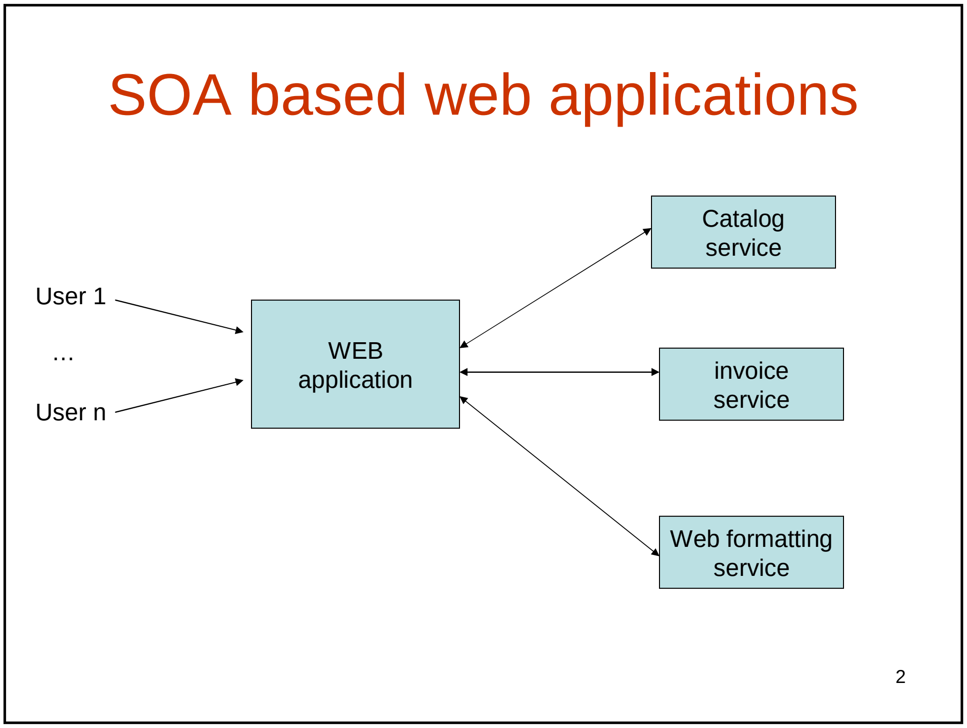
Testing Service Oriented Architecture Based Web Applications

