

# CASE STUDIES OF SUMMER MODEL-BASED TESTING FRAMEWORK

V. Kuliamin, N. Pakulin, A. Tugaenko

Institute for System Programming, Russian Academy of Sciences

# OUTLINE

## ❑ Description of the tool

- Intended domain
- Main features
- Similarities and differences with other tools

## ❑ Case studies

- SMPT protocol
- Part of DOM API
- Part of Google Web UI

# THE TOOL

---

- Summer  
Test development and execution framework
- Features
  - Java API testing (and where possible to link up)
  - xUnit-like test presentation
  - Unit – Component levels
  - Black-box testing
  - MBT features
    - EFSM-based testing
    - Software contracts as test oracles

# VERY SIMPLE TEST EXAMPLE (JUST XUNIT)

```
@Test
public class TestClass {

    Account target = new Account(); // object under test

    @Test
    public void testDeposit() {
        int oldBalance = target.getBalance();
        int deposited = 5;

        target.deposit(deposited);
        assertEquals(target.getBalance(), oldBalance + deposited
                    , "Balance should grow by deposited sum");
    }
}
```

# ELABORATED TEST EXAMPLE

```
@Test
public class TestClass {

    Account target = new Account();

    @State
    public int balance() { return target.getBalance(); }

    int[] sums = new int[]{0, 1, 2, 3, 5, 17, 238};
    public boolean bound() { return balance() < 350; }

    @Test
    @DataProvider(name = "sums")
    @Guard(names = "bound")
    public void testDeposit(int sum) { ... target.deposit(sum); ... }

    @Test
    @DataProvider(name = "sums")
    public void testWithdraw(int sum) { ... target.withdraw(sum); ... }
}
```

# **SIMILAR TOOLS AND ADDITIONS**

---

- TestNG (one of most powerful xUnit tools, Java)
  - Testware hierarchy : test suites – tests – test classes – test methods
  - Setup-teardown methods on all hierarchy levels
  - Test methods grouping and selection by groups
  - Test methods sequencing
  - Test data providers
- NModel (MBT tool extending xUnit, C#)
  - State-based testing
  - Guard conditions
  - Compositions of test classes
- Additions
  - Stateful software contracts (described separately of tests)
  - Aspect-based linking of external components : contracts, coverage models, etc.
  - Combinations of data providers for parameters
  - More flexible data providers, guard conditions, state definitions

# CONTRACT EXAMPLE

```
public class AccountContract {
    int balance;
    int maxCredit;

    public boolean withdrawPost(int sum) {
        if (Contract.oldValue(balance) - sum > maxCredit)
            return assertEquals(Contract.intResult(), sum
                , "Result should be equal to the argument")
                && assertEquals(balance, Contract.oldValue(balance) - sum
                , "Balance should be increased on the argument");
        else
            return assertEquals(Contract.intResult(), 0
                , "Result should be 0")
                && assertEquals(balance, Contract.oldValue(balance)
                , "Balance should not change");
    }
}
```

# ASPECT-BASED LINKING

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns= ... >
    ...
    <bean id="accountContract" class="mbtest.tests.account.AccountContract">
        <property name="checkedObject" ref="accountImpl"/>
    </bean>

    <bean id="accountContractExecutor" class="mbtest.contracts.ContractExecutor">
        <property name="postcondition"
                  value="mbtest.tests.account.AccountContract.withdrawPost(int)"/>
        <property name="updater"
                  value="mbtest.tests.account.AccountContract.transferUpdate"/>
        <property name="contract" ref="accountContract"/>
    </bean>

