Tutorial
Integration Testing of Composite Applications using TTCN-3

MCeTech'08 Montréal

Bernard Stepien, Liam Peyton, Pierre Seguin

Université d'Ottawa | University of Ottawa

uOttawa

L'Université canadienne
Canada's university

www.uOttawa.ca

Motivation

- **Large Systems are complex**
  - Enterprises deploy Composite Applications that leverage a shared infrastructure (Services, Components)
  - Services and Components are linked and distributed
  - Composite Applications, Services, Components are deployed and upgraded independently of each other.

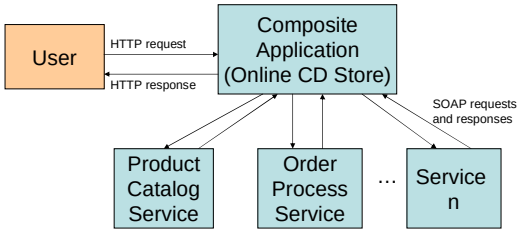
2

Integration Testing is complex

- **An observed fault at the level of user interaction could be:**
  - a fault or quality of service issue (performance, security, scalability, etc.) in the application or process logic
  - a fault or quality of service issue in any of the components used by the application
  - an unintended interaction in combining components

3

Distributed nature of composite applications



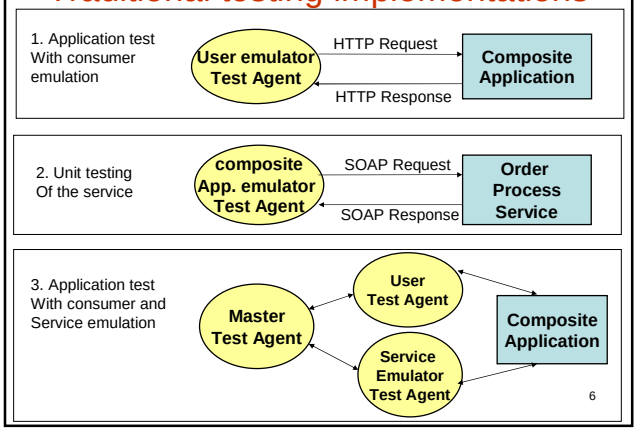
4

Traditional testing approach

- Unit testing of application and components:
  - Test the composite application by emulating user behavior.
  - Test the underlying services by emulating the composite application.
  - Test the composite application by emulating the user and the underlying services.

5

Traditional testing implementations



6

What needs to be tested?

1.Message correctness

- User receives correct responses to specific requests.
- Composite application sends correct requests to services.
- Service produces correct responses to specific requests from composite application.
- Composite application produces correct responses to the user relative to specific service responses.

7

What needs to be tested?

2.Quality of service

- Performance
- Scalability
- Security
- Under multi-user conditions
- Under multi-applications using the same services conditions

8

Deficiencies of unit testing

- individual unit tests can only verify the unit in isolation **(maintenance issue)**
  - independent of other applications, services and components that are being upgraded and introduced independently
- when a "unit tested" unit is deployed into a SOA unintended interactions can result in faults. **(interaction issue)**
  - Multi-user behavior:
    - Competing for resources.
    - Application logic mix-ups between different user sessions.
  - Caching of messages.
- Unit testing is unable to isolate or diagnose the cause of an observed fault to specific components within an SOA **(diagnosis issue)**

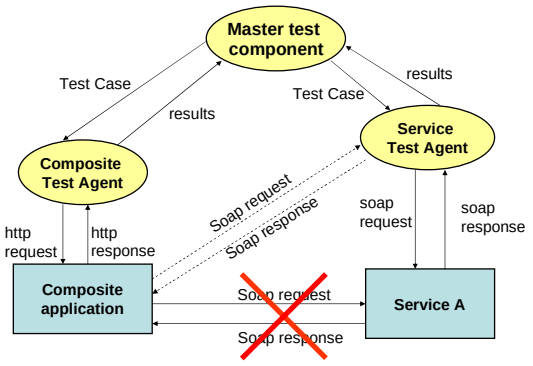
9

Integration testing approach

- Test all messages flowing between all components of a composite application
- Test both sides of all interactions (expected requests, expected responses)

10

Proposed integration testing architecture (grey-box testing)



11

Integration testing strategy

- Integration testing can be composed by re-using part or all of unit testing code.
- The additional requirements consists in:
  - For a given user action, being able to correlate events at different interfaces of the system.
  - Coordinate the testing of the various components.

12

What does testing consist of?

1. Test specification
   - Specify test data
   - Specify test behavior as sequences of events
   - Specify test outcome (pass/fail)
2. Perform the test
   - Manage communication with SUT
   - Invoke test cases
   - Code or decode messages
3. Analyzing test results
   - Details to understand results (expected vs actual values)
   - Tracing of test events
   - Produce reports

13

How can we implement a test

- By writing an anti-product using a conventional programming language (Java, C, C++, visual-basic, …).
- By using off-the-shelf testing products.
- By using open source Frameworks .
- By using languages specialized for testing purposes.

