Towards an IP-oriented Testing Framework
the *IPv6 Testing Toolkit*

Ariel Sabiguero[1,2], Anthony Baire[2], Alexandra Desmoulin[2], Annie Floch[2], Frédéric Roudaut[2], and César Viho[2]

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
`asabigue@fing.edu.uy`
`http://www.fing.edu.uy/inco`

[2] IRISA
Campus de Beaulieu
35042 Rennes CEDEX, France
{`asabigue,abaire,adesmoul,afloch,roudaut,viho`}`@irisa.fr`,
`http://www.irisa.fr/armor`

Abstract. TTCN-3 is an abstract language for specification of Abstract Test Suites. Coding of TTCN-3 values into physically transmittable messages and decoding of bitstrings into their TTCN-3 representation has been removed from the language itself and relayed to external and specialized components, called CoDec. CoDec development is a must for IPv6 protocol testing as standard ones do not cope with the requirements. To achieve adequate software engineering practices, a set of types, tools and definitions were developed. This paper unveils gray areas in TTCN-3 architecture and presents a methodological approach to minimize the complexity of CoDec development. Even though the initial field of application is IPv6 testing, the main tool introduced -the CoDec Generator- is a valuable tool in any testing application domain. This CoDec Generator is developed within the framework called IPv6 Testing Toolkit.

Keywords: IPv6 Testing, TTCN-3, automatic CoDec generation

1 Introduction

The Testing and Test Control Notation version 3 (TTCN-3) is a standard test specification and implementation language [1–6]. It is designed to provide a framework for the precise definition of test procedures for black-box testing of responsive communicating systems [7, 8]. TTCN-3 has been defined to be abstract enough so as to be used for any kind of testing activity, from Abstract Test Suite specification to Executable Test Suite execution. It is supposed to allow an easy, efficient and yet powerful description of abstract test suites in a platform independent manner. Even though it cannot be considered a new language, there is not enough maturity and know-how in the community regarding its application.

TTCN-3 is a strong-typed language which presents some difficulties when trying to work with complex, low level oriented data. Network protocols hold themselves several hard to predict behaviors related to flow flags, options and other aspects that require the ability to handle unknown sizes, number of options, etc. TTCN-3 language provides basic matching capability based on wildcards like ? and *.

Coding of TTCN-3 values into transmittable messages and decoding of bitstrings into their TTCN-3 representation has been removed from the core language itself and is done in an external component. These operations of coding and decoding are relayed to a specialized component named CoDec. The CoDec is interfaced with the TTCN-3 Executable (TE) through the Test Control Interface-Coding Decoding (TCI-CD) interface. As CoDec are not standard in TTCN-3, required ones *might* be present or not in tools. This is due to the fact that even though the TCI-CD interface is standard, there is no minimal requirement for CoDec implementation. Tool-provided generic CoDec were inadequate for working with IP traffic. TTCN-3 allows the possibility of implementing new CoDec, specific to the communication problem being addressed. They have to be coded in a "lower level" programming language like Java or C++, which we name *platform language*. The concept underneath the word platform shows that the choice between C++ and Java is not only a matter of taste, but it defines the support that you may get from the environment selected.

The design of a test system architecture that separates implementation details from the test definition itself helps achieving a high level of abstraction in the test definition language. On the other hand, it imposes additional complexity to the test development process: handling of communicable types has to be done both in the abstract TTCN-3 specification and in the platform language specialized CoDec. Every time the low-level types are reviewed, pieces of highly coupled yet independent code, developed in different languages, have to be altered and kept synchronized manually.

The objective of the present work is to present and discuss the experience gathered producing test suites for different IPv6 protocols in TTCN-3 language, code engineering options and how we solved the puzzle to allow reusability and maintainability of test suites. During this process, a framework for the testing of IPv6 based protocols was defined and implemented. This framework, named *IPv6 Testing Toolkit*, provides enough flexibility for software reuse with minimal or no code modifications. Existing works, like [9, 10] considered during the development.

