

Using XML to Test Web Software Services

Jeff Offutt

Information & Software Engineering
George Mason University
Fairfax, VA USA
www.ise.gmu.edu/faculty/ofut/

Joint research with Suet Chun Lee, GMU PhD student
Thanks in part to Dick Lipton

Supported by NIST and NASA.

Modern Web Sites

Web sites have continuously evolved in the last decade

We have moved from:

- ftp/email ...
- to gopher ...
- to simple html pages ...
- to web sites ...
- to dynamic html ...
- to web commerce ...

with amazing speed!

Modern Web Sites

- **Web sites are now too complicated for individuals to manage.**
- **They need to be engineered by teams of people with diverse talents:**
 - Programming skills
 - Graphics design
 - Usability
 - Information layout and engineering
 - Data communications
 - Data base

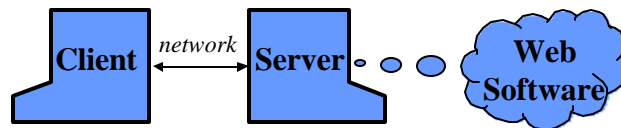
We need web site engineering

© Offutt & Lee

3

Web Sites and Software

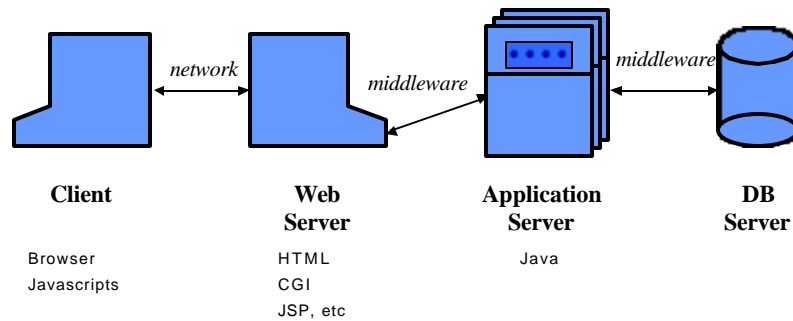
- **Web Page : Data that fits in one browser screen.**
- **Web Site : A number of connected web pages.**
- **Web Site Software : Software that makes web sites dynamic.**



© Offutt & Lee

4

Multi-tiered Web Software Systems



Client-server ... 3-tier ... N-tier ...

© Offutt & Lee

5

Important Web Software Quality Attributes

1. **Reliability**
2. **Usability**
3. **Security**

4. **Availability**
5. **Scalability**
6. **Maintainability**
7. **Performance**

Customers have little "site loyalty" and will switch quickly, thus time to market is much less important than in other application areas.

(but still important!)

© Offutt & Lee

6

Web Sites are Much More Complex

- **1995: Web sites were 100% interface**
- **1998: Web sites were about 90% interface**
- **2001: Web sites are about 50% interface**

There is a shortage of knowledgeable, skilled web programmers and software engineers.

Summary Changes in Web Technologies

1997

- Static web pages
- “Soft brochures”
- Webmasters
- HTML, CGI, JavaScript

1998-1999

- Dynamic HTML
- Programs (poorly written)
- Confused webmasters
- ASP, CSS, ...

2000-2001

- Functional websites
- ECommerce +
- WebManager + programmers, DB, network, UI, graphics designers, ...
- Java J2EE (JSP, Servlets, beans), {HT,U,X}ML, Component-based

Summary Concerns of Software

Traditional

1. Efficiency of process
(time to market)
2. Efficiency of execution



50. Reliability
51. Safety
52. Maintainability
53. Security



Web Software

1. Reliability
2. Usability
3. Scalability
4. Security
5. Availability
6. Maintainability
7. Performance



© Offutt & Lee

9

Complexities of Web Site Software

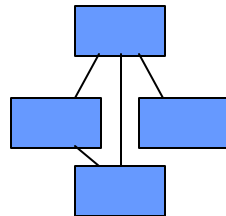
- **Heterogeneous**
 - Multiple hardware platforms
 - Multiple software platforms
 - Multiple software languages
 - Multiple software models (imperative, declarative, functional, ...)
 - Multiple organizations building the software
- **Concurrent and distributed**
- **High quality requirements**
- **High degree of reuse and third party components**
- **Loosely coupled ...**

© Offutt & Lee

10

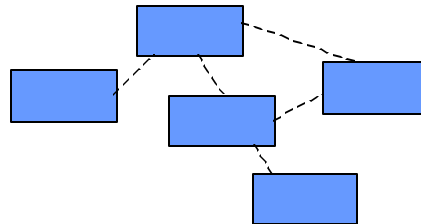
Complexities of Web Site Software

Loosely coupled



Traditional systems

Connected by calls and message passing
High and moderate coupling



Web-based systems

Connected with network protocols
Loose and *extremely* loose coupling

How can we ensure the reliability of this type of system?

