How to implement TTCN-3 codecs and adapters efficiently

Anthony Baire, César Viho

June 2010

TTCN-3 Asia User Conference

This work has been partially funded on behalf of the French Government under the public procurement number 06 42 214 by the DGA Information Superiority Unit
Scope of the Tutorial

• TTCN-3 is an abstract language, it focuses on test logics, e.g:
  – describe of the messages sent and received
  – define the timing constraints

• other implementation details (*communications, encoding, logging, ...*) are not addressed in the language
Scope of the Tutorial

TTCN-3 Abstract Test Suite

testcase TC_Example()
runs on DNSClient
{
    dns_port.send(a_DNSQuery("www.irisa.fr"));
    dns_port.receive(a_DNSAnswer("131.254.254.46"));
}

how to transmit the message to the SUT and receive the reply?

how the messages shall be formatted?
Scope of the Tutorial

• Implementation details (*communications, encoding, logging, ...*) are handled in separate modules:
  – either provided by the TTCN-3 tool suite
  – or implemented in the platform language (C, C++ or Java) by the test developer
Layout of a TTCN-3 Executable Test Suite

- **User**
  - Test Management
    - (user interface)
  - Component Handling
    - (distributed execution)
  - System Adapter
    - (communications with the SUT)
  - Platform Adapter
    - (external functions, timers)
- **Test Executable**
  - (TTCN-3 Test Suite)
  - CoDec
    - (messages formatting)
- **Test Logging**
  - (test reports / user interface)
- **System Under Test (SUT)**

**Languages and Interfaces**
- TTCN-3
- C/C++ or Java
Layout of a TTCN-3 Executable Test Suite

- Test Management
  - (user interface)
- Test Executable
  - (TTCN-3 Test Suite)
- CoDec
  - (message formatting)
- Test Logging
  - (test reports / user interface)
- System Adapter
  - (communications with the SUT)
- Platform Adapter
  - (external functions, timers)
- Component Handling
  - (distributed execution)
- System Under Test (SUT)
Test Suite Development Process

- Test Suite & Protocol Specifications
- Message/Port type definitions (in TTCN-3)
- Testcases implementation (in TTCN-3)
- CoDec & Adapters implementation (in C/C++ or Java)
- Executable Test Suite
Tutorial motivation

• The internal design of codecs and adapters is not addressed by the TTCN-3 Standard
  → this is left free to the developer

• It is not the focus of the test but still a critical task
  – errors in the adapters/codecs give a biased view of what's happening in the test and they are more difficult to detect
  – developing the CoDec is error-prone because there is a lot of redundant code to be written
  – merging several codecs/adapters
Tutorial Purpose

• Provide good practises for developing the codec & adapter ensuring:
  – quick development
  – reliability
  – reusability

• These objectives will be facilitated thanks to T3DevKit, a free software toolkit for TTCN-3
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
Accessing the System Under Test (SUT)

• In a TTCN-3 test suite, the System Under Test is viewed as a black-box component with one or several ports

• The actual communications are implemented in C/C++ or Java in a separate module: the System Adapter
Implementing a System Adapter

- The SA interacts with the compiled test suite using a standardised interface: the **TRI** (*TTCN-3 Runtime Interface*)

- the **TRI** can be used in C/C++ or Java
CoDec Development

• TTCN-3 uses an abstract representation for the messages (similar to ASN.1)

• Messages exchanged with the SUT need to be encoded in a binary format
  – protocol messages that are based on ASN.1 can be encoded/decoded automatically using the adequate rules (BER, PER, ...)
  – for the other protocols it is necessary to develop an external CoDec (Coding/Decoding module)
CoDec Module Features

ENCODING

Message to be sent to the SUT (generated by the TTCN-3 test)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>154</td>
<td>44</td>
</tr>
<tr>
<td>132.154.14.21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1025</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Message reported to the TTCN-3 test

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>154</td>
<td>44</td>
</tr>
<tr>
<td>10.1.2.20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1025</td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DECODING

Actual binary message transmitted to the SUT

01001100010111011...