The paper is organized as follows. In Section 2 we present intermediate works that addressed the problem from a different approach and help motivating current framework. Section 3 summarizes the main problems addressed, puts together the experience gathered through previous experiments and points out the decisions that lead to the development of the toolkit. Afterward, in Section 4 the CoDec Generator is introduced. The toolkit with a few examples is shown in 5. The work concludes in Section 6, where also future lines of work are presented.

2 Highlights of intermediate solutions

With the goal of producing an adequate framework for IPv6 protocol testing, several different approaches for test engineering were experimented. Some of them are worth to be described, some others are not.

Since our first approach into the TTCN-3 world, the main problems found were the low level data handling requirements imposed by IPv6. The problems ranged from IPv6 network addresses manipulation to reception of unexpected -but still valid and conformant- data. State-of-the-art IPv6 testing requirements do not impose highly complicated signaling patterns, but very detailed composition of messages and a bitwise inspection of incoming messages. TTCN-3 language design addresses mainly the high level part of the testing problem, relaying the low level, bit oriented work to specialized pieces of code not specified in TTCN-3.

2.1 Some TTCN-3 considerations

Before going into further details, lets just review some TTCN-3 concepts. According to [6], a TTCN-3 test system can be thought conceptually as a set of interacting entities, each implementing a specific test functionality. Figure 1 shows the general structure of a TTCN-3 test system. We will focus on the main concepts addressed by this work.

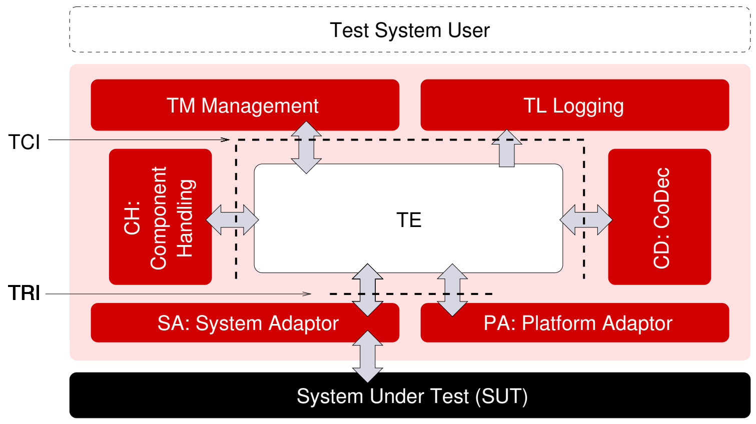

Fig. 1. Conceptual architecture of TTCN-3

The TTCN-3 Executable (TE) interprets and executes TTCN-3 modules. The Test Logging (TL) entity performs test event logging and presentation to the Test System User. SA, which stands for SUT Adaptor (System Under Test Adaptor), "adapts" communications between the TTCN-3 system and the

SUT. The Platform Adaptor (PA) implements external functions and provides a TTCN-3 system with a single notion of time. TE can be distributed among several test devices. The Component Handling (CH) implements communication between distributed test system entities. The Test Management (TM) entity is responsible for overall management of a test system. Finally, the Coding and Decoding (CD) entity is responsible for the encoding and decoding of TTCN-3 values into bitstrings suitable to be sent to the SUT. All this definitions can be found (and were taken from) [1, 5, 6].

TTCN-3 standards do not define some other concepts associated to the TTCN-3 world, which are required here. Due to the fact that our approach is the application of TTCN-3 technology in a cutting-edge field, we are concerned with implementation and field problems. Some of those problems are not addressed by TTCN-3 standards. Let us take them from the Go4IT project[1]. The project is also concerned with tools and practical testing issues. They found the lack of some definitions too, and they propose definitions for them. The Go4IT project defines the TTCN-3 Development Environment as the following set of tools: an Integrated Development Environment (IDE) for ATS edition; an adaptation layer IDE; runtime modules; debugger. We shall refer briefly to the TTCN-3 Development Environment as a TTCN-3 tool or simply, tool.

