Component-based Framework for Model Based Testing

Victor Kuliamin
Institute for System Programming
Moscow, Russia
Disclaimer and Outline

- Work in progress

- Motivation – integration problems
- Existing approaches
- Proposed solution
- Some examples
Motivation

• A lot of promising model based testing (MBT) techniques emerge every year
  – They need to be evaluated in various contexts
  – They can be combined in different ways
  – To integrate them all in one tool is very expensive
  – Most of them are not useful in each separate project
  – They also need integration in development process

⇒ Need for a component MBT frameworks, which allow rapid integration of emerging techniques in chosen combinations
Additional Observations

• Various other development tools (analyzers, debuggers, monitors, etc.) also evolve quickly
  – It is advantageous to use them along with MBT
  – MBT based on their own languages or non-standard extensions (not supported by standard tools for base language) are hard to maintain

⇒ Library- and standard extensions-based solutions are preferable
Perspective Approaches I

Generic unit testing tools (xUnit, JUnit descendants)

- Lightweight, “stealth” frameworks
- Configurable
- Easy to integrate with other tools and components

• TestNG [2003]

  http://www.testng.org
  – Use of annotations instead of naming conventions
  – Expected exceptions and timeouts before checks
  – Elaborated hierarchy of test components, test groups
  – Setup and tear-down methods for each kind of groups
  – Dependencies of test methods and groups
  – Test data as test method arguments, test object factories
  – External configuration in XML files
Perspective Approaches II

MBT tools of the same style

• ModelJUnit [2004]
  http://czt.sourceforge.net/modeljunit/index.html
  JUnit + EFSM-based MBT engine
  – Test class considered as EFSM
  – Test methods are possible actions
  – State is calculated by special method
  – Guardians (also methods) can be set for actions

• NModel [2007]
  http://nmodel.codeplex.com/
  – EFSM-based engine, attributes instead of special names
  – Test data as test method arguments
  – Composition of models
  – Support for model analysis
Additional Features I

• Other modeling techniques than EFSMs
  – Interface contracts and assertions
    • Behavior driven development [2003]

```java
specify { robot.moveForward() }.Must.Not.Throw();

Specify.That( max(x, y) ).
  And.Must.Either.Be(x).Or.Be(y);
```
Additional Features II

• Other modeling techniques than EFSMs
  – Interface contracts and assertions
    • Behavior driven development [2003]
    • CodeContracts [2009]

• Auxiliary components for unit testing
  – httpUnit, dbUnit, etc.
  – Mocks and spies
  – Test data generators

• Model-based test coverage measurement
Core Architecture Ideas

• Use of annotations and libraries to implement MBT concepts
  – Easy integration with a lot of development tools

• Dependency injection
  – Easy configurability and extensibility
  – Non-invasive integration of various components

• Aspect-based configuration
  – Non-invasive monitoring of system under test and event processing
Dependency Injection

Centralized Integration

Integration based on Dependency Injection
Architecture Scheme

- External components (mocks, etc.)
- Coverage models
- Aspect bindings
- Configuration
- Adapters
- Covarage measurement engine
- Contract checking engine
- EFSM MBT engine
- Aspect weaving engine
- Dependency injection engine
Prototype Implementation

• Base language – Java
  – Reflection
  – Annotations
  – Plentitude of tools
  – A lot of auxiliary libraries for unit testing

• Dependency injection and aspect engine – Spring framework [www.springsource.org]
  – Long usage and evolution history
Example

Bank account

• **Single operation** `transfer(int sum)`
  – `sum > 0`: deposit
  – `sum < 0`: withdraw
• Stores current balance (`int`, overflow prohibited)
• Credit is possible (maxCredit bounds possible withdrawals)
• All transfers are monitored and can be banned by external validity checker
• All transfers and their results are logged for audit

Test system

• EFSM with state (balance, transferAllowed)
• Test methods for transfer, and switching on/off transfer allowance in validity checker stub
• Contract for transfer method
• Multi-aspect coverage model
• Spy on logging

Stub and spy library – Mockito [http://code.google.com/p/mockito/]

Account Example Class Diagram

- Account
- AccountImpl
- SUT
- Permitter
  - Transfer validity checker
- AuditLog
  - Operation logger
- $AuditLog
  - Auto-created by Mockito
- AccountCoverage
  - Coverage model
- AccountContract
  - Contract of main functionality
- AccountTest
  - EFSM test model
- $Permitter
  - Auto-created by Mockito Control stub
- AccountSpy
  - Contract of logging aspect
Stateful contracts