Binary message received from the SUT

01001100010111011...
Implementing an external CoDec

- The CD interacts with the compiled test suite using a standardised interface: the **TCI (TTCN-3 Control Interface)**
- the **TCI** can be used in C/C++ or Java
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
T3DevKit Features

- T3DevKit: a toolkit for developing TTCN-3 tests:
  - a C++ library (t3devlib) which provides:
    - an object-oriented framework to manipulate TTCN-3 entities (values, ports, timers, ...)
    - an implementation of the TRI & TCI interfaces
    - default codecs
    - some debugging features
  - a CoDec generator (t3cdgen)
  - a set of portable build scripts based on waf
T3DevKit Layout

User (TTCN-3 Test Suite and Adapters Developer)  User (ETS User)

- Code developed by the User: TTCN-3 ATS & C++ Codets, Pixits, SA, and PA
- Standard TRI & TCI APIs (C)
- T3DevKit API (used by the User for SA & PA development; used by the Generated CoDec)
- T3DevKit: T3DevLibs (TCI and TRI compatible) and T3CDGen (Codec Generator)
- Generated CoDec (by the CoDec Generator)
- Compiled ATS (by the Compiler)
- TTCN-3 Development and Execution environment

T3DevKit in the Standard TTCN-3 Edition & Execution Environment
T3DevKit Motivations

• Why use a toolkit?
  – TRI & TCI are low-level interfaces, they do not provide:
    • functions to manipulate data
    • a generic framework for the internal design of the modules
  – we want to avoid “reinventing the wheel”
    → lots of features are not specific to one test suite
      – an object-oriented generic framework
      – memory management
      – defaults implementations (codecs, timers, ...)
      – synchronisation between C++ & TTCN-3 type definitions
      – interactions with the TE
Waf Motivations

• Why build the test suites with waf?
  – Portability (TTCN-3 tool, OS, c++ toolchain)
  – Simplicity
    • A unique build procedure for every environment
    • Automatic detection of the tools
  – Extensibility
    • Write new python modules to integrate new tools (eg. the CoDec generator)
Waf workflow

Test Suite Sources
- TTCN-3 Sources
- Codec & Adapters Sources
- Portable Build Script

WAF modules
- gcc
- msvc
- icc
- compiler_c
- compiler_c++
- ibm_rst
- picottcn
- compiler_ttcn
- ...

WAF build automation tool

Third party WAF modules
- t3devkit
  (codec generator ...)

Executable Test Suite
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
The HelloWorld Example

• Purpose:
  – display the string "Hello World!" to the console

• Objective:
  – demonstrate how to develop a basic System Adapter
module HelloWorld
{
    type port ConsolePort message { inout charstring; } 

type component HelloTester { port ConsolePort console; } 

testcase ExampleHelloWorld()
    runs on HelloTester
    {
        console.send("Hello World!");
    }

control {
    execute (ExampleHelloWorld());
}
}
Implementing the System Adapter (1/3)

### C++ Header

```cpp
class ConsolePort : public t3devlib::Port
{
    public:
        ConsolePort (t3devlib::PortId& id);

    protected:
        bool Send (const t3devlib::ComponentId& from,
                   const t3devlib::Bitstring& msg);
};
```

- Define a class that inherits from `t3devlib::Port`.
- Reimplement the function `Send()`. 
Implementing the System Adapter (2/3)

**C++ implementation of Console Port**

```c++
bool ConsolePort::Send (const ComponentId& from,
                       const Bitstring& msg)
{
    cout.clear();

    cout << endl;
    cout.write ((char*) msg.GetValueBin(),
                msg.GetLength()/8);
    cout << endl;
    cout << endl;

    return cout.good();
}
```

- write the message to the standard output
- return true in case of success
Implementing the System Adapter (3/3)

C++ Initialisation function for the SA

```cpp
namespace t3devlib {

void SAInit() {
    Port::RegisterType ("HelloWorld", "ConsolePort", 
                        &createPort<ConsolePort>);
}
}
```

function called by T3DevKit when initialising the System Adapter

register this port type so that T3DevKit knows how to handle it
Building the Test Suite

**HelloWorld waf script**

```python
def configure(conf):
    conf.check_tool ('compiler_ttcn3 t3devkit')

def build(bld):
    bld.ttcn3_ets(
        features = 't3devkit',
        target = 'HelloWorld',
        te_source = 'HelloWorld.ttcn',
        sa_source = 'ConsolePort.cpp sa-init.cpp'
    )
```