By Bernard Stepien, Liam Peyton, Pulei Xiong, Pierre Seguin

uOttawa
L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

www.uOttawa.ca



SOA based web applications

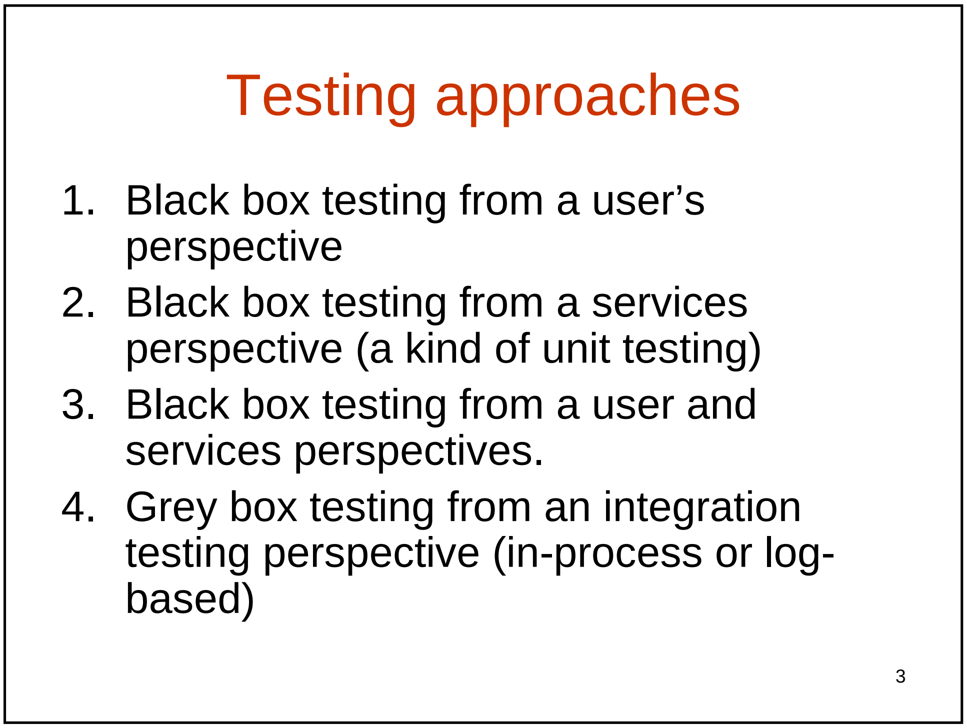
User 1
…
User n

WEB application

Catalog service

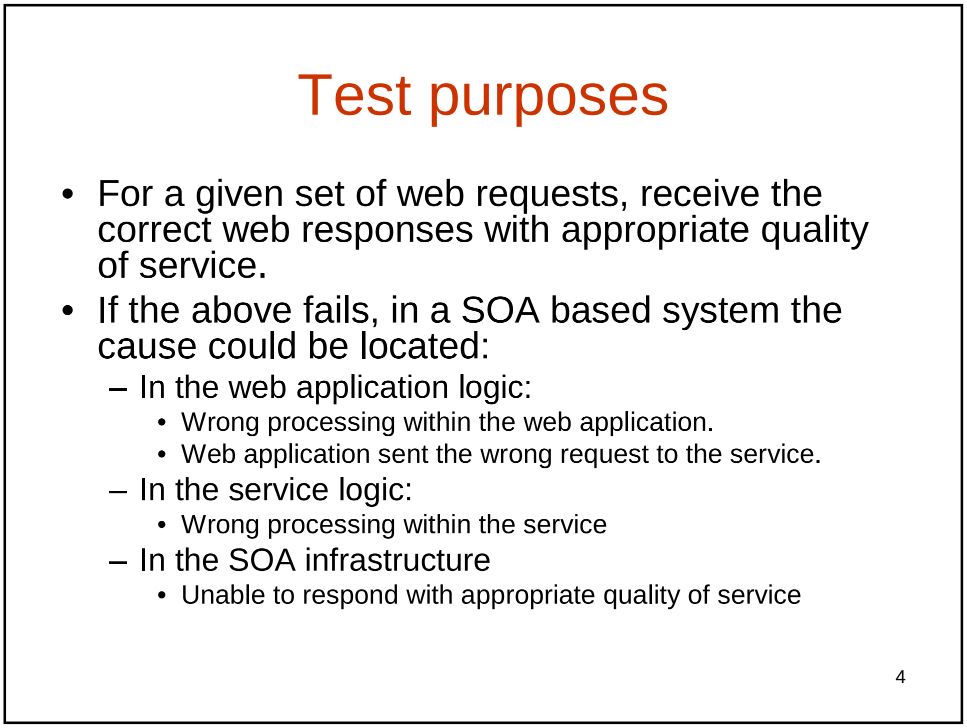
invoice service

Web formatting service



Testing approaches

1. Black box testing from a user's perspective
2. Black box testing from a services perspective (a kind of unit testing)
3. Black box testing from a user and services perspectives.
4. Grey box testing from an integration testing perspective (in-process or log-based)

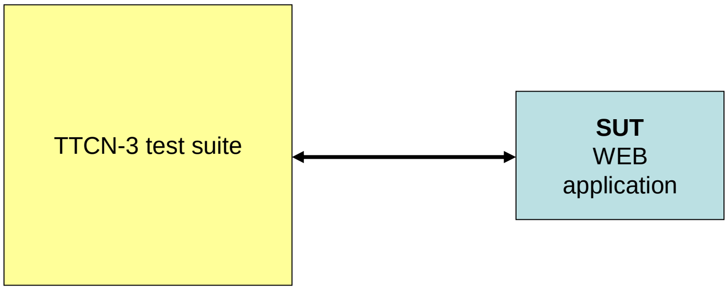


Test purposes

- For a given set of web requests, receive the correct web responses with appropriate quality of service.
- If the above fails, in a SOA based system the cause could be located:
  - In the web application logic:
    - Wrong processing within the web application.
    - Web application sent the wrong request to the service.
  - In the service logic:
    - Wrong processing within the service
  - In the SOA infrastructure
    - Unable to respond with appropriate quality of service

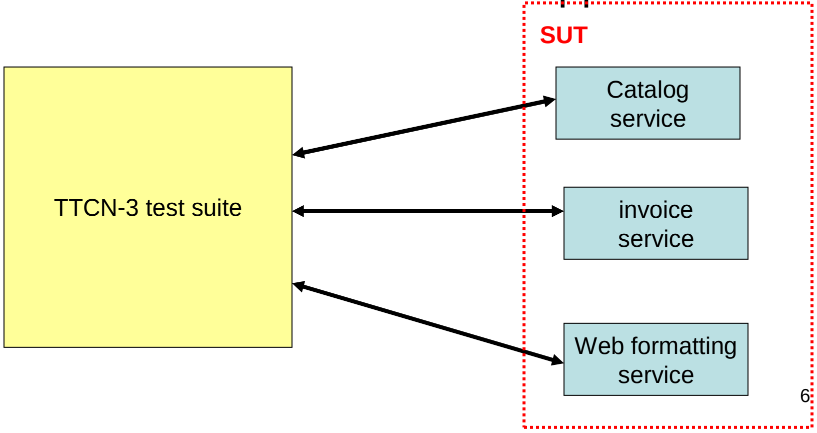
Black box testing
from a user’s perspective

- The SUT is observed only via the web application.
- The services are not visible to the tester



Black box testing
from a services perspective

- The SUT is the collection of services
- The ATS simulates the web application



First two approaches solutions

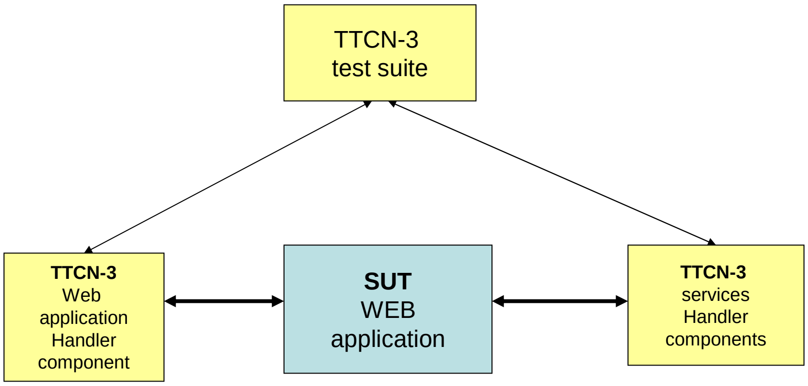
- *Already covered in previous research:*
  - *Web testing:*
    - *http://www.site.uottawa.ca/~bernard/Testing_a_servlet.pdf*
  - *SOAP application testing:*
    - *http://www.site.uottawa.ca/~bernard/TestingWebServices.pdf*

Problems with the black box separate approaches

- With the first approach (web application testing), if there is a failure we won't be able to determine it's cause accurately:
  - In the web application
  - In the underlying services
- With the second approach, we may conclude that the services are OK, but we can not guarantee that:
  - the web application places the correct service requests.
  - the web application processes correctly the service responses.

