October 24, 2013

TTCN4SOA™: a TTCN-3 architecture and framework for SOA testing

Fabio De Rosa[2], Ariele P. Maesano[1], Libero Maesano[2]
Ariele.Maesano@lip6.fr, {fabio.de-rosa, libero.maesano}@simple-eng.com

[1] Laboratoire d'Informatique de Paris 6 (UPMC), Paris, France
[2] SIMPLE ENGINEERING, 31-35, rue St. Ambroise, 75011 Paris, France

Table of contents

1. SOA Conceptual Framework
2. SOA Functional Testing
3. TTCN4SOA™ Testing Architecture
4. TTCN4SOA™ framework in TTCN-3 language
5. Conclusion

What is SOA

- In the digital economy* tens, hundreds, thousands, … applications, systems, devices will be connected and will collaborate **without human intermediation**, putting in place the **automation of business processes** that **support economic, administrative and social activities**
- Service Oriented Architecture (SOA) is **the design and implementation style** that allows organizations to put in practice **dynamic collaborations** of **loosely coupled** systems, in order to achieve **flexible, dependable** and **secure** business process automation in the digital economy

* W. Brian Arthur, "The second economy." McKinsey Quarterly, October 2011

SOA conceptual framework

- The collaboration among systems is carried out through the **exchange** (offering and invoking) of **services** - a **provider** doing something for / on behalf of a **consumer** - described/regulated by **contracts**
- **Service contracts** are:
  - **models** of the **service functions, interfaces** and **external behaviours** (including **security** and **quality of service** aspects) of the **service provider** and **consumer**
  - **informal** vs. **formal**
  - **partial** vs. **complete**
- Examples: The **API economy** - services offered through API on the Internet - are described by **formal** (executable) **interface models** (WSDL, WADL, REST/JSON, …) + **informal function** and **behaviour models** (documentation)
- A **services architecture** is a network of **participant systems**, playing roles defined by **service contracts**, that collaborate in order to achieve business goals

What is SOA testing

- Whatever its completeness and formality level, a service contract is a black-box model – it does not include any information about internals - of the systems that claim to implement it
- SOA functional (in the large) testing: stimulating and observing the behaviours of the participants at their interfaces in order to assess their compliance with the involved service contracts
- SOA vulnerability testing: testing the participants' resilience when submitted to malicious attacks (outside the scope of this presentation)
- SOA functional testing of a services architecture is black-box testing of participants and, when some interactions among them are observable, grey-box testing of the overall architecture
- SOA information hiding: participant white-box models and internals (codes) are hidden each other
- Techniques such as formal verification, model checking and white-box testing are not applicable to services architectures – only by system owners to theirs own systems

Why SOA testing is important

- How **stakeholders' confidence** on the services architecture **participants' compliance** with the service contracts can be increased and strengthened ?
  By **SOA (functional) testing**
- SOA (functional) testing is *by definition* **model-based testing** – it is **testing implementations (participants) against models (contracts)**:
  - **more complete** the **contract (model)** is – more large the **testing extent** (all **aspects** of the contract – e.g. security, quality of service …- can be tested)
  - **more formal** the **model (contract)** is – more **automatable** the test **generation** and **run** are

Why SOA testing is hard

- lack of observability
- lack of trust in the employed engineering methods
- lack of direct control of the service implementation lifecycles
- dynamic (run time) binding of systems with service roles
- uncertainty of the test verdicts (false positives and negatives)
- organizational complexity of multi-owner services architectures
- elastic demand of computational resources
- increasing scale factor of the services architectures
- high labor costs
- high equipment costs
- questionable efficacy of human-based testing activity

SOA testing (generation and run) automation is a must

BN4SAT Project*

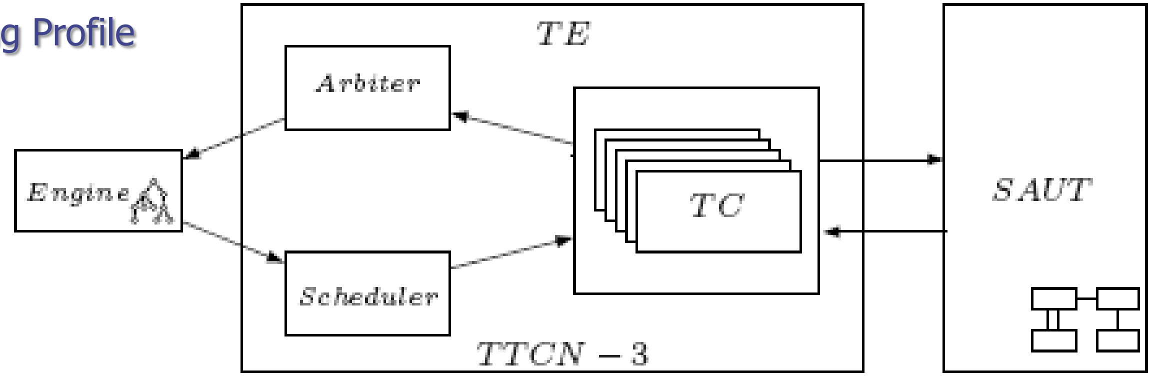
- The on-going research on SOA testing spans three main areas:
  - the **automated generation** of test cases (accuracy, robustness, safety, fault-tolerance …) from models (contract models, SUT …) for service unit and composition testing
  - the **automation of the test execution** and **arbitration environments**
  - **advanced (inference-based) methods and tools for static (priority-based)** and **dynamic** (based on the verdicts of previously executed test cases) **scheduling**
- Bayesian Networks for Services Architecture Testing (BN4SAT) project is centred on the application of a **Bayesian network** (BN) inference approach and technology for the scheduling of test suites
  - It appears to be adequate for **failure seeking** and **troubleshooting** (identification of faulty participants) in services architecture that are only **partially observable**
- TTCN4SOA™ is the TTCN-3 execution framework of the BN4SAT architecture

(*) The project is conducted in cooperation between the Laboratoire d'Informatique de Paris VI (LIP6 - Université Pierre et Marie Curie - Paris VI - France), the Centre National de la Recherche Scientifique (CNRS - France) and Simple Engineering, a European group specialized in the design of services architectures. The project is partially funded by the Association Nationale de la Recherche et de la Technologie (ANRT - France)