- **Load the waf modules supporting**
  - TTCN-3 compilers
  - T3DevKit
- **Build a TTCN-3 Executable Test Suite**
  using T3DevKit's features (generator...)
- **Source files:**
  - Test Executable (TTCN-3)
  - System Adapter (C++)
Execution

abaire@kakrafoon:/~git/t3devkit/examples/HelloWorld$ twaf configure
Checking for program gcc,cc          : ok /usr/local/bin/gcc
Checking for program cpp             : ok /usr/bin/cpp
Checking for program ar              : ok /usr/bin/ar
Checking for program ranlib          : ok /usr/bin/ranlib
Checking for gcc                      : ok
Checking for Rational Systems Tester : ok /opt/telelogic/tau/bin/t3cg
Checking for program g++,c++          : ok /usr/local/bin/g++
Checking for program ar              : ok /usr/bin/ar
Checking for program ranlib          : ok /usr/bin/ranlib
Checking for g++                      : ok
Checking for t3devkit                 : ok /usr/local/stow/t3devkit/share/t3devkit/config.py
Checking for t3devkit supports 64-bit integer : not found
Checking for t3devkit supports debugging : not found
'configure' finished successfully (0.189s)
abaire@kakrafoon:/~git/t3devkit/examples/HelloWorld$ twaf

'build' finished successfully (8.991s)

16:35:37 abaire@kakrafoon:/~git/t3devkit/examples/HelloWorld$ twaf --run

13:09:18 (null):0 [(null)::(null) '00000000'O] Info 0 Module HelloWorld registered
13:09:18 ../ttcn/HelloWorld.ttcn:41 [(null)::(null) '00000000'O] Info 0 Module HelloWorld initialized
13:09:18 ../ttcn/HelloWorld.ttcn:41 [(null)::(null) '00000000'O] Info 0 Module HelloWorld - module parameters initialized
13:09:19 ../ttcn/HelloWorld.ttcn:54 [(null)::(null) '00000000'O] TcExecute HelloWorld::ExampleHelloWorld {} 0
13:09:19 ../ttcn/HelloWorld.ttcn:54 [(null)::(null) '00000000'O] CCreate [HelloWorld::HelloTester '00010000'O] MTC 0
13:09:19 ../ttcn/HelloWorld.ttcn:54 [(null)::(null) '00000000'O] TcStart HelloWorld::ExampleHelloWorld {} 0
13:09:19 ../ttcn/HelloWorld.ttcn:47 [HelloWorld::HelloTester '00010000'O] PMap [HelloWorld::HelloTester '00010000'O]::console[-1] [HelloWorld::HelloTester '00010000'O]::console[-1]
13:09:20 ../ttcn/HelloWorld.ttcn:50 [HelloWorld::HelloTester '00010000'O] Encode "Hello World !" 0 '48656C6C6F20576F726C642021'O user encoder Hello World !
13:09:20 ../ttcn/HelloWorld.ttcn:50 [HelloWorld::HelloTester '00010000'O] MSend_m [HelloWorld::HelloTester '00010000'O]::console[-1] [HelloWorld::HelloTester '00010000'O]::console[-1] "Hello World !" '00000000'O 0 '48656C6C6F20576F726C642021'O 0
13:09:20 (null):0 [HelloWorld::HelloTester '00010000'O] CStop [HelloWorld::HelloTester '00010000'O]
13:09:20 (null):0 [HelloWorld::HelloTester '00010000'O] TcStop
13:09:20 ../ttcn/HelloWorld.ttcn:54 [(null)::(null) '00000000'O] TcStarted HelloWorld::ExampleHelloWorld {} 0
13:09:20 (null):0 [HelloWorld::HelloTester '00010000'O] CTerminated none
13:09:20 ../ttcn/HelloWorld.ttcn:54 [(null)::(null) '00000000'O] TcTerminated HelloWorld::ExampleHelloWorld {} none
13:09:21 (null):0 [(null)::(null) '00000000'O] CtrlTerminated

Stages:
1. configure
2. build
3. run
HelloWorld Summary

• System Adapter
  ✔ write a C++ class with a specialised Send() function
  ✔ register the port type at SA initialisation (in SAInit)

• Makefile
  ✔ store the list of TTCN-3 & C++ files and the root module name in the adequate T3DK_XXXXX variables
  ✔ include T3DevKit's generic makefile

• CoDec
  ✔ nothing to do (using the default codec for charstring)
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
The Quiz Example

• Purpose:
  – implement a simple Question/Answer game in TTCN-3

• Objectives:
  – develop a fully functional System Adapter
  – demonstrate how to generate a basic CoDec
The Quiz Example

A Question/Answer game in TTCN-3

Quiz
pass !

