T3UC 2011
Bled, Slovenia

New architecture for Integration of TTCN-3 concrete layer with frameworks

Bernard Stepien, Liam Peyton
School of Information Technology and Engineering

Ming Shang
Research in Motion ltd

uOttawa
L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

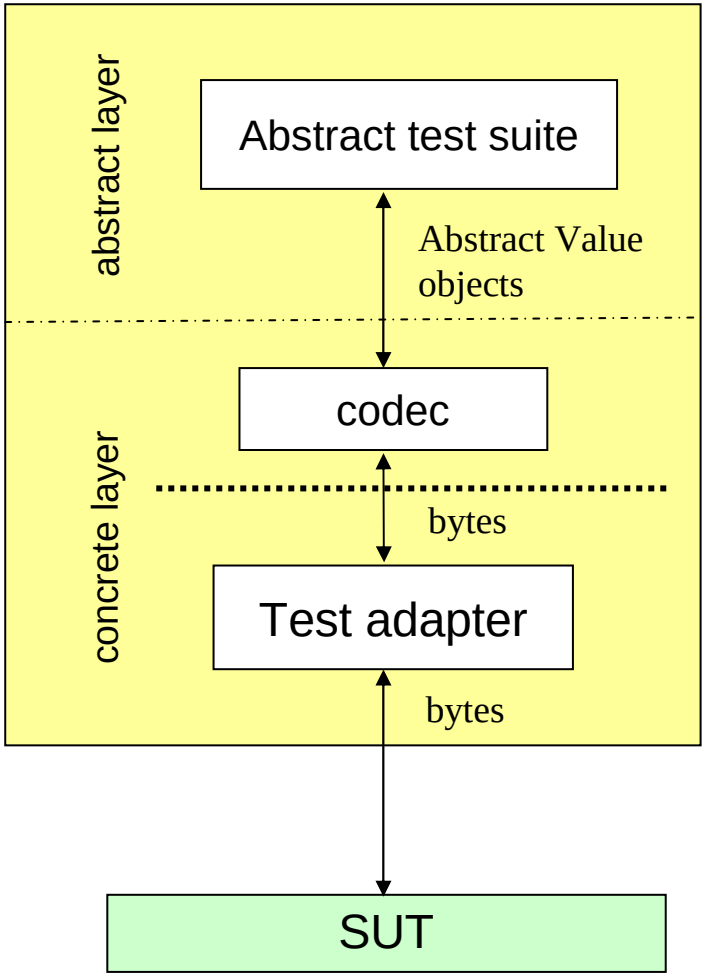
Introduction

- TTCN-3 is a powerful framework for defining and managing Web Application testing
- Web Application testing supported by concrete layer Unit Testing Frameworks (jUnit etc.)
- The implementation of TTCN-3 currently is not convenient or optimized for efficient integration with such frameworks
- We present a proposed architectural enhancements to TTCN-3 implementations, consistent with the TTCN-3 spec which address these shortcomings
- We present a series of examples to show the how to exploit this new architecture for improved web application testing

TTCN-3 Background

- TTCN was originally conceived for :
  - telecommunications applications
  - Implies sending and receiving compacted information as bytes.
- The architecture of TTCN-3 execution tools consists:
  - of a strict sequence of invoking coding or decoding methods of the implemented codec class to obtain bytes
  - Sending messages by invoking the *triSend* method of the implemented test adapter

Byte stream oriented architecture



- Multi-level separation of concerns
  - Abstract layer
  - Concrete layer
    - Codec
    - test adapter

Functionalities of codecs

Incoming messages

- Parsing incoming bytes to determine field values.
- Constructing an abstract TTCN-3 Value object using field values.

Bytes:

```
6162636 47B78797a
```

Abstract value

```
template MyType t_1 := {
   field_1 := “abcd”,
   field_2 := 123,
   field_3 := “xyz”
}
```

Functionalities of codecs

outgoing messages

- Extracting concrete values from TTCN-3 abstract Value instances.
- Encode messages as bytes using these concrete values.

Abstract value

```
template MyType t_1 := {
   field_1 := "abcd",
   field_2 := 123,
   field_3 := "xyz"
}
```

Bytes:

61626364 7B 78797a

Codec activity phases

- On data sent:
  - Extract concrete values from the abstract values
  - Encode outgoing byte oriented message
- On data received:
  - Extract concrete field values from byte oriented message.
  - Construct an abstract Value object from the concrete field values.

Functionality of test adapters

- Establish concrete communication with the SUT
- Send or receive **bytes** over the concrete communication
- Enqueue responses to the message queues
- Invoke methods on objects
- Commonly used methods:
  - triSend()
  - triEnqueueMsg()

Codec class methods characteristics

```
public TriMessage encode(Value value) {

}
```

```
public Value decode(TriMessage message,
        Type decodingHypothesis) {
```

- Parameters provide only
  - Abstract Values
  - Concrete messages
  - Expected abstract data type in the abstract receive statement.

Test adapter class characteristics

```
public TriStatus triSend(
   final TriComponentId componentId,
   final TriPortId tsiPortId,
   final TriAddress address,
   final TriMessage sendMessage) {
```

```
package org.etsi.ttcn.tri;

public interface TriMessage {
   public byte[] getEncodedMessage();

   public void setEncodedMessage(byte[] message);
   public int getNumberOfBits();
   public void setNumberOfBits(int amount);
   public boolean equals(TriMessage message);
}
```

- Parameters provide only:
  - Component id
  - port id
  - Address
  - Message to be sent as bytes
- TriMessage talks only about bytes

Functionality of test frameworks

- Parse messages
- Encode messages
- Send or receive messages over a communication channel
- Invoke methods with parameter values for performing communication operations.

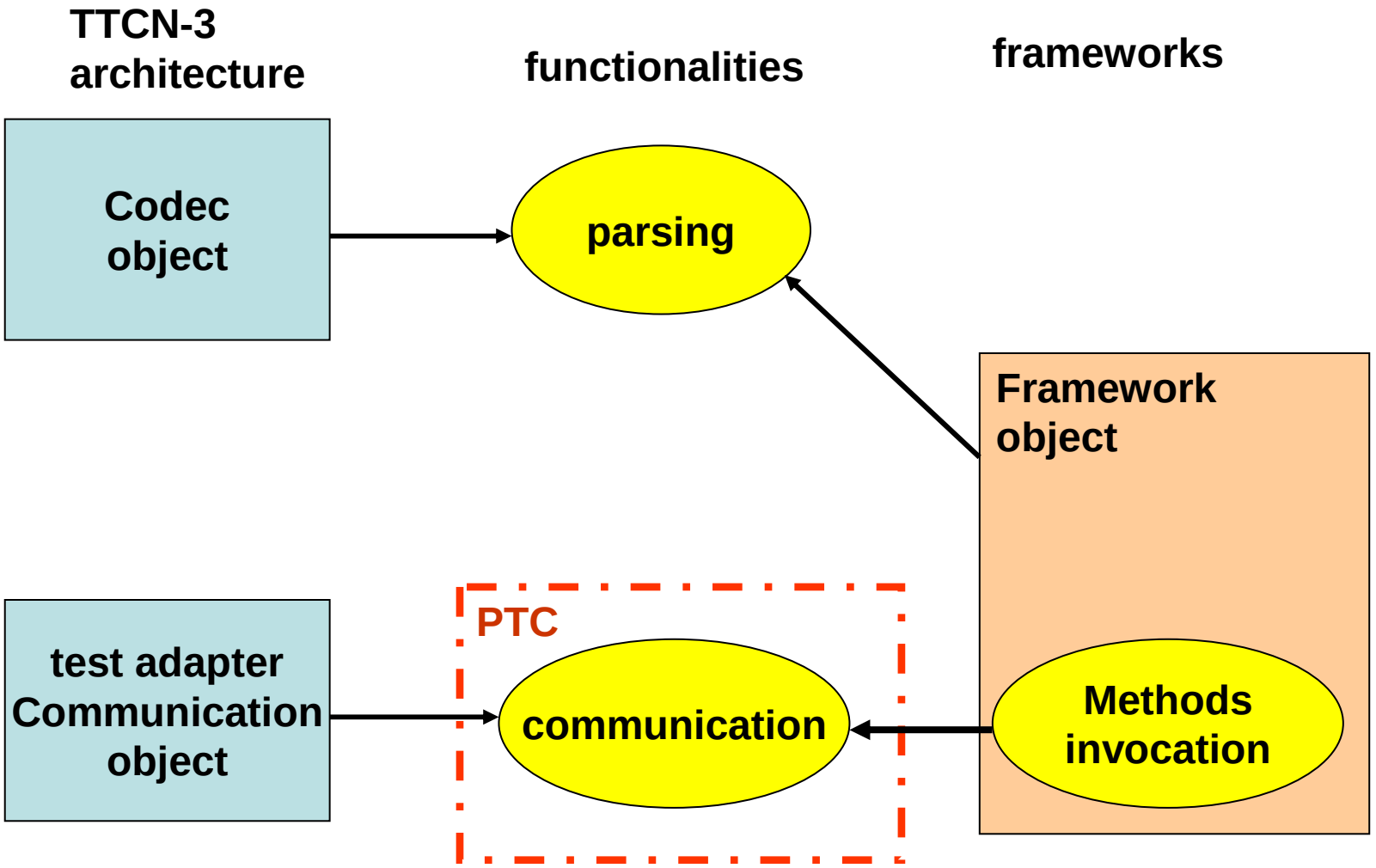
Characteristics of frameworks

- Object oriented
- A single object handles all functionalities
- No separation of functionalities using separate objects

Architectural clash
TTCN-3 vs frameworks

- A framework object must be associated with a parallel test component (PTC)
- Especially in multi-PTC instances contexts
- In TTCN-3, parsing functionalities and communication functionalities are separated in two different objects.
- But only the test adapter knows about PTCs
- In frameworks, most of the time parsing and communication are merged in a single object.

Separation of concerns clash



Frameworks integration problems

identified problems

- Framework methods require detailed individual concrete field values, not a single record of encoded bytes.
- TCI codec transforms detailed abstract values into a single byte stream.
- Framework objects are associated with PTCs.
- TCI codec has no indication of PTC and thus can not retrieve the corresponding framework object instance.

Frameworks integration problems

possible solutions

- Location of the framework object
  - In the codec?
  - In the test adapter?
- Decision factor: association of framework objects with parallel test components
- The TTCN-3 separation of concerns between codec and communication functionalities may not exist with framework objects.

Location of the framework object
in the codec

- Valid if only codec functionalities of the framework object are used
- The codec is not the right place for using communication functionalities of the framework object.
- The are no TTCN-3 standard defined ways to determine the PTC from the codec using standard defined methods (encode decode) which is important in the case of multiple PTCs.