14

Purpose of testing tools and frameworks

- Help designing tests.
- Reduce the coding effort for test execution.
- Reduce the coding effort for test results presentation and analysis.
- Help understand the test system.
- Help understand the results of a test.
- Help debugging.

15

Categories of testing tools

- Generic tools and frameworks
- Targeted tools and frameworks (for specific applications)
  - Web testing
  - Specific telecom protocols (SIP, SS7, 3GPP)
- Frameworks that address only part of the testing problem.
  - httpUnit: handles only the communication management and codec of web applications.

16

Advantages/disadvantages

- Generic languages are labor intensive.
- Off-the-shelf tools are limited.
- Off-the-shelf tools depend on the existence of the vendor.
- Open source frameworks are not necessarily reliable. (no one feels responsible)
- Standard high-level languages save considerable work effort and are supported by a variety of vendors. If one vendor fails, your test suite will still work on another's vendors tool.

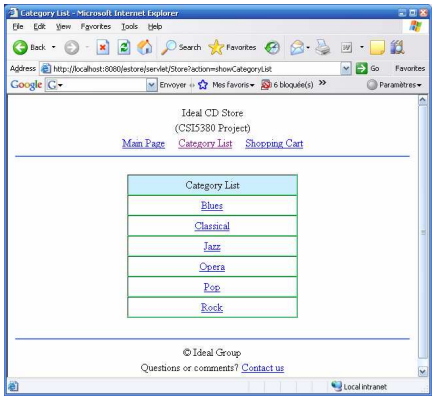
17

Tool evaluation criteria summary

Kind of tool	Coding effort	Risk	Maintenance	Code ownership
Generic Programming language	high	low	high	yes
Off-the-shelf tool	low	high	low	no
High-level specialized programming language	medium	low	low	yes

18

Example web page



19

JUnit source of inspiration

single values testing

- JUnit in conjunction with other frameworks such as HtmlUnit is presented in litterature as simple, easy to use and understand

example taken from the HtmlUnit Documentation:

```
public void testHtmlUnitHomePage() throws Exception {
    final WebClient webClient = new WebClient();
    final URL url = new URL("http://htmlunit.sourceforge.net");
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );
}
```

20

Testing a web page JUnit

multiple values testing

```
public void testCategories() {
  List urlList = new LinkedList();
  String[] theLinkNames = {"Main Page", "Category List", "Shopping Cart",
        "Blues", "Classical", "Jazz", "Opera", "Pop", "Rock", "Contact us"};

  final WebClient webClient = new WebClient();
  assertNotNull(webClient);
  try {
    final URL url = new URL("file:categories_list.html");
    final HtmlPage theCurrentPage = (HtmlPage)webClient.getPage(url);

    assertNotNull(theCurrentPage);
    assertTrue(theCurrentPage.getWebResponse().getStatusCode() == 200);
    assertTrue(theCurrentPage.getTitleText().equals("Category List"));

    int textPosition = theCurrentPage.asText().indexOf("Ideal CD Store");
    assertTrue(textPosition >= 0);

    List theLinks = theCurrentPage.getAnchors();

    int n = theLinks.size();

    assertEquals(n, 10);

    for(int i=0; i<n; i++) {
      HtmlAnchor theAnchor = (HtmlAnchor) theLinks.get(i);
      assertEquals(theAnchor.asText(), theLinkNames[i]);
      urlList.add(theAnchor.getHrefAttribute());
    }
  }  catch( Exception e) {...}
}
```


Testing a web page in TCL

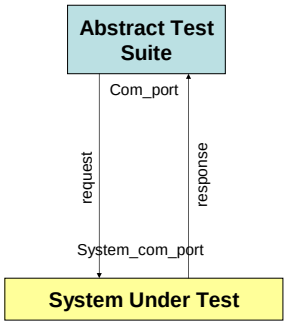
using regular expression feature

```
package require http 1.0

proc testCategoriesPage {} {
        puts "testing categories page test"

        set categoriesPage [http_get http://localhost:8080/estore/servlet/Store?action=showCategoryList -query]
        set categoriesPageData    [ http_data $categoriesPage ]
        set pageStatus [ http_status $categoriesPage ]

        puts $categoriesPageData

        puts "-----------------------------------------------------"

        if { $pageStatus != "ok" } {
                puts "page status not ok - set verdict to fail"
                return
        }

        set textFound [ regexp "<html>.* <title>.*Category List.*</title>.*Ideal CD Store.*CSI5380 Project.*<.*\
href=.*>.*Main Page.*<.*href=.*>.*Category List.*<.*href=.*>.*Shopping Cart.*<.*href=.*>.*Blues.*<.*href=.*>.*Classical.*\
<.*href=.*>.*Jazz.*<.*href=.*>.*Opera.*<.*href=.*>.*Pop.*<.*href=.*>.*Rock.*" $categoriesPageData ]

        if { $textFound == 1 } {
                puts "categories page has matched the expectation - verdict pass"
        } else {
                puts "categories page has NOT matched the expectation - verdict fail"
        }
}

testCategoriesPage
```