    <aop:config>
        <aop:aspect id="accountContractAspect" ref="accountContractExecutor">
            <aop:pointcut id="accoutTransfer"
                           expression="execution(* mbtest.tests.account.Account.withdraw(..))"/>
            <aop:around pointcut-ref="accoutTransfer" method="execute"/>
        </aop:aspect>
    </aop:config>
</beans>
```

# CASE STUDIES

---

- ❑ Testing
  - SMTP protocol implementations (against SMPT RFC)
  - Part of Xerces DOM implementation (against DOM API standard)
  - Part of Google WebUI (against simple intuition)
- ❑ Main goal : to check the following ideas
  - Idea 1  
Flexibility of component architecture facilitates usage of generic tools in various domains
  - Idea 2  
Modular testware (separate components : test oracles, test sequence generator, test data generators, test coverage measurement) helps to achieve comprehensive testing with less effort

# SMPT CASE STUDY

- ❑ Simple Mail Transfer Protocol

RFC 5321 [2008]

- ❑ Client

- Basic actions:

<connect>

[HELO | EHLO] ...

( MAIL FROM: <...> (RCPT TO: <...>)+ DATA (<line>)\* . )+

QUIT

- Additional : NOOP, RSET { VRFY, EXPN, HELP }

- ❑ Server responses : [2-5][0-2|5][0-9] ...

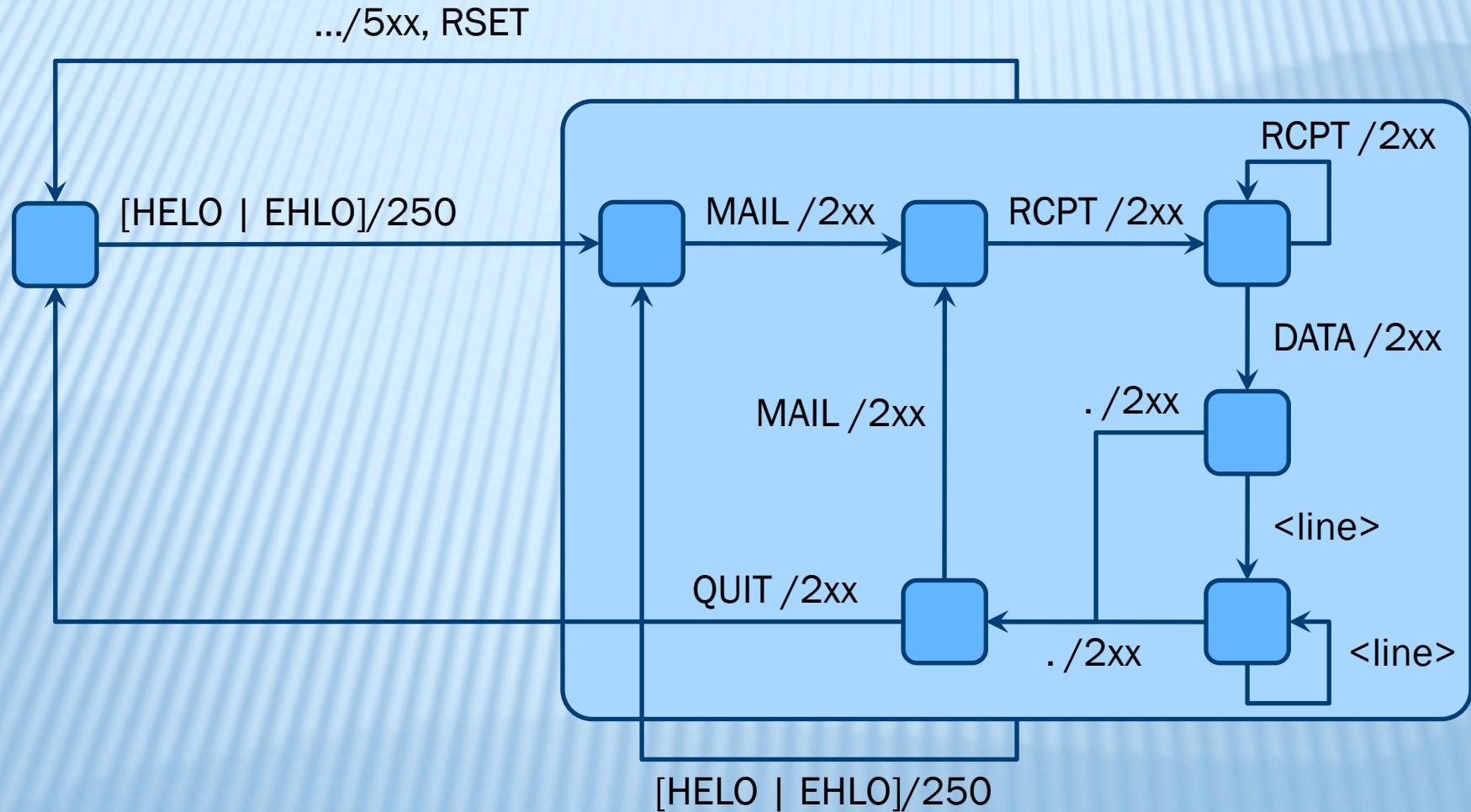