Black box testing from a user and services perspective

- The SUT is the web application only
- The web application is disconnected from the real services.
- A TTCN-3 parallel test component emulates the services messaging.

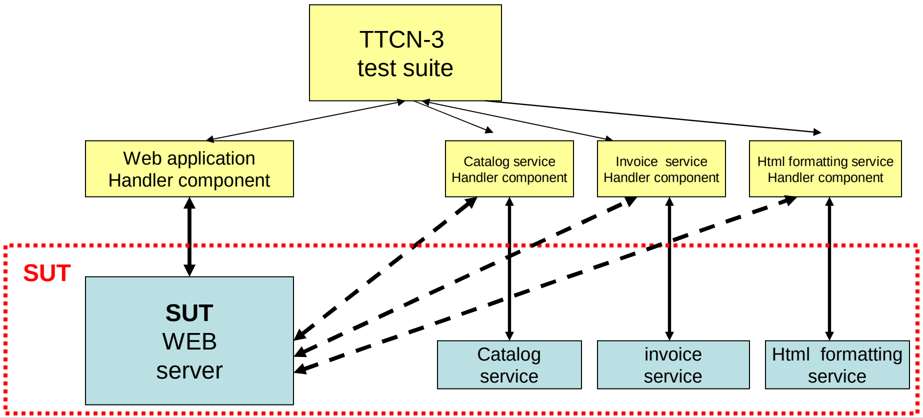


Advantages of approach 3

- This approach fully verifies that the web application sends the correct requests to the services, but only according to the tester’s opinion.
- Once this verified, if the web response to the user is still wrong, we can then conclude with confidence that the problem is located in the web application processing.

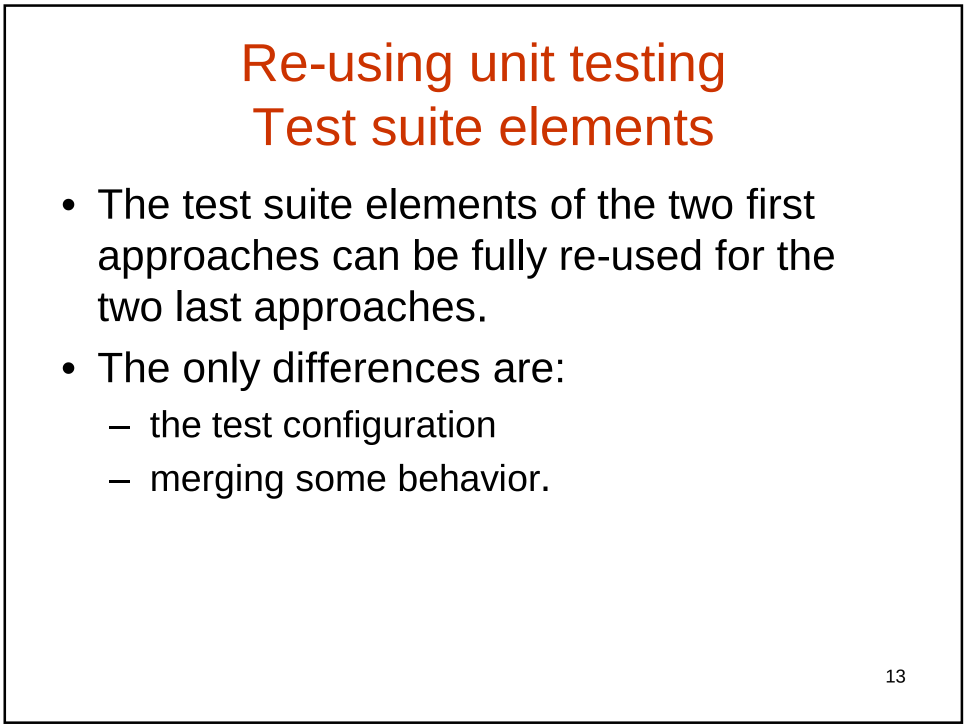
Grey box testing from an integration testing perspective

- All messages between all components are tested.
- The test suite intercepts the communication between the web application and the services and verifies them.