Test Implementation using TTCN-3

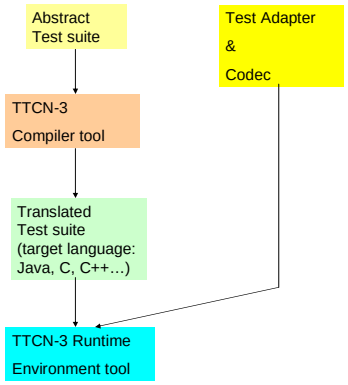
- TTCN-3 is well adapted to the nature of our integration testing problem:
  - Communication **ports**.
  - The **template** language construct maps to the fine grained structuring requirements of integration testing.
  - The **parallel test component** language construct (PTC) maps to the concept of testing agent.
  - The **complex data type matching mechanism** is very powerful and fully abstracts message validation.
  - The **set-based matching mechanism** is very powerful and particularly useful for addressing multiple user message flows
  - The **parametrization** of test cases, templates and test components improves clarity and flexibility.
  - **Strong typing** enables the detection of many errors at design stage instead of at run time.
  - The **separation of concerns** between the abstract and the concrete layers enables to focus on the abstract view of the testing problem.

Basic TTCN-3 events



- Send a request to the SUT
- Receive a response from the SUT and match it to a predefined template

TTCN-3 test environment



25

TTCN-3 separation of concerns

- Between abstract test suite and adaptation layer where communication and coding/decoding takes place.
- Between behavior and conditions governing behavior (behavior tree and templates).
- Between test behavior and test coordination (parallel test components that represent test agents)

26

Modeling a web page with data types

```
type record WebResponseType {
   integer statusCode,
   charstring title,
   charstring content,
   linkListType  links optional,
   formSetType forms optional,
   TableSetType tables optional
}

type record linkType {
   charstring text,
   charstring link
}

type set of linkType linkListType;

type set of charstring RowCellSetType;

type record tableRowType {
   RowCellSetType cells
}

type set of tableRowType tableRowSetType;

type record TableType {
   tableRowSetType rows
}

type set of TableType TableSetType;
```

27

TTCN-3 template concept is a test oracle

- Based on data types (has field names).
- Looks like an assignment of values but also provides the capability of specifying matching rules.
- Allows re-usability among templates (building blocks).
- Doesn't require complex if-then-else constructs. The TTCN-3 receive() construct and the underlying matching mechanism handles the verification of the oracle without any programming efforts.
- Is a kind of function, thus parametric.
- Has a useful modifies features that enables to build a new template derived from an existing one.

28

TTCN-3 template example

```
template WebResponseType categoriesPageResponse := {
  statusCode := 200,
  title := " Category List ",
  content := pattern "*Ideal CD Store*(CSI5380 Project)*",
  links := categoriesPageLinks("Main Page"),
  forms := omit,
  tables:= omit
}
```

```
template linkSet categoriesPageLinks(charstring myText) := {
  { text := myText, URL := (url_1, url_2) },
  {"Category List", ?},
  {"Shopping Cart", ?}, {"Contact us", ?}
  {"Blues", ?}, {"Jazz", ?}, {"Classical", ?},
  {"Opera", ?}, {"Pop", ?}, {"Rock", ?}
}
```

```
template charstring url_1 := "http://www.mycompany.com/mylink.html"
```


TTCN-3 matching mechanism

- Specify that an incoming message must match a template.
- No detailed coding of the matching of complex messages is required. That was the role of the template.
- Matching is specified on a named communication port.

```
web_port.receive( categoriesPageResponse ) { … }
```

In TTCN-3, the receive statement means both receive data from the communication media and match it against the template

TTCN-3 behavior tree concept

- A behavior tree is composed of nested alternate responses to given requests.
- Requests and responses are abstracted using TTCN-3 templates.
- Alternatives can be abstracted into functions called altsteps.

```
alt {
    [] webApplPort.receive(checkStockRequest) {
            servicePort.send(checkStockRequest); …
            alt {
                [] servicePort.receive(productDetailsConfirmation) { }
                [] servicePort.receive(outofStockNotification) { }
                [] servicePort.receive { setverdict(fail) } // unexpected request
                [] serviceTimer.timeout { setverdict(fail) }
            }
    }
    [] webApplPort.receive { setverdict(fail) } // unexpected request
}
```