2.2 Initial Problems

The test development process induced by TTCN-3 mainly splits the work in two different development tasks. The first one directly related to the definition of the message exchange, related to the abstract idea of test execution. The second one addresses crafting and bitwise coding and decoding of exchanged messages. These two different tasks, even though tightly related, are addressed by different experts with deep IPv6 skills, using different languages: the first task requires TTCN-3 specification, while the second one, platform language codification, in our case C++.

These two different tasks are combined through the TCI-CD interface, a TTCN-3 specialized interface for the decoupling of abstract and low level message handling. Its goal is to permit the data exchange between the TTCN-3 data structures and the platform language, which is in charge of performing low level or specialized tasks. To avoid obscure data manipulation practices, we decided to completely map TTCN-3 types into platform language ones and vice versa. In this way, we would share a common modeling of the communication messages and objects both in the platform language and in the TTCN-3 test specification. Low level manipulation would be done in the C++ view of the data, while test related decisions would be taken on the high level model done in TTCN-3 types. The link between these two representations is given by the encode and decode operations of the TCI interface, which were developed too.

After our first complete implementation of a test suite following this approach [11], we realized that this process (named CoDec development from here

[1] http://www.go4-it.org/

on) is tedious and error-prone. Whenever there is a C++ or TTCN-3 requirement that forces some change in the type definition, the counterpart also has to be corrected accordingly. Differences in the expressiveness of the type definition structures of both languages induces non transparent data transformation procedures.

Additionally, IPv6 is not just a protocol, but a protocol suite. It forces us to handle different, simultaneous, not always related IPv6 message exchanges. This forces us to be able to handle all possible incoming IPv6 messages, even though we are interested in testing some specific behavior. Transmission is not a problem, as we are in control of which messages to transmit.

2.3 Reusing an existing tool: Ethereal

Trying to minimize the complexity of the CoDec developed, the decision taken was to make an attempt to use an existing tool for solving the decodification of incoming packets. The goal is to avoid the complexity of manually decoding an arbitrary incoming IPv6 packet and having the task done by the Ethereal tool [12]. Ethereal is a well known tool extensively used in the IP world. It provides several benefits like: it is well known, it evolves with new protocols, it is maintained, it is community-validated and it is free. Considering all these benefits, we decided to perform the low level decoding of incoming messages using Ethereal and interface it with TTCN-3 through the TCI-CD interface.

Several transformation formats and tasks were required to interface Ethereal with TTCN-3. Data received through the Platform Adaptor reaches the TCI-CD interface in a TTCN-3 format. Even though it is the bitstring representation of the packet, it is received as a TTCN-3 string. It has to be transformed into something understandable by Ethereal. Then, Ethereal's output has to be parsed and used to assemble the TTCN-3 objects that will hold the received message. Amongst the different Ethereal output formats, PDML (Packet Details Markup Language) [13] format was chosen, an XML representation of network packets. The library `libxml` was used to parse the PDML description of the packet and have access to all of its parts.

Even though it was possible to reuse the tool and avoid the complexity of packet parsing, interfacing Ethereal is not a minor task. Most of all, not all problems were solved using Ethereal.

Problems due to the use of Ethereal
Even though Ethereal tool greatly solved the problem of parsing incoming messages, it does only provide that. Message transmission is done independently from Ethereal. This solution lacks of a symmetrical treatment for transmission and reception operations.

Moreover, as PDML is neither standard nor stable, thus the mapping has to be reviewed every time Ethereal is updated. Ethereal also does not decode parts of the packets which are important for our purposes (i.e. content of padding fields), thus, it is necessary to patch Ethereal to meet our requirements. The amount of C++ code to maintain did not shrink, and the solution become more complicated for deploying, as an additional external dependency was added. The complexity was shifted from packet parsing into interfacing and data transformation, but still a simple and elegant solution is missing.

The sum of all these problems suggested us to abandon this approach.

3 Summary of main addressed problems

This Section summarizes technological needs and problems faced before the development of the CoDec Generator. It shows relevant problems and solutions found, our vision of what is required from a testing tool and the issues that motivated our decisions.