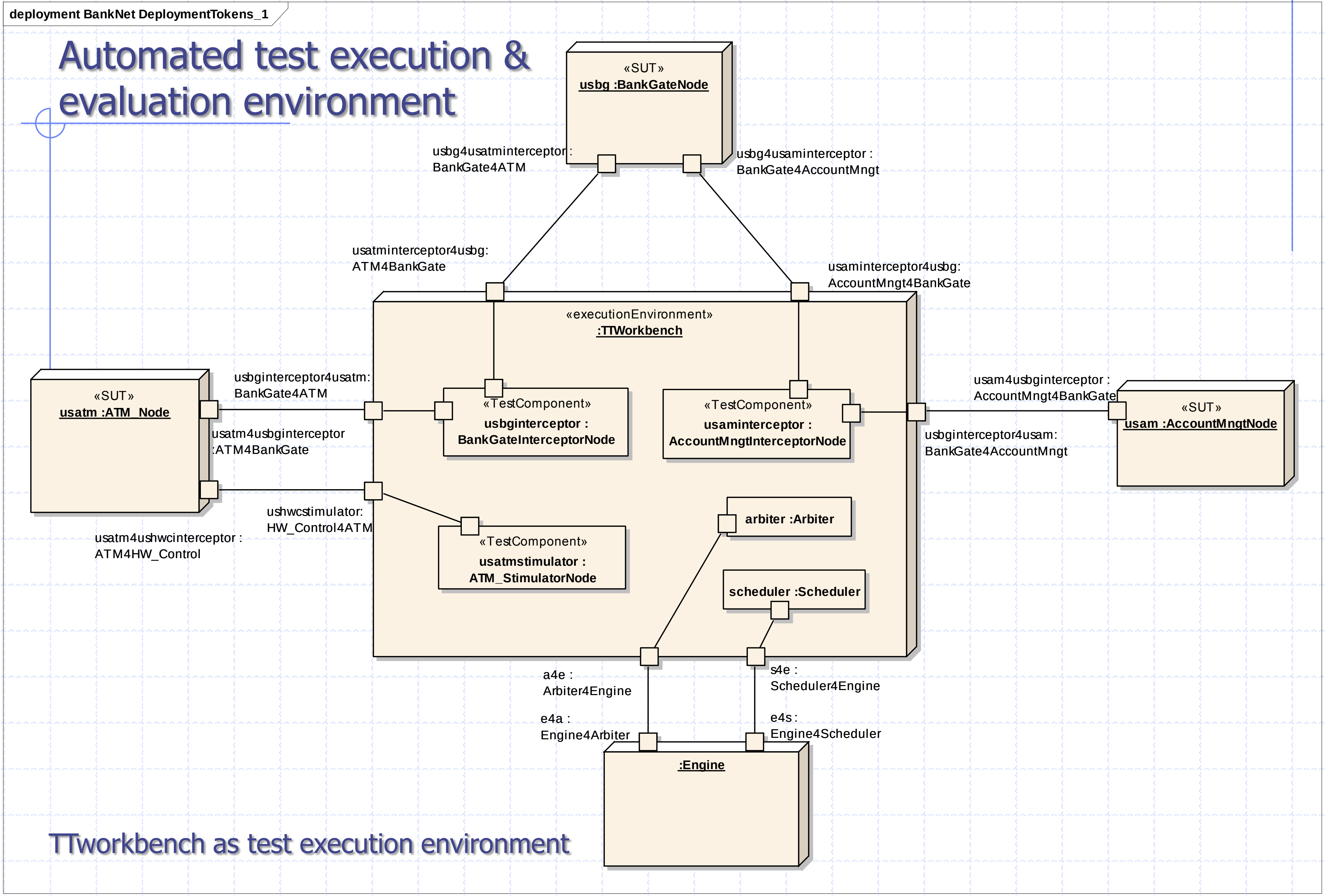
TTCN4SOA™: Testing Architecture

- **Executor,** a TTCN-3 execution environment with specialized **test components**:
  - *SUT proxies*, that are charged to convey interactions to and from the SUTs through the TRI interface,
  - *Emulators* - that emulate the behaviour of some elements of the SAUT, either by sending stimuli to a SUT (*stimulator*) or by emulating a SUT response, in order to run a test (*stub*)
  - *Interceptors* - that sit, transparently, between two connected SUT, and observe the interaction and perform different tasks
- **Arbiter** collects local test verdicts generated by the Test Components and produces final test verdict standard value (*pass*, *fail*, *error*, *inconc*)..
- **Scheduler** drives the execution of test runs (it starts and stops a test run).
- **Engine**, which is notified by the Arbiter with test verdicts, and manages the Scheduler by handling the test strategy on the basis of probabilistic inference on the acquired test verdicts

Compliant with OMG UML Testing Profile
http://www.omg.org/spec/UTP/1.2/



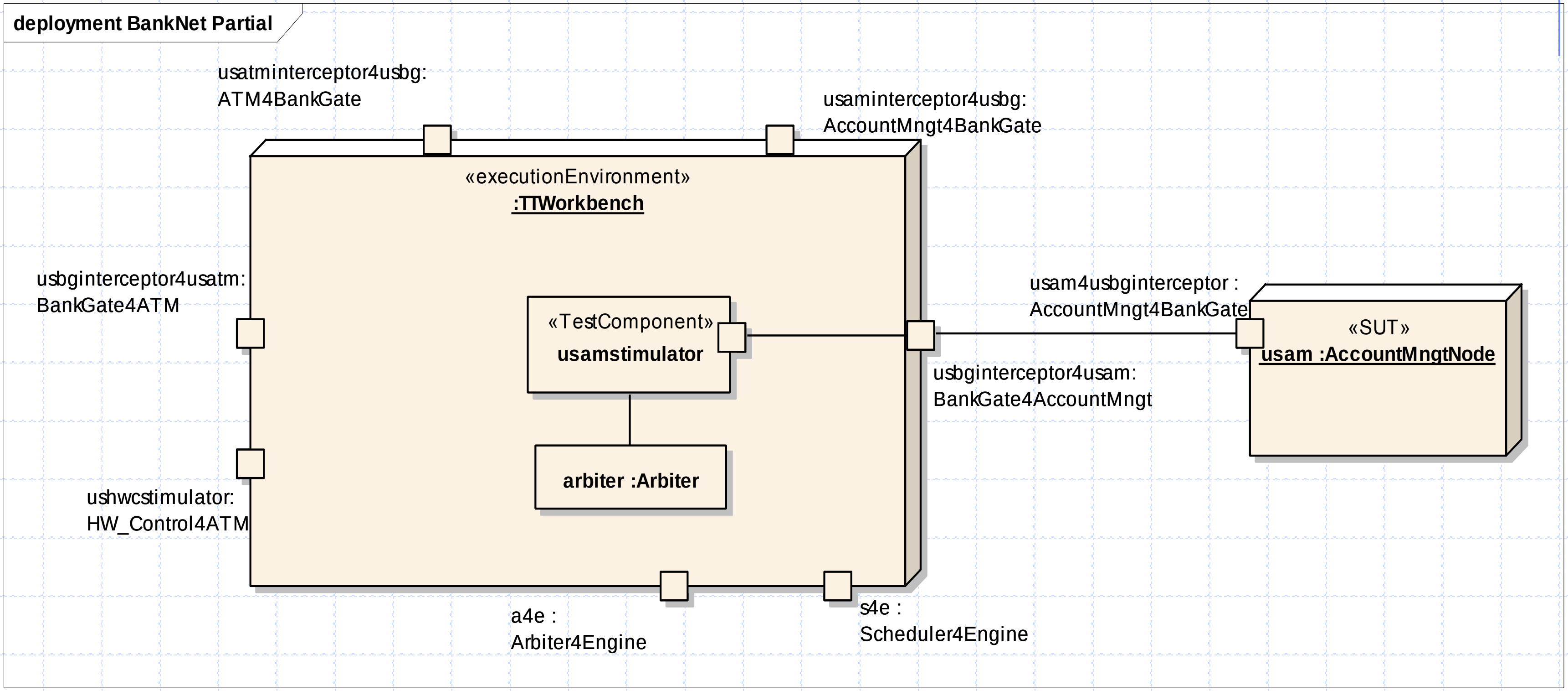
Automated test execution & evaluation environment


deployment BankNet DeploymentTokens_1
«SUT»
usbg :BankGateNode
usbg4usatminterceptor : BankGate4ATM
usbg4usaminterceptor : BankGate4AccountMngt
usatminterceptor4usbg: ATM4BankGate
usaminterceptor4usbg: AccountMngt4BankGate
«executionEnvironment»
:TTWorkbench
«SUT»
usatm :ATM_Node
usbginterceptor4usatm: BankGate4ATM
usatm4usbginterceptor :ATM4BankGate
«TestComponent»
usbginterceptor : BankGateInterceptorNode
«TestComponent»
usaminterceptor : AccountMngtInterceptorNode
usam4usbginterceptor : AccountMngt4BankGate
«SUT»
usam :AccountMngtNode
usbginterceptor4usam: BankGate4AccountMngt
ushwcstimulator: HW_Control4ATM
usatm4ushwcinterceptor : ATM4HW_Control
«TestComponent»
usatmstimulator : ATM_StimulatorNode
arbiter :Arbiter
scheduler :Scheduler
a4e : Arbiter4Engine
s4e : Scheduler4Engine
e4a : Engine4Arbiter
e4s : Engine4Scheduler
:Engine