TTCN-3 Test agents configuration

- A Master test component orchestrates all test agents behavior.
- A test agent is mapped to a TTCN-3 parallel test component.
- A test component is started with a specific test case as parameter.
- Behavior of various test agents can be coordinated.

```
testcase CompositeWebApplicationTesting() runs on MTCType … {
        var ServiceComponentType theServiceComponent;
        var UserComponentType theUserComponent[2];

        theUserComponent[0] := UserComponentType.create;
        theUserComponent[1] := UserComponentType.create;
        theServiceComponent := ServiceComponentType.create;

        // map all ports here …

        theServiceComponent.start(
                serviceEventsTest(expectedMsgTemplate));

        theUserComponent[0].start(User_1_events());
        theUserComponent[1].start(User_2_events());

        theUserComponent[0].done;
        theUserComponent[1].done;

        servCoordPort.send("end test");

        all component.done;

        log("testcase SOABasedWebTesting completed");
}
```

TTCN-3 Verdicts

- Kinds of verdicts:
  - Pass
  - Fail
  - Inconclusive
- TTCN-3 records both passed and failed tests
- JUnit shows only failed tests.
- TTCN-3 is better for tracing because it is based on event tracing.

33

The TTCN-3 adaptation layer handling communication with SUT

```
public class WebTesting_TestAdapter extends TestAdapter
                                    implements TriCommunicationSA, TriPlatformPA, TciEncoding {
  …
  public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId, TriAddress address,
                                                                    TriMessage sendMessage) {

    Byte [] mesg = sendMessage.getEncodedMessage();
    String String theUrlStr = new String(mesg);

     if(tsiPortId.getPortName().equals("systemUserWebPort")) {

      final WebClient webClient = new WebClient();

      try {
            final URL url = new URL(theUrlStr);

            theCurrentPage = (HtmlPage) webClient.getPage(url);

            TriMessageImpl rcvMessage = new TriMessageImpl(theCurrentPage.asText().getBytes());

            myCte.triEnqueueMsg(tsiPortId, new TriAddressImpl( new byte[] {}),  componentId, rcvMessage);
       } catch ( … ) { … }
      …
  }
…
}
```

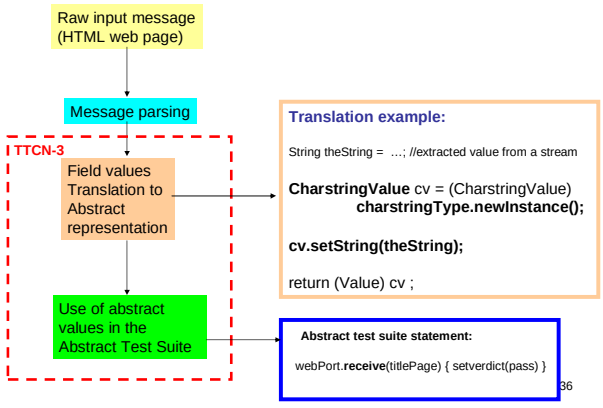
34

Principles of the codec

- Extract a value from the input stream.
- Build an internal representation of this value using the tools API.
- Return it to the abstract layer.
- A TTCN-3 codec is type driven.

35

Translation API



36

Complex types Codec example

Abstract type definition:

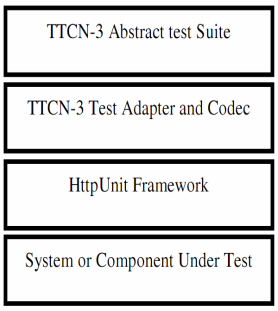
```
type record weatherResponse {
   charstring location,
   charstring date,
   charstring kind,
   integer temperature,
   integer windVelocity,
   charstring conditions
}
```

```
RecordValue theWeatherResponseValue = (RecordValue) type.newInstance();

For each field:

IntegerValue theTemperatureValue = (IntegerValue) integerType.newInstance();

theTemperatureValue.setInt(-25);

theWeatherResponseValue.setField("temperature", theTemperatureValue);
```

37

Test Adapter use of external Frameworks

- Test campaign is specified at the abstract layer level
- Codecs are used to translate between concrete data structures and abstract ones
- Adapters are used to communicate with the SUT or CUT
- Codecs and adapters use HttpUnit for communication with the SUT or CUT



38

TTCN-3 test adapter and codecs coding effort

- Writing a test adapter for TTCN-3 is a fixed effort that is not repeated for subsequent testing using the same data types.
- Nokia has reported at T3UC'06 that the adapter represented only **25%** of the coding effort in a large test application, while the abstract layer represents 75%
- Adapters can be efficiently structured and their components re-usable among different testing projects.
- Thus, test adapter writing efforts largely depend on classic software development structuring techniques and management.