- ❑ Extensions

RFC 4954 (AUTH) { RFC 1652, 1879, 2034, 2920, 3030, 3207, 3461, 3463, 3865, 3885, 4095, 4405, 4865, 4954, 5336 }

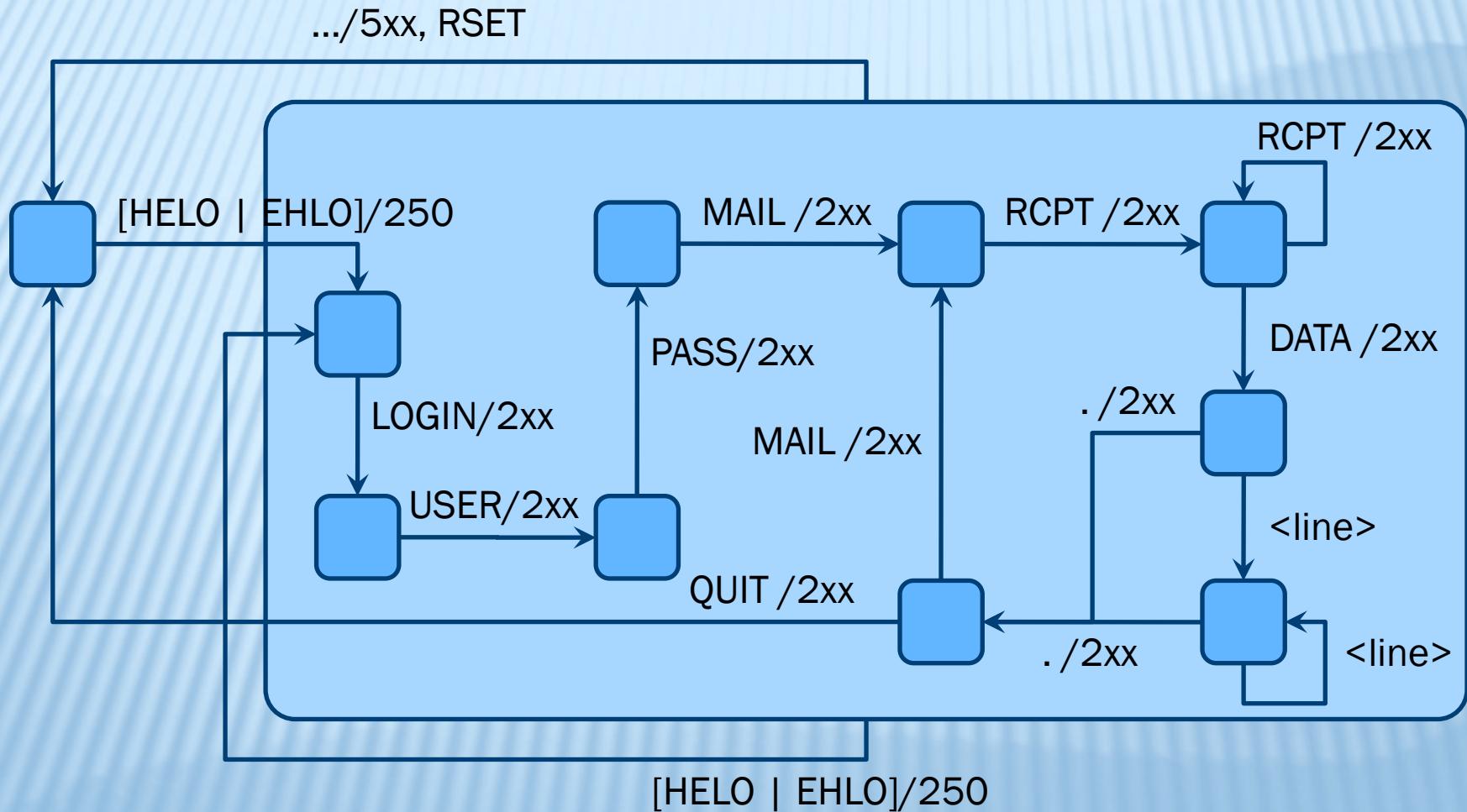
# MODULAR TEST MODEL

- ❑ Separate CONNECT-DISCONNECT test model
  - Can be used with other over-transport protocols
- ❑ Separate basic SMTP test model
- ❑ Separate AUTH PLAIN test model
- ❑ Possibility to add other extensions

# TEST STATE MACHINE (SLIGHTLY SIMPLIFIED)



# TEST STATE MACHINE WITH AUTH PLAIN



# SMTP CASE STUDY STATISTICS (LOC)

Testware module	Test model	Contract	Other	Total
Connect-Disconnect	90	140		230
Basic SMTP	200	480		680
Authentication	140	300		440
Auxiliary			680	680
Configuration			230	230
Total	430	920	910	2260

# DOM CASE STUDY

- ❑ DOM API Standard
  - Document Object Model – internal representation of web pages in browsers
- ❑ Node interface
- ❑ SUT – Xerces for Java [[xerces.apache.org](http://xerces.apache.org)]

## appendChild modified in DOM Level 3

Adds the node newChild to the end of the list of children of this node  
If the newChild is already in the tree, it is first removed.

Parameters:

newChild of type `Node` [p.56]

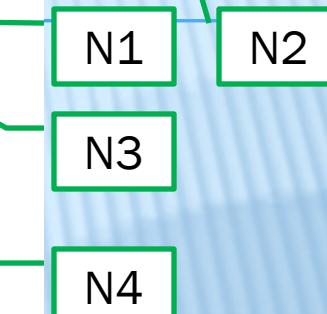
The node to add.

If it is a `DocumentFragment` [p.40] object, the entire contents of the document fragment are moved into the child list of this node

Return Value

`Node` [p.56]

The node added.



### Exceptions

`DOMException`  
[p.31]

E3

HIERARCHY REQUEST ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors [p.205] or this node itself, or if this node is of type `Document` [p.41] and the DOM application attempts to append a second `DocumentType` [p.115] or `Element` [p.85] node.

E1

E2

E4

E5

E6

WRONG DOCUMENT ERR: Raised if newChild was created from a different document than the one that created this node.

E7

E8

NO MODIFICATION ALLOWED ERR: Raised if this node is readonly or if the previous parent of the node being inserted is readonly.

E9

E10

NOT SUPPORTED ERR: if the newChild node is a child of the `Document` [p.41] node, this exception might be raised if the DOM implementation doesn't support the removal of the `DocumentType` [p.115] child or `Element` [p.85] child.

# TRACING REQUIREMENTS



Package Hierarchy

- domproc
- domtest
  - src
    - DOMTest.java
    - DOMTestOld.java
    - NodeContract.java
    - TreeState.java
    - TypeTest.java
    - TypeTreeState.java
  - domtest.xml
- JRE System Library [JavaSE-6]
- Referenced Libraries