• Both contracts support their own copy of SUT state (values of `balance` and `maxCredit`)
• To synchronize it with the SUT state they need update methods

```java
public void transferUpdate(int sum) {
    if(balance + sum > maxCredit
        && checkedObject.getPermitter()
            .isPermittedTransfer(checkedObject, sum))
        balance += sum;
}
```
public boolean transferPost(int sum) {
    // Validity check result
    boolean permission = checkedObject.getPermitter()
        .isPermittedTransfer(checkedObject, sum);

    if (Contract.oldBooleanValue(balance + sum > maxCredit) && permission) {
        // The transfer is correct and possible
        return Contract.assertEquals(Contract.intResult(), sum,
            "Result should be equal to the argument")
            && Contract.assertEquals(balance, Contract.oldIntValue(balance) + sum,
                "Balance should be increased by the argument")
            && Contract.assertEquals(maxCredit, Contract.oldIntValue(maxCredit),
                "Max credit should not change");
    } else {
        // The transfer is impossible
        return Contract.assertEquals(Contract.intResult(), 0,
            "Result should be 0")
            && Contract.assertEquals(balance, Contract.oldIntValue(balance),
                "Balance should not change")
            && Contract.assertEqualsInt(maxCredit, Contract.oldIntValue(maxCredit),
                "Max credit should not change");
    }
}
Contract of Logging Aspect

```java
public void transferLogSpy(int sum)
{
    // Validity check result
    boolean permission = checkedObject.getPermitter()
        .isPermittedTransfer(checkedObject, sum);

    // Whether the transfer possible at all
    boolean possible = (balance + sum > maxCredit) && permission;

    // Calls to spy are verified in order-independent way
    if(possible)
    {
        Mockito.verify(logSpy).logKind("SUCCESS");
        Mockito.verify(logSpy).logNewBalance(balance);
    }
    else if(!permission)
        Mockito.verify(logSpy).logKind("BANNED");
    else
        Mockito.verify(logSpy).logKind("IMPROPER");

    Mockito.verify(logSpy).logOldBalance(oldBalance);
    Mockito.verify(logSpy).logSum(sum);
}
```
Coverage Model

```java
public void transferCoverage(int sum)
{
    // Validity check result
    boolean permission = checkedObject.getPermitter()
        .isPermittedTransfer(checkedObject, sum);

    if(balance + sum > maxCredit) Coverage.addDescriptor("Possible transfer");
    else
        Coverage.addDescriptor("Too big sum");

    if(permission) Coverage.addDescriptor("Permitted");
    else Coverage.addDescriptor("Not permitted");

    if(balance == 0) Coverage.addDescriptor("Zero balance");
    else if(balance > 0) Coverage.addDescriptor("Positive balance");
    else Coverage.addDescriptor("Negative balance");

    if(sum == 0) Coverage.addDescriptor("Zero sum");
    else if(sum > 0) Coverage.addDescriptor("Positive sum");
    else Coverage.addDescriptor("Negative sum");
}
```
@Test public class AccountTest {
    Account account;
    @Mock Permitter permitterStub;

    boolean permission = true;

    // Init stubs and configure permitterStub to return true on call to isPermittedTransfer()
    public AccountTest() {
        MockitoAnnotations.initMocks(this);
        Mockito.when(permitterStub.isPermittedTransfer(Mockito.<Account>any(), Mockito.anyInt())).thenReturn(permission);
    }

    public void setAccount(Account account) {
        this.account = account;
        account.setPermitter(permitterStub);
    }

    // Current permission and balance are two components of the test state
    @State
    public boolean getPermission() { return permission; }

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

    ...
}
@Test public class AccountTest
{
    ...
    @Test(dependsOnMethods="testWithdraw")
    @DataProvider(name = "sumArray")
    @Guard(name = "bound")
    public void testDeposit(int x)  { account.transfer(x);  }

    @Test(dependsOnMethods="switchPermission")
    @DataProvider(name = "sumArray")
    public void testWithdraw(int x)  { account.transfer(-x);  }

    // Switch permission and configure permitterStub to its value true on call to isPermittedTransfer()
    @Test
    public void switchPermission()
    {
        permission = !permission;
        Mockito.when(permitterStub.isPermittedTransfer(Mockito.<Account>any(), Mockito.anyInt()))
            .thenReturn(permission);
    }

    // Guardian for deposits to bound the possible balance values
    public boolean bound() { return getBalance() < 5 || !permission;  }

    // Source of test data for both transfer test methods
    public int[] sumArray = new int[]{0, 1, 2, 3, 4};
}
More Realistic Example

• DOM API in Java
  – SUT – Xerces for Java [xerces.apache.org]

• Node children manipulation
  – appendChild(Node n)
  – removeChild(Node n)
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]

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.265] 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.

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

NO_MODIFICATION_ALLOWED_ERR: Raised if the node is readonly or if the previous parent of the node being inserted is readonly.