Question: 2+2 ?

Answer: 4
Quiz TTCN-3 source (testcase)

module Quiz
{

testcase Quizz_2_plus_2()
runs on QuizComponentTester
{

timer t;
qportTester.send (Addition (2, 2));

t.start (20.0);
alt {
    [] qportTester.receive (Result: 4) { setverdict (pass); }

    [] qportTester.receive { setverdict (fail); }
    [] t.timeout { setverdict (fail); }
}
}
module Quiz
{

type port QuizPort message {
    out Operation;
    in Result;
};

type record Operation {
    Operand a,
    Operator operator,
    Operand b
};
type integer Operand;
type charstring Operator length (1);
type integer Result;

message to be sent, eg: Operation: { 2, "+", 2 }
message to be received, eg: Result: 4
module Quiz
{

    template Operation Addition (integer val_a, integer val_b) :=
    {
        a := val_a,
        operator := "+",
        b := val_b
    }
}
Implementing the System Adapter

• Implement a C++ class deriving from t3devlib::Port

  – Sending messages
    → implement a Send() function (see HelloWorld)

  – Receiving messages
    → implement a listening thread that reads the input from the console and reports it to the TTCN-3 runtime system
Implementing Message Reception (1/2)

**Initialisation of the Port**

```cpp
bool QuizPort::Map(const PortId& port_id) {
    if (mMappedPort != NULL) {
        cerr << "QuizPort: cannot map to more than one port";
        return false;
    }
    mThread.reset(new boost::thread(boost::bind(&QuizPort::QuizThread, this)));
    mMappedPort = &port_id;
    return true;
}
```

- Reimplement the `Map()` function (called at the beginning of the testcase)
- Allow only one mapped port (for simplicity)
- Start a listening thread on this port
- Remember the id of the mapped port
Implementing Message Reception (2/2)

Listening thread for QuizPort

```cpp
void QuizPort::QuizThread ()
{
    cin.clear();
    string buff;
    while (getline (cin, buff).good()) {
        buff = buff.substr (0, buff.find_first_of("
"));
        EnqueueMsg (*mMappedPort, buff.c_str(), buff.size() * 8);
    }
}
```

- **Read a line from the console**
- **Remove the trailing newline characters**
- **Forward the message to the TTCN-3 runtime system**
Implementing the CoDec for Quiz

• Two parts:

  – implement the codecs for the subtypes:
    • Operand, Result (integers)
    • Operator (charstring)

  – implement the codec for the structured type:
    • Operation (record)
Reminder: TTCN-3 user-defined types used in Quiz

module Quiz
{

type integer Operand;

type integer Result;

type charstring Operator length (1);

type record Operation {
    Operand a,
    Operator operator,
    Operand b
}
};
Encoding/Decoding the subtypes

Codec header file quiz_codec.h

#include <t3devlib/t3devlib.h>
#include <t3devlib/generator.h>

namespace t3devlib {
    namespace gen {
        T3DEVLIB_ASCII_INTEGER_DEFINITION(Quiz, Operand, Signed);
        T3DEVLIB_ASCII_INTEGER_DEFINITION(Quiz, Result, Signed);
        T3DEVLIB_FIXED_STRING_DEFINITION(Quiz, Operator, Charstring, 8);
    }
}

encode and decode the integer subtypes Quiz.Operand and Quiz.Result in textual format

encode and decode the charstring subtype Quiz.Operator as a fixed-length string (8-bit long)
Encoding/Decoding the record type

nothing to do!