```

    @SuppressWarnings("unchecked")
    public boolean appendChildNormalPost(Node n)
    {
        boolean common =
            Contract.assertIdentical(parent, Contract.<Node>oldValue(parent)
                , "Parent node should be preserved")
            && Contract.assertIdentical(owner, Contract.<Node>oldValue(owner)
                , "Owner document should be preserved")
            && Contract.assertIdentical(Contract.<Node>result(), n
                , "[N4]" + " The node added should be returned");

        if(n instanceof DocumentFragment)
        {
            List<Node> appendedNodes = Contract.<List<Node>>oldValue(toList(n.getChildNodes()));

            for(int i = 0; i < appendedNodes.size(); i++)
                common &= Contract.assertTrue(target.isSameNode(appendedNodes.get(i).getParentNode())
                    , "[N3]" + " This node should become parent for all appended nodes");

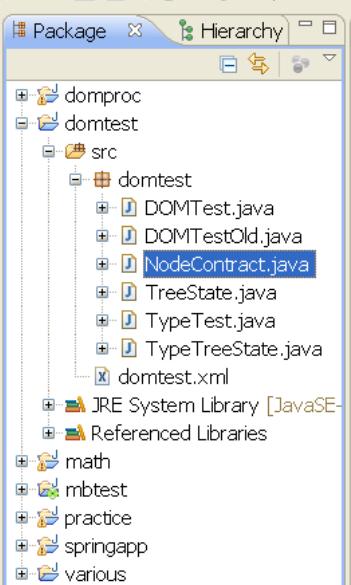
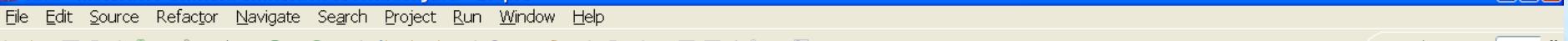
            return common
                && Contract.assertIdentical(n.getChildNodes().getLength(), 0
                    , "The new contents of the document fragment should be empty")
                && Contract.assertEquals(Utils.getPostfix(children, appendedNodes.size()), appendedNodes
                    , "[N3]" + " The entire contents of the document fragment should be moved in the end of the child list of this node")
                && Contract.assertEquals(Utils.getPrefix(children, appendedNodes.size()), Contract.<Object>oldValue(((ArrayList<Node>)children).subList(0, children.size() - 1))
                    , "All other children should be preserved");
        }
        else
        {
            common &=
                Contract.assertTrue(target.isSameNode(n.getParentNode())
                    , "[N1]" + " This node should become appended node parent")
                && Contract.assertIdentical(Utils.getLast(children), n
                    , "[N1]" + " The node added should be appended to the list of children");

            Node oldParent = Contract.<Node>oldValue(n.getParentNode());

            if(target.isSameNode(oldParent))
            {
                int ind = Contract.<Integer>oldValue(children.indexOf(n));

                if(ind == 0)
                    return common
                        && Contract.assertEquals(children.subList(0, children.size() - 1), Contract.<Object>oldValue(((ArrayList<Node>)children).subList(0, children.size() - 1))
                            , "All other children should be preserved");
                else if(ind == Contract.<Integer>oldValue(children.size() - 1))
                    return common
                        && Contract.assertEquals(children, Contract.<Object>oldValue(((ArrayList<Node>)children).clone()))
                            , "All other children should be preserved");
            }
        }
    }
}