39

TTCN-3 tools

- About 7 vendors.
- Some academic Open Source versions.
- Compilers and runtime environments.
- Runtime GUIs, APIs.
- Features
- Off-the-shelf codecs.
- Abstract types libraries (XML, IDL, WSDL)

40

Reducing coding efforts

- How to measure coding effort?
  - Number of lines of code.
  - Error detection (design time or runtime?)

41

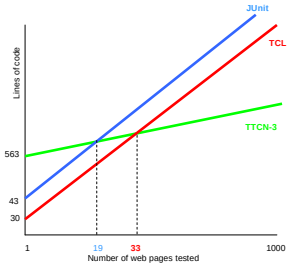
Web page testing example
tools comparison statistics

- JUnit: **43** lines
- TCL/TK: **30** lines
- TTCN-3:
  - **Abstract test suite**: **63** lines
  - Adaptation layer: 200 lines
  - Codec: 300 lines
  - Total lines: **563** lines

42

TTCN-3 coding effort comparison

Fixed coding effort:	
type definitions:	26 lines
Behavior definitions:	20 lines
module/control	4 lines
Test adapter:	200 lines
Codec:	300 lines
Total fixed part:	**550 lines**
Variable coding effort:	
Templates definitions:	12 lines
Control part:	1 line
Total variable part:	**13 lines**
Total for one page:	**563 lines**



43

Important remark about fixed and variable parts of code

- All three approaches can be decomposed into fixed and variable code parts in a similar way with similar coding effort savings.
- However, the main difference between TTCN-3 and JUnit or TCL is that with TTCN-3 there is a **model** that **forces** the tester to decompose the problem that way.

44

Separation of concern

A post mortem example

- A company spent two person/years to develop a test suite for a web application using JUnit and httpUnit.
- The test suite was hard to maintain due to the intensive use of httpUnit methods buried deep in the code.
- A number of items could not be tested because httpUnit did not provide appropriate features for that purpose.
- Converting to more appropriate htmlUnit would have required massive changes (80% of the code consisted in invocations to httpUnit methods).
- The test suite was merely scrapped and thus never used.

45

Separation of concerns example

JUnit

```
final URL url = new URL("http://htmlunit.sourceforge.net");
final HtmlPage page = (HtmlPage) webClient.getPage(url);
assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );

final URL url = new URL("http://anotherpage.com");
final HtmlPage page = (HtmlPage) webClient.getPage(url);
assertEquals( "htmlunit - Welcome to another page", page.getTitleText() );
```

TTCN-3

```
Web_port.send("http://htmlunit.sourceforge.net");
Web_port.receive("htmlunit - Welcome to HtmlUnit") { setverdict(pass) }

Web_port.send("http://htmlunit.sourceforge.net");
Web_port.receive("htmlunit - Welcome to HtmlUnit") { setverdict(pass) }
```

46

Differences JUnit/TTCN-3

- In the JUnit version, there are 6 lines of code.
- In the JUnit version every line is invoking a method of the HttpUnit framework.
- In the TTCN-3 version, the abstract layer has only 4 lines of code.
- In the TTCN-3 version, there is no reference to the HttpUnit framework at all.

47

consequences of the TTCN-3 separation of concerns

- If you were to re-write the preceding code using a different framework, like htmlUnit:
  - With JUnit you would have to rewrite all of the 6 lines of codes.
  - With TTCN-3 you would have to re-write only the codec that is common to both URL invocations.
- With TTCN-3 you could save 33% of lines of code.
- The TTCN-3 abstract code can be fully re-usable regardless of the framework used.

48

Advantages of TTCN-3

- TTCN-3 is a standard, thus a test suite can be circulated among users practically without documentation.
- TTCN-3's separation of concern improves clarity and imposes an efficient programming style.

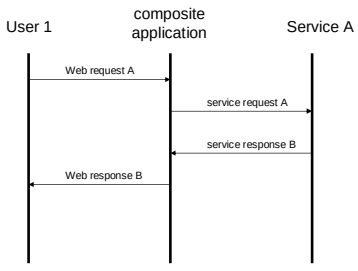
49

Problems with SOA testing

- Correlation gap
  - With multiple users
  - With multiple concurrent composite applications accessing the same services
- Cached messages
- performance

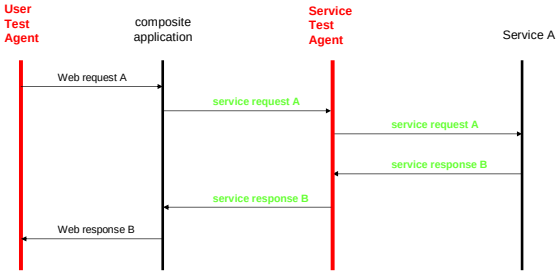
50

Composite system use case message flow example



51

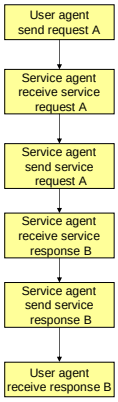
Test agent architecture
corresponding message flow