It is worth mentioning that the problems introduced by the use of Ethereal, are not due to Ethereal itself or the reuse of existing tools. Not all the different usages of Open/Free solutions were discarded as happened with the previous example. Functional extensions were successfully added to Ethernet ports using the libraries `libpcap`[14] and `libnet`[15], as presented in [16]. These extensions are now included in all of our tools. What was addressed and solved are some specific requirements of IPv6 protocol testing, but did not solve the main issues regarding an adequate protocol testing framework. The main problems that were faced and had to be addressed are presented in this Section.

3.1 CoDec specific problems

As stated before in Section 2, CoDec development, integration and maintenance represent the main issue for us using TTCN-3 for IPv6 testing. This is due to the fact that manual synchronization of types has to be done in two languages: C++ and TTCN-3. Moreover, there is a group of operations whose natural place is the CoDec itself. For example, operations like checksum and length calculation can be seen more like a transmission problem than a test logic problem, thus the coding process is a natural place for performing these operations. We would like to integrate these operations in the test development process in a more automated way.

Another aspect that becomes clear after working with CoDec implementation is that the TCI-CD is only an API designed for data exchange, not for data manipulation. Standard TTCN-3 data manipulation from the platform language is required for easy and efficient CoDec development. We realized that there are no standard libraries for manipulation of TTCN-3 data in an intuitive, efficient and uniform way across the different types. Standard operations in languages like C++ (i.e. casting, type conversion, etc.) cannot be performed in TTCN-3 in a simple and type independent way. To ease CoDec generation we find it necessary to be able to develop a library that provides a value-type handling similar to the one used in the platform language.

At a certain moment in time, three different groups of our team were coding different IPv6 test suites for independent protocols. We faced severe problems for

IPv6 core protocol type definitions due to the fact that there is not a methodology or tool support for separating test specific issues from standard (library-like) routines or processes.

Several other small factors also accounted, but we might want to point out as a last relevant problem that our near-future requirements imposed us the need of a strong workload on the CoDec side. Encryption and Security handling can be seen as further layers of encapsulation of message encoding operations. This requires that good software engineering practices are applied to all the software development process. TTCN-3 does not provide means for adequate handling of these operations and completely manual implementation of all operations would become not feasible, or at least, extremely complex.

3.2 Empirical observations

Apart of previously mentioned problems, our TTCN-3 experience also showed empirical facts that oriented us in subsequent decisions. The first one, that is almost evident, is that there is a high level of redundancy between TTCN-3 and C++ code. Due to the fact that the development process applied started from the abstract side, it is the the C++ code that repeats TTCN-3 structures. The C++ code which implements the CoDec is only a mean for representation conversion between physical messages and TTCN-3 data types.

Other relevant observation performed is that TTCN-3 type definition already holds most of the information required for coding and decoding. Most of the platform language code mainly repeats TTCN-3 one. The addition is a few metadata information (like type length for some data types) and specific algorithms for coding/decoding particular fields in non-standard ways. Also precedence in coding/decoding operations has to be specified, as calculating the length field and afterward the checksum is not the same than the reverse order.

3.3 Approach followed

The objective is to simplify test suite development process by minimizing CoDec development and maintenance work. This can be achieved separating all that can be automatized from what really has to be provided (because cannot be expressed in TTCN-3 language). This separation can be done by extending TTCN-3 adding the missing logic, dependencies and semantic which are not present in the standard language.

The tool that automatically performs these tasks will be referred as *the CoDec generator* from now on and will be the subject of Section 4. The initial cost of development a CoDec Generator is higher than simply developing a CoDec for a single test suite. The main advantage of the CoDec Generator is that it can be reused through different tests. In this way, the test dependent part of the CoDec remains independent from the CoDec generator. The CoDec generator becomes a part of our test platform and test development process. Only the platform language code and logic added to the TTCN-3 abstract specification is part of the test suite. The pieces of platform language code which serve as input for the CoDec Generator will be referred as *codets*.