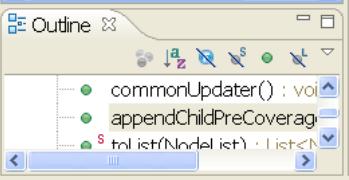
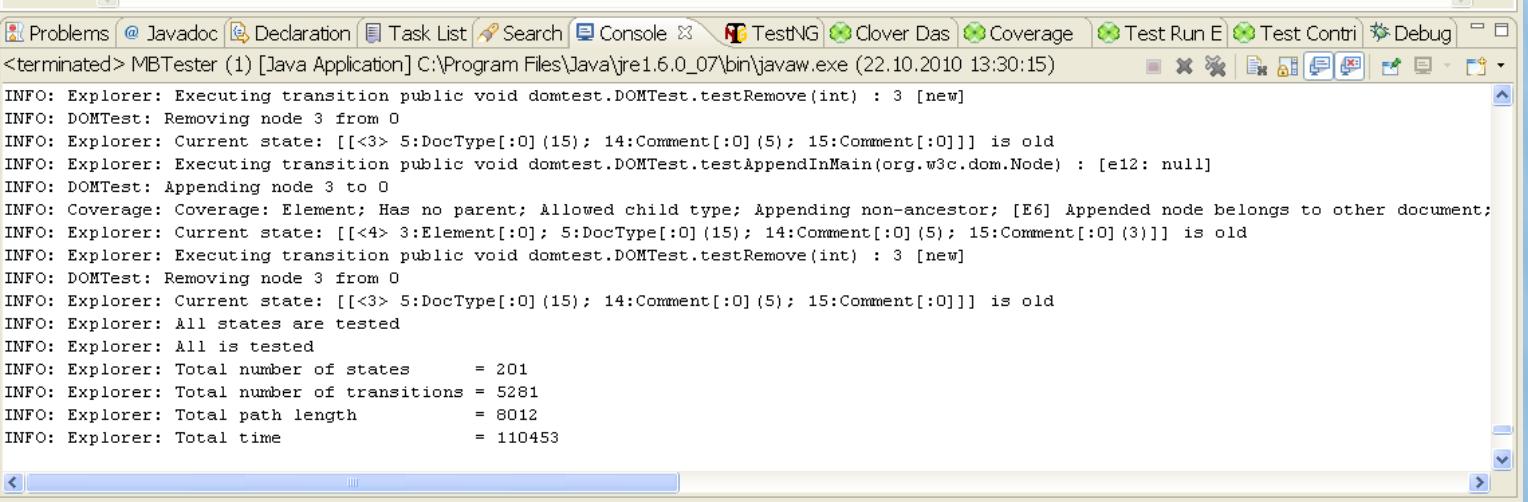
```



```

6392 public void appendChildPreCoverage(Node n)
6392 {
6392     if(n instanceof DocumentFragment)
6392     {
6392         Coverage.addDescriptor("[N3] DocumentFragment");
6392         if(n.getChildNodes().getLength() > 1) Coverage.addDescriptor("More than 1 child");
6392         else if(n.getChildNodes().getLength() == 1) Coverage.addDescriptor("Single child");
6392         else Coverage.addDescriptor("No children");
6392     }
6392     else if(n instanceof Document)
6392     {
6392         Coverage.addDescriptor("Document");
6392         if(containsDifferentChildOfType(DocumentType.class, n))
6392             Coverage.addDescriptor("[E4] Has another DocumentType child");
6392         if(containsDifferentChildOfType(Element.class, n))
6392             Coverage.addDescriptor("[E5] Has another Element child");
6392     }
6392     if(target instanceof Document)
6392     {
6392         Coverage.addDescriptor("Appending to document");
6392         if(target.isSameNode(n)) Coverage.addDescriptor("The same document");
6392         else Coverage.addDescriptor("[E6] Another document");
6392     }
6392     else
6392     {
6392         Coverage.addDescriptor("Appending to non-document");
6392         if(target.getOwnerDocument() == null)
6392             Coverage.addDescriptor("This node belongs to no document");
6392         else if(target.getOwnerDocument().isSameNode(n))
6392             Coverage.addDescriptor("This node belongs to appended document");
6392         else
6392             Coverage.addDescriptor("Appending to different document");
6392     }
6392 }

```

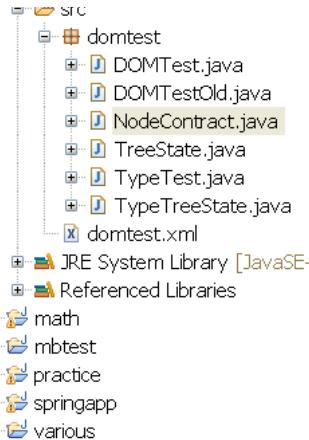




DOMException

[p.31]

HIERARCHY\_REQUEST\_ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors [p.205] or this node itself, or if this node is of type Document [p.41] and the DOM application attempts to append a second DocumentType [p.115] or Element [p.85] node.



