Generating Test Cases for XML-based Web Component Interactions Using Mutation Analysis

Suet Chun Lee
BUSINEX, Inc.
San Diego, CA USA
Suetl@businex.com

A. Jeff Offutt
George Mason University
Fairfax, VA USA
www.ise.gmu.edu/faculty/ofut/

Support from NSF is greatly appreciated.

Multi-tiered Web Software Systems

Client
Browser
Javascipts
Applet

Web Server
HTTP
HTML, XML pages
CGI script
JSP, ASP
Servlet, Javabeans, etc

Application Server
Middleware
EJB

DB Server
Middleware
Client-server ... 3-tier ... N-tier ...
Peer-to-Peer (P2P) Web Component Interactions

Web Component C1

Web Component C2

Web Component C3

Web Component C4

Web Component C5

Web Component C6

Web Component C7

Web Component C8

Web Component C9

The Problem and a Solution

Solution: A testing technique --> Interaction Mutation (IM)

Web Software Quality Attributes
- Correctness
- Usability
- Security
- Availability
- Scalability
- Maintainability
- Performance

Test Level (Scope)
- Unit testing
- Integration testing
- System Testing
- Interaction vs. integration
- Example interaction faults: Timing error, version mismatch, wrong assumptions on the interacting party, etc.

Consumers'

Primary Concern

Suppliers'

Primary Concern
Mutation Analysis

- **White-box method to develop test cases** → guides testers to create effective tests
- **Test Process**
  - A test case that kills a mutant can also detect the fault
  - Test case exercises that part of the program code
- **Rationale**
  - Computationally expensive if program is large

Formulation of the Solution

- **XML** → data exchange between heterogeneous web components
- **Specification for the P2P interaction**
  - Elements and element definitions involve in the message exchange
  - The messages
  - Test requirement → test specification
  - Test criterion → when to stop the test
- **Model the interaction and formally represent the test criterion** → build program to use the model as a test framework
Example XML Document and its DTD

```xml
<?xml version = "1.0">
<AUTHORIZED_USERS>
  <AUTHORIZED_USER>
    <USER_ID>jenny</USER_ID>
    <PASSWORD>jen</PASSWORD>
  </AUTHORIZED_USER>
</AUTHORIZED_USERS>
```

<!ELEMENT AUTHORIZED_USERS (AUTHORIZED_USER)>  
<!ELEMENT AUTHORIZED_USER (USER_ID, PASSWORD)>  
<!ELEMENT USER_ID (#PCDATA)>  
<!ELEMENT PASSWORD (#PCDATA)>  

A Model for Web Component Interactions

- A formal model of web component interactions is needed to automate testing
- Model* consists of
  - DTDs (a language to specify the grammar of XML documents) to formally define interaction vocabularies
  - Messages
  - Constraints

*Extends Fan and Simeon’s DTD structure and integrity constraints
Define Interaction Specification Model (ISM)

- DTD structure $S = (E, P, R)$, where:
  - $E$ is a set of elements
  - $P$ maps elements to element definitions
  - $R$ maps attributes of elements to the attribute types
- $M = (\text{request}, \text{response})$, request and response are XML document constructs defined in $S$
- $L_M$ is a constraint language
  - Two constraints defined: memberOf and lengthOf
- $\Sigma$ = a set of basic XML constraints expressed in $L_M$
- ISM = $(S, M, \Sigma)$

An Interaction Specification in DTD

```xml
<!-- Collection of authorized users -->
<!ELEMENT AUTHORIZED_USERS (AUTHORIZED_USER)*)
<!ELEMENT AUTHORIZED_USER (USER_ID, PASSWORD)>
<!-- Request message -->
<!ELEMENT SIGNON_REQUEST (USER_ID, PASSWORD)>
<!-- Response message -->
<!ELEMENT SIGNON_RESPONSE (USER_ID)>
<!ATTLIST SIGNON_RESPONSE AUTHENTICATION (ALLOW | DENY) #REQUIRED>

<!-- DTD for computer accounts -->
<!ELEMENT USER_ID (#PCDATA)>
<!ELEMENT PASSWORD (#PCDATA)>
```
**Model the Interaction Specification Using ISM**

\[ S = (E, P, R) \]

\[ E = \{ \text{USER\_ID, PASSWORD, AUTHORIZED\_USERS, AUTHORIZED\_USER, SIGNON\_REQUEST, SIGNON\_RESPONSE} \} \]

\[ P(\text{SIGNON\_REQUEST}) = (\text{USER\_ID, PASSWORD}) \]

\[ P(\text{SIGNON\_RESPONSE}) = \ldots \text{ etc.} \]

\[ R(\text{SIGNON\_RESPONSE AUTHENTICATION}) = \text{string values} \]

\[ M = (\text{request, response}) \]

\[ \text{request} = \{ P(\text{SIGNON\_REQUEST}) \} = \{ (\text{USER\_ID, PASSWORD}) \} \]

\[ \text{response} = \{ P(\text{SIGNON\_RESPONSE}) \} = \{ (\text{USER\_ID}) \} \]

\[ \Sigma \text{ consists of memberOf and lenOf constraints:} \]

\[ \Sigma = \{ \text{SIGNON\_REQUEST \{USER\_ID, PASSWORD\}} \not\subseteq \text{AUTHORIZED\_USER \{USER\_ID, PASSWORD\}}, \]

\[ | \text{PASSWORD [String Value S]} | \neq 3 \} \]