(the CoDec generator will generate a default CoDec automatically for the 'Operation' record type)
Building the Test Suite

Quiz waf script

def configure(conf):
    conf.check_tool ('compiler_ttcn3 t3devkit')

def build(bld):
    bld.ttcn3_ets(
        features = 't3devkit',
        target = 'Quiz',
        te_source = 'Quiz.ttcn',
        sa_source = 'QuizPort.cpp sa-init.cpp',
        t3cdgen_header = 'quiz_codec.h')

tell T3DevKit to include our header when generating the CoDec
Execution

abaire@kakrafoon:~/git/t3devkit/examples/Quiz$ twaf
Waf: Entering directory `/home/abaire/git/t3devkit/examples/Quiz/__build__'
[cxx link: __build__/default/__t3devkit__/pa_init_8.o __build__/default/c++/QuizPort_8.o __build__/default/c+/sa-init_8.o __build__/default/t3devkit__/cd_init_8.o __build__/default/t3devkit__/gen_classes_8.o __build__/default/t3devkit__/root_module_8.o __build__/default/t3rts_conditional_8.o __build__/default/codec_plugin_8.o __build__/default/tau__ts_modules_8.o __build__/default/tau__Quiz_8.o -> __build__/default/Quiz_ETS]
Waf: Leaving directory `/home/abaire/git/t3devkit/examples/Quiz/__build__'
'build' finished successfully (10.321s)

abaire@kakrafoon:~/git/t3devkit/examples/Quiz$ twaf --run
Waf: Entering directory `/home/abaire/git/t3devkit/examples/Quiz/__build__'
Test case started: Quiz.Quizz_2_plus_2
--------------------------------
-- Quizz Question --
Enter the result of 2+2
-> 4
Test case terminated --> pass
Waf: Leaving directory `/home/abaire/git/t3devkit/examples/Quiz/__build__'
'build' finished successfully (6.328s)
Quiz Summary

• System Adapter
  ✓ reimplement Port::Map() and launch a listening thread
  ✓ forward the received messages using Port::Enqueue()

• CoDec
  ✓ record type: nothing to do (automatically generated)
  ✓ subtypes: write a header to define each type and associate it with a codec

• Makefile
  ✓ include our new header in the CoDec generation
Summary

1. The TTCN-3 standard interfaces TRI & TCI

2. Overview of T3DevKit Features

3. Examples
   a) HelloWorld
   b) Quiz
   c) DNS
The DNS Example

- **Purpose:**
  - implement a real-life protocol (DNS)

- **Objective:**
  - demonstrate how to customise the CoDec for a complex protocol
The DNS Example

- Implement a DNS Client in TTCN-3 so as to test a DNS Server

Question: where is www.irisa.fr?

Pass!

Answer: www.irisa.fr is at 131.254.254.46
DNS messages in TTCN-3

[RFC 1035] Domain Implementation and Specification

4. MESSAGES

4.1. Format

```
+-----------------------------+
| HEADER                      |
+-----------------------------+
| QUESTION  | the question for the name server |
+-----------------------------+
| ANSWER    | RRs answering the question       |
+-----------------------------+
| AUTHORITY | RRs pointing toward an authority |
+-----------------------------+
| ADDITIONAM | RRs holding additional information |
+-----------------------------+
```
module DNS
{
  type record DNSMessage {
    DNSHeader header,
    DNSQuestions questions,
    DNSResourceRecords answers
    // authorities & additional info omitted
    // to keep the example simple
  };
}

type set of DNSQuestion DNSQuestions;
type set of DNSResourceRecord DNSResourceRecords

the actual number of fields is variable
DNS header in TTCN-3

[ RFC 1035 ] Domain Implementation and Specification

4.1.1. Header section format

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ID |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| QR | Opcode | AA | TC | RD | RA | Z   | RCODE |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    | QDCOUNT |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    | ANCOUNT |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    | NSCOUNT |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
    +---+---+---+---+---+---+---+---+---+---+---+---+---+
    | ARCOUNT |
```

- Padding field (filled with zeros)
- Number of entries in each section of the message
module DNS
{

type record DNSHeader {
    UInt16      Id,
    boolean    QRflag,
    UInt4       Opcode,
    boolean    AAflag,
    boolean    TCflag,
    boolean    RDflag,
    boolean    RAflag,
    // 3 padding bits omitted
    UInt4       Rcode
    // QDCount, ANCount, NSCount & ARCount omitted
};

These fields will be handled inside the CoDec only (not necessary in the test suite)

NOTE: this is just a design choice
4.1.2. Question section format

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   | QNAME |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   | QTYPE  |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   | QCLASS |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

[RFC 1035] Domain Implementation and Specification
module DNS
{
    type record DNSQuestion {
        DNSName Name, 
        UInt16 Type, 
        UInt16 Class
    }
}
DNS resource record in TTCN-3

[RFC 1035] Domain Implementation and Specification

4.1.3. Resource record format