52

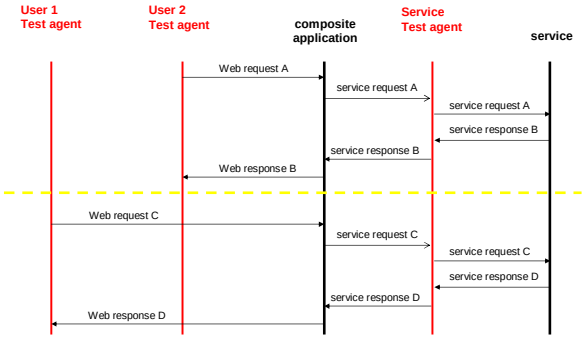
Test scripting

- For a single user, end-to-end testing of a use case may look like a simple linear sequence of events to be verified:
  - The user test agent sends a request
  - The service test agent receives a service request and forwards it to the service
  - The service test agent receives a service response and forwards it to the composite application.
  - The user test agent receives the response



Multiple user message flow

ideal case

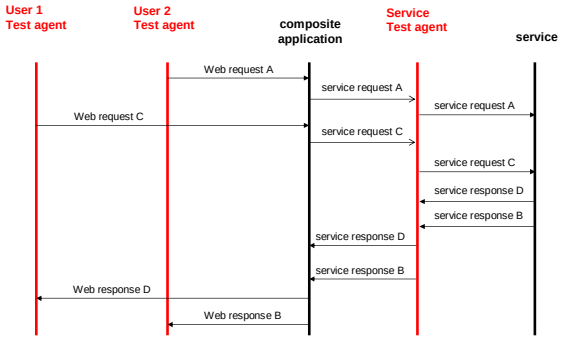


Correlation gap
Multi-user problem

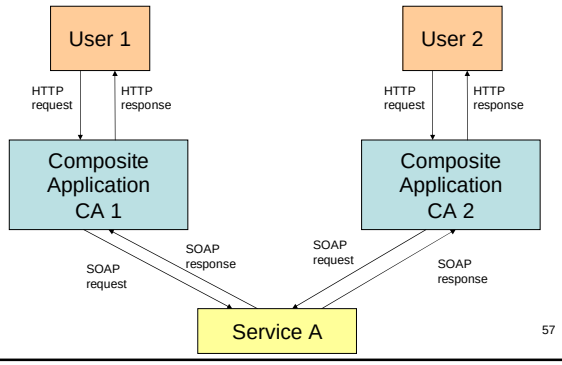
- The well separated end-to-end message flows for each user are only an ideal case.
- Both composite application and service applications may disturb this idealistic view of the problem.
- Messages may be interleaved. Therefore, the order of arrival and departure of messages at underlying services can no longer be correlated with the order of initial requests.
- Caching may remove some messages. (not addressed in these slides, see paper)

Multiple user message flow

one of many realistic cases



Multiple composite applications accessing the same services



Service test agents

- Is where the correlation gap must be handled.
- The solution is to specify the service agent as a kind of case statement that can handle any request at any time.
- Once a request has arrived, the sequence of events resulting from it are processed sequentially.
- This explains it's recursive nature. (handle one request at a time and move on to the next)
- Two levels of processing:
  - Validate an incoming request
  - Validate the corresponding response (see slide 22)

Service agent example

- One altstep function per message received from composite application that encapsulates a behavior tree.
- A catch all receive for undesired messages.
- One coordination function to end the test agent process.

```
function serviceEventsTest(
          RequestType expectedRequests,
          ResponseType expectedResponses) … {
  …
  alt {
    [] A1_behavior() { … }
    …
    [] B2_behavior() { … }
    [] soaWebPort.receive {setverdict(fail)}
    [] endTestBehavior(expectedRequests)
  }
}
```


Individual service request-response behaviors

```
altstep A1_behavior() runs on SOAComponentType {
  …
    [] webApplPort.receive(checkStockRequest)  -> incomingRequest {
          checkIfCached(incomingRequest);
          updateReceivedRequests(incomingRequest );

          servicePort.send(checkStockRequest); …
          alt {
                [] servicePort.receive(productDetailsConfirmation) -> incomingResponse {
                      updateReceivedResponses(incomingResponse)
                 }
                [] servicePort.receive(outofStockNotification) -> incomingResponse {
                      updateReceivedResponses(incomingResponse)
                  }
                [] servicePort.receive { setverdict(fail) } // unexpected request
                [] serviceTimer.timeout { setverdict(fail) }
          };
            repeat
    }
}
```

Verifying test completeness

- So far we have checked that when a message has been received from a composite application, it was indeed expected.
- Now, we need to verify that all expected messages have been received. Only then can we set the test verdict to pass.

```
altstep endTestBehavior(RequestType expectedRequests,
                        ResponseType expectedResponses) … {
  [] serviceCoordPort.receive("end of test") {
    if(match(expectedRequests, receivedRequests)
       &&  match(expectedResponses, receivedResponses)) {
      setverdict(pass);
    }
    else {
      setverdict(fail);
    };
    evaluateQOS()
  }
}
```

61

Completeness checking and alternative behaviors

- When users compete for resources, some will be able to fulfill their requests, other will not.
- Thus, sets of expected requests and expected responses are necessary to determine correctness in the case of alternative responses (check stock example).