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.
public boolean appendChildNormalPost(Node n) {
    boolean common =
        ContracthandleChange1(parent, Contract.&Node&oldValue(parent)
            , "Parent node should be preserved")
        && Contract.handleChange1(owner, Contract.&Node&oldValue(owner)
            , "Owner document should be preserved")
        && Contract.handleChange1Contract.&Node&oldValue(result), n
            , "[N1]" + " The node added should be returned");

    if (n instanceof DocumentFragment) {
        List.&Node& appendedNodes = Contract.&List.&Node&.oldValue(new
            .List.&Node&.getChildrenNodes());

        for (int i = 0; i < appendedNodes.size(); i++)
            common &= Contract.&Node&.handleChange1(appendedNodes.get(i).getParentNode()
                , "[N1]" + " This node should become parent for all appended nodes");

        return common &&
            Contract.handleChange1(n.getChildrenNodes().getLength(), 0
            , "The new contents of the document fragment should be empty")
            && Contract.handleChange1(new.&List.&Node&.getPrefix(children, appendedNodes.size()), appendedNodes
            , "[N1]" + " The entire contents of the document fragment should be moved in the end of the child list of this node");
    } else
        common =
            Contract.handleChange1(target, Contract.&Node&.handleChange1(n.getParentNode()
                , "[N1]" + " This node should become appended node parent")
            && Contract.handleChange1(new.&List.&Node&.getFirst(children), n
                , "[N1]" + " The node added should be appended to the list of children");

    Node oldParent = Contract.&Node&.handleChange1(n.getParentNode());

    if (target.isSameNode(oldParent)) {
        int ind = Contract.&Integer&.handleChange1(children.size(), n);

        if (ind == 0)
            return common &&
                Contract.handleChange1(children.subList(0, children.size() - 1), Contract.&Object&.handleChange1((&List.&Node&)[], "All other children should be preserved");
        else if (ind == Contract.&Integer&.handleChange1(children.size() - 1))
            return common &&
                Contract.handleChange1(children, Contract.&Object&.handleChange1((&Array.&List.&Node&)[], children.clone()), "All other children should be preserved");
    }
@Test
@DataProvider(name = "nodes")
public void testAppendInMain(Node n)
    {
        logger.info("Appending node " + index(n) + " to 0");
        try
        {
            mainDoc.appendChild(n);
        }
        catch(NoSuchElementException e)
        {
            logger.info("Exception caught " + e.getClass().getName() + ": " + e.getMessage());
        }
    }

@Test
@DataProvider(name = "indexIterator")
public void testRemove(int i)
    {
        Node n = mainDoc.getFirstChild().item(i);
        logger.info("Removing node " + i + " from 0");
        try
        {
            mainDoc.removeChild(n);
        }
        catch(NoSuchElementException e)
        {
            logger.info("Exception caught " + e.getClass().getName() + ": " + e.getMessage());
            e.printStackTrace(System.out);
        }

        public int index(Node n)
        {
            int n = numberOfMainDocChildren();
            int[] indices = new int[n];
            for(int i = 0; i < n; i++) indices[i] = i;
            return Util.ArrayToIndex(indices).iterator();
        }
    }
public void appendChildPreCoverage(Node n) {
    if (n instanceof DocumentFragment) {
        Coverage.addDescriptor("[N1] DocumentFragment");
        if (n.getChildNodes().getLength() > 1) Coverage.addDescriptor("More than 1 child");
        else if (n.getChildNodes().getLength() == 1) Coverage.addDescriptor("Single child");
        else Coverage.addDescriptor("No children");
    } else if (n instanceof Document) {
        if (containsDifferentChildOfThisType(DocumentType.class, n))
            Coverage.addDescriptor("[E4] Has another DocumentType child");
        if (containsDifferentChildOfThisType(Element.class, n))
            Coverage.addDescriptor("[E5] Has another Element child");
        if (target instanceof Document) {
            Coverage.addDescriptor("Appending to document");
            if (target.isSameNode(n)) Coverage.addDescriptor("The same document");
            else Coverage.addDescriptor("[E6] Another document");
        } else {
            Coverage.addDescriptor("Appending to non-document");
            if (target.getDocument() == null) Coverage.addDescriptor("This node belongs to no document");
            else Coverage.addDescriptor("This node belongs to appended document");
        }  
    } else {
        Coverage.addDescriptor("[D0] Not a Document");
    }
}
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 or this node itself, or if this node is of type Document or Element and the DOM application attempts to append a second DocumentType or Element node.
Conclusion

• There is a lot of features to implement yet
• But many are implemented with low effort
• (Almost) All external tools and libraries are open source (exception – clover is taken for its good integration in Eclipse)

• Dependency injection and aspects allow non-invasive composition of models
Thank you!
Questions?

Victor Kuliamin
Institute for System Programming,
Software Engineering Department

kuliamin@ispras.ru
www.ispras.ru/~kuliamin

www.unitesk.com
petrenko@ispras.ru