```
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |
| NAME  | TYPE  | CLASS | TTL   | RDLENGTH | RDATA |
+-------+-------+-------+-------+-------+-------+-------+
```
module DNS
{

type record DNSResourceRecord {
    DNSName Name,
    UInt16 Type,
    UInt16 Class,
    UInt32 TTL,

    // the RdLength field is omitted
    DNSResourceData Rdata
};;

type union DNSResourceData {
    DNSName name,
    IPAddress ip,
    octetstring str
};;

the resource data has several possible formats
module DNS
{
    type integer UInt4 (0..15);
    type integer UInt16 (0..65535);
    type integer UInt32 (0..4294967295);

    type charstring IPAddress;
    type charstring DNSName;
}

must be binary encoded (32-bit field)
eg: "131.254.254.46" => 0x83fefe2e

must be encoded in a DNS-special format (possibly with compression)
eg: "www.irisa.fr" => "\x03www\x05irisa\x02fr\0"
Implementing the DNS CoDec

- Automatically generated by T3DevKit...

- ...but additional informations are necessary:
  - how many entries in DNSQuestions & DNSAnswers ?
  - omitted fields in DNSHeader
  - which union variant in DNSResourceData?
  - what is the length of a DNSResourceData?
  - special formats for IPAddress & DNSName
Implementing the DNS CoDec

• Solution: customise the generated CoDec by inserting additional code at specific points

  – this customised code is written in special functions named *codets*

  – *codets* may be hooked in lots of places in the encoding/decoding processes, eg:
    • PreDecode(), PostDecode()
    • PreEncodeField()
    • ...

How does the generated CoDec looks like?

**TTCN-3 Definitions**

```c
type union DNSResourceData {
    DNSName name,
    IPAddress ip,
    octetstring str
};
```

**Generated CoDec (C++)**

```c
class DNSResourceData : public t3devlib::Union {
    public:
    void Encode (Buffer&);
    void Decode (Buffer&);
    DNSName& Get_name();
    IPAddress& Get_ip();
    Octetstring& Get_str();

    enum {
        id_name = 0,
        id_ip = 1,
        id_str = 2
    };
};
```

- One C++ class generated for each type.
- T3DevKit's base class specialised for encode/decode functions.
- Field accessors.
- Field index IDs.
Customising the CoDec
Example: length of DNSResourceData

module DNS
{
    type record DNSResourceRecord {
        DNSName Name,
        UInt16 Type,
        UInt16 Class,
        UInt32 TTL,
        // the RdLength field is omitted
        DNSResourceData Rdata
    };

type union DNSResourceData {
    DNSName name,
    IPAddress ip,
    octetstring str
};

the binary length size of the resource data is given by the RdLength field
Decoding DNSResourceData

Decoding _codet_ for DNSResourceData