Location of the framework object

in the test adapter

- Appropriate since the test adapter has a standard way to determine the PTC instance.
- TCI Codecs will keep the same structure.
- However the framework object needs to have access to the detailed concrete fields of the message.
- In some cases, codec and test adapter functionalities need to be mixed.

Possible approaches

- Message oriented testing
- Procedure oriented testing

Using message oriented testing

- Currently not efficient because of the byte stream bottleneck
- Requires double encoding/decoding phase

Using procedure oriented testing

Solution

- Invoke the frameworks methods using TTCN-3 procedure oriented testing

Problem

- What are you testing?
  - The SUT?
  - The framework?
- Violates the central TTCN-3 concept of independence of the abstract layer relative to the concrete layer.

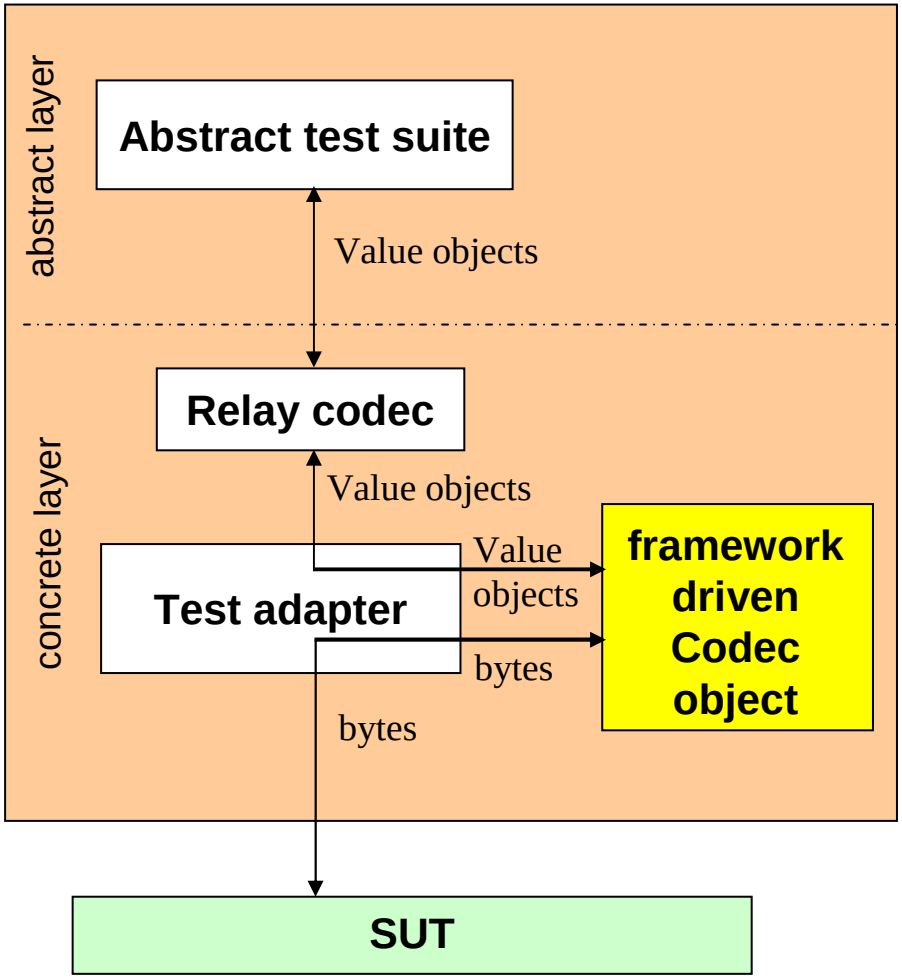
Solution

an extension to TRI/TCI

- Manage framework objects in the test adapter only
- Forward abstract data values directly to the test adapter when sending
- Build the abstract data values in the test adapter for receiving
- Forward abstract data values to and from the relay codec
- Avoid using tool vendor proprietary features

Our proposed TTCN-3 architecture

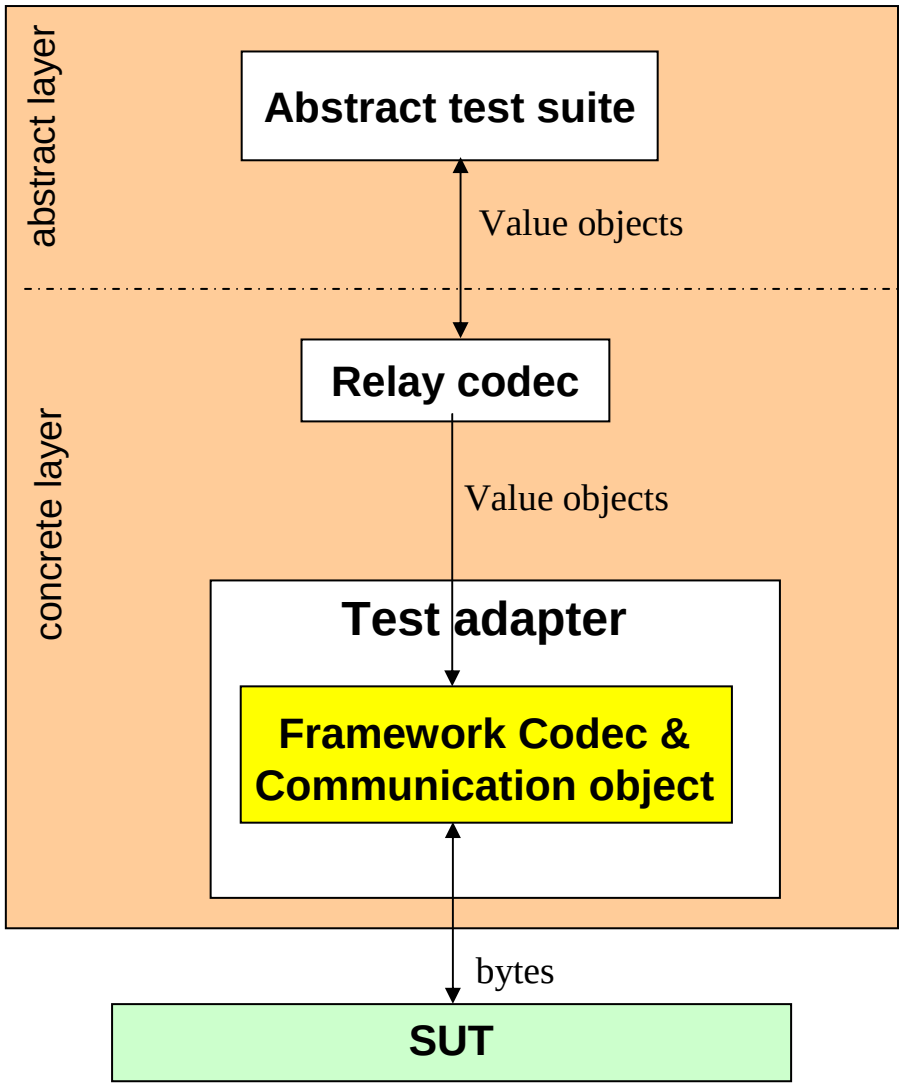
framework codec only



- When PTC association is necessary.

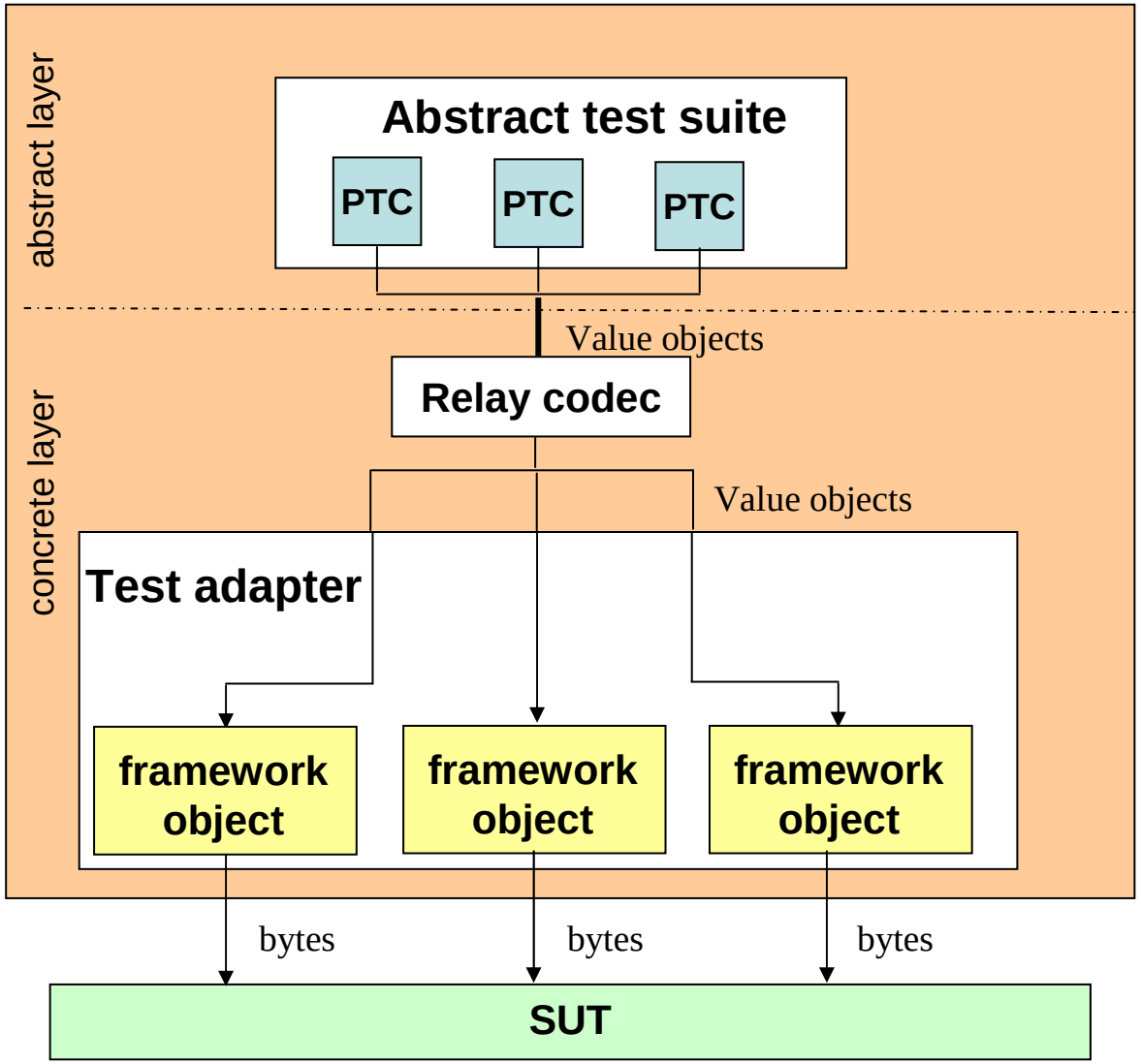
Our proposed TTCN-3 architecture

framework codec & communication



Our proposed TTCN-3 architecture

multiple PTCs configuration



Relationship to Standard and Tool Vendors

- The proposed implementation changes are consistent with TTCN-3 standard
- The standard specifies that the TriMessage implementation is left to the discretion of the tool implementers.
- Our model has been prototyped with one tool vendor
- Have been rolled out in a mid 2010 version

Solution implementation details

- Modify content of abstract class TriMessage by adding a TTCN-3 abstract Value attribute.
- Ensure the tool accepts this new TriMessage object.
- Move specific data type codec invocation to the test adapter
- Requires cooperation of the tool vendor

What happens to the traditional Tci codec?