TTworkbench as test execution environment

Stimulator

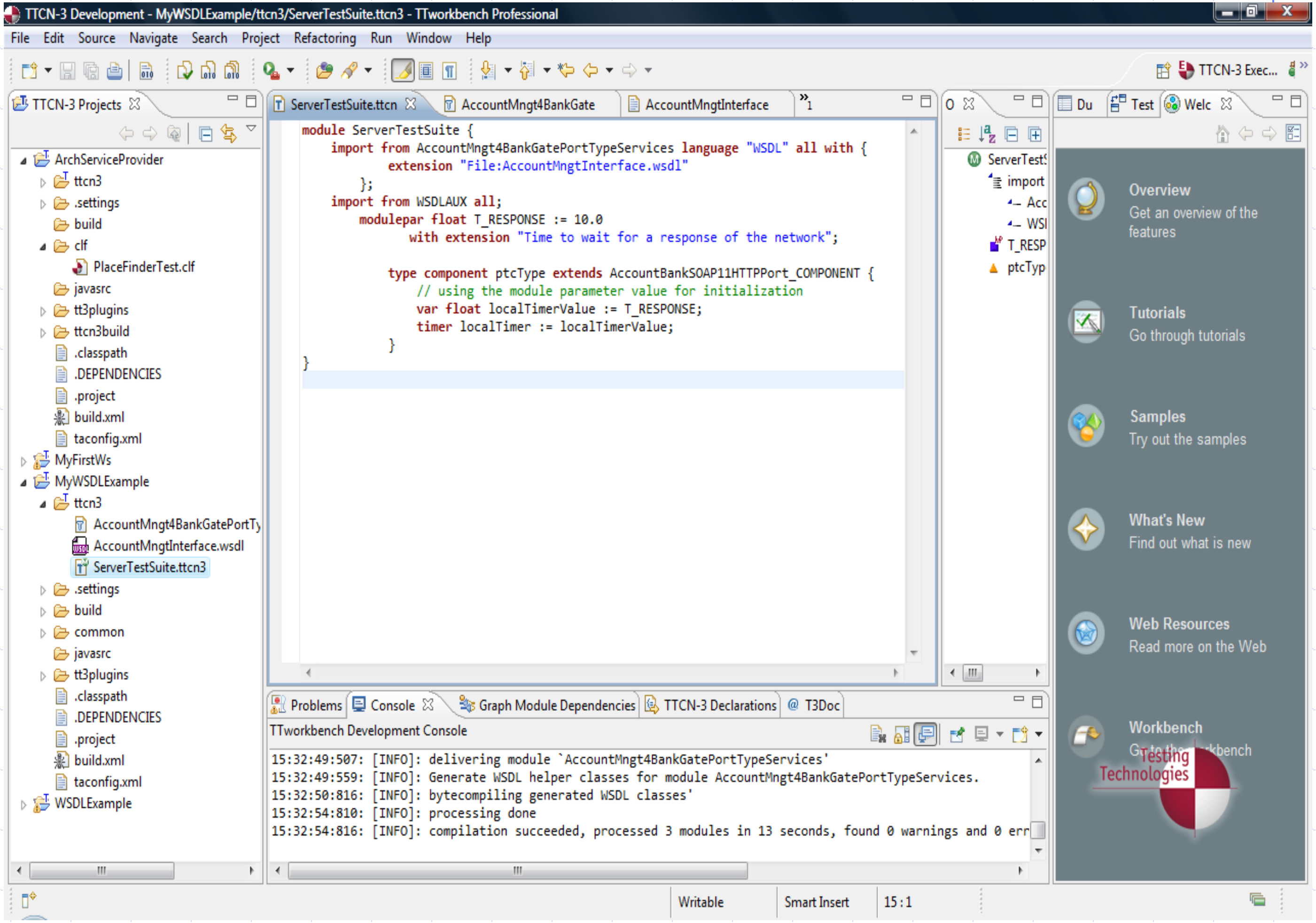
- Stimulator test component for *AccountMngt* service*



* This example has been taken from the OMG UML Testing Profile v1.2 and adapted to SOA scenarios

SUT Proxy

- SUT Proxy: Automatic Generation from *AccountMngt* WSDL file



```
module ServerTestSuite {
    import from AccountMngt4BankGatePortTypeServices language "WSDL" all with {
          extension "File:AccountMngtInterface.wsdl"
        };
    import from WSDLAUX all;
        modulepar float T_RESPONSE := 10.0
             with extension "Time to wait for a response of the network";

          type component ptcType extends AccountBankSOAP11HTTPPort_COMPONENT {
              // using the module parameter value for initialization
              var float localTimerValue := T_RESPONSE;
              timer localTimer := localTimerValue;
          }
}
```

```
15:32:49:507: [INFO]: delivering module `AccountMngt4BankGatePortTypeServices'
15:32:49:559: [INFO]: Generate WSDL helper classes for module AccountMngt4BankGatePortTypeServices.
15:32:50:816: [INFO]: bytecompiling generated WSDL classes'
15:32:54:810: [INFO]: processing done
15:32:54:816: [INFO]: compilation succeeded, processed 3 modules in 13 seconds, found 0 warnings and 0 err
```