4 The CoDec Generator

The CoDec Generator is a generic tool that fully automatizes the task of CoDec development. It takes the TTCN-3 code and *codets* (additional logic developed in the platform language) and produces a CoDec that implements the TCI-CD interface, providing the required coding and decoding facilities. Even though it was developed while addressing IPv6 protocol testing, the CoDec generator was carefully designed and developed as an "universal" tool, and can be used for CoDec generation in any testing domain. The only bias introduced by our IPv6 requirements is the order in which features were developed and that the platform language for which it is currently implemented is C++.

The underlying idea behind CoDec Generator was already presented as our work methodology during previous sections: each TTCN-3 type is mapped to a platform language object and customized conversion means are provided using codets. The idea of having different levels of abstraction, and thus, different data models in the CoDec and in the TTCN-3 abstract test specification was discarded as it would always require a very specialized CoDec. In such case, the CoDec expert and the TTCN-3 expert would have different views of the problem and would not even share a common data model of the problem.

The mapping implemented by the CoDec Generator is not performed directly to platform language objects, but to a hierarchy of objects designed to provide a comfortable framework for data handling inside the CoDec. We will return to this point in 4.2. The rest of this section provides a quick glimpse of the CoDec Generator.

4.1 Architecture

The CoDec Generator implements a TTCN-3 parser, built using bison[17]/flex[18] Open/Free tools. The implemented parser is responsible of extracting basic type information and structure from standard TTCN-3 code. Even though only type information is strictly required for CoDec Generation, a parser that accepts the complete TTCN-3 language was developed, which is another spin-off of this work. Type information is further augmented (as shown in 4.3) with codets that perform specific operations between the TTCN-3 object and the low level, platform language managed codification.

The way in which the CoDec Generator is integrated into the native TTCN-3 framework is straightforward and simple. As it does not produce changes into the TTCN-3 code, no particular care has to be taken while developing TTCN-3 ATS. The CoDec Generator shall be invoked prior to TTCN-3 link-edition phase, so the actual CoDec is generated for the ETS. The CoDec Generator may produce platform language sources, objects or libraries, according to the TTCN-3 tool requirements. Depending on the options provided by TTCN-3 tool, it can be included into user defined link edition commands and invoked transparently from the tool environment.

4.2 testing toolkit `library` - `ttlib::`

The `ttlib::` provides basically functions for data type management and data coding and decoding. The library provides adequate definitions that allow the mapping of all TTCN-3 types and data structures into special platform objects. Platform objects were engineered using platform language (C++ or Java) Object-Oriented properties (inheritance, polymorphism, etc.) so as to allow homogeneous and simple access to all types. The figure 2 shows the class-inheritance diagram for the objects that maps TTCN-3 primitive types.

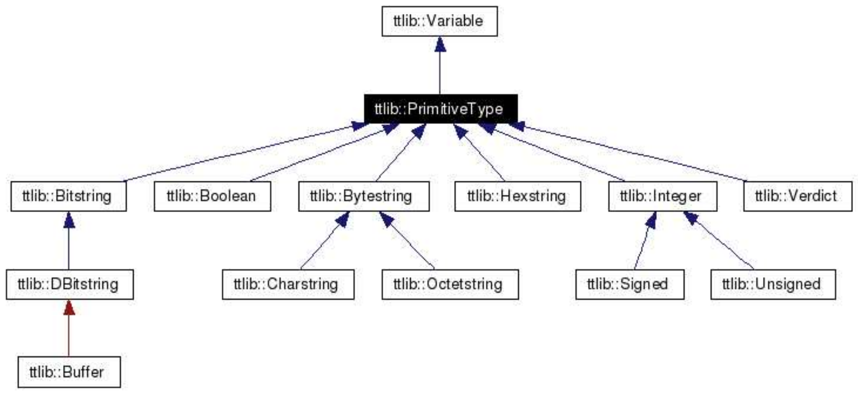

Fig. 2. Platform basic type hierarchy.

This ensures a minimum interface (set of member methods) available for all the objects. All objects implement their own `Encode()` and `Decode()` methods, main reason for a CoDec. Methods like `GetValueHexa()` and `SetValueHexa()` are intended to provide a uniform handling of the value, regardless the object itself. The implementation would be subclass dependent as the semantic might differ from a `Charstring` to an `Integer`, but a uniform way of accessing primitive types is provided.