- Traditional codec is maintained but now has two functionalities:
  - Handle traditional byte oriented messages.
  - Forward abstract Value to the test adapter for user selected data types.
- Asymmetric codecs are possible:
  - Decoding is handled in the test adapter
  - Encoding is handled in the traditional TCI codec
  - Abstract Value objects are constructed in the traditional TCI codec.

Modification of the TriMessage interface

```
package org.etsi.ttcn.tri;

public interface TriMessage {
   public byte[] getEncodedMessage();
   public Value getValue();

   public void setEncodedMessage(byte[] message);
   public void setValue(Value value);

   public int getNumberOfBits();
   public void setNumberOfBits(int amount);
   public boolean equals(TriMessage message);
}
```

- Add an attribute for Value objects
- Add corresponding getters and setters

Creating a relay codec
encode()

```
public TriMessage encode(Value value) {

    TciValueMessage vmsg = new TciValueMessage(value);

    return vmsg;
}
```

Creating a relay codec
decode()

```
public Value decode(TriMessage message, Type type) {

   Value returnedValue = null;

   if(message instanceof TciValueMessage) {

      TciValueMessage bimpl = (TciValueMessage) message;

      returnedValue = bimpl.getValue();
   }
   else {
            // handle byte oriented messages here
   }

   return returnedValue;
}
```

Handling Value objects in the test adapter

```
public TriStatus triSend(final TriComponentId componentId,
                         final TriPortId tsiPortId, final TriAddress address,
                         final TriMessage sendMessage) {

   Value tciValue = null;

   if (sendMessage instanceof LocalMessage) {
      tciValue = ((LocalMessage) sendMessage).getContent();

      if(tciValue.getType().getName().equals("MyType")) {

         // invoke the framework object method that performs codec
         // functionalities and the framework object's method invocation
      }
   }

…
}
```

Summary of workarounds

respecting compliance to the standard

- Serialization of either abstract values or concrete objects
  - Objects must be serializable
  - Impacts on performance
- Make framework objects available to the codec (at codec instantiation time)
  - No association possible with PTCs
- Encode in a temporary byte stream and decode it again in the test adapter to restore field values
  - Impacts on performance and doubles the amount of coding.
- Use procedure oriented testing
  - Dependence on framework in the abstract layer

Research approach

- This research was conducted in two steps
  - We created our own TriMessage implementation
  - We used our TriMessage Implementation through the direct invocation of the tool's enqueue() method
- In order to avoid problems with new tool releases, we asked the tool vendor to adopt our solution
- This resulted in a compromise
  - To ensure not violating the standard
  - To ensure compatibility with other tool features

Examples

- Web application testing example
- SIP in a concurrent context example

Example I:
Web application testing example
htmlUnit framework

HtmlUnit classes and methods

simulating a browser

- Class: **WebClient**
- Methods:
  - communication methods:
    - getHomePage()
    - getPage(URL)
    - getCache()
    - ...

- Class: **HtmlPage**
- Methods:
  - Parsing methods:
    - getBody()
    - getTitleText()
    - getForms()
    - getElementsByTageName()
- Class **HtmlForm**:
  - getButtonByName()
- Class **HtmlInput**:
  - Click()

Codec strategies

- One shot codec
- Decomposed codec

Codec strategies

one shot codec

- A given data type is processed entirely at once, i.e. all of its fields and sub-fields.
- This is appropriate only for byte stream oriented applications

Codec strategies

decomposed codec

- Is appropriate when the framework object has a different structure than the abstract data type.
- TTCN-3 recordValue fields values are used as parameter values for a method invocation:
  - At once through a single framework object method invocation.
  - In a staggered way via several different framework object method invocations.

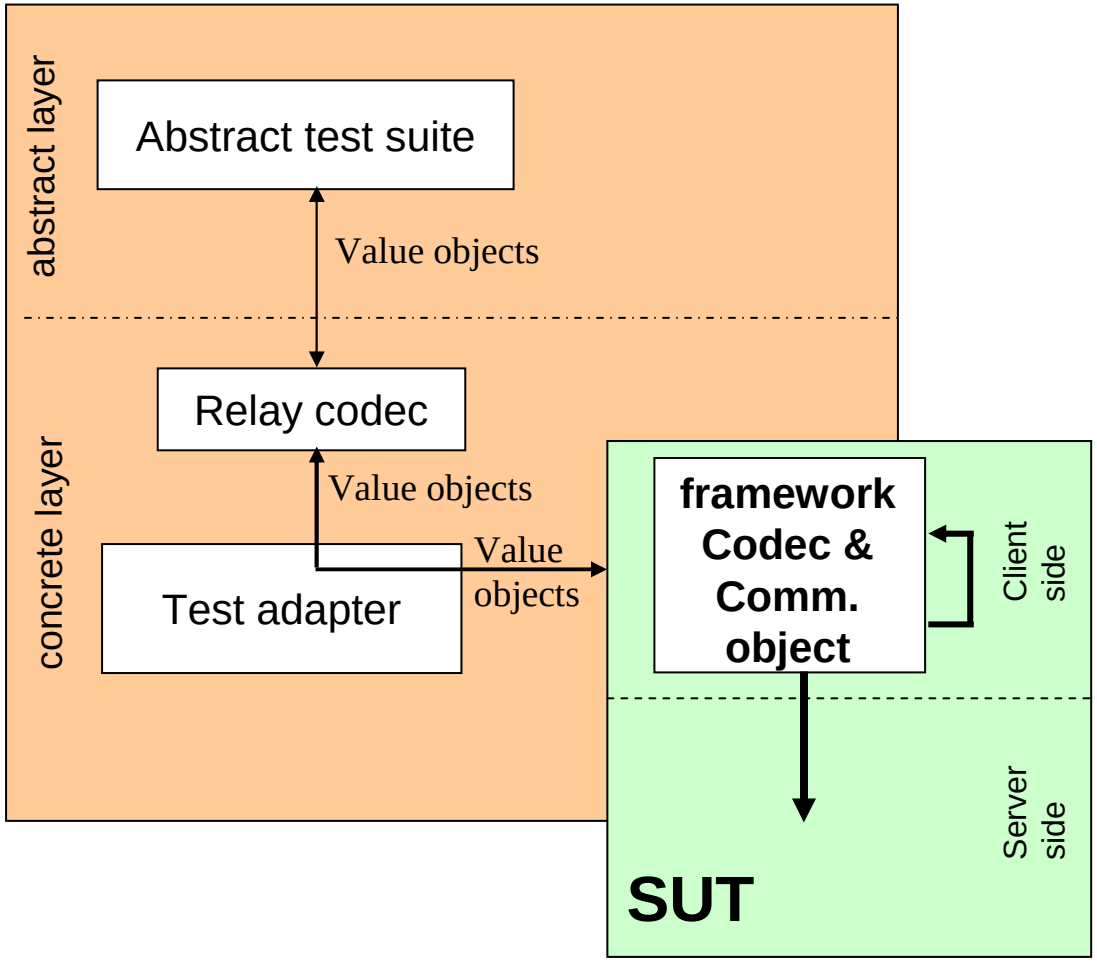
What is the SUT in web applications?

- Distinction between:
  - Server side
  - Client side
- Server side
  - The SUT is the web site
- Client side
  - The SUT is the framework object located in the test adapter itself that executes javascript methods
  - There are no messages going anywhere

Client side operations

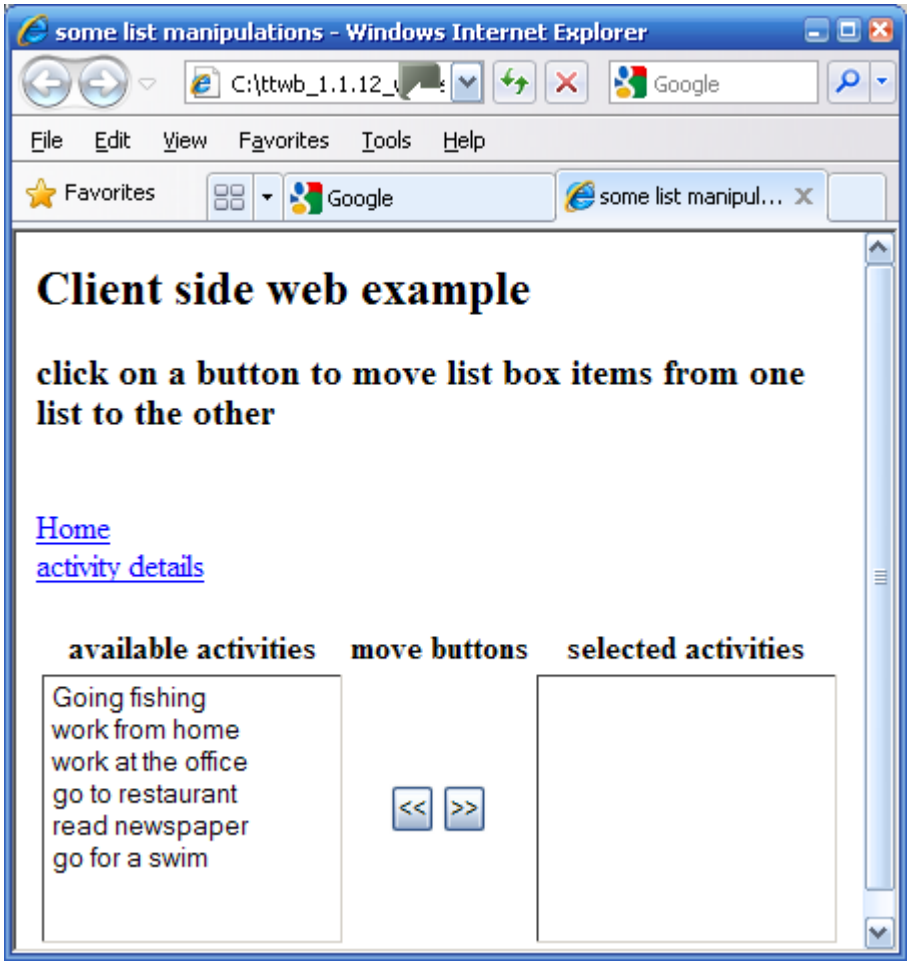
- There are no messages sent to the server
- Processing is performed inside the framework object (client side)