Specific platform Adapter and Codec library are automatically generated

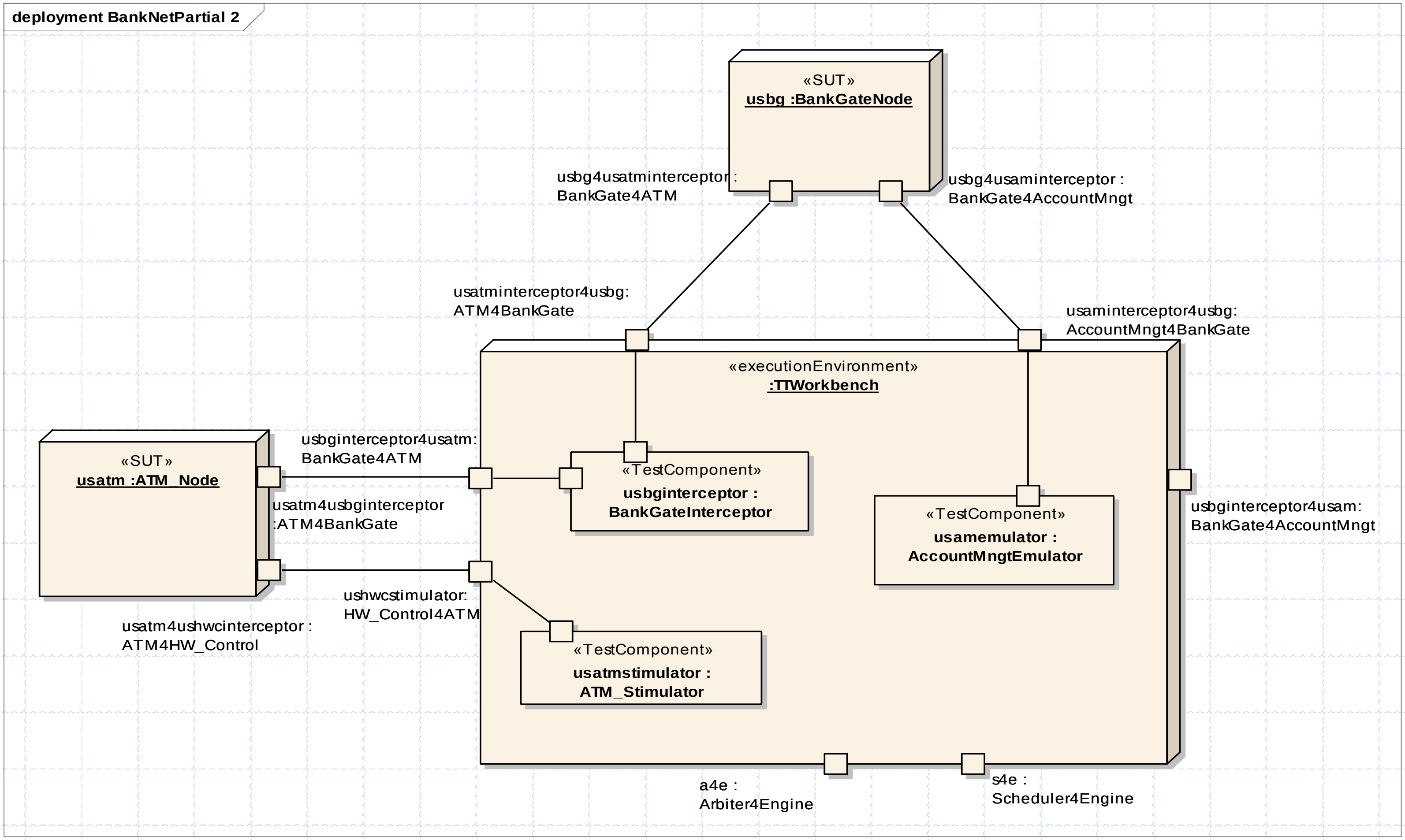
Stimulator test logic

- TTCN-3 source code snapshot for test logic of the *AccountMngtStimulator* component

```
module AccountMngtStimulator {
      ...
 template MoneyType money := {fAmount := 1000, fCurrency := "Euro"}
 template DebitMsgType debit_1 := {account := "123456", amount := money}
 template DebitRespMsgType debit_1_resp := {response := true}
 testcase tcDebit()
 runs on ptcType system AccountBankSOAP11HTTPPort_COMPONENT {
      map(mtc:AccountBankSOAP11HTTPPort_PORT,
      system:AccountBankSOAP11HTTPPort_PORT);
      AccountBankSOAP11HTTPPort_PORT.call(debit_op:{debit_1}, localTimerValue) {
        [] AccountBankSOAP11HTTPPort_PORT.getreply(debit_op: {- } value
      debit_1_resp) {
                setverdict(pass);
        }
        [] AccountBankSOAP11HTTPPort_PORT.getreply(debit_op:{-} value ?) {
                setverdict(fail, "Response mismatch");
        }
        [] AccountBankSOAP11HTTPPort_PORT.catch(debit_op, SystemException:?) {
                setverdict(fail, "Communication exception");
        }
        [] AccountBankSOAP11HTTPPort_PORT.catch(timeout) {
                setverdict(fail, "Timeout occurred.");
        }
      };
 }
}
```

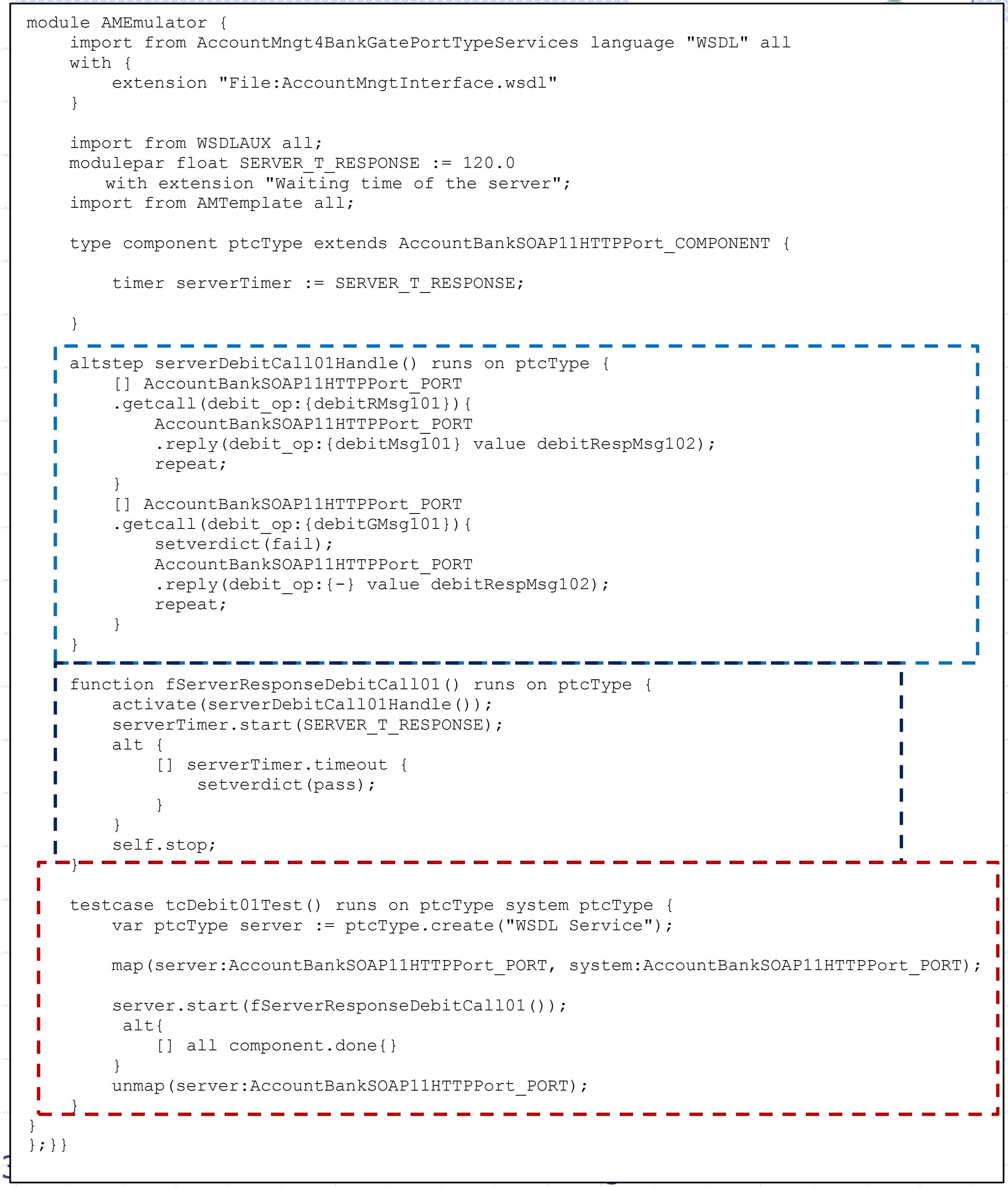
Emulator

- Emulator test component for *AccountMngt* service



Emulator Test Logic

- TTCN-3 source code for test logic of emulator component:
  - *alternative step*
  - *function*



```
module AMEmulator {
    import from AccountMngt4BankGatePortTypeServices language "WSDL" all
    with {
        extension "File:AccountMngtInterface.wsdl"
    }

    import from WSDLAUX all;
    modulepar float SERVER_T_RESPONSE := 120.0
       with extension "Waiting time of the server";
    import from AMTemplate all;

    type component ptcType extends AccountBankSOAP11HTTPPort_COMPONENT {

        timer serverTimer := SERVER_T_RESPONSE;

    }

    altstep serverDebitCall01Handle() runs on ptcType {
        [] AccountBankSOAP11HTTPPort_PORT
        .getcall(debit_op:{debitRMsg101}){
            AccountBankSOAP11HTTPPort_PORT
            .reply(debit_op:{debitMsg101} value debitRespMsg102);
            repeat;
        }
        [] AccountBankSOAP11HTTPPort_PORT
        .getcall(debit_op:{debitGMsg101}){
            setverdict(fail);
            AccountBankSOAP11HTTPPort_PORT
            .reply(debit_op:{-} value debitRespMsg102);
            repeat;
        }
    }

    function fServerResponseDebitCall01() runs on ptcType {
        activate(serverDebitCall01Handle());
        serverTimer.start(SERVER_T_RESPONSE);
        alt {
            [] serverTimer.timeout {
                setverdict(pass);
            }
        }
        self.stop;
    }

    testcase tcDebit01Test() runs on ptcType system ptcType {
        var ptcType server := ptcType.create("WSDL Service");

        map(server:AccountBankSOAP11HTTPPort_PORT, system:AccountBankSOAP11HTTPPort_PORT);

        server.start(fServerResponseDebitCall01());
         alt{
            [] all component.done{}
        }
        unmap(server:AccountBankSOAP11HTTPPort_PORT);
    }
}
};}}
```