All variables, specializations of class `ttlib::Variable`, provide a method `Dump` that allows textual representation of instance's value. This method receives as a parameter an output stream and is intended to provide an aid for CoDec debugging. Providing a comprehensive guide to the library is beyond the scope of this section.

4.3 Codet: platform language code extension.

In the general case, direct TTCN-3 type conversion into platform objects and back is not feasible for complex protocols. Many protocols not only handle unknown size payloads, but inclusion of unknown options, making it difficult to handle simple type matching and standard codification rules. To help the (de)coding process, the CoDec Generator accepts codets that perform specialized handling. This allows the test developer to separate (de)coding logic from test logic and also to place logic that naturally belongs to (de)coding process in the CoDec. This approach follows the same design principle of TTCN-3, but addresses relevant software engineering aspects. If no additional input is provided, the CoDec Generator will produce -if possible- a CoDec that directly maps TTCN-3 types into bitstrings and vice versa.

Different options of "logic extensions" were considered during the design of this version of the CoDec Generator. Maybe the most appealing ones were those who extended TTCN-3 language. We faced problems: access to a compiler was required so as to modify it and implement our extensions; our ATS would be non-portable. Other option considered was to perform the extensions inside comment blocks. In this way our ATS would still be specified in standard TTCN-3 language, but parsing becomes more complex and non standard. The implemented option considers independent files for both TTCN-3 code and the extensions.

```
type union ICMPv6OptionSingleType {
    SLLOptionType                     SLLOpt,
    TLLOptionType                     TLLOpt,
    RedirectHdOptionType              RedirectOpt,
    MTUOptionType                     MTUOpt,
    PrefixOptionType                  PrefixOpt,
    ICMPv6UndefinedOptionType         UndefinedOpt,
    AdvertisementIntervalOptionType   AdvertisementInterval,
    HomeAgentInformationOptionType    HomeAgentInformationOpt
}
```

Fig. 3. TTCN-3 type definition for a ICMPv6 options field

Figure 3 shows TTCN-3 type definition for the option field of ICMPv6 and figure 4 shows the codet to be executed prior to the actual decoding of the field, specifically, for guessing the type of the option field. We can see that the matching is done based on TTCN-3 type names and predefined member names. These member names correspond to the TTCN-3 type that is being coded and the moment that the operation is to be performed. We will refer to this possible entry points as *codec hooks*. Possible codec hooks for decoding are: `PreDecode`, `PreDecodeField`, `PostDecodeField`, `PostDecode`. The CoDec

```
inline void ICMPv6OptionSingleType::PreDecode (Buffer& buffer)
  throw (DecodeError) {
       UInt8 type;
       int position = buffer.GetPosition();
       buffer.Read (type, 8);
       buffer.SetPosition (position);
       SetHypChosenId (map_icmpv6_opttype.GetValue(type));
  }
```

Fig. 4. Codet for determining the option type for `ICMPv6OptionSingleType`

generator will replace standard handling for the customized one, according to the definitions provided, if present. The symmetric processing is applied during coding time, and the possible codec hooks are: `PreEncode`, `PreEncodeField`, `PostEncodeField` and `PostEncode`.

It can be seen on the function definition at figure 4 that the CoDec Generator also provides a framework for handling `DecodeError` exception issuing. `EncodeError` exceptions are handled too.

4.4 Summary

The CoDec Generator is a generic tool that automatizes the CoDec development task. It is based on a parser that extract required and available logic already present in standard TTCN-3 ATS and complements it with codets, pieces of platform language code, to build the effective CoDec. As it extracts most of type information from TTCN-3 ATS, the task of repeating TTCN-3 type structure in the platform language is done automatically, removing the error-prone task of type structure synchronization from the test developer. Only codets need to be maintained. The amount of test-specific code becomes smaller, making it easier to maintain and evolve.