© Offutt & Lee

11

Communicating in Loosely Coupled Systems

- **In traditional software, components and component authors negotiate details about structure of data that is exchanged**
 - Types
 - Formats
 - Order
- **This is difficult in extremely loosely coupled software**
 - Authors cannot communicate
 - Components may not know who they interact with until execution time
- **XML is a standard that allows data items to be exchanged:**
 - Independent of type
 - Without regard to format
 - In arbitrary order
- **XML tags allow components to infer information about type, format and order**

© Offutt & Lee

12

The Problem Context



- **Heterogeneous web software interactions**
- **Components communicate by using an agreed upon standard for data exchange – XML**

Testing: A technique to check the functional correctness of the interaction:

Interaction Mutation (IM)

© Offutt & Lee

13

Background

- **Web Components**
- **eXtensible Markup Language (XML)**
- **Testing and Mutation Analysis**

© Offutt & Lee

14

Web Components

- **Web component**: A software module or collection of modules that execute on the world wide web
- **Examples:**
 - Java server pages
 - Java servlets
 - JavaScripts
 - Active server pages
 - Java beans and non-bean classes
 - Databases

eXtensible Markup Language (XML)

- A universal format for structured documents and data on the web
- A recommendation of the World Wide Web Consortium (W3C)
- Enables data to be exchanged among heterogeneous software and hardware platforms

Example XML Document and its DTD

```
<?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)>
```

© Offutt & Lee

17

Background: Software Testing

- **Unit and Module Testing**: Testing individual procedures and groups of related procedures
- **Integration Testing**: Testing for interface problems and incompatibilities between objects
- **System Testing**: Testing a complete system from an external perspective

We are focusing on integration testing of web applications

© Offutt & Lee

18

Testing Criteria

- **Test requirement**
 - Something in the software that needs to be tested
- **Test specification**
 - Description of input needed to satisfy a requirement
- **Test criterion**
 - Imposes a collection of test requirements

Informal Definition: A goal or stopping rule for testing

Testing and Mutation Analysis

- **Mutation testing**
 1. Modifies the source code
 2. Produces mutants
 3. Guides testers to create effective tests
 4. Test cases that kill mutants also find faults
- **Fundamental mutation premise: If a fault exists, there is usually a mutant that will only be killed by a test case that also detects the fault**
- **Mutation testing is computationally expensive**
- **It is practical for small software components ... most web components are small!**

Interaction Mutation (IM) Analysis

- **Traditional mutation analysis:**
 - modifies the program source
 - is primarily used for unit and module testing
 - uses test cases that are input values to program units
- **Interaction mutation analysis:**
 - mutates the of web service data interactions (the messages)
 - is used for integration testing of web software components
 - uses test cases that are XML messages between web software components
- **IM creates test cases as XML messages from the XML grammar description (DTDs)**

A Model for Web Service Interactions

- **A formal model of web service interactions is needed to automate testing**
- **Model uses:**
 - formally defined DTDs
 - messaging interfaces
 - constraints
- **Extends Fan and Simeon's DTD structure and integrity constraints**

Definition

Interaction Specification Model (ISM)

- **DTD structure $S = (E, P, R)$, where:**
 - E are the elements
 - P maps elements to element definitions
 - R maps attributes of elements to the attribute type
- **Lm = constraint language**
 - Two constraints currently defined in Lm: *membership* and *length*
- **M = (request, response), request and response are defined in S**
- **S = a set of basic XML constraints expressed in Lm**
- **ISM = (S, M, Σ)**

ISM Example

```
<!-- DTD for Computer Accounts -->
<!ELEMENT USER_ID (#PCDATA)>
<!ELEMENT PASSWORD (#PCDATA)>

<!-- Collection of authorized users -->
<!ELEMENT AUTHORIZED_USERS (AUTHORIZED_USER) * >
<!ELEMENT AUTHORIZED_USER (USER_ID, PASSWORD)>

<!-- XML request sent, SignOn A -->
<!ELEMENT SIGNON_REQUEST (USER_ID, PASSWORD)>

<!-- XML response sent, Authenticate B -->
<!ELEMENT SIGNON_RESPONSE (USER_ID)>
<!ATTLIST SIGNON_RESPONSE AUTHENTICATION (ALLOW | DENY) #REQUIRED>
```

ISM Example (2)

$S = (E, P, R)$

$E = \{ \text{USER_ID, PASSWORD, AUTHORIZED_USERS, AUTHORIZED_USER, SIGNON_REQUEST, SIGNON_RESPONSE} \}$

$P(\text{SIGNON REQUEST}) = (\text{USER ID, PASSWORD})$

$P(\text{SIGNON_RESPONSE AUTHENTICATION}) = \text{string values}$

$M = (\text{request, response})$

$\text{request} = \{ P(\text{SIGNON REQUEST}) \}$

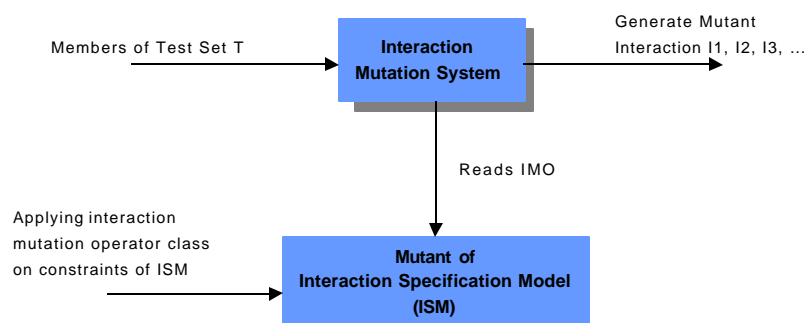
$\text{response} = \{ P(\text{SIGNON RESPONSE}) \}$

