation}(\_, Pd, Rd, Ct), SD_{in}, SD_{out}).
\end{aligned}
$$

Step 5: Testing Process

- Test sequence scheduler
- Guard generation and evaluation
- Oracle generation
- Step mode execution
- Test result evaluation

Testing Process



Test Sequence Scheduler

- Motivation: Mitigate the impact of prior tests
- Strategy: Delay deletion
- Method:
  - Test sequences are classified into four groups: read-only, insertion-only, mixed insertion and deletion, and deletion-only
  - Scheduled in that order

Guard Generation

- Motivation
  - Maintain data inheritance
  - Generate each test case as a whole
- Strategy: Guard
  - Prolog query
  - Determine test case generation possibility
  - Constructed by concatenating the heads of corresponding Dynamic Input Specification rules

Guard Generation (Example)

- Test sequence for Modify Reservation path:
- 2->3->2->3'->2->4->5->2->4->5'->2->4->6->7
- Corresponding guard query:

$$
\begin{aligned}
?\text{-}\ & input_3(Fn_1, Ln_1, C\#_1, AD_{in}, AD_1),\\
& input_{3'}(Fn_2, Ln_2, C\#_2, SD_{in}, SD_1),\\
& input_4(Fn_3, Ln_3, C\#_3, AD_1, AD_2),\\
& input_5(Pd_1, Rd_1, Ct_1, AD_2, AD_3),\\
& input_4(Fn_4, Ln_4, C\#_4, AD_3, AD_4),\\
& input_{5'}(Pd_2, Rd_2, Ct_2, SD_1, SD_2),\\
& input_4(Fn_5, Ln_5, C\#_5, AD_4, AD_5),\\
& input_6(Pd_3, Rd_3, Ct_3, C\#_5, AD_5, AD_6, SD_2, SD_3).
\end{aligned}
$$

Guard Evaluation

- Using the Prolog interpreter
- Successful evaluation
  - Test Case
- Unsuccessful evaluation
  - ADi: bad state for test case generation
  - Test sequence: put back to the queue.

Test Result Evaluation

- Step mode execution
  - Transaction by transaction
- Test Result Evaluation
  - After each transaction commits
  - ADi: checked by an oracle

Partial Oracle Generator

- Direct comparison impossible
  - Database state is invisible
  - Costly to instantiate an oracle AD
- Partial testing is feasible
  - Partial oracle: partial correctness property
  - e.g. update correctness

Partial Oracle Definition

- A set of Prolog rules
- Two possible forms

$$\begin{aligned}&\text{1) insert\_}\textit{Relation}\text{\_Failed :-}\\&\qquad \text{insert\_AD\_}\textit{Relation}(A_1,\ldots,A_k),\\&\qquad \neg\text{AD\_}\textit{Relation}(A_1,\ldots,A_k).\\&\text{2) delete\_}\textit{Relation}\text{\_Failed :-}\\&\qquad \text{delete\_AD\_}\textit{Relation}(A_1,\ldots,A_k),\\&\qquad \text{AD\_}\textit{Relation}(A_1,\ldots,A_k).\end{aligned}$$

- Generated automatically based on the update specification.

Partial Oracle Evaluation

- Evaluated by executing Prolog queries
  - ?- insert_Relation_Failed.
  - ?- delete_Relation_Failed.
- Successful evaluation: update failed
- Unsuccessful evaluation: update succeeded

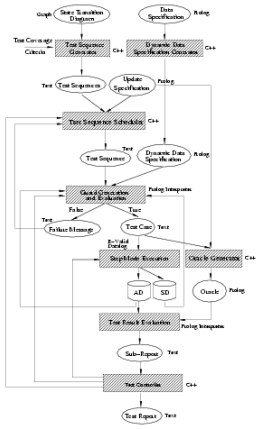
Analysis of the Test Report

- Three Error Categories
  - Modeling Error
    - Wrong input/update specification
  - Buggy Implementation of this Particular Transaction
  - Buggy Implementation of the Previous Transactions
    - Due to the limitation of the partial oracle

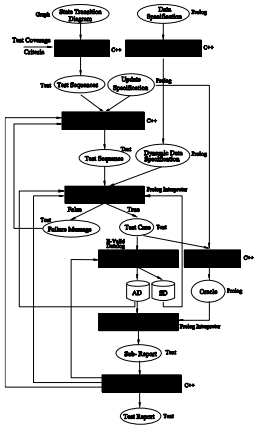
Outline

- Motivation
- Two approaches
- AutoDBT
- Implementation plan
- Related work
- Conclusion and future work

Implementation Plan



Implementation Plan



Implementation Plan

- Test Sequence Generator: C++
- Dynamic Data Specification Generator: C++
- Test Sequence Scheduler: C++
- Guard Generator and Evaluator: Prolog Interpreter

Implementation Plan

- Step Mode Executor: E-valid
- Oracle Generator: C++
- Test Result Evaluator: Prolog interpreter
- Test Controller: C++

Related Work

- Testing of web application without incorporating testing the application database
- Database testing
  - Performance testing of the DBMS
- No related work on testing web database application

Conclusions

- Identify an open problem
  - Testing web database application
- Design of AutoDBT
  - Extends the functional testing to include database update
  - Model dynamic behavior of AD with DEFSM
  - Valid test case generation: Guard
  - Partial oracle generation and evaluation

Future Work

- Implementation of AutoDBT
- Investigate other kinds of testing
- Improve the feedback on failed tests
  - A hint on fixing the problem
- Improve the guards
  - Populate the database with “good” data