62

Tool results inspection features comparison

63

JUnit tool features

Failure traces

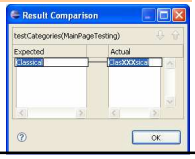
```
junit.framework.ComparisonFailure: expected:<...> but was:<...XXX>
        at junit.framework.Assert.assertEquals(Assert.java:81)
        at junit.framework.Assert.assertEquals(Assert.java:87)
        at MainPageTesting.matchWebPage(MainPageTesting.java:127)
        at MainPageTesting.testCategoriesPage(MainPageTesting.java:101)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:585)
        at junit.framework.TestCase.runTest(TestCase.java:154)
        at junit.framework.TestCase.runBare(TestCase.java:127)
        at junit.framework.TestResult$1.protect(TestResult.java:106)
        at junit.framework.TestResult.runProtected(TestResult.java:124)
        at junit.framework.TestResult.run(TestResult.java:109)
        at junit.framework.TestCase.run(TestCase.java:118)
        at junit.framework.TestSuite.runTest(TestSuite.java:208)
        at junit.framework.TestSuite.run(TestSuite.java:203)
        at org.eclipse.jdt.internal.junit.runner.junit3.JUnit3TestReference.run(JUnit3TestReference.java:128)
        at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:460)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:673)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:386)
        at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:196)
```

64

JUnit mismatch display feature

- JUnit shows only what did not match
- It shows only the first mismatch
- Usable only for assertEquals() assertions
- Does not help for assertTrue() assertions

assertEquals(X, “Classical”);




```
public void testAssertTrue(){

    int X = 10;
    assertTrue(X == 5);
}
```

In the above, JUnit does not display The value of the variable X

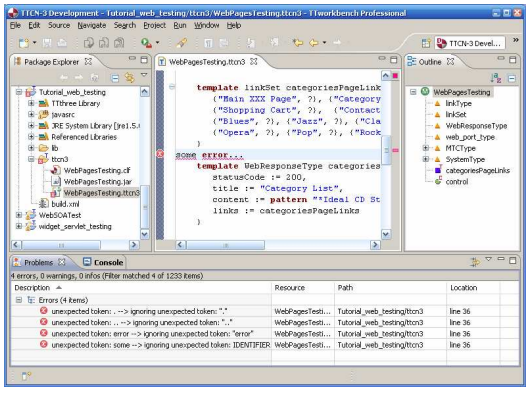
TCL/TK results analysis features

- Basically there are none
- However, because of TK, it is easy to create a custom GUI to display results and improve results analysis
- With TK, GUIs for displaying results can be considered as very flexible. Other tools have only fixed features that a user can not modify.

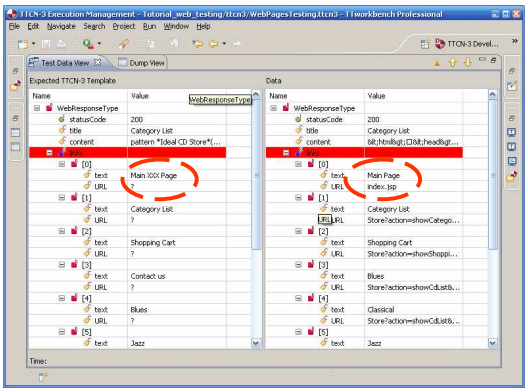
TTCN-3 tools features

- **Matching mechanism overview**: in case of mismatch, the values of **all** the fields that caused the mismatch can be viewed along with the correct values for other fields.
- **Logging**: each event gets logged and thus the sequence of events can be thoroughly inspected. Thus tracing without the need of a classical debugger.
- **Event traceability**: Logs are not limited to display failures, they show successful events too. This improves traceability.