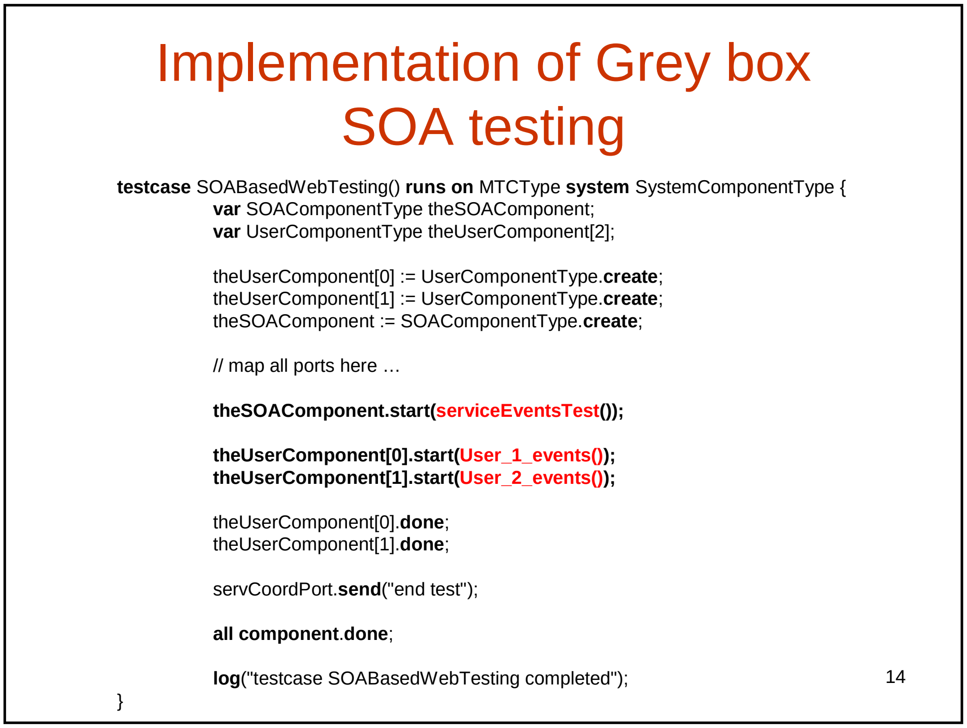
Reasons for integration testing

- Unit testing does not guarantee that components will work together correctly.
- Multi-user traffic could reveal faults:
  - In the web application (mixing client side data between users)
  - In the services (mixing server side data between responses and time outs due to load)
  - Inability of the SOA infrastructure to deliver responses with an appropriate quality of service



Re-using unit testing Test suite elements

- The test suite elements of the two first approaches can be fully re-used for the two last approaches.
- The only differences are:
  - the test configuration
  - merging some behavior.



Implementation of Grey box SOA testing

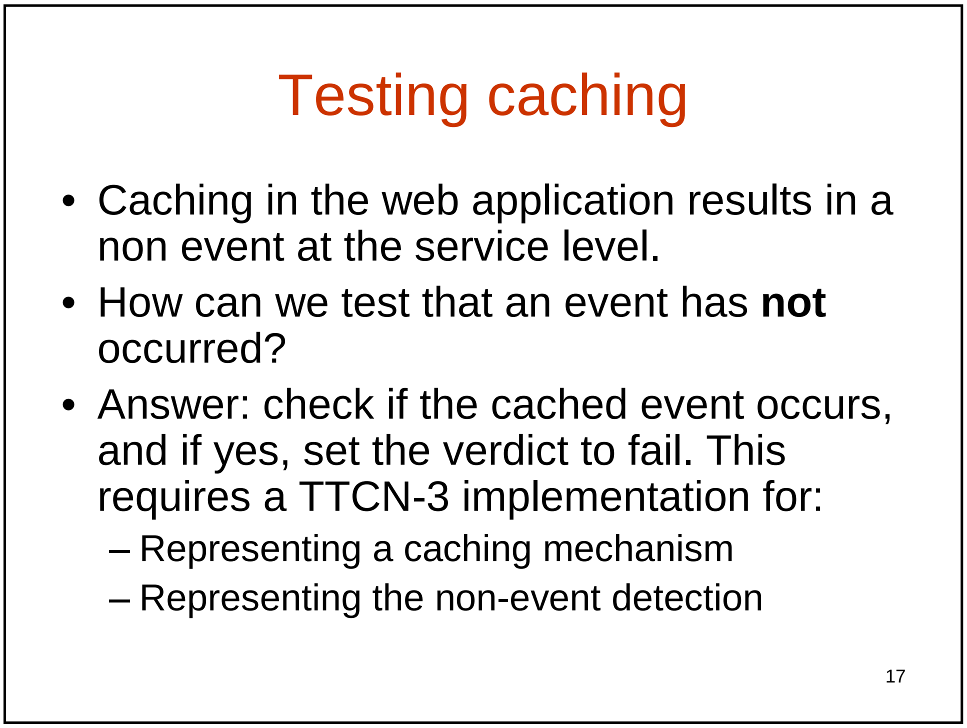
```
testcase SOABasedWebTesting() runs on MTCType system SystemComponentType {
        var SOAComponentType theSOAComponent;
        var UserComponentType theUserComponent[2];

        theUserComponent[0] := UserComponentType.create;
        theUserComponent[1] := UserComponentType.create;
        theSOAComponent := SOAComponentType.create;

        // map all ports here …

        theSOAComponent.start(serviceEventsTest());

        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");
}
```

User behavior specification

```
function User_1_behavior() runs on UserComponentType {
        var ResponseType theResponse;
        timer theTimer;
        var integer i;

        userWebPort.send(web_req_A);
        theTimer.start(5.0);
        alt {
                [] userWebPort.receive(web_resp_B) {
                        log("user 1 has received web_resp_B");
                        theTimer.stop;
                        setverdict(pass)
                }
                [] userWebPort.receive(ResponseType:?) -> value theResponse {
                        log("user 1 has receive wrong response: " & theResponse);
                        setverdict(fail)
                }
                [] theTimer.timeout {
                        log("User 1 timed out");
                        setverdict(inconc)
                }
        }
}
```