Client side computing configuration



Example client side testing

forms with list boxes



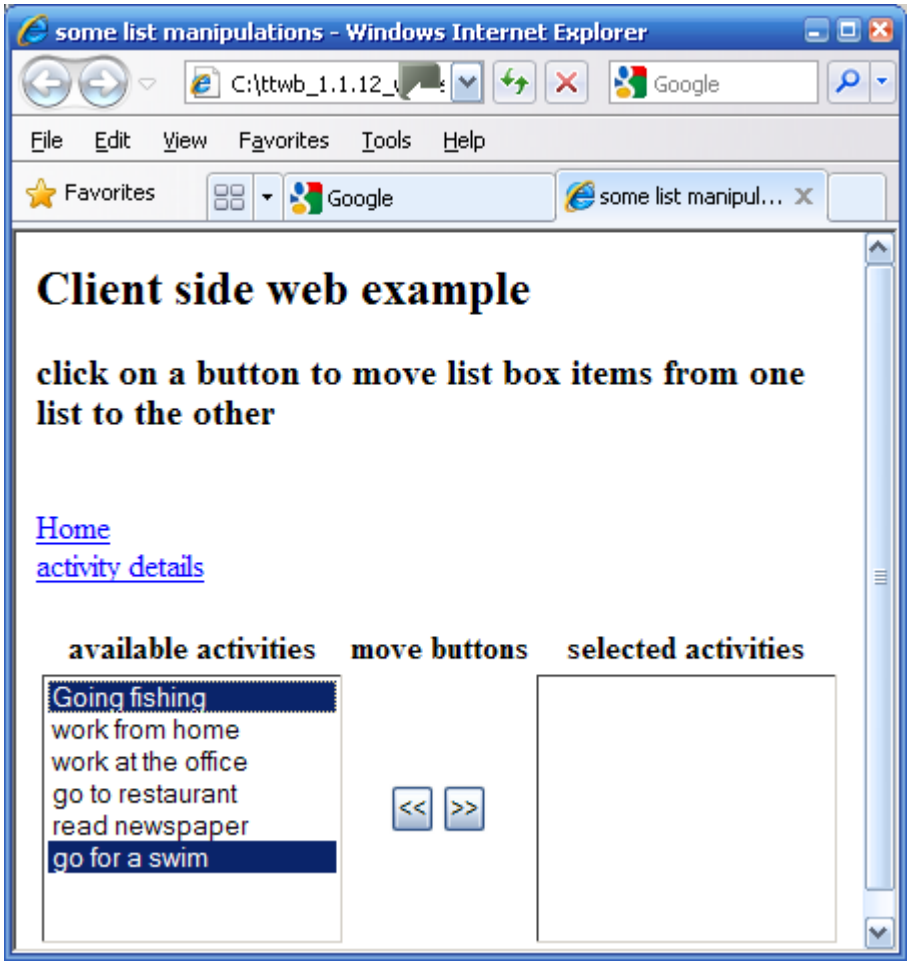
- Items are moved from one box to another using javascript
- Test that when left box item is selected and right arrow is clicked, the item is:
  - Redisplayed in the right box
  - Removed from the left box

HTML code

```
<html>
  <head>
    <title>some list manipulations</title>
      <script language="javascript">
        …
      </script>
  </head>
  <body>
    <h2>Client side web example</h2>
    <h3>click on a button to move list box items from one list to the other</h3> <br>
    <A href="home.html">Home</A> <br>
    <A href="activity_details.html">activity details</A>

    <form name="act_sel_form" method="post" onsubmit="javascript:document.act_sel_form.save.disabled=true;"
                  action="/webadmin/en/telco/jsp/ProcessAddUser.jsp">
      <table>
        <tr>
          <th>available activities</th>                    <th>selection buttons</th>  <th>selected activities</th>
        </tr>
        <tr>
          <td>
            <SELECT name="list1" size="8" multiple class="selList" tabindex="9" style="width:150px">
              <OPTION VALUE="50">Going fishing</OPTION>
              <OPTION VALUE="2">work from home</OPTION>
              <OPTION VALUE="3">work at the office</OPTION>
              <OPTION VALUE="5">go to restaurant</OPTION>
              <OPTION VALUE="1">read newspaper</OPTION>
              <OPTION VALUE="4">go for a swim</OPTION>
            </SELECT>
          </td>
          <td>
            <INPUT type="button" tabindex="10" onClick="move(document.act_sel_form.list2,document.act_sel_form.list1)"
                                                                    value="<<" name="button4" class="button">
            <INPUT type="button" tabindex="11" onClick="move(document.act_sel_form.list1,document.act_sel_form.list2)"
                                                                    value=">>" name="button42" class="button">
          </td>
          <td>
            <SELECT NAME="list2" size="8" multiple tabindex="12" class="selList" style="width:150px“/>

          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

forms with list boxes

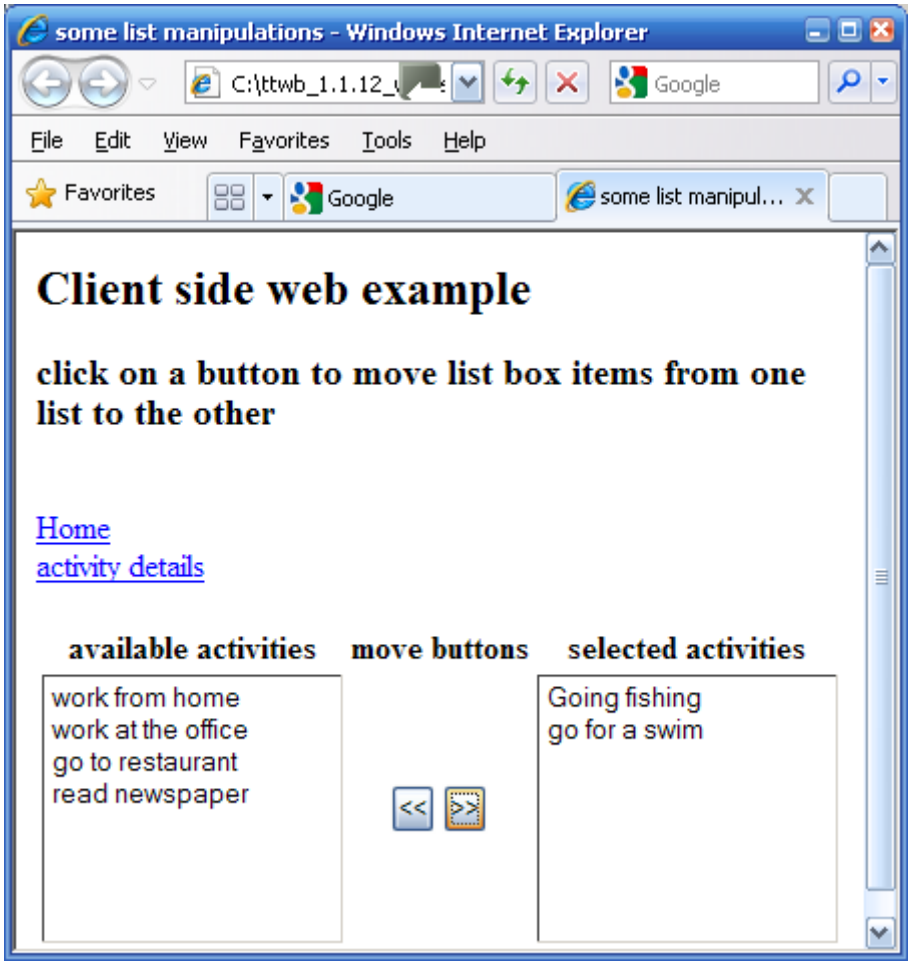
item selection test



- Two items are selected
- WebClient object invokes methods to set the items to “selected”
- The response HTML should show that the attribute “selected” is equal to “true”

forms with list boxes

button click for item move test



- After clicking the ">>" button, the two selected items should have been moved to the right hand list box

Forms with list boxes

abstract layer data type definitions

```
type record formSelectOptionType {
        charstring optionValue,
        boolean selected
}

type set of formSelectOptionType formSelectOptionSetType;

type union formElementKind {
    formInputValueType input,
    formSelectOptionSetType options
}

type record formElementType {
    charstring name,
    formElementKind formElement
}

type set of formElementType formElementSetType;
```

```
type record FormType {
    charstring name,
    charstring formAction,
    charstring kindMethod,
    formElementSetType elements
}
```

Forms with list boxes

initial state template definitions

```
 template formSelectOptionSetType list1_options_before := {
        { optionValue := "Going fishing", selected := false},
        { optionValue := "work from home", selected := false},
        { optionValue := "work at the office", selected := false},
        { optionValue := "go to restaurant", selected := false},
        { optionValue := "read newspaper", selected := false},
        { optionValue := "go for a swim", selected := false}
}

template formElementSetType beforeSelectFormElements := {
        { name := "list1", formElement := {options := list1_options_before} },
        { name := "button4", formElement := {input := "<<"}},
        { name := "button42", formElement := {input := ">>"}},
        { name := "list2", formElement := {options := {}} }
}

template FormType initialFormContent := {
        name := "act_sel_form",
        formAction := "/webadmin/en/telco/jsp/ProcessAddUser.jsp",
        kindMethod := "post",
        elements := beforeSelectFormElements
}
```

Forms with list boxes

response templates definitions

```
template formSelectOptionSetType list1_options_after_1 := {
        { optionValue := "Going fishing", selected := false},
        { optionValue := "work from home", selected := false},
        { optionValue := "work at the office", selected := false},
        { optionValue := "go to restaurant", selected := false},
        { optionValue := "read newspaper", selected := false},
        { optionValue := "go for a swim", selected := true}
}
```

After selecting: “go for a swim”

```
template formSelectOptionSetType list1_options_after_2 := {
        { optionValue := "Going fishing", selected := true},
        { optionValue := "work from home", selected := false},
        { optionValue := "work at the office", selected := false},
        { optionValue := "go to restaurant", selected := false},
        { optionValue := "read newspaper", selected := false},
        { optionValue := "go for a swim", selected := true}
}
```

After selecting: “going fishing”

```
template formSelectOptionSetType list1_after_move_right := {
        { optionValue := "work from home", selected := false},
        { optionValue := "work at the office", selected := false},
        { optionValue := "go to restaurant", selected := false},
        { optionValue := "read newspaper", selected := false}
    }
```

After clicking on The “>>” button

Forms with list boxes

request templates definitions