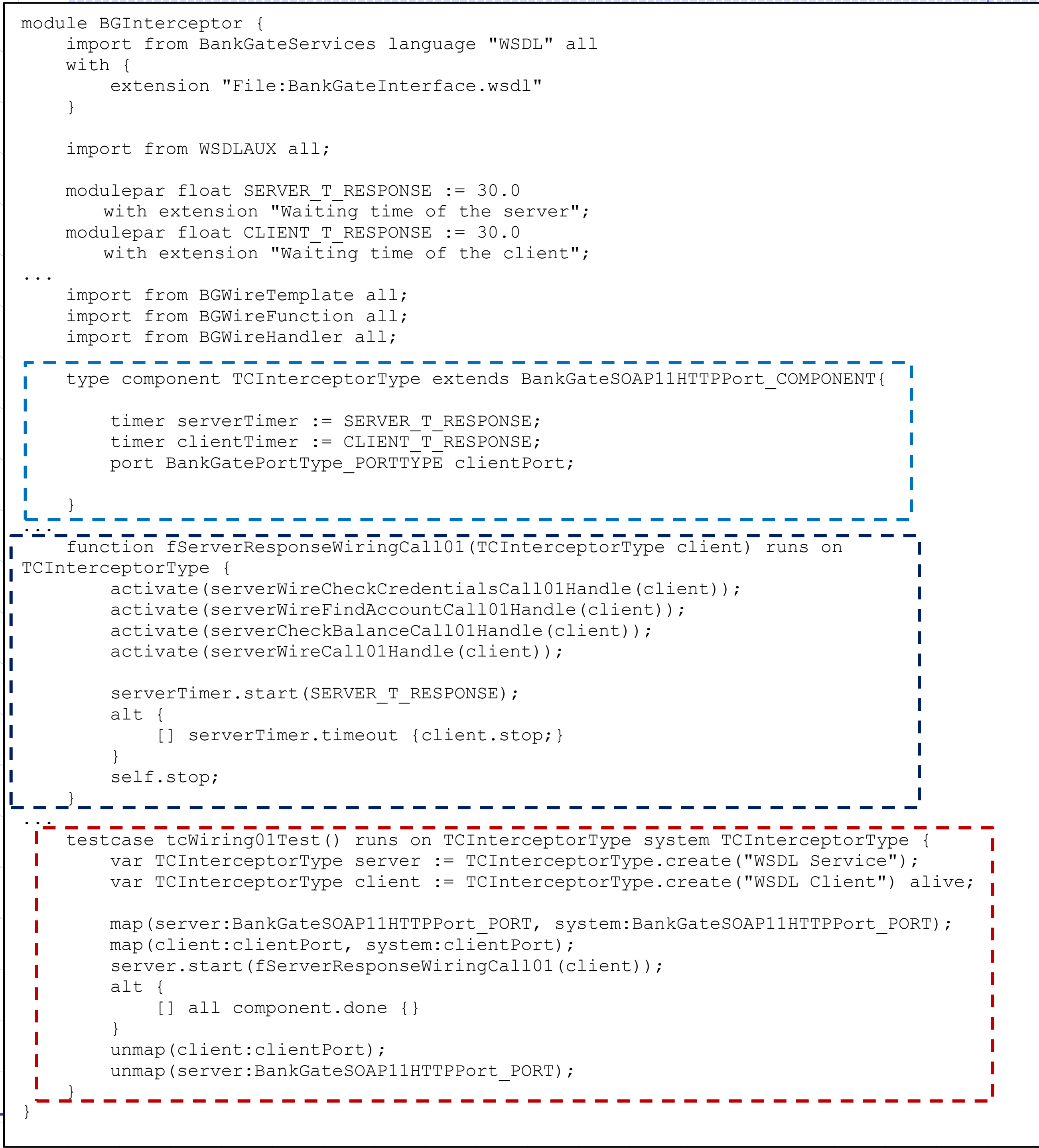
Interceptor

A TTCN-3 *Interceptor component* is structured as follows:

- A ServiceInterceptor module, which is the main module containing test cases
  - For each operation provided by the service and for each test case defined within the module, a special function is declared: this function controls all test case logic execution by activating specific test case handlers
- A ServiceOperationHandler module for each operation provided by the service. This module contains alternative step definitions able to handle one specific test case under execution
- A ServiceOperationFunction module for each operation provided by the service. This module contains functions implementing the consumer part of the interceptor (i.e., these functions are able to send and / or receive messages to/from the service under test)
- A ServiceOperationTemplate module for each operation provided by the service. This module contains request and/or response and/or fault message templates for a specified service operation

Interceptor Test Logic

- TTCN-3 source code for test logic of interceptor component
  - *Main module*
  - *Operation handler module*
  - *Operation function module*
  - *Operation template module*



```
module BGInterceptor {
    import from BankGateServices language "WSDL" all
    with {
        extension "File:BankGateInterface.wsdl"
    }

    import from WSDLAUX all;

    modulepar float SERVER_T_RESPONSE := 30.0
       with extension "Waiting time of the server";
    modulepar float CLIENT_T_RESPONSE := 30.0
       with extension "Waiting time of the client";
...
    import from BGWireTemplate all;
    import from BGWireFunction all;
    import from BGWireHandler all;

    type component TCInterceptorType extends BankGateSOAP11HTTPPort_COMPONENT{

        timer serverTimer := SERVER_T_RESPONSE;
        timer clientTimer := CLIENT_T_RESPONSE;
        port BankGatePortType_PORTTYPE clientPort;

    }
...
    function fServerResponseWiringCall01(TCInterceptorType client) runs on
TCInterceptorType {
        activate(serverWireCheckCredentialsCall01Handle(client));
        activate(serverWireFindAccountCall01Handle(client));
        activate(serverCheckBalanceCall01Handle(client));
        activate(serverWireCall01Handle(client));

        serverTimer.start(SERVER_T_RESPONSE);
        alt {
            [] serverTimer.timeout {client.stop;}
        }
        self.stop;
    }
...
    testcase tcWiring01Test() runs on TCInterceptorType system TCInterceptorType {
        var TCInterceptorType server := TCInterceptorType.create("WSDL Service");
        var TCInterceptorType client := TCInterceptorType.create("WSDL Client") alive;

        map(server:BankGateSOAP11HTTPPort_PORT, system:BankGateSOAP11HTTPPort_PORT);
        map(client:clientPort, system:clientPort);
        server.start(fServerResponseWiringCall01(client));
        alt {
            [] all component.done {}
        }
        unmap(client:clientPort);
        unmap(server:BankGateSOAP11HTTPPort_PORT);
    }
}
```

Service Operation Handler for Wire

- TTCN-3 source code snapshot for the *BankGateWireHandler* module

```
module BGWireHandler {

    import from BGWireTemplate all;
    import from BGWireFunction all;
    import from BGInterceptor all;
    import from BankGateServices all;

    altstep serverWireCall01Handle(TCInterceptorType client) runs on TCInterceptorType {

        [] BankGateSOAP11HTTPPort_PORT
        .getcall(wireMoney_op:{wireReqMsg601}){

            client.start(fcWireCall01());
            alt{
                [] client.done{}
            }

            BankGateSOAP11HTTPPort_PORT
            .reply(wireMoney_op:{wireReqMsg601} value wireRespMsg602);
            setverdict(pass);
            repeat;
        }
        [] BankGateSOAP11HTTPPort_PORT
        .getcall(wireMoney_op:{wireGeneralMsg}){
            BankGateSOAP11HTTPPort_PORT.raise(wireMoney_op, wireRespFaultMsg601("404"));
            setverdict(fail);
            repeat;
        }
    }
...
}
```