A TTCN-3 Tool editor/compiler

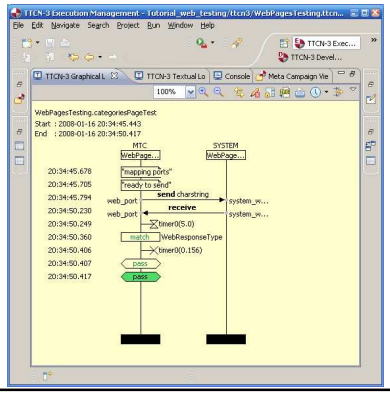


Matching mechanism overview



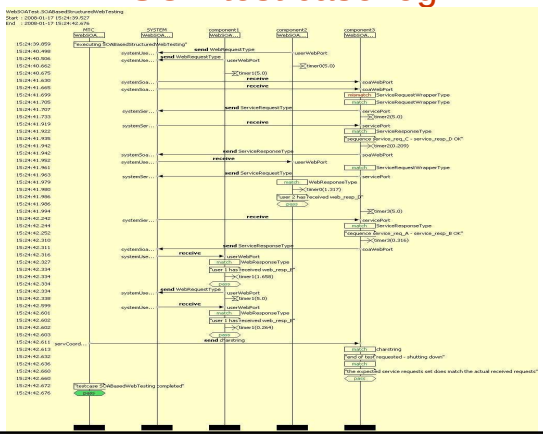
69

Logging (graphical/textual)



70

SOA test case log



71

JUnit stack dump

- Shows only the points of method invocation.
- Doesn't show the sequence of events that led to a point of failure or success.
- JUnit is good for software testing where nested method calls are the basic events.
- JUnit is not good for discrete events sequences.

72

Web testing vendor features
from Testing Tech

- Instant access to WSDL/SOAP based web services
- Automatic import of WSDL specifications into TTCN-3 that are translated into TTCN-3 data types.
- Zero-coding efforts (codec/adapter)
- Seamless usability within any TTCN-3 test application
- Multiple test components and multiple port mapping
- W3C Web Service Description Language (WSDL) v1.1
- W3C SOAP v1.1 and v1.2 Candidate Recommendation

73

Zero-coding-effort?

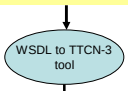
- Automated transformation of WSDL to TTCN-3 types
- Automated transformation of WSDL to TTCN-3 test components and ports
- Automated generation of the CODEC.

Your coding effort:

- Templates containing test data.
- Test behavior containing sequences of events and alternate events trees

WSDL
complex type example

```
<xsd:complexType name="PlaceFinderOptions">
  <xsd:sequence>
    <xsd:element name="dataSource" nillable="true" type="xsd:string" />
    <xsd:element name="filterCountry" nillable="true" type="xsd:string" />
    <xsd:element name="filterExtent" nillable="true" type="ns3:Envelope" />
    <xsd:element name="filterType" nillable="true" type="xsd:string" />
    <xsd:element name="resultSetRange" nillable="true" type="ns1:ResultSetRange" />
    <xsd:element name="searchType" nillable="true" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```




```
type record PlaceFinderOptions {
   String dataSource optional,
   String filterCountry optional,
   Envelope filterExtent optional,
   String filterType optional,
   ResultSetRange resultSetRange optional,
   String searchType optional
  }
```

75

Your template
(test data)

```
template PlaceFinderOptions findPlaceOptionsStrictRequest := {
  dataSource := "ArcWeb:ESRI.Gazetteer.World",
  filterCountry := "US",
  filterExtent := {
    coordSys := { datumTransformation := "dx", projection := "4326" },
    maxX := -116.0,
    maxY :=  36.0,
    minX := -118.0,
    minY :=  -20.0 },
  filterType := "",
  resultSetRange := { count := 5, startIndex := 0 },
  searchType := "exactMatch"
}
```

76

Your test case

```
testcase tcFindStrictPlace() runs on ptcType
                                        system PlaceFinderSampleHttpPort_COMPONENT {
  map (mtc: PlaceFinderSampleHttpPort_PORT, system: PlaceFinderSampleHttpPort_PORT);

  PlaceFinderSampleHttpPort_PORT.call (findPlace_op: {
    placeStrictReq, findPlaceOptionsStrictRequest }, localTimerValue) {
    [] PlaceFinderSampleHttpPort_PORT.getreply (findPlace_op: {-, - }
          value tStrictResponse) { // this is the expected result
          setverdict (pass);
    }
    [] PlaceFinderSampleHttpPort_PORT.getreply (findPlace_op: { -, -  }
          value ?) { // in any other case this is not good
          setverdict (fail);
    }
    [] PlaceFinderSampleHttpPort_PORT.catch (timeout) {
          // and if the SUT does not return this is not good either
          setverdict (fail);
    }
  }
}
```

References

- TTCN-3 standards:
  - http://www.ttcn-3.org/StandardSuite.htm
- Papers and tutorials:
  - http://www.ttcn-3.org/Tutorials.htm
  - http://www.site.uottawa.ca/~bernard/ttcn.html
- Tools:
  - Testing Tech: http://www.testingtech.de
  - Telelogic: http://www.telelogic.com
  - OpenTTCN: http://www.openttcn.com
  - TRex: http://www.trex.informatik.uni-goettingen.de/trac

78

Conclusions

- TTCN-3 provides the tools and framework for addressing the complexities of enterprise applications and SOA.

79