```
template selectOptionRequestType selectRequestTemplate_1 := {
        formName :=  "act_sel_form",
        listBoxName := "list1",
        optionName := "go for a swim"
}

template selectOptionRequestType selectRequestTemplate_2
                                        modifies selectRequestTemplate_1 := {
        optionName := "Going fishing"
}
```

Forms with list boxes

behaviors definitions

```
function ListboxManipulationTest() runs on ptcType {

        web_port.send(theListBoxExamplePage);
        alt {
           [] web_port.receive(listboxPageResponse) -> value theCurrentResponse {
              log("got the list manipulation page OK");

              SelectOptionTest(selectRequestTemplate_1, selectResponseTemplate_1);

              SelectOptionTest(selectRequestTemplate_2, selectResponseTemplate_2);

              ClickMoveButtonTest()
           }
           [] web_port.receive(anyResponse) {
              setverdict(fail);
              stop
           }
        }
}
```

Forms with list boxes

behaviors definitions

a generic behavior function

```
function SelectOptionTest(selectOptionRequestType theRequest,
                selectOptionResultType theResponse) runs on ptcType {

      web_port.send(theRequest);
      alt {
         [] web_port.receive(theResponse) {
            log("received the correct select option state");
            setverdict(pass)
           }
         [] web_port.receive {
            log("did not received the correct select option state");
            setverdict(fail);
            stop
           }
      }
}
```

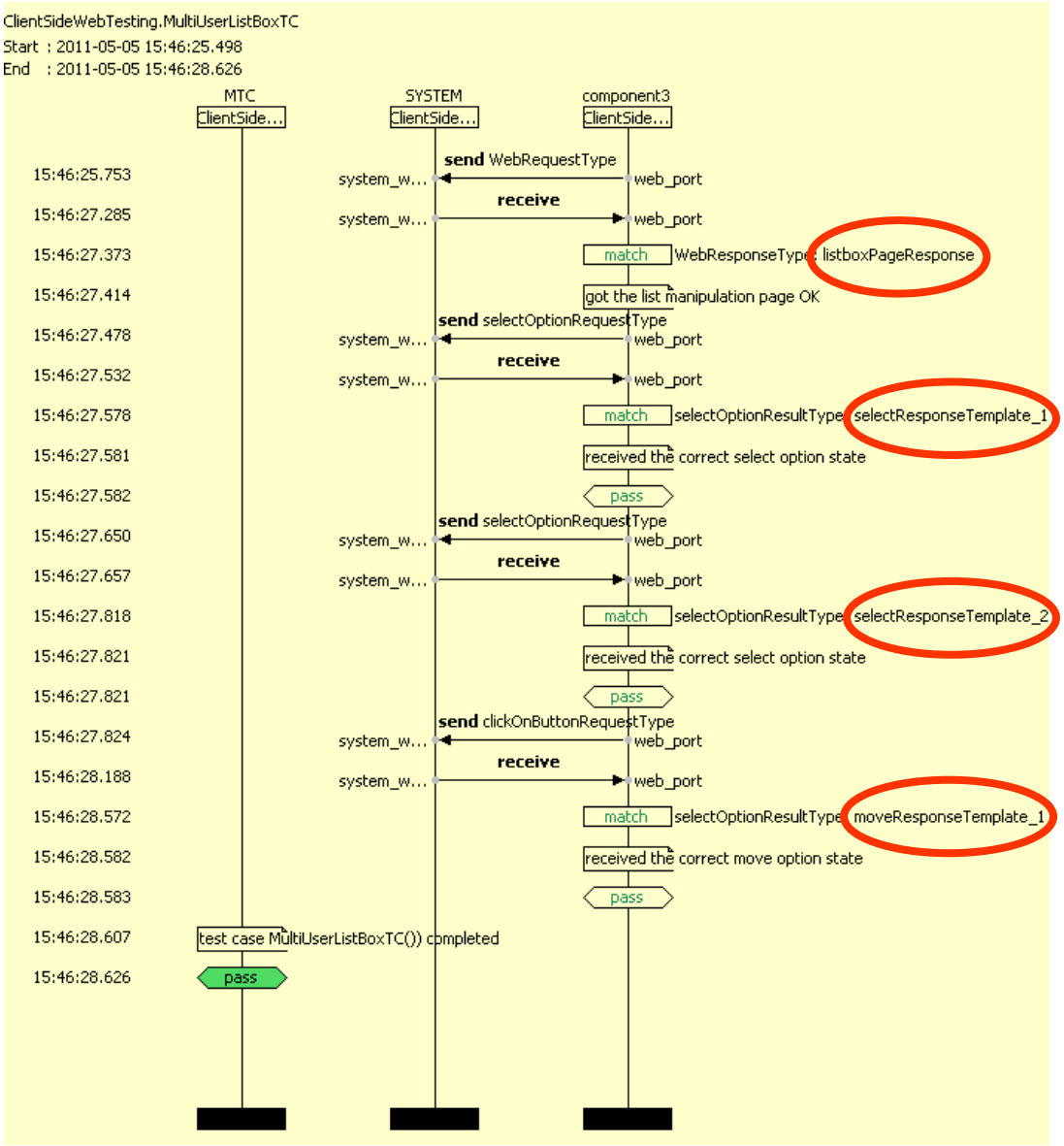
Forms with list boxes

behaviors definitions

```
function ClickMoveButtonTest() runs on ptcType {

        web_port.send(moveRequestTemplate_1);
        alt {
            [] web_port.receive(moveResponseTemplate_1) {
                log("received the correct move option state");
                setverdict(pass)
               }
            [] web_port.receive {
                log("did not received the correct move option state");
                setverdict(fail);
                stop
               }
        }
}
```

Forms with list boxes

test execution results



Forms with list boxes

codec encode – bytes approach

```
public TriMessage encode(Value value) {
   …
   if(msgType.getTypeClass() == TciTypeClass.RECORD) {
      String theRecordName = value.getType().getName();

      if(theRecordName.equals("WebRequestType")) {
         theWebMsg = Encode_WebRequestType((RecordValue) value);
      }
      else if(theRecordName.equals("selectOptionRequestType")) {
         theWebMsg = Encode_SelectOptionRequestType((RecordValue) value);
      }
      else if(theRecordName.equals("clickOnButtonRequestType")) {
         theWebMsg = Encode_ClickOnButtonRequestType((RecordValue) value);
      }
   }
   …
}
```

Forms with list boxes

codec Encode_SelectOptionRequestType

- No other choice than create a string and concatenate the parameter values

```
public String Encode_SelectOptionRequestType(RecordValue value)
  {
  String theSendCommand = "select option ";

  CharstringValue theFormNameValue = (CharstringValue) value.getField("formName");
  CharstringValue theListboxNameValue = (CharstringValue) value.getField("listBoxName");
  CharstringValue theOptionNameValue = (CharstringValue) value.getField("optionName");

  String theFormNameString = theFormNameValue.getString();
  String theListboxNameString = theListboxNameValue.getString();
  String theOptionNameString = theOptionNameValue.getString();

  theSendCommand += "#"+theFormNameString+"#"
                    +theListboxNameString+"#"+theOptionNameString+"#";

  return theSendCommand;
  }
```

Forms with list boxes

Test Adapter framework PTC object mapping

```
public TriStatus triSend(TriComponentId componentId,
                TriPortId tsiPortId, TriAddress address, TriMessage sendMesg) {

   byte [] mesg = sendMessage.getEncodedMessage();
   String theSendRequest = new String(mesg);

   if(theSendRequest.indexOf("URL") > -1) {
      String theUrlStr = substr(theSendRequest, 4);

      final WebClient webClient = new WebClient();
      webClients.put(componentId.getComponentId(), webClient);
      try {
        final URL url = new URL(theUrlStr);
        theCurrentPage = (HtmlPage) webClient.getPage(url);

        String response = theCurrentPage.getWebResponse().getContentAsString();
        TriMessageImpl rcvMessage = new TriMessageImpl(response.getBytes());

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

Forms with list boxes

Test Adapter handling selections

byte stream approach

```
  if(theSendRequest.indexOf("select option") > -1) {

     // parsing of the received string
     StringTokenizer t = new StringTokenizer(theSendRequest ,"#");

    String formName = t.nextToken();
    String listboxName = t.nextToken();
    String optionName = t.nextToken();

    selectFormOption(theCurrentPage, formName, listboxName, optionName);

     TriMessageImpl rcvMessage = new TriMessageImpl(theMessageDisp.getBytes());
     myCte.triEnqueueMsg(tsiPortId, new TriAddressImpl( new byte[] {}),
                                                          componentId, rcvMessage);
  }
}
```

Forms with list boxes

Test Adapter handling selections

Value objects approach

```
if(TciValue.getType.getName.equals("selectOptionRequestType")) {

   RecordValue formSubmitValue = (RecordValue) TciValue ;

   // parsing of the received string
   StringTokenizer t = new StringTokenizer(theSendRequest ,"#");

  String formName = formSubmitValue.getField("formName").getString();
  String listboxName = formSubmitValue.getField("listBoxName").getString();
  String optionName = formSubmitValue.getField("optionName").getString()

  selectFormOption(theCurrentPage, formName, listboxName, optionName);

   TriMessageImpl rcvMessage = new TriMessageImpl(theMessageDisp.getBytes());
   myCte.triEnqueueMsg(tsiPortId, new TriAddressImpl( new byte[] {}),
                                                    componentId, rcvMessage);
  }
}
```

Advantages of using Value objects in the test adapter

- Avoids an extra encoding and decoding of framework method parameter values
- This is particularly important when the submitted form is large

Client side operations

```
 static void selectFormOption(HtmlPage page, String theFormName,
                                          String theListBoxName, String theOptionName) {
    final HtmlForm theForm = page.getFormByName(theFormName);

    final HtmlSelect theSelect = findSelect(theForm, theListBoxName);

    final HtmlOption theOption =  findOption(theSelect, theOptionName);

    theOption.setAttributeValue("selected", "yes");
}
```

WebClient object

maintaining a state

- The HtmlPage object theCurrentPage goes through different states:
  - First item set to “selected”
  - Second item set to “selected”
  - Two items moved from the left list box to the right list box
- Thus, three separate abstract send have change the state of the same framework object (theCurrentPage).