Service Operation Function for Wire

- TTCN-3 source code snapshot for the *BankGateWireFunction* module

```
module BGWireFunction {

    import from BGWireTemplate all;
    import from BGInterceptor all;
    import from BankGateServices all;

    function fcWireCall01() runs on TCInterceptorType {

        clientPort.call(wireMoney_op:{wireMsg601}, CLIENT_T_RESPONSE) {
            [] clientPort.getreply(wireMoney_op:{wireMsg601} value wireRespMsg602) {
                setverdict(pass, "Response from Bankgate matches");
            }
            [] clientPort
            .getreply(wireMoney_op:{wireMsg601} value ?) {
                setverdict(fail, "Response mismatch");
            }
            [] clientPort
            .catch(wireMoney_op, FaultMsgType:wireFaultMsg("503")) {
                setverdict(fail, "Received fault message");
            }
            [] clientPort
            .catch(wireMoney_op, FaultMsgType:wireFaultMsg("404")) {
                setverdict(fail, "Received fault message");
            }
            [] clientPort
            .catch(wireMoney_op, SystemException:?) {
                setverdict(fail, "Communication exception");
            }
            [] clientPort.catch(timeout) {
                setverdict(fail, "Timeout occurred.");
            }
        };
    }
...
}
```

Service Operation Template for Wire

- TTCN-3 source code snapshot for the *BankGateWireTemplate* module

```
module BGWireTemplate {

    import from BankGateServices all;

    /* Wire request message templates */
    template WireMoneyMsgType wireGeneralMsg := {
        wsaInfo := {
            messageID := ?,
            action_ := ?,
            relatesTo := omit
        },
        swiftNumber := ?,
        sourceAcc := ?,
        targetAcc := ?,
        amount := {fAmount := ?, fCurrency := ?}
    }
    template WireMoneyMsgType wireReqMsg601 := {
            wsaInfo := {
                        messageID := "m-usatm-usbg-601",
                        action_ := "WireMoneyRequest",
                        relatesTo := omit
            },
            swiftNumber := "US3007",
            sourceAcc := "123456",
            targetAcc := "734456",
            amount := {fAmount := 2000, fCurrency := "Euro"}
    }
    template WireMoneyMsgType wireMsg601 := {
        wsaInfo := {
            messageID := "m-usatm-usbg-601",
            action_ := "WireMoneyRequest",
            relatesTo := ""
        },
        swiftNumber := "US3007",
        sourceAcc := "123456",
        targetAcc := "734456",
        amount := {fAmount := 2000, fCurrency := "Euro"}
    }
...

}
```

Conclusion

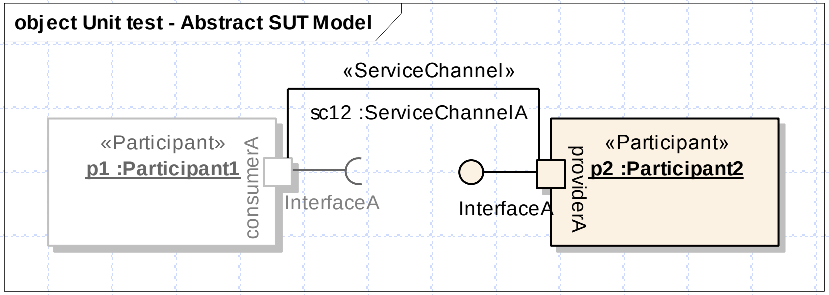
- We have presented our experience in designing and implementing the proposed TTCN4SOA™ architecture and framework in TTCN-3 standard and on top of the TTworkbench platform used as test execution environment
- BN4SAT outcomes constitute a background of the ongoing EC FP7 research project MIDAS (Model and Inference Driven Automated testing of Services architectures – project number: 318786)
- The MIDAS goal is the implementation of a test facility for services architectures that:
  - automates the test generation, scheduling, execution and arbitration for unit and integration (choreography) testing,
  - targets model-based functional, security and usage-based testing with different methods
  - is available on self-provisioning and pay-per-use basis as a TaaS (Test as a Service) running on a cloud infrastructure

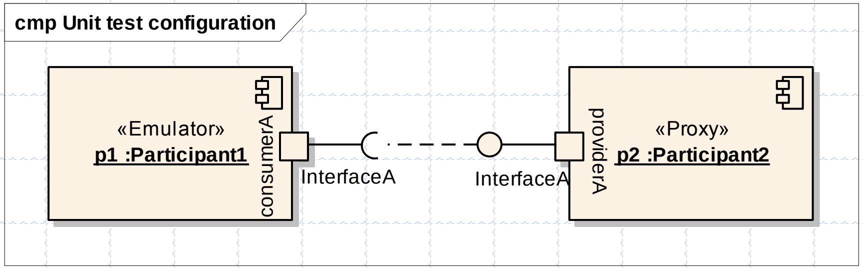
Thanks

Questions?