5 The IPv6 Testing Toolkit

The *"IPv6 Testing Toolkit"* (also referred as *the toolkit*) is a set of data, functions and basic mechanisms dedicated to TTCN-3 test development and execution of IPv6 test suites. The original idea was to provide a library that provides a higher level of abstraction and specialization to the language TTCN-3 for IPv6 test suite development. As TTCN-3 definition does not provide means for compiled and packaged code distribution, the toolkit is a collection of tightly coupled Open/Free tools, C++ libraries and pieces of TTCN-3 routines and type definitions.

5.1 Scope

The objective of the toolkit is to allow quick design of IPv6 test suites. To achieve that, it addresses TTCN-3 design and maintenance issues, as much as providing off-the-shelf type and data structures required for adequate protocol handling. For our team, it was not possible to address a stable data type definition until we solved the CoDec generation issue. At a certain moment in time, there were 3 engineers developing different abstract test suites in parallel for different IPv6 protocols. Until the CoDec Generator was developed, it was impossible to share the code base of IPv6 definitions. Once the CoDec Generator was available, it became possible to share common definitions of base types and tools through a version managed source repository.

Team efforts on IPv6 testing required working in different parts of IPv6 protocols, ranging from core protocols to OSPFv3 and Network Mobility. Details of protocols implemented are given in 5.2. Figure 5 shows a graphical representation of the components, that helps understanding their correlation.

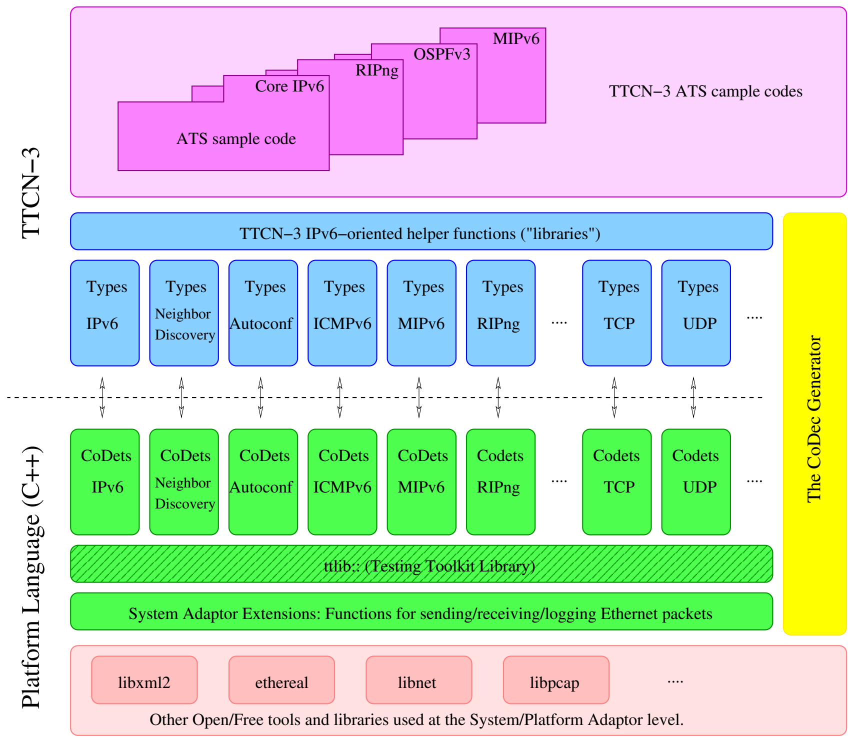

Fig. 5. Graphical representation of toolkit elements

Main types and data structures are readily available and can be used as sort of building-blocks to design test suites. This cannot be achieved completely as TTCN-3 does not provide mechanisms for function definition overriding or library-style distribution.

5.2 The `ttlib-ipv6::` library

As an specialization of the `ttlib::` (library presented in 4.2), the `testing toolkit library` for `IPv6` complements the former by adding functions and tools for handling IPv6 protocols. It comprises TTCN-3 types, functions and C++ codets required for handling IPv6 testing in any toolkit-like environment (not only for our TTCN-3 tool).