Σ consists of memberOf and lenOf constraints:
 $\Sigma = \{ \text{SIGNON REQUEST (USER ID, PASSWORD)}$
 $\subseteq \text{AUTHORIZED USER (USER ID, PASSWORD)}$
 $| \text{PASSWORD [String Value S] } | = 3$
 $\}$

© Offutt & Lee

25

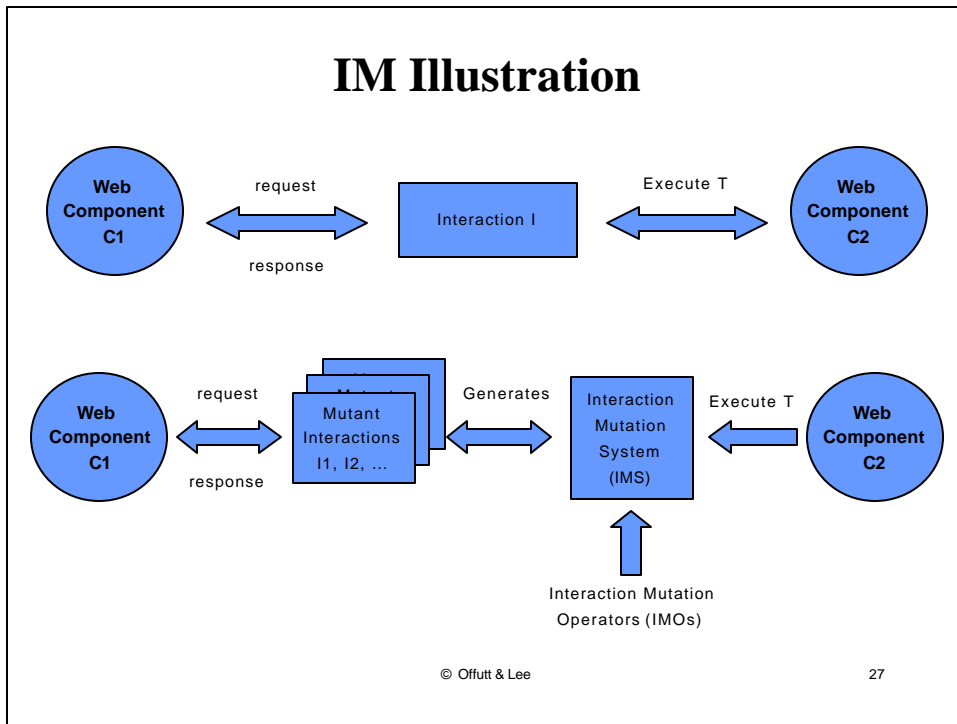
Interaction Mutation Test Process



© Offutt & Lee

26

IM Illustration



Example: Authentication



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

Request and Response Messages

Request Message

```
<? xml version=1.0?>
<SIGNON REQUEST>
  <USER ID>Jenny</USER ID>
  <PASSWORD>jen</PASSWORD>
</SIGNON REQUEST>
```

Response Message

```
<? 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 version=1.0?>
<SIGNON REQUEST>
  <USER ID>Jeff</USER ID>
  <PASSWORD>jen</PASSWORD>
</SIGNON REQUEST>
```

Response Message

```
<? xml version=1.0?>
<SIGNON RESPONSE
  AUTHENTICATION="DENY">
  <USER ID>Jeff</USER ID>
</SIGNON RESPONSE>
```

Assumed: The pair "Jeff, jen" is not in the authorized user database.

Response is different from that of original message, so mutant is "killed"

Use NOT lenOf IMO to Mutate Request Message

Mutated Request Message

```
<? xml version=1.0?>
<SIGNON REQUEST>
  <USER ID>Jenny</USER ID>
  <PASSWORD>jenXXX</PASSWORD>
</SIGNON REQUEST>
```

Response Message

```
<? xml version=1.0?>
<SIGNON RESPONSE
  AUTHENTICATION="DENY">
  <USER ID>Jenny</USER ID>
</SIGNON RESPONSE>
```

Given: Jenny's password is of length 3.

Response is different from that of original message,
so mutant is "killed"

© Offutt & Lee

31

Applications

- Can be used anywhere that XML is used
- Useful when third-party software components whose source is not available
- Most newer web applications use XML for communication
 - Between tiers
 - Between software and database
 - Between heterogeneous software components
- Loosely coupled software applications

© Offutt & Lee

32

Conclusions

- **New testing technique for validating functional correctness of web component interactions → IM**
 - Source code is not needed (allowing third party vendors)
 - Create mutants of XML message
- **Interaction Specification Model**
- **Test criteria → constraints**
- **IMO classes, IMOs**

Future Work

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

Relevant papers: www.ise.gmu.edu/~ofut/rsrch/web.html