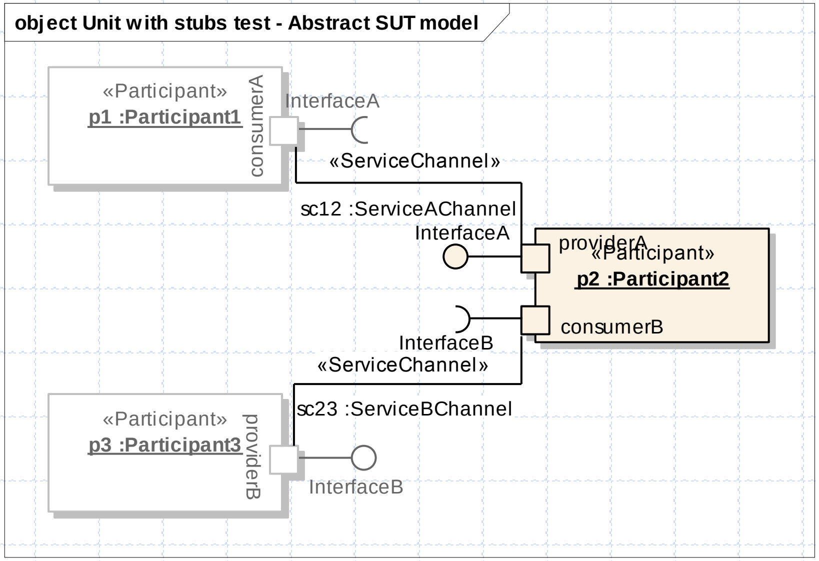
Templates

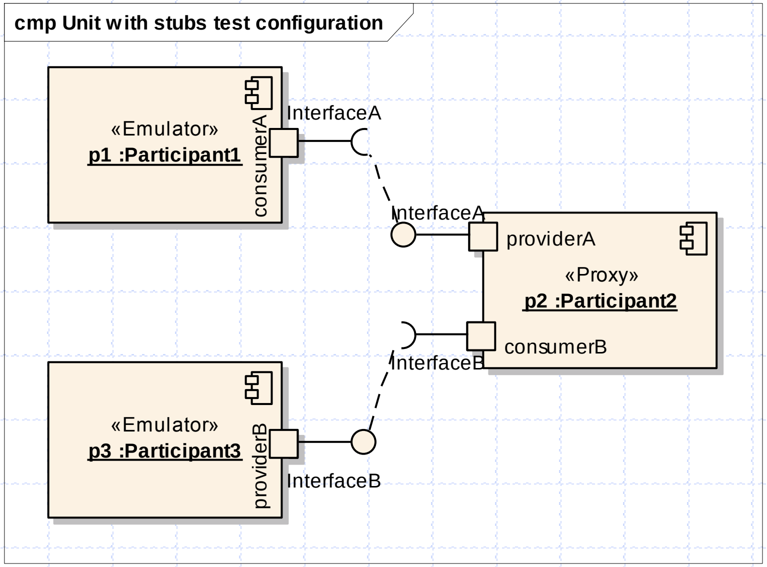
Unit test





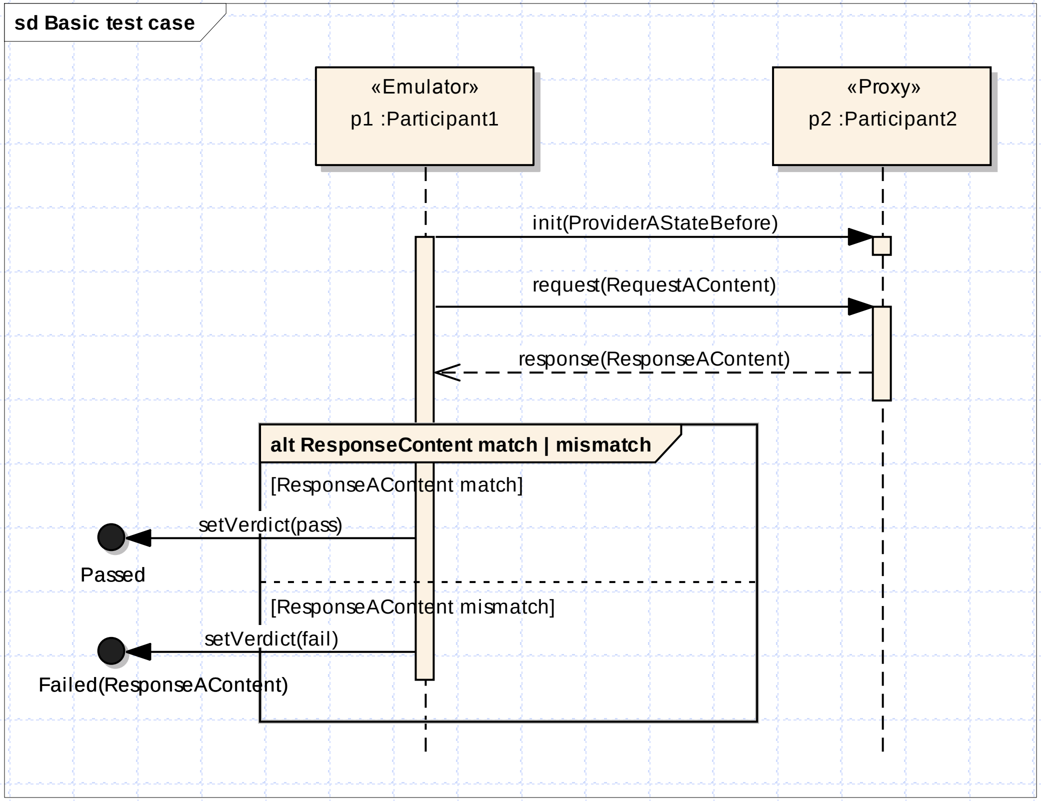
Unit test with stubs





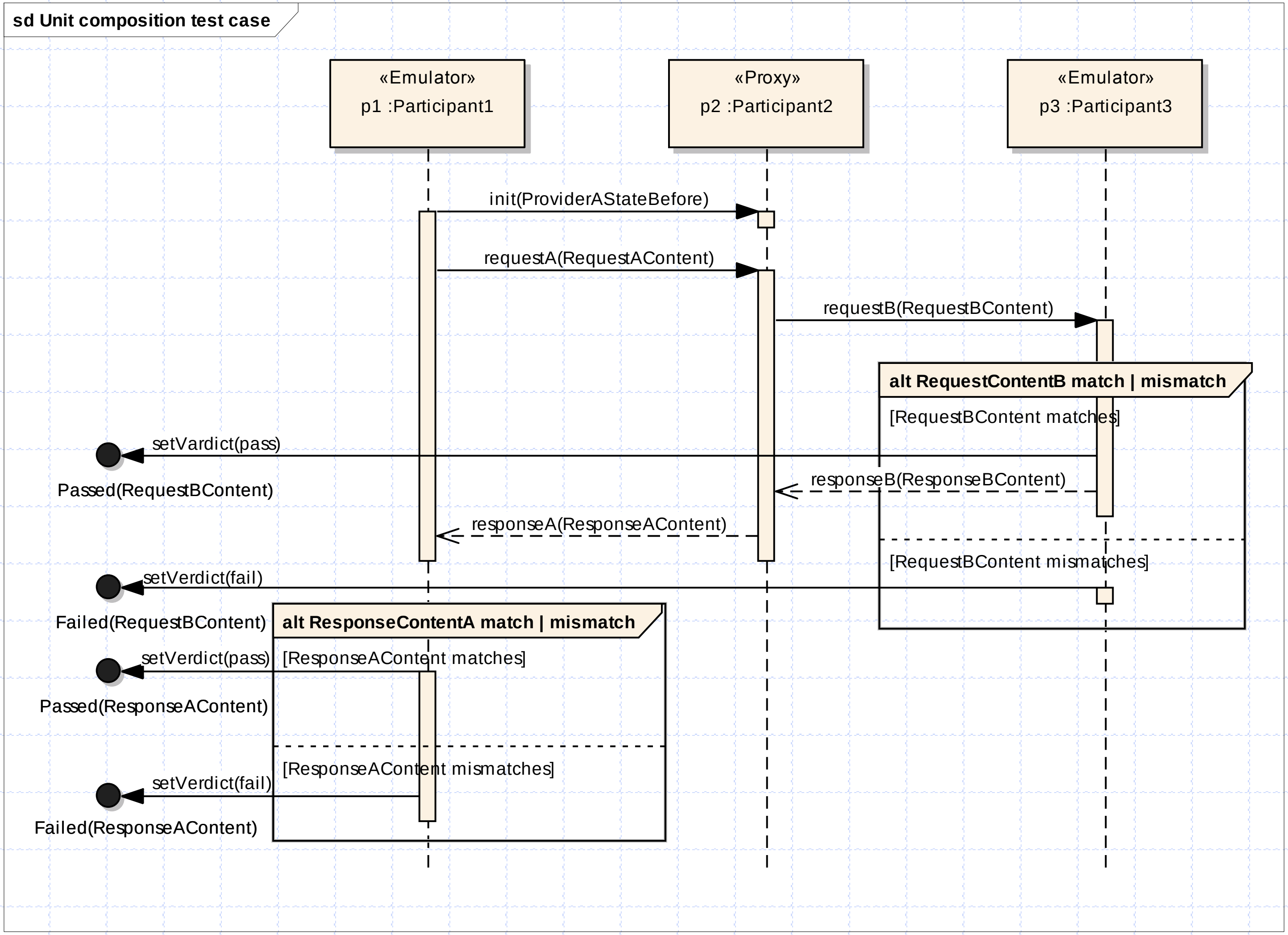
Basic test case pattern

Unit test



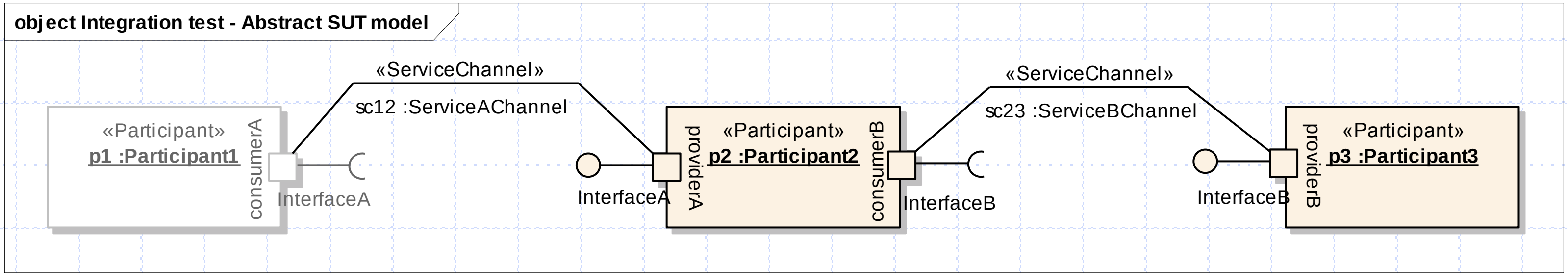
Basic test case pattern (II)