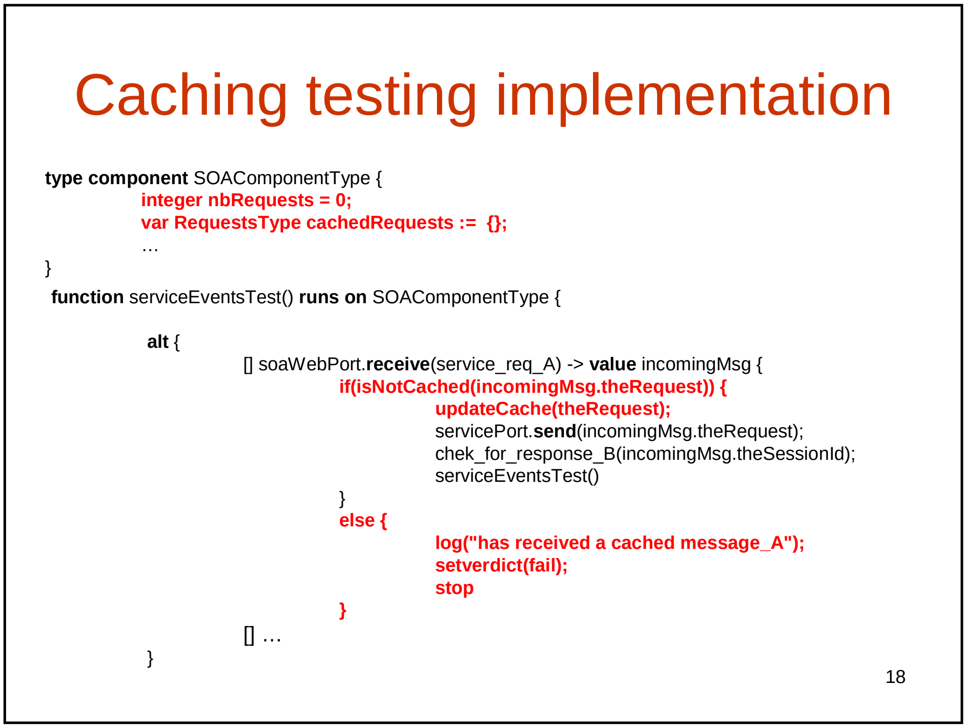
Testing challenges

- In Web pages, data is mixed with formatting information.
- Caching: Requests to the web application do not produce always a request to an underlying service.
- Service messages can not always be correlated directly to a specific user’s web applications messages.



Testing caching

- Caching in the web application results in a non event at the service level.
- How can we test that an event has **not** occurred?
- Answer: check if the cached event occurs, and if yes, set the verdict to fail. This requires a TTCN-3 implementation for:
  - Representing a caching mechanism
  - Representing the non-event detection



Caching testing implementation

```
type component SOAComponentType {
        integer nbRequests = 0;
        var RequestsType cachedRequests :=  {};
        …
}
 function serviceEventsTest() runs on SOAComponentType {

        alt {
                [] soaWebPort.receive(service_req_A) -> value incomingMsg {
                        if(isNotCached(incomingMsg.theRequest)) {
                                updateCache(theRequest);
                                servicePort.send(incomingMsg.theRequest);
                                chek_for_response_B(incomingMsg.theSessionId);
                                serviceEventsTest()
                        }
                        else {
                                log("has received a cached message_A");
                                setverdict(fail);
                                stop
                        }
                [] …
        }
```

TTCN-3 features for caching

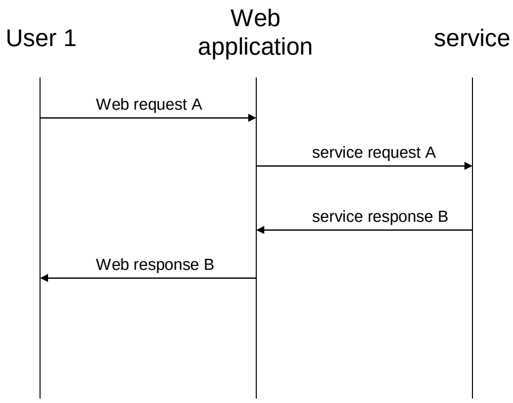
- Caching is very easy to implement in TTCN-3 because:
  - Easy building of lists or sets containing cached complex messages.
  - Easy lookup of the cache due to the powerful TTCN-3 matching mechanism for complex types.
  - Possibility to select messages subject to caching.

```
function isNotCached(RequestType theRequest) runs on SOAComponentType return  boolean {
        var integer i;

        for(i:=0; i < nbRequests; i:=i+1) {
                if(match(theRequest,  cachedRequests[i])) {
                        return false;
                }
        }

        return true;
}
```

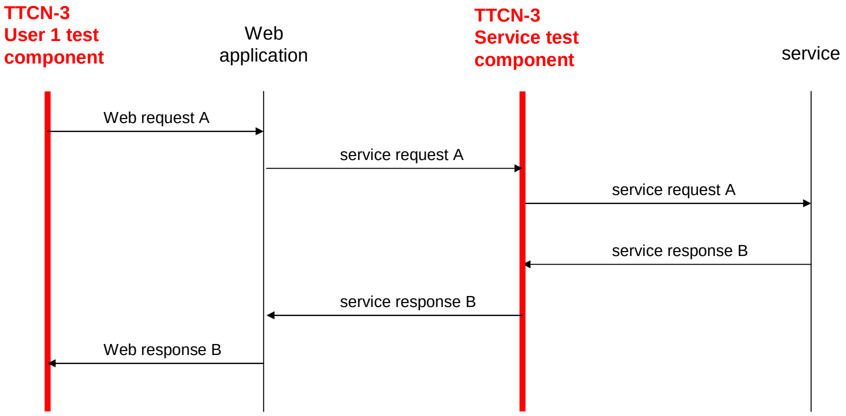
Correlation gap handling

- Temporal ordering problem. The web application may place its requests to the service in a different order as received from the users.
- Potential lack of indicators to assign a service request to a specific user.
- End-to-end tracking may work for a single user, but does not work in the case of multiple users.