```

    class) {
        ption().code =
        tFragment)
    }
    .either(!allChildrenAllowed(target, n)
        , "[E1]" + " this node is of a type that does not allow children of the type of the newChild node"
        .or(getAncestors().contains(n))
        , "[E2]" + " the node to append is one of this node's ancestors"
        .or(n == target
        , "[E3]" + " the node to append is this node itself")
        .or((target instanceof Document) && (n instanceof DocumentType) && containsDifferentChildOfType(DocumentType.class, n)
        , "[E4]" + " this node is of type Document and the DOM application attempts to append a second DocumentType"
        .or((target instanceof Document) && (n instanceof Element) && containsDifferentChildOfType(Element.class, n)
        , "[E5]" + " this node is of type Document and the DOM application attempts to append a second Element")
        .onOpposite()
        .either(!isAllowedChild(target, n)
            , "[E1]" + " this node is of a type that does not allow children of the type of the newChild node"
            .or(getAncestors().contains(n))
            , "[E2]" + " the node to append is one of this node's ancestors")
        .or(n == target
        , "[E3]" + " the node to append is this node itself")
        .or((target instanceof Document) && (n instanceof DocumentType) && containsDifferentChildOfType(DocumentType.class, n)
        , "[E4]" + " this node is of type Document and the DOM application attempts to append a second DocumentType"
        .or((target instanceof Document) && (n instanceof Element) && containsDifferentChildOfType(Element.class, n)
        , "[E5]" + " this node is of type Document and the DOM application attempts to append a second Element")
    )

```

The code block shows the logic for raising the HIERARCHY\_REQUEST\_ERR exception. It handles five cases (E1-E5) where the new child node is not allowed. A red circle highlights the entire conditional block. A blue box labeled 'Document' encloses the first case (E1). Another blue box labeled 'Element' encloses the second case (E2). A blue box labeled 'DocumentType' encloses the third case (E3). Two blue boxes labeled 'Comment' enclose the fourth and fifth cases (E4 and E5).