Complex form submission example

Testing the submission of a tax return

- It is a form with a large number of input fields of all kinds (input, check boxes, etc...)
- Using the encode to create a byte oriented buffer is unpractical for the following reasons:
  - Impacts performance due to double coding/decoding
  - Can be quite complex due to dependencies to do flattening of hierarchical types

Workarounds to share WebClient objects

- Instantiate the codec with the WebClient instance
- What about the current page?
  - It can only be set in the test adapter upon invoking a url (a communication operation)
  - It has to be linked to a PTC
  - The Codec doesn’t know PTCs
- Create a Web application dedicated codec
- This would be a duplication of efforts. The HtmlUnit framework does everything already

Example II: JAIN SIP framework

Problem statement

- A third party product communicates with a RIM product via SIP messages over a single communication channel.
- The third party product is emulated using TTCN-3
- The emulator uses multiple PTCs to simulate called parties.
- Incoming messages need to be dispatched among PTCs from the test adapter
- The key to the dispatching mechanism is in a message concrete field value.
- Thus, the incoming messages need to be parsed in the test adapter

Dispatching SIP Message to PTC

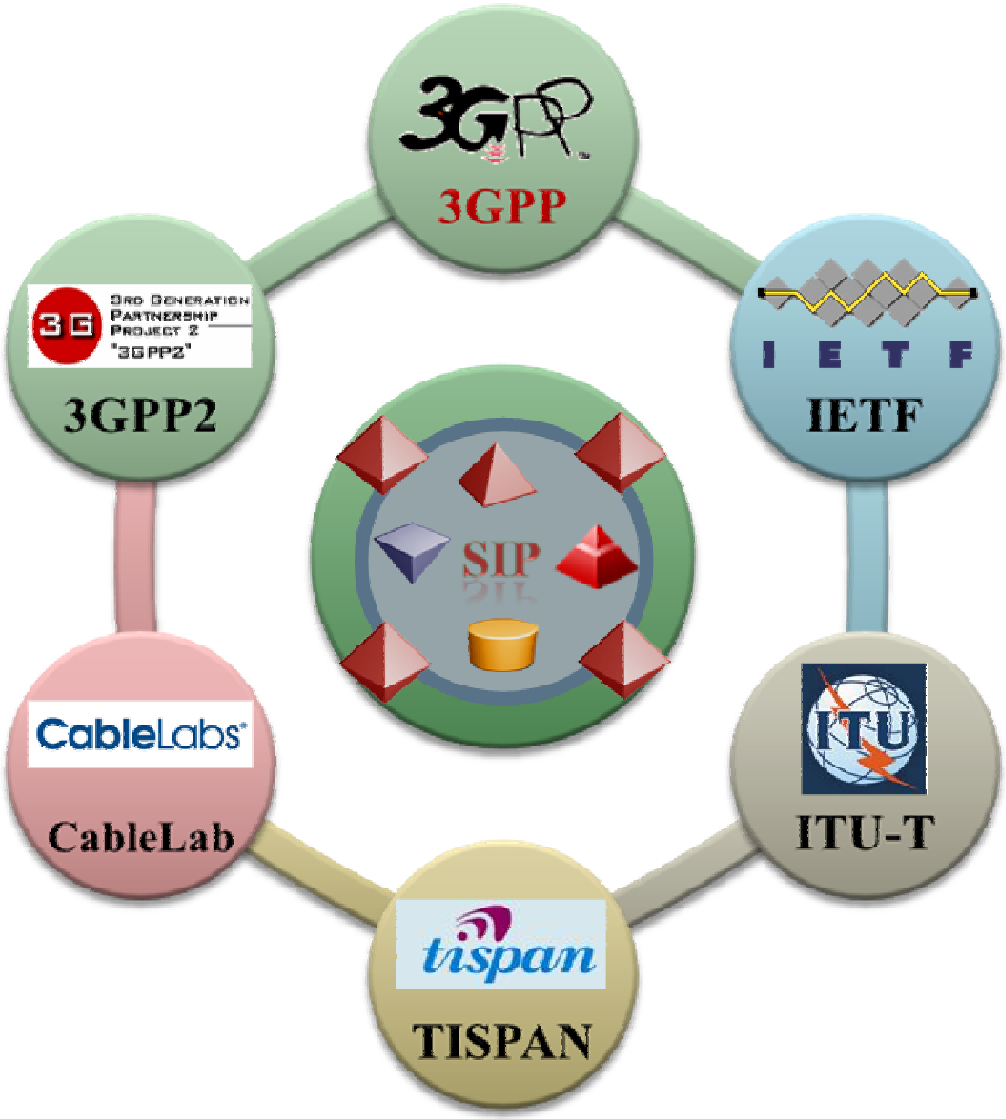
- To solve this multiple PTCs problem, incoming SIP message needs to be identified and then sent to certain PTC (e.g. SIP messages being dispatched according to the SIP callid).
- **The identification process has to be executed at Test adapter stage to determine to which PTC this SIP message belongs to**.
- To avoid double parsing (at both codec and test adapter) problem, we propose here that parsed SIP message object should be carried by the TciValueMessage and sent to codec for further processing.

SIP

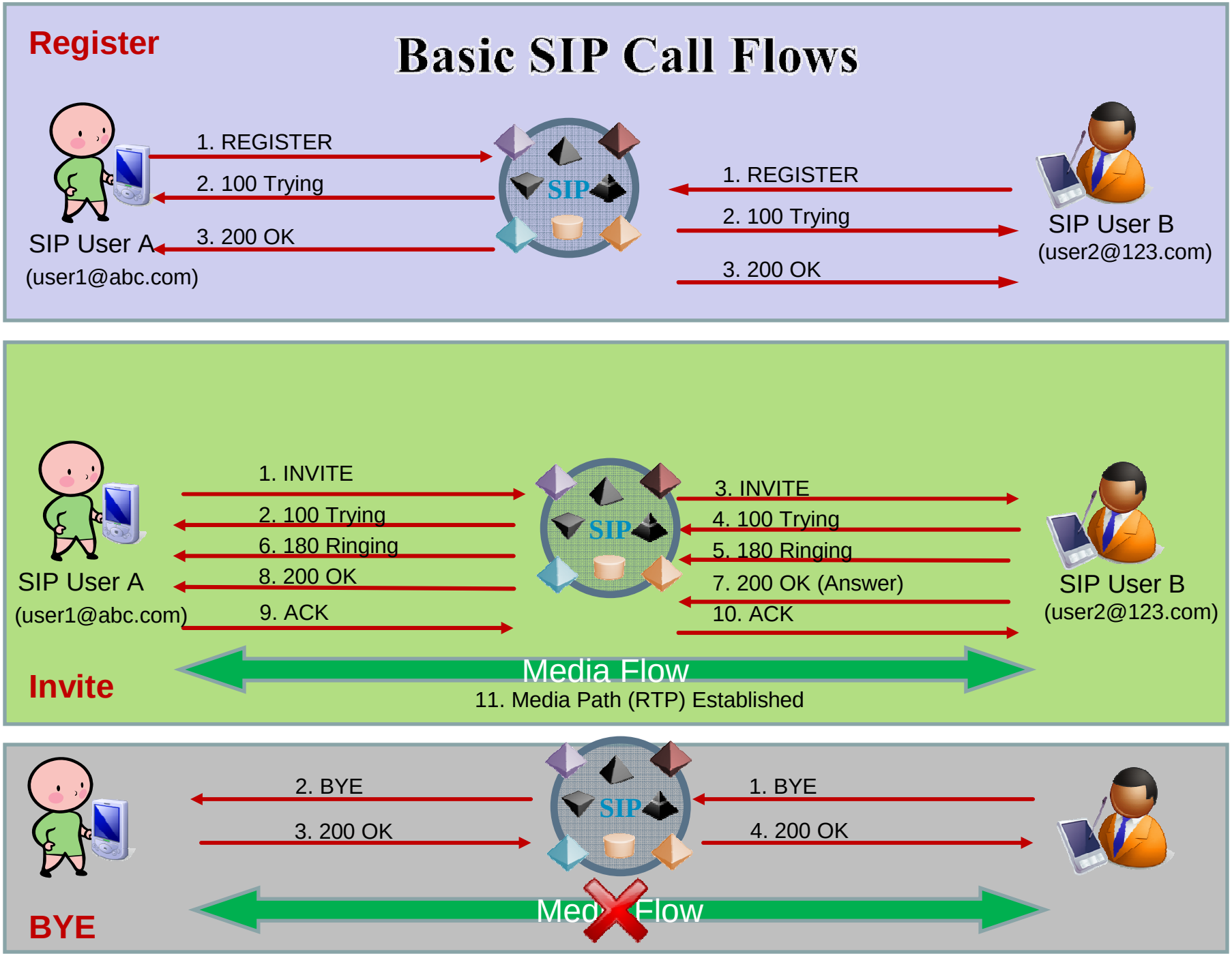
- **SIP** stands for **S**ession **I**nitiation **P**rotocol.
- **SIP** is an application-layer session control (signaling) protocol based on plain-text.
- **SIP is** a signaling protocol for creating, modifying and destroying dialogs between multiple endpoints.
- **SIP** is defined in RFC 3261 and has a set of extensions based on this core.

Related standardization bodies

SIP is an IETF specification that has been adopted by the communications industry in the form of 3GPP, 3GPP2, CableLabs and ITU