Single user end to end tracking message flow



Single user end to end tracking testing architecture



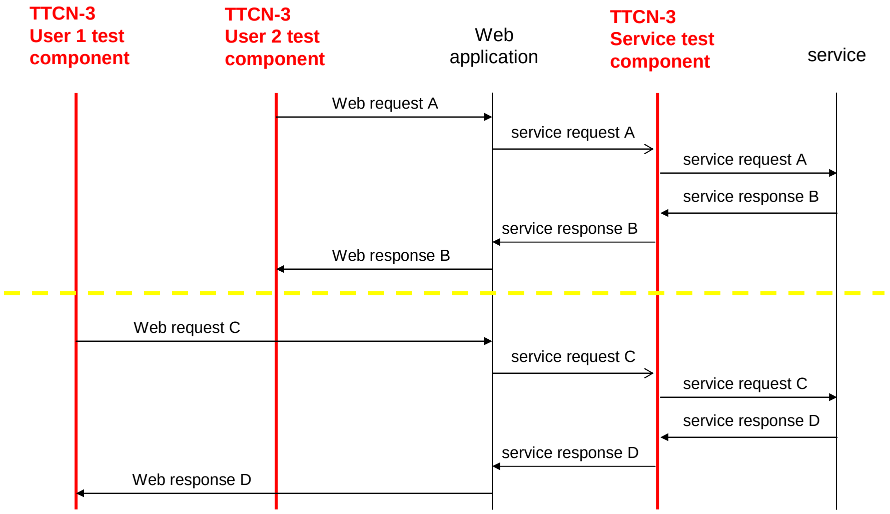
Single user
TTCN-3 behavior implementation

- Simplified behavior using only the MTC

```
userWebPort.send(web_req_A);

soaWebPort.receive(service_req_A) -> value incomingMsg {

servicePort.send(incomingMsg.theRequest); //service req A

servicePort.receive(service_resp_B)

soaWebPort.send(service_resp_B) to incomingMsg.theSessionId

userWebPort.receive(web_resp_B) {

setverdict(pass)
```

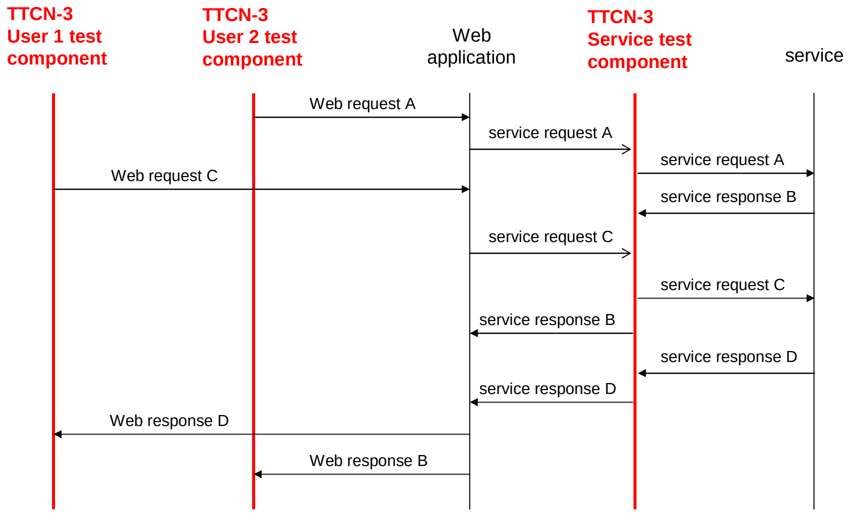
Multiple user message flow

ideal case



Multiple user message flow

one of many realistic cases



Multiple user handling

- Each user is portrayed by a TTCN-3 parallel test component.
- The service handler is portrayed by a single TTCN-3 parallel test component.
- For each service request received the service handler performs two kinds of checking:
  - It checks if such a message was expected, if yes, then forwards it to the service
  - It enforces the expected response from the service and if successful forwards the service response to the web application.

- The MTC tells to the service handler what requests to expect but not in which order. This is handled in a template:

```
template RequestsType expectedRequests :=
                                        { "service_req_A", "service_req_C"};
```

- At the end of the test, the service handler checks if the set of messages it was told to expect by the MTC matches the set of actually received messages.

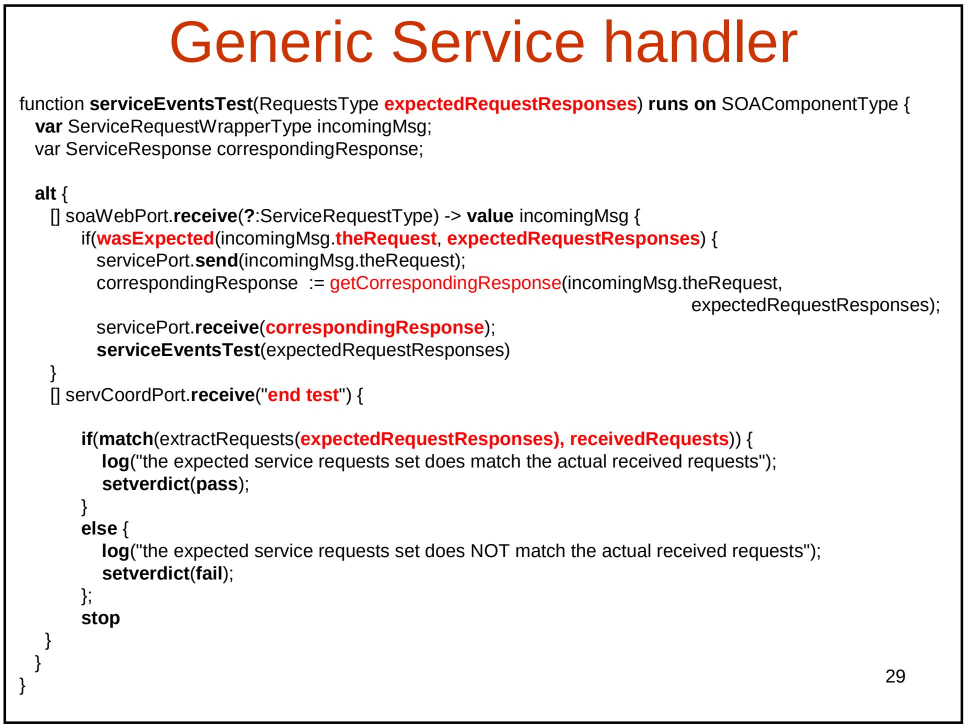
```
if(match(expectedRequests, receivedRequests)) {
      setverdict(pass);
}
else {
      setverdict(fail);
};
```

Service test component re-usability

- There are two ways to implement this architecture:
  - Hard code the service component with each expected alternative message in a function.
  - Make a generic service message handler that checks if a received message is present in the expected messages list.