```

Problems Declaration Task List Search Console TestNG Clover Coverage Test Run E Test Contrib Debug
<terminated> MBTester (1) [Java Application] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (22.10.2010 13:35:48)
INFO: Coverage: Coverage: Element; Has no parent; Allowed child type; Appending non-ancestor; [E6] Appended node belongs to other document;
INFO: DOMTest: Exception caught org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.
INFO: Explorer: Current state: [[<4> 3:Element[:0] (15); 5:DocType[:0] (3); 14:Comment[:0] (5); 15:Comment[:0]]] is old
INFO: Explorer: Executing transition public void domtest.DOMTest.testAppendInMain(org.w3c.dom.Node) : [xs:schema: null] [new]
INFO: DOMTest: Appending node 5 to 0
INFO: Coverage: Coverage: DocumentType; Has parent; A child of this node; Appended node will be moved; Appended node parent is changeable;
ERROR: Contract: HIERARCHY_REQUEST_ERR is raised so
Either [E1] this node is of a type that does not allow children of the type of the newChild node
Or [E2] the node to append is one of this node's ancestors
Or [E3] the node to append is this node itself
Or [E4] this node is of type Document and the DOM application attempts to append a second DocumentType
Or [E5] this node is of type Document and the DOM application attempts to append a second Element
should hold
ERROR: Contract: Exceptional postcondition failed
INFO: DOMTest: Exception caught org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.
INFO: Explorer: Current state: [[<4> 3:Element[:0] (15); 5:DocType[:0] (3); 14:Comment[:0] (5); 15:Comment[:0]]] is old
INFO: Explorer: Executing transition public void domtest.DOMTest.testAppendInMain(org.w3c.dom.Node) : [xs:schema: null] [new]
INFO: DOMTest: Appending node 6 to 0
INFO: Coverage: Coverage: DocumentType; Has no parent; Allowed child type; Appending non-ancestor; [E6] Appended node belongs to other document;
INFO: DOMTest: Exception caught org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.
INFO: Explorer: Current state: [[<4> 3:Element[:0] (15); 5:DocType[:0] (3); 14:Comment[:0] (5); 15:Comment[:0]]] is old
INFO: Explorer: Executing transition public void domtest.DOMTest.testAppendInMain(org.w3c.dom.Node) : [xs:schema: null] [new]

```

The log output from the MBTester application shows several INFO and ERROR messages. Red circles highlight the error messages related to the HIERARCHY\_REQUEST\_ERR exception, specifically the ones involving the DocumentType and Element cases. A blue arrow points from the 'Element' box in the code diagram to the 'Element' case in the log output.

# WEB APPLICATION CASE STUDY

- Google Web UI
- No ready Java API  
so, use  
WebUI Driver
  - Selenium RC
  - <http://seleniumhq.org/>

The screenshot shows a Google search results page with the query "model based testing book". The results are filtered under the "Everything" category. The first result is a link to "Scholarly articles for model based testing book" with three sub-links: "Practical model-based testing: a tools approach - Utting - Cited by 353", "A taxonomy of model-based testing - Utting - Cited by 109", and "Model-Based Software Testing - El-Far - Cited by 120". Below this is a link to "Amazon.com: Model Based Testing (MBT) related books" with a sub-link to "www.amazon.com/Model-Based-Testing...books/..../R1L5EF0VK75... Cached". The next result is "Amazon.com: Practical Model-Based Testing: A Tools Approach ..." with a sub-link to "www.amazon.com... Computer Science > Software Engineering - Cached". The following result is "Model-Based Testing: Black or White?" with a video thumbnail, a link to "video.google.com/videoplay?docid=5521890509476590796", and a description: "Recently, he co-authored the first industry-oriented book specifically on model-based testing, and developed the open-source ModelJUnit ...". The last result shown is "Practical Model-Based Testing - Elsevier" with a link to "www.elsevier.com/wps/productions\_home/709817 - Cached" and a description: "10 Jun 2011 – Practical Model-Based Testing: This book gives a practical introduction to model-based testing, showing how to write models for testing ...". At the bottom of the page, there is a decorative footer with the word "Goooooooooooooogle" followed by a right-pointing arrow and a "Next" link.

# TEST EXCERPT

---

```
@Test public class GoogleTest {
    Selenium sdriver;
    int page = 0;
    @State public int getPage() { return page; }

    @BeforeTest public void init() {
        sdriver = new DefaultSelenium("localhost", 4444, "*firefox", "http://www.google.com");
        sdriver.start();
        sdriver.open("/");
    }

    boolean containsTransitionTo(int n) { return sdriver.isElementPresent("link=" + n); }

    @Test @DataProvider(name = "pageNumber")
    @Guard(names = {"isResultsPage", "containsTransitionTo"})
    public void moveToPage(int n) {
        sdriver.click("link=" + n);
        if(n != 1) sdriver.waitForCondition("selenium.isTextPresent(
            'Page " + n + " of about results') == true", "20000");
        else sdriver.waitForCondition("selenium.isTextPresent(
            'Page " + page + " of about results') != true", "20000");
        page = n;
        commonCheck();
    }
}
```

# CONCLUSION

## □ Idea 1

Flexibility of component architecture facilitates usage of generic tools in various domains

- Seems to be true : API components, protocols, Web UI can be tested by uniform mechanisms

## □ Idea 2

Modular testware helps to achieve comprehensive testing with less effort

- Not enough data, maybe wrong
- Note : tests' maintainability requires additional effort, but modular tests are more maintainable by construction
- So, such an idea seems to be formulated inadequately

# Thank you for attention!