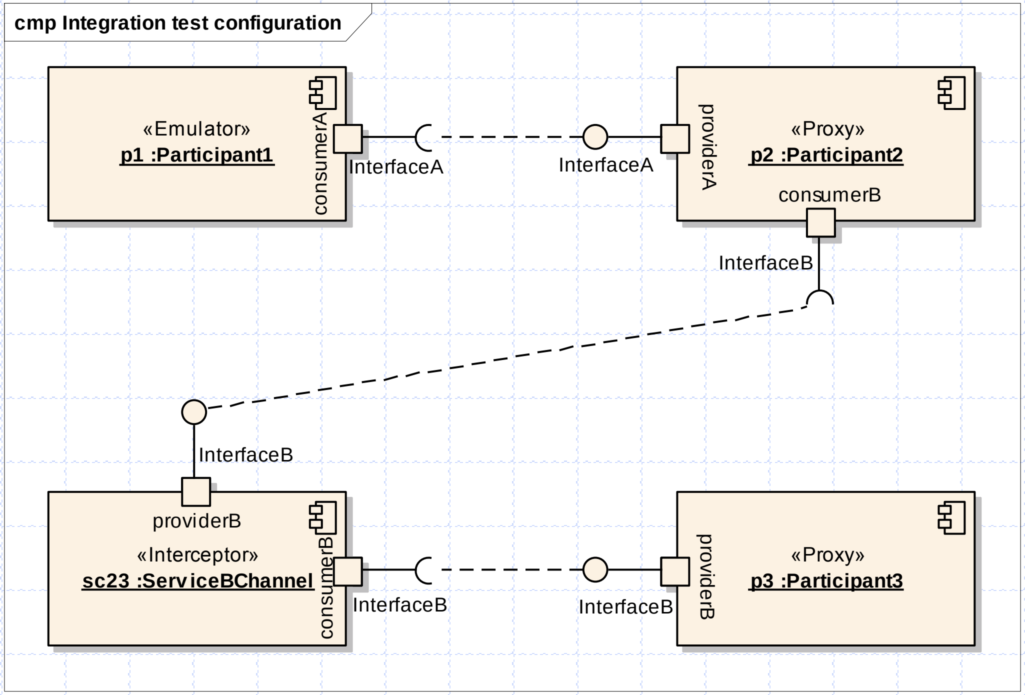
Unit test with stubs


sd Unit composition test case
«Emulator»
p1 :Participant1
«Proxy»
p2 :Participant2
«Emulator»
p3 :Participant3
init(ProviderAStateBefore)
requestA(RequestAContent)
requestB(RequestBContent)
alt RequestContentB match | mismatch
[RequestBContent matches]
setVardict(pass)
Passed(RequestBContent)
responseB(ResponseBContent)
responseA(ResponseAContent)
[RequestBContent mismatches]
setVerdict(fail)
Failed(RequestBContent)
alt ResponseContentA match | mismatch
setVerdict(pass)
[ResponseAContent matches]
Passed(ResponseAContent)
[ResponseAContent mismatches]
setVerdict(fail)
Failed(ResponseAContent)

Templates (II)

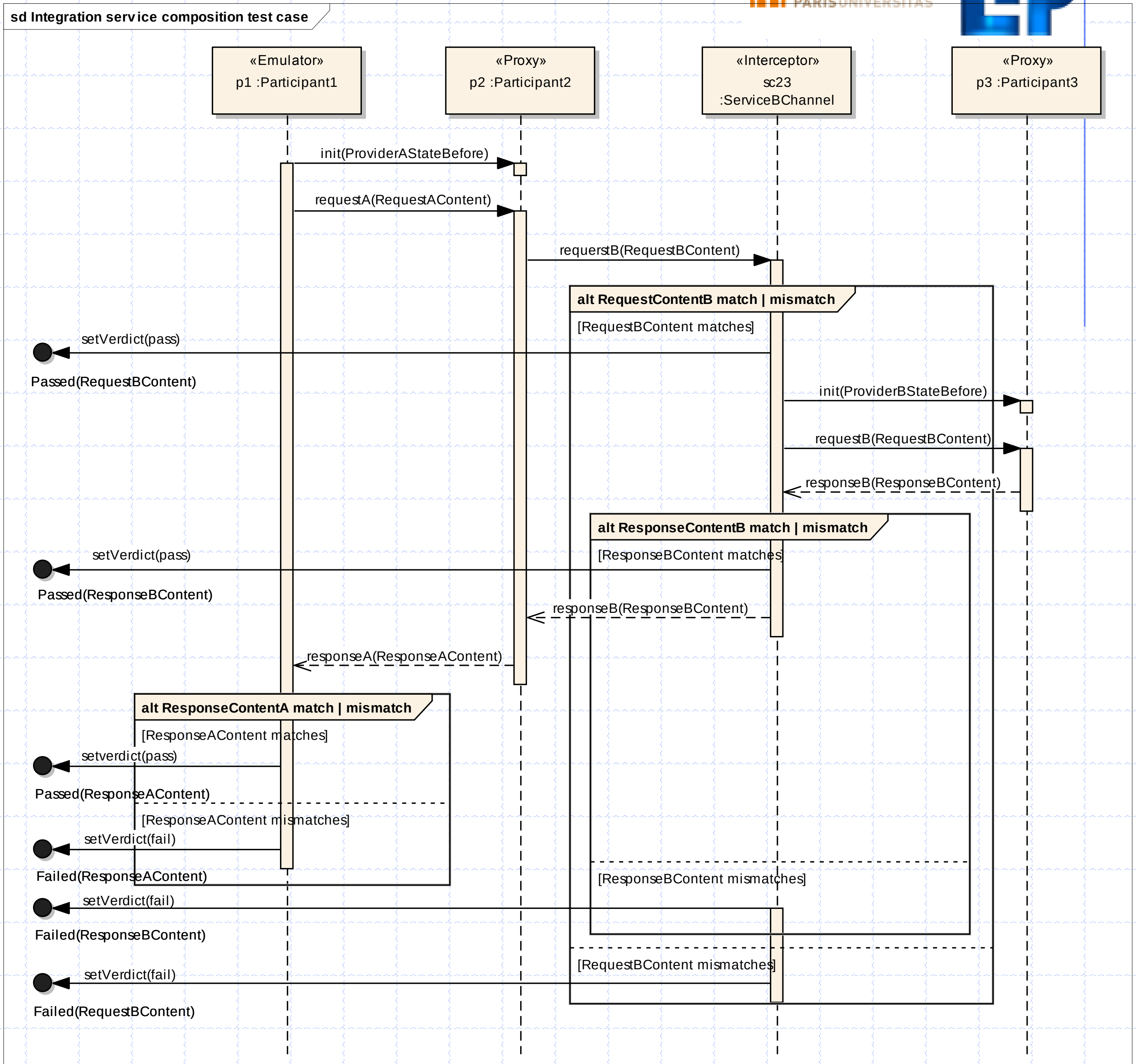
Integration test





Basic test case pattern (III)

Integration test


sd Integration service composition test case
«Emulator»
p1 :Participant1
«Proxy»
p2 :Participant2
«Interceptor»
sc23
:ServiceBChannel
«Proxy»
p3 :Participant3
init(ProviderAStateBefore)
requestA(RequestAContent)
requerstB(RequestBContent)
alt RequestContentB match | mismatch
[RequestBContent matches]
setVerdict(pass)
Passed(RequestBContent)
init(ProviderBStateBefore)
requestB(RequestBContent)
responseB(ResponseBContent)
alt ResponseContentB match | mismatch
[ResponseBContent matches]
setVerdict(pass)
Passed(ResponseBContent)
responseB(ResponseBContent)
responseA(ResponseAContent)
alt ResponseContentA match | mismatch
[ResponseAContent matches]
setverdict(pass)
Passed(ResponseAContent)
[ResponseAContent mismatches]
setVerdict(fail)
Failed(ResponseAContent)
[ResponseBContent mismatches]
setVerdict(fail)
Failed(ResponseBContent)
[RequestBContent mismatches]
setVerdict(fail)
Failed(RequestBContent)