Basic SIP Call Flows



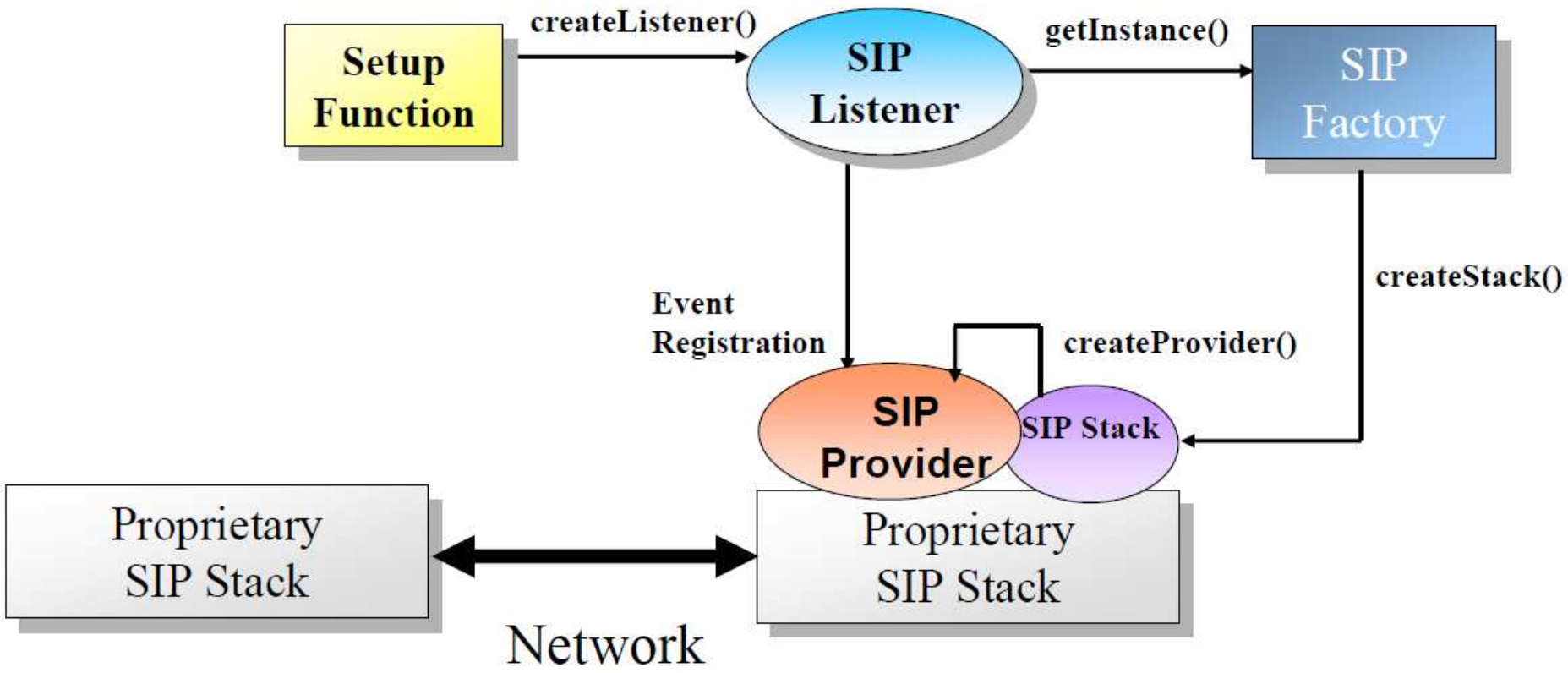
JAIN SIP framework

- JAIN stands for Java APIs for Integrated Networks
- Sun’s initiative to provide a set of API specification for various telephony (voice and data) services.
- JAIN SIP is one of these activities to provide SIP protocol support using Java

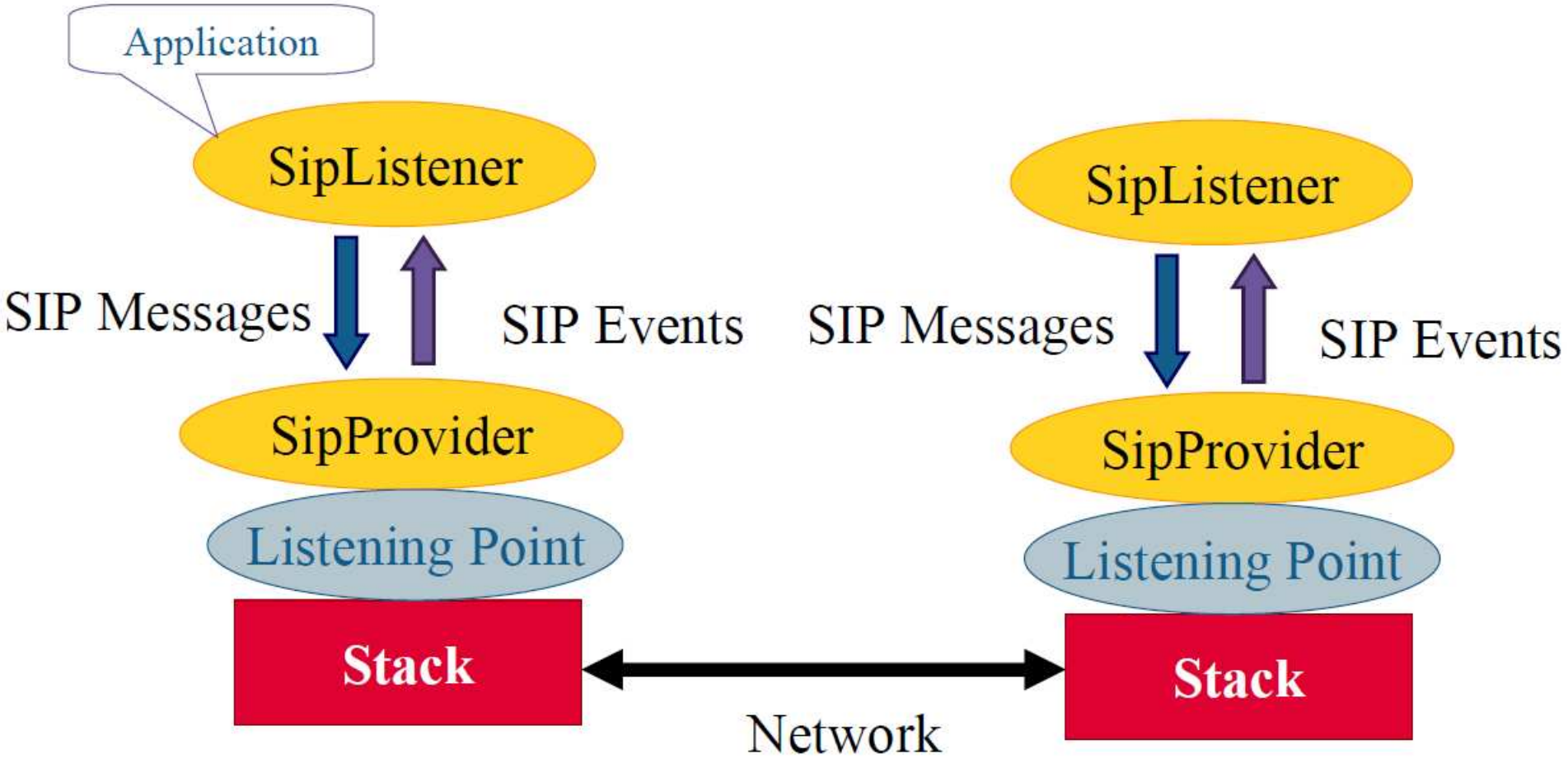
JAIN SIP framework (2)

- JAIN SIP provides specification only
- The National Institute of Standards and Technology (NIST) provides an implementation of the JAIN-SIP which serves as a reference implementation (RI).

JAIN SIP Object Architecture



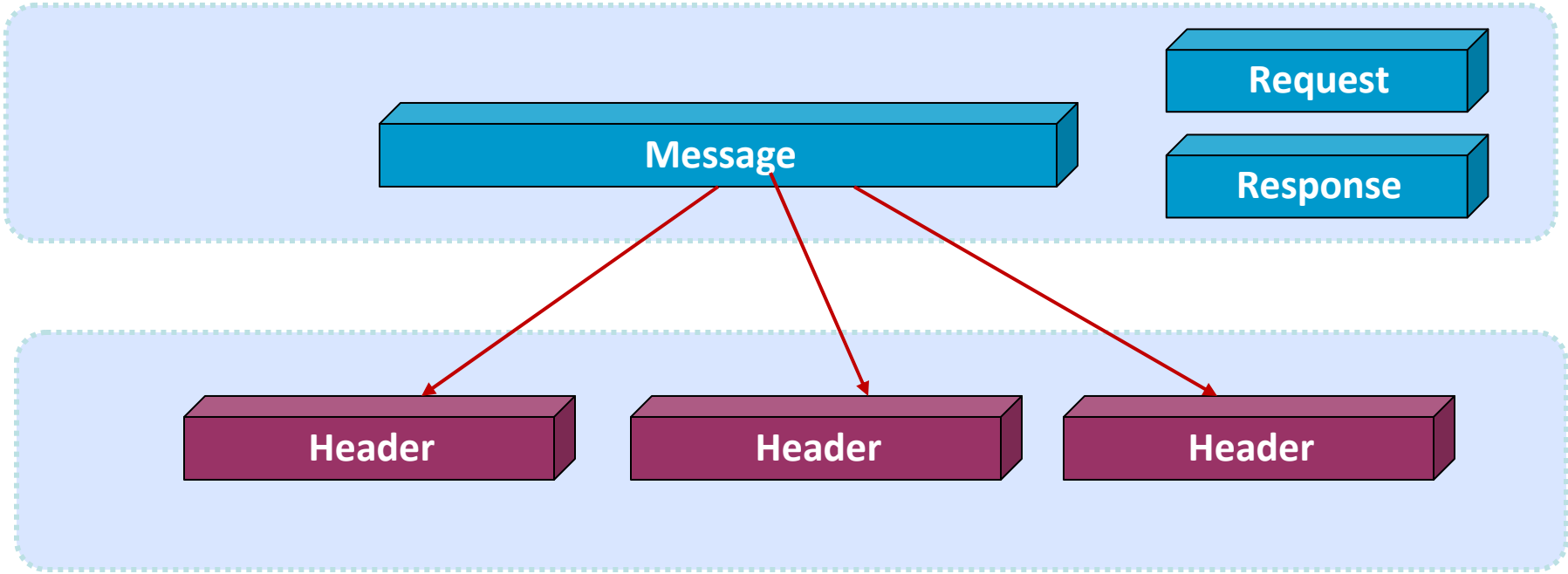
JAIN SIP Message Flow



JAIN SIP Message Object Family

- There are two type of messages in SIP, which JAIN SIP defines as interfaces: **Request** and **Response**
- **Request** messages are sent from the client to server, which contain a specific method type that identifies the type of Request.
- **Response** messages are sent from server to client in response to a Request, which contain a specific status code that identifies the type of Response
- Messages may also contain multiple **Headers** of the same type

JAIN SIP Message Object Family (2)



NIST SIP Usage in TTCN-3

- NIST SIP is a fully-implemented SIP stack instead of SIP codec, which means it maintains the state machine for each SIP session.
- TTCN-3 requires that each individual incoming/outgoing SIP message must be controlled by the abstract test case instead of the SIP stack.