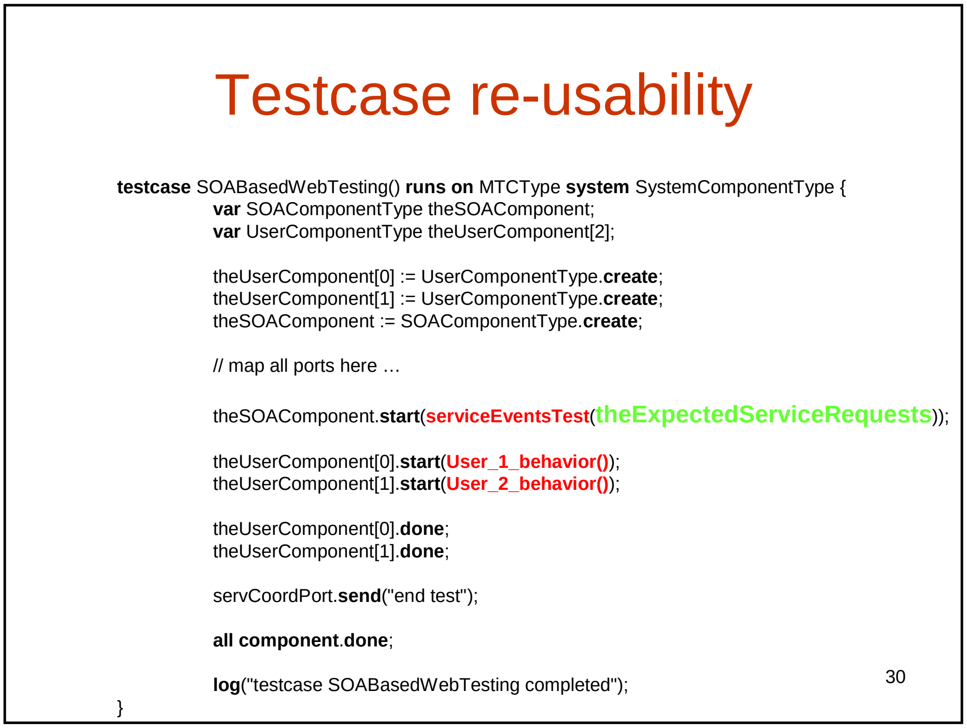
Hard coded Service handler

```
function serviceEventsTest(RequestsType expectedRequests) runs on SOAComponentType {
  var ServiceRequestWrapperType incomingMsg;

  alt {
    [] soaWebPort.receive(service_req_A) -> value incomingMsg {
        servicePort.send(incomingMsg.theRequest);
        check_response_B(incomingMsg.theSessionId);
        serviceEventsTest(expectedRequests)
    }
    [] soaWebPort.receive(service_req_C)  -> value incomingMsg {
        servicePort.send(incomingMsg.theRequest);
        check_response_D(incomingMsg.theSessionId);
        serviceEventsTest(expectedRequests)
    }
    [] servCoordPort.receive("end test") {

        if(match(expectedRequests, receivedRequests)) {
           log("the expected service requests set does match the actual received requests");
           setverdict(pass);
        }
        else {
           log("the expected service requests set does NOT match the actual received requests");
           setverdict(fail);
        };
        stop
   } …
```



Generic Service handler

```
function serviceEventsTest(RequestsType expectedRequestResponses) runs on SOAComponentType {
  var ServiceRequestWrapperType incomingMsg;
  var ServiceResponse correspondingResponse;

  alt {
    [] soaWebPort.receive(?:ServiceRequestType) -> value incomingMsg {
         if(wasExpected(incomingMsg.theRequest, expectedRequestResponses) {
           servicePort.send(incomingMsg.theRequest);
           correspondingResponse  := getCorrespondingResponse(incomingMsg.theRequest,
                                                               expectedRequestResponses);
           servicePort.receive(correspondingResponse);
           serviceEventsTest(expectedRequestResponses)
    }
    [] servCoordPort.receive("end test") {

         if(match(extractRequests(expectedRequestResponses), receivedRequests)) {
            log("the expected service requests set does match the actual received requests");
            setverdict(pass);
         }
         else {
            log("the expected service requests set does NOT match the actual received requests");
            setverdict(fail);
         };
         stop
    }
  }
}
```