```cpp
void DNSResourceData::PreDecode (Buffer& buffer) throw (DecodeError) {
    UInt16 Rdlength;
    Rdlength.Decode (buffer);
    SetHypLength (Rdlength.GetValue() * 8);
}
```

function called automatically before decoding a DNSResourceData

Read a 16-bit integer from the buffer (field RdLength)

tell T3DevKit how big is this value (we set an hypothesis)

=> in case of mismatch, T3DevKit will throw an exception later
Decoding DNSResourceData

module DNS
{

type record DNSResourceRecord {
    DNSName Name,
    Uint16 Type,
    Uint16 Class,
    Uint32 TTL,
    // the RdLength field is omitted
    DNSResourceData Rdata
};

type union DNSResourceData {
    DNSName name,
    IPAddress ip,
    octetstring str
};

the variant of the union depends on the class & type of the entry
function called automatically after decoding each field in DNSResourceRecord

once the Class field is decoded, we decide which variant to decode in the next DNSResourceData

Decoding DNSResourceData

void DNSResourceRecord::PostDecodeField(int id, Buffer& buffer) throw (DecodeError) {
    if (id == id_Class) {
        DNSResourceData::SetHypChosenId(DNSResourceData::id_str);
        if (Get_Class().GetValue() == 1) { // class IN
            if (Get_Type().GetValue() == 1) { // Type A
                DNSResourceData::SetHypChosenId(DNSResourceData::id_ip);
            } else if (Get_Type().GetValue() == 5) { // Type CNAME
                DNSResourceData::SetHypChosenId(DNSResourceData::id_name);
            }
        }
    }
}}
Decoding the DNS Header

module DNS
{
    type record DNSHeader {
        UInt16 Id,
        boolean QRflag,
        UInt4 Opcode,
        boolean AAflag,
        boolean TCflag,
        boolean RDflag,
        boolean RAflag,
        // 3 padding bits omitted
        UInt4 Rcode
    }
};

We must decode these additional fields and make the correct hypothesis when decoding the Questions & Answers sections.
void DNSHeader::PostDecodeField (int id, Buffer& buffer) throw (DecodeError)
{
    switch (id) {
    case id_RAflag: {
        // padding : need to skip three bits
        Unsigned(3).Decode (buffer);
        break;
    }
    }
}

function called automatically after decoding each field in DNSHeader

after decoding the RAflag field, we skip the next 3 padding bits
Decoding the DNSHeader

Decoding codet for DNSHeader

```c
 Decoding codet for DNSHeader

 case id_Rcode: {
    UInt16 count;
    count.Decode (buffer); // QD count
    DNSQuestions::SetHypSize (count.GetValue());
    count.Decode (buffer); // AN count
    DNSResourceRecords::SetHypSize (count.GetValue());
    count.Decode (buffer); // NS count (ignored)
    count.Decode (buffer); // AR count (ignored)
    break;
}
```

- **read the QD count field (16-bit integer)** and use its value as an hypothesis when decoding the DNSQuestions set of idem for the answer count
- **read the NS count and AR count fields** (so as to be aligned with the end of the header when leaving)
Encoding & Decoding the IP Address

Manual CoDec implementation codec for IPAddress

class IPAddress : public t3devlib::Charstring
{
public:
    const char* GetModuleName() const { return "DNS"; }
    const char* GetTypeName() const { return "IPAddress"; }

    void Encode (Buffer& buffer) throw (EncodeError);
    void Decode (Buffer& buffer) throw (DecodeError);
};
void IPAddress::Decode (Buffer& buffer) throw (DecodeError)
{
    Unsigned ip(32);
    ip.Decode (buffer);
    SetValue (inet_ntoa ((const struct in_addr*) ip.GetValueBin()));
}
Decoder for the DNS Name

- similar to the decoder of IPAddress's
  (there is nothing new)

=> the complete example is available on the CD
Decoder for NameServers and Additional Info

- this example does not address the last two sections in the DNSMessage
  → need to skip the sections when decoding the message

**Code for DNS Message**

```cpp
void DNSMessage::PostDecode (Buffer& buffer)
    throw (DecodeError)
{
    buffer.SetPosition (buffer.GetEndMarker());
}
```

move the buffer cursor to the end of the message
(T3DevKit would report an error if the buffer was not fully decoded)
Implementing the DNS Encoder

• specific actions to be implemented in the encoder:
  – adding missing fields (eg. QD/AN/NS/Al count)
  – computing the length of ResourceRecords
  – custom encoder for IPAddress and DNSName
    ➔ `Encode()` member function

(see the complete example on the CD)
module DNS
{

type record DNSResourceRecord {
    DNSName Name,
    UInt16 Type,
    UInt16 Class,
    UInt32 TTL,
    // the RdLength field is omitted
    DNSResourceData Rdata
};

type union DNSResourceData {
    DNSName name,
    IPAddress ip,
    octetstring str
};

the RdLength field must be filled with the length of the DNSResourceData in the encoded message
Encoding the length of DNSResourceData

**dns_codec.h**