**Generate ISM’**

Applying interaction mutation operator class on constraints of ISM

\[ \Sigma = \{ \text{const}1, \text{const}2, \ldots \} \]

\[ \Sigma’ = \{\text{IMO}1, \text{IMO}2, \ldots \} \]

\[ \Sigma = \{ \text{SIGNON\_REQUEST \{USER\_ID, PASSWORD\}}, \]

\[ \text{AUTHORIZED\_USER \{USER\_ID, PASSWORD\}}, | \text{PASSWORD [String Value S]} | \neq 3 \} \]

\[ \Sigma’ = \{ \text{SIGNON\_REQUEST \{USER\_ID, PASSWORD\}}, \]

\[ \text{AUTHORIZED\_USER \{USER\_ID, PASSWORD\}}, | \text{PASSWORD [String Value S]} | \neq 3 \} \]
IM Mutant Generation Process

Test case $t$ → Interaction Mutation System (IMS) → Mutant of Interaction Specification Model (ISM') → Generate Mutant Interaction $I_1, I_2, I_3, \ldots$

Example:
- # of test cases = 10
- # of IMOs = 10
- Total # of mutant interactions = 100

Interaction Mutation Test Process

Web Component C1 request → Interaction I response → Interaction Mutation System (IMS) → Execute T

Web Component C2 request → Mutant of Interaction Specification Model (ISM') → Generates Interaction Mutation Operators (IMOs) → Execute T

Web Component C2 request → response
## Mutation Analysis vs. Interaction Mutation (IM) Analysis

<table>
<thead>
<tr>
<th></th>
<th>Mutation Analysis</th>
<th>Interaction Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basis</strong></td>
<td>Usually source code and some formal specs.</td>
<td>ISM</td>
</tr>
<tr>
<td><strong>What to mutate</strong></td>
<td>Program source code</td>
<td>XML messages</td>
</tr>
<tr>
<td><strong>Test level</strong></td>
<td>Unit &amp; module testing</td>
<td>Integration testing</td>
</tr>
<tr>
<td><strong>Test cases</strong></td>
<td>Input and output values of the program units</td>
<td>XML messages exchanged between interacting web components</td>
</tr>
</tbody>
</table>

### IM Example: Authentication

- A web component C1 requires users to login
- Authentication for C1 is provided by C2
- C1 sends an XML message requesting C2 to authenticate a user
- C2 does the verification and responds to C1 with an XML message that allows or denies the user access
### Request and Response Messages

**Request Message**
```xml
<? xml version=1.0?>
<SIGNON_REQUEST>
  <USER_ID>Jenny</USER_ID>
  <PASSWORD>jen</PASSWORD>
</SIGNON_REQUEST>
```

Assume user Jenny with password “jen” is in the authorized user database.

**Response Message**
```xml
<? xml version=1.0?>
<SIGNON_RESPONSE AUTHENTICATION="ALLOW">
  <USER_ID>Jenny</USER_ID>
</SIGNON_RESPONSE>
```

### Use NOT MemberOf IMO to Mutate Request Message

**Mutated Request Message**
```xml
<? xml version=1.0?>
<SIGNON_REQUEST>
  <USER_ID>Jeff</USER_ID>
  <PASSWORD>jen</PASSWORD>
</SIGNON_REQUEST>
```

Assume user Jeff is not in the authorized user database.

**Response Message**
```xml
<? xml version=1.0?>
<SIGNON_RESPONSE AUTHENTICATION="DENY">
  <USER_ID>Jeff</USER_ID>
</SIGNON_RESPONSE>
```

Response is different from that of the original message, so mutant is “killed”
Use NOT lenOf IMO to Mutate Request Message

<table>
<thead>
<tr>
<th>Mutated Request Message</th>
<th>Response Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;? xml version=1.0?&gt;</code></td>
<td><code>&lt;? xml version=1.0?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;SIGNON_REQUEST&gt;</code></td>
<td><code>&lt;SIGNON_RESPONSE</code></td>
</tr>
<tr>
<td><code>&lt;USER_ID&gt;Jenny&lt;/USER_ID&gt;</code></td>
<td><code>&lt;AUTHENTICATION=&quot;DENY&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;PASSWORD&gt;jenXXX&lt;/PASSWORD&gt;</code></td>
<td><code>&lt;USER_ID&gt;Jenny&lt;/USER_ID&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/SIGNON_REQUEST&gt;</code></td>
<td><code>&lt;/SIGNON_RESPONSE&gt;</code></td>
</tr>
</tbody>
</table>

Assume Jenny’s password consists of 3 characters.

Response is different from that of the original message, so mutant is “killed”

Conclusions

- New testing technique for validating semantic correctness of web component interactions → IM
  - Create mutants of XML message
  - Source code is not needed (allowing third party vendors)

- Interaction Specification Model
- Test criteria → constraints
- IMO class
Future Work

- Define additional classes of interaction mutation operators
- DTD → schema language
- Extend the technique to handle composite multilateral interactions
- Construct a test environment
- Empirical validation