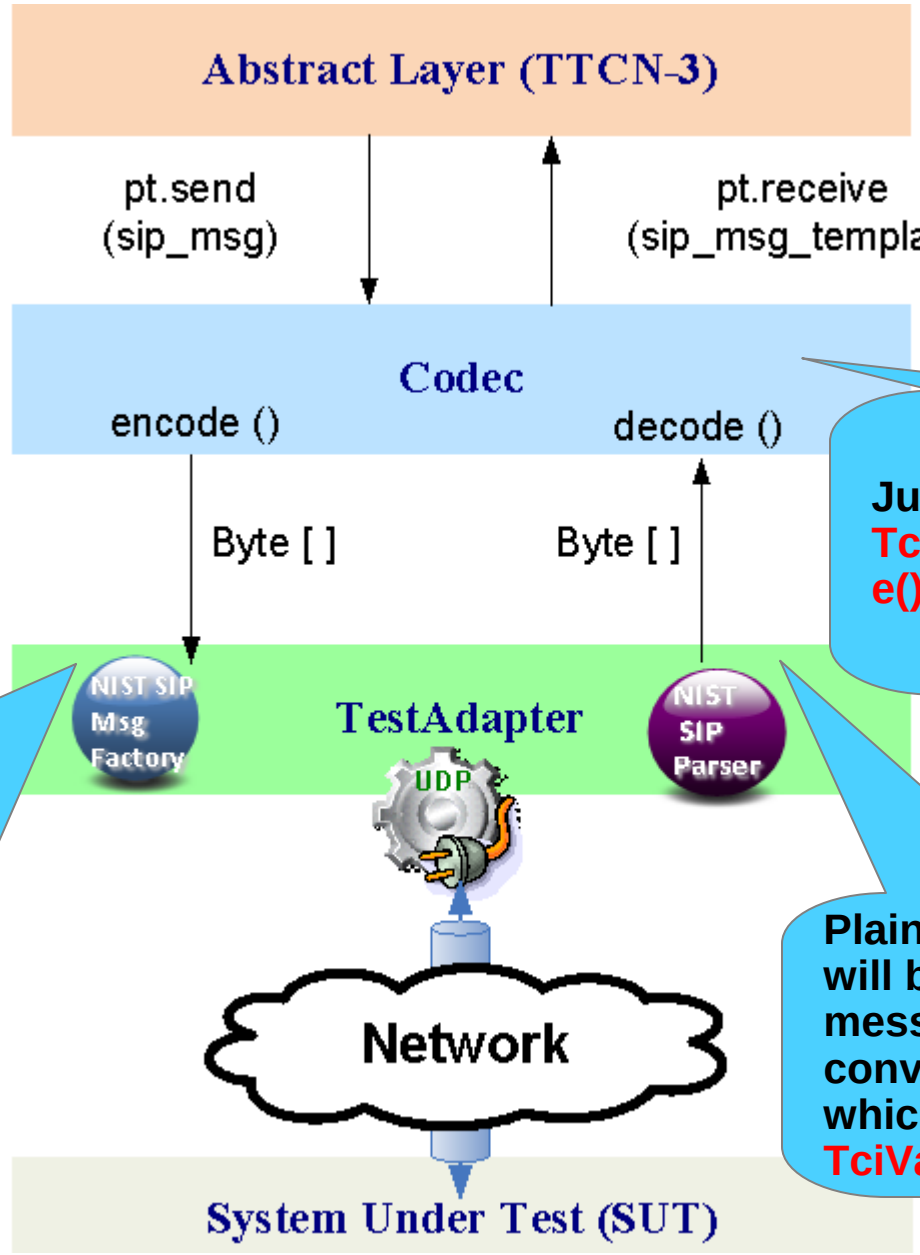
NIST SIP Usage in TTCN-3 (2)

- Instead of using NIST SIP stack, we only use the SIP message parsing functionality provided by NIST SIP.
- A UDP or TCP based test adapter has to be created to send/receive SIP message.

NIST SIP Usage in TTCN-3 (3)

```
//at test adapter

//NIST SIP Parser
private final StringMsgParser parser = new StringMsgParser();

public gov.nist.javax.sip.message.SIPMessage recvAndParse(DatagramChannel sc)
{

    ByteBuffer recvBuffer = ByteBuffer.allocate(8096);
    //receive the sip incoming message from the UDP socket
    sc.receive(recvBuffer);

    //prepare the byte stream for parser to parse next step
    recvBuffer.flip();

    //parse from the SIP byte stream message to SIPMessage object
    SIPMessage message = parser.parseSIPMessage(recvBuffer.array());

    Return message;
}
```

NIST SIP Usage in TTCN-3 (4) - message flows



TTCN-3 SIP Type system will be translated to NIST SIP message object using NIST SIP message factory based on the Value object carried by TciValueMessage

Just simply return TciValueMessage.getValue() to get the Value object.

Plain SIP message string will be parsed to SIP message object, and then converted to Value object which will be carried by TciValueMessage.

How to get Type info at TestAdapter

- There is no Type object available at TestAdapter, so the previous solution relies on the SIP message ←→ Type mapping beforehand.
- In the following example, hard coded approach is being employed. In the real situation, more complex configuration mechanisms could be applied like configuration file

How to get Type info at TestAdapter (2)

```
//at test adapter
public Value convert (gov.nist.javax.sip.message.SIPMessage message) {

   if(message instanceof Request) { //sip request message

       Type type = RB.getTciCDRequired().getTypeForName(
            "SIP_Types.Request");
        …
   }
   else if (message instanceof Response) { //sip response message
        Type type = RB.getTciCDRequired().getTypeForName(
            "SIP_Types.Response");
         …
   }

   return returnedValue;
}
```

Alternative approach

codec phases splitting

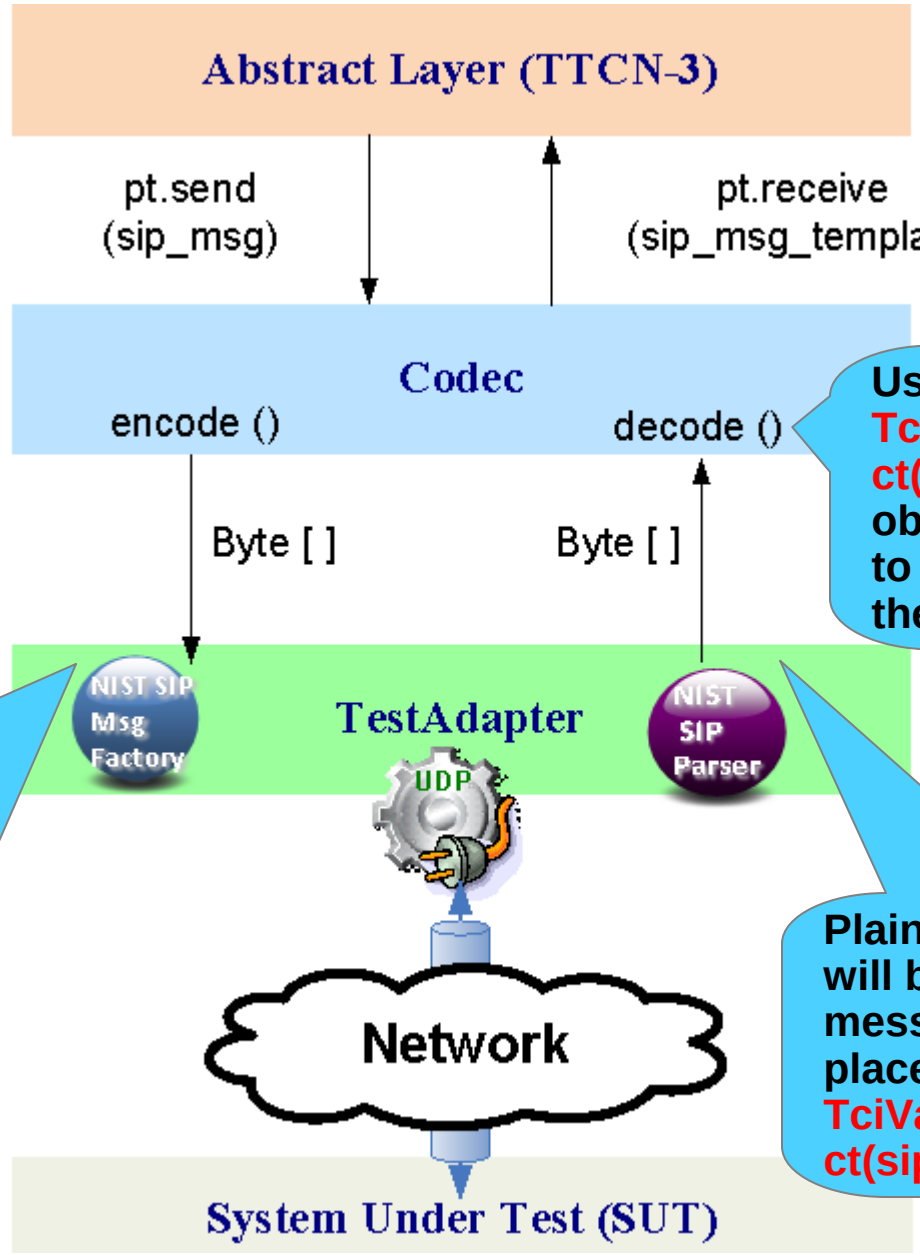
- Use JAIN-SIP framework to decode SIP message and construct a JAIN-SIP object in the test adapter
- The JAIN-SIP object is used to perform dispatching to the relevant PTC (test adapter)
- Forward the JAIN-SIP object to the traditional codec
- The traditional codec then only constructs the abstract Value object using the already decoded message contained in the JAIN-SIP object.

Alternative approach

- It is also possible that the SIP object → Value object converting process takes place at the codec stage.
- In this case, we propose that the TciValueMessage interface to be modified to enable carrying extra Object information

```
package org.etsi.ttcn.tri;

public interface TciValueMessage extends TriMessage{
   public byte[] getEncodedMessage();
   public Value getValue();
   public Object getObject();

   public void setEncodedMessage(byte[] message);
   public void setValue(Value value);
   public void setObject(Object object);

   public int getNumberOfBits();
   public void setNumberOfBits(int amount);
   public boolean equals(TriMessage message);
}
```

Alternative approach



TTCN-3 SIP Type system will be translated to NIST SIP message object using NIST SIP message factory based on the Value object carried by TciValueMessage

Using TciValueMessage.getObject() to get the SIPMessage object, and then convert to Value object based on the Type object.

Plain SIP message string will be parsed to SIP message object, and then place this object using TciValueMessage.setObject(sipMessage).

Alternative Approach - TestAdapter

```
//at test adapter

public TciValueMessage convert(DatagramChannel sc) {
    //parse from the SIP byte stream message to SIPMessage object
    SIPMessage message = recvAndParse(sc);

    //create an empty TciValueMessage
    TciValueMessage tciValue = new TciValueMessageImpl();

    //insert the SIPMessage object into TciValueMessage
    tciValue.setObject(message);

    Return tciValue;
}
```

Alternative Approach - Codec

```
//at codec
public Value decode(TriMessage message, Type type) {
    Value returnedValue = null;
   if(message instanceof TciValueMessage) {
      TciValueMessage bimpl = (TciValueMessage) message;
      returnedValue = bimpl.getValue();

      //getValue() returns null while getObject() contains SIPMessage
      if(null == returnedValue && null != bimpl.getObject()) {
          SIPMessage sipMessage = (SIPMessage) bimpl.getObject();
          //convert the SIPMessage object to corresponding Value object based on the
          //Type object here
          returnedValue =  convertFromSIPMessageToValue(sipMessage, type)
       }
   }
   else {
            // handle byte oriented messages here
   }
   return returnedValue;
}
```

Conclusion

- Our new codec/test adapter architecture avoids double encoding/decoding when using frameworks.
- It is useful both for non-telecom and telecom applications
- Future work: determine new application domains