```c
#define DEFINITIONS_DNSResourceData() \
    int mPosition;
```

Define a new attribute in the class DNSResourceData (the generator will include this macro)

**dns_codets.cpp**

```c
void DNSResourceData::PreEncode (Buffer& buffer) 
    throw (EncodeError) 
{
    Unsigned(16).Encode(buffer);
    mPosition = bufferGetPosition();
}
```

Insert a blank 16-bit field (to store RdLength later)

remember our current location in the buffer (for later use)
Encoding the length of DNSResourceData

dns_codets.cpp

```cpp
void DNSResourceData::PostEncode (Buffer& buffer)
    throw (EncodeError)
{
    int current_position = buffer.GetPosition();

    Unsigned Rdlength (16, (current_position - mPosition) / 8);

    buffer.SetPosition (mPosition - 16);
    Rdlength.Encode (buffer);

    buffer.SetPosition (current_position);
}
```

- Remember the current position in the buffer.
- Compute RdLength and write it into the buffer at the correct location.
- Move the buffer cursor back to the end of the resource record.
Building the Test Suite

DNS Makefile

```python
def configure(conf):
    conf.check_tool ('compiler_ttcn3 t3devkit')

def build(bld):
    bld.ttcn3_ets(
        features = 't3devkit',
        target = 'DNS',
        te_source = 'DNS.ttcn',
        sa_source = 'DNSPort.cpp sa-init.cpp',
        cd_source = 'dns_codets.cpp',
        t3cdgen_header = 'dns_codec.h'
    )
```

Source files for the CoDec

T3DevKit will scan them to detect which codets are implemented
CoDec Generation Process

Abstract Test Suite (TTCN-3)

```c
type record DNSMessage {
    DNSHeader   header
    ...
}
```

Codets (C++)

```c
void DNSMessage ::PostDecode(...) {
    ...
}
```

T3DevKit's CoDec Generator

Generated CoDec

```c
class DNSMessage :
    public t3devlib::Record {
    public:
        void Get_header();
        ...
        void Encode(...);
        void Decode(...);
        void PostDecode(...);
    }
```
Execution

```bash
abaire@kakrafoon:/~git/t3devkit/examples/DNS$ DEBUG_CODEC=1 twaf --run
Waf: Entering directory `/home/abaire/git/t3devkit/examples/DNS/__build__'
Test case started: DNS.ExampleResolveIrisa

******************** Encode: DNSMessage ********************
DNSMessage (record)
  header
    DNSHeader (record)
      Id                                      12345 - 0x3039 (16 bit unsigned integer)
      QRflag                                  false (boolean)
      Opcode                                  0 - 0x0 (4 bit unsigned integer)
      AAflag                                  false (boolean)
      TCflag                                  false (boolean)
      RDflag                                  true (boolean)
      RAflag                                  false (boolean)
      Rcode                                   0 - 0x0 (4 bit unsigned integer)
  questions
    DNSQuestions (set of)
      DNSQuestion (record)
        Name                                  "www.irisa.fr" (charstring)
        Type                                  1 - 0x1 (16 bit unsigned integer)
        Class                                 1 - 0x1 (16 bit unsigned integer)
  answers
    DNSResourceRecords (set of)

Encoded to: '3039010000100000010000000000037777770569726973610266720000010001'O

************************************************************
*************** Decode: DNSMessage, 1520bits ***************

Test case terminated --> pass
Waf: Leaving directory `/home/abaire/git/t3devkit/examples/DNS/__build__'
'build' finished successfully (0.096s)
abaire@kakrafoon:/~git/t3devkit/examples/DNS$
```

enable CoDec debugging (T3DevKit feature)
DNS Example Summary

- **CoDec**
  - implement special actions in codets => eg: `PreDecode()`
  - make predictions when decoding
    - binary length of a field => `SetHypLength()`
    - variant of a union => `SetHypChosenId()`
    - size of a set of => `SetHypSize()`
  - new class attributes can be defined using a C macro
  - custom formats (IPAddress, DNSName): implement `Encode()` and `Decode()` directly

- **Makefile**
  - give the list of files containing codets
Questions ?

Contacts: abaire@irisa.fr / viho@irisa.fr

Complete source code (T3DevKit & examples) available at: http://t3devkit.gforge.inria.fr/