Testcase re-usability

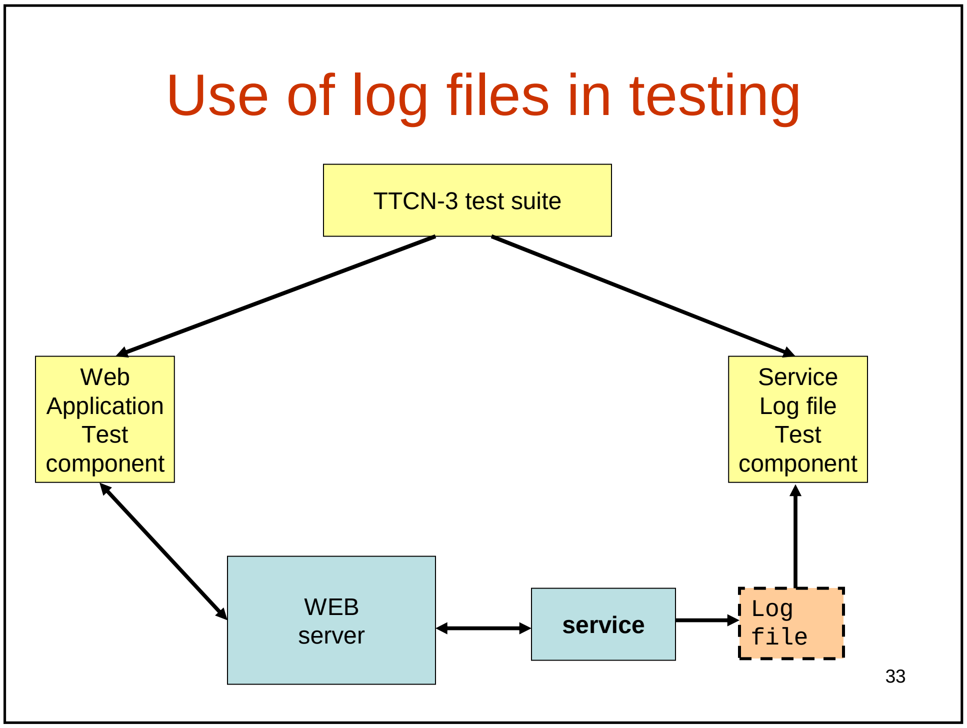
```
testcase SOABasedWebTesting() runs on MTCType system SystemComponentType {
        var SOAComponentType theSOAComponent;
        var UserComponentType theUserComponent[2];

        theUserComponent[0] := UserComponentType.create;
        theUserComponent[1] := UserComponentType.create;
        theSOAComponent := SOAComponentType.create;

        // map all ports here …

        theSOAComponent.start(serviceEventsTest(theExpectedServiceRequests));

        theUserComponent[0].start(User_1_behavior());
        theUserComponent[1].start(User_2_behavior());

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

        servCoordPort.send("end test");

        all component.done;

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

Advantages of the generic service handler

- The service handler does not need to be rewritten for each test campaign.
- Expected tuples of service requests/responses can be implemented with:
  - Templates for the sets of request/response tuples.
  - Parametric functions that take the expected Request/Response tuples sets.
- All of this thanks to the powerful TTCN-3 matching mechanism.

Using TTCN-3 to verify log files

- Perform the black box testing of the web application from a user’s point of view.
- Use the services log files as part of the SUT:
  - Decode the log files
  - Create a TTCN-3 function containing a behavior to verify log files automatically
- This approach moves the status of log files from post-mortem analysis to a fully active status
- Advantages: it does not disturb the normal operation of the SUT.



Use of log files in testing

TTCN-3 test suite

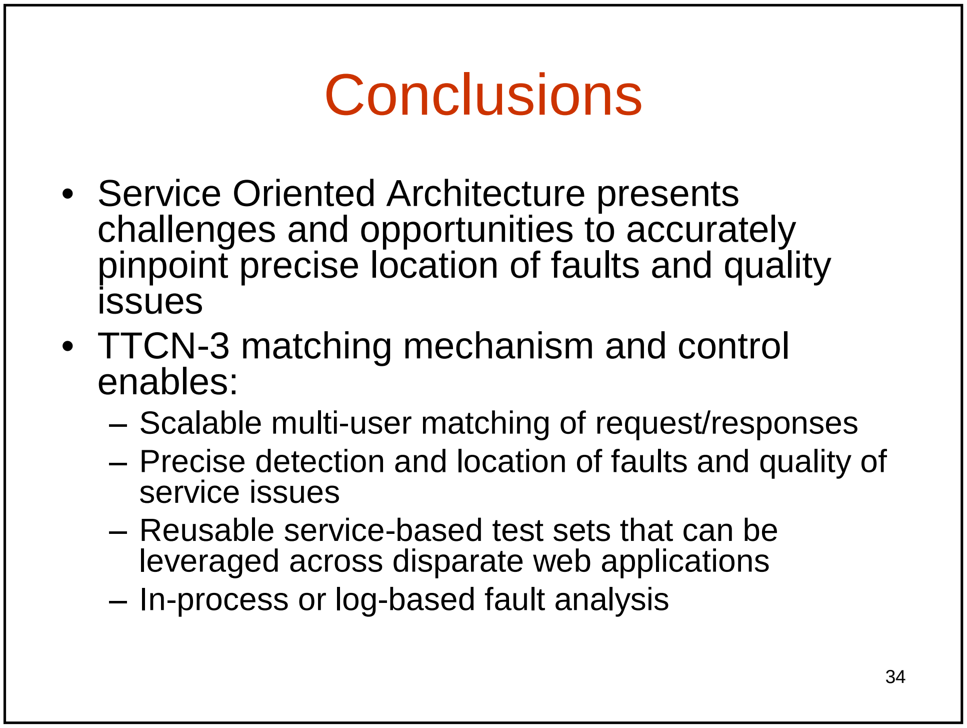
Web Application Test component

Service Log file Test component

WEB server

service

`Log file`



Conclusions

- Service Oriented Architecture presents challenges and opportunities to accurately pinpoint precise location of faults and quality issues
- TTCN-3 matching mechanism and control enables:
  - – Scalable multi-user matching of request/responses
  - – Precise detection and location of faults and quality of service issues
  - – Reusable service-based test sets that can be leveraged across disparate web applications
  - – In-process or log-based fault analysis

Contact information

- E-mail:
  - – bernard@site.uottawa.ca
  - – lpeyton@site.uottawa.ca
  - – xiong@site.uottawa.ca
- Further reading:
  - – http://www.site.uottawa.ca/~bernard/ttcn.html
  - – http://www.site.uottawa.ca/~lpeyton/