The library `ttlib-ipv6::` addresses the following standards:

- **IPv6 transmission over Ethernet**
  RFC2464
- **IPv6 and Options**
  RFC2460, RFC2675, RFC2711, RFC3775, RFC3963
- **IPv6 Extension Headers**
  RFC2460, RFC3775
- **ICMPv6 and options**
  RFC2461, RFC2463, RFC2710, RFC3775
- **IPSEC**
  AH RFC2402, ESP RFC2406
- **NEtwork MObility**
  RFC3963
- **Routing protocols**
  RIPng RFC2080, OSPFv3 RFC2740, BGP4+ RFC1771 RFC2858
- **Transport protocols**
  UDP (RFC2460 RFC768), TCP (RFC2460 RFC793)

Implementation aspects, like endianness, are implemented for standard IPv6 over Ethernet coding and might require revision prior to the porting to other deployment scenarios.

5.3 An IPv6/ICMPv6 example

The objective of the example is to show the usage of the toolkit for two simple operations in ICMPv6 packet transmission: length and checksum calculation. Even though these operations have to be performed on a packet-by-packet basis, they were hard to implement in our first test suites [11]. It was understood that it is an accessory to the test suite, but generic CoDec provided by tool vendors do not support this kind specialized operations easily. On the other hand, TTCN-3 is not adequate for this kind of bit-oriented calculations, and it is clearly a transmission problem. We will do a high level presentation on how to engineer this problem using the toolkit, without getting into deep technical issues which are beyond the scope of this paper.

For a complete understanding of how operations are performed, it would be required to get into details of the TTCN-3 type definition. Let us just concentrate on the functional handling of properties and leave aside complete type description and function invocation details. These aspects alone require a complete section for a good understanding.

Figure 6 shows an extract of the codet that is executed at `PostEncode` time. `PostEncode` is the right moment for length calculation, because it is the last access provided before encoding is finished and low level representation is returned to the TTCN-3 invoking call. At this point, it is supposed that all upper level protocol data is already assembled, source and destination IPv6 addresses too. Length can be thus calculated, and only afterward the checksum (because it covers also the length field, that has to be filled beforehand).

```
  inline void FrameType::PostEncode (Buffer& buffer) throw (EncodeError) {
<snip>
         // IPv6 layer: compute the payload length if not given
         Ipv6HeaderType& ipv6 = layer.Get_ipv6();
         if (ipv6.computeLength_) {
                 UInt16& len = ipv6.Get_PayloadLength();
                 buffer.SetPosition (beginning_of_layer[id]);
                 len.SetValue (buffer.GetBitsLeft()/8 - 40);
                 buffer.SetPosition (buffer.GetPosition() + 32);
                 buffer.Write (len);
          }
<snip>

          // ICMPv6 layer: compute the checksum if not given
          ICMPv6MessageType& icmpv6 = layer.Get_icmpv6();
          if (icmpv6.computeChecksum_) {
                 Unsigned& checksum = icmpv6.Get_Checksum();
                 checksum.SetValue (
                       ChecksumIPv6 (ip6_pshdr, buffer, id, 2)
                       );
                 buffer.SetPosition (beginning_of_layer[id]);
                 buffer.SetPosition (buffer.GetPosition() + 16);
                 buffer.Write (checksum);
           }
<snip>
  }
```

Fig. 6. ICMPv6 Codet fragment for checksum and length calculation

The tags `<snip>` indicates parts of the source code that were removed for the sake of simplicity. The first thing that is performed is the length calculation and stored in the buffer in the right position. More code is removed to keep the example simple. The checksum calculation function, `ChecksumIPv6()`, is part of the library `ttlib-ipv6::` and can be simply invoked. Afterward the checksum is stored in the transmission buffer and the packet assembly is completed.

The removed pieces of code do not affect calculation and that the remaining part of the codet is what really performs the calculation. This example shows a good compromise between what can be automatized, what shall be provided by the toolkit and what has to be specified in the platform language in the form of a codet. This codet is part of the toolkit and shall be used, unless it is required to replace it due to some test requirement.

6 Conclusions & future work

In this work, we presented the *IPv6 Testing Toolkit*, developed in the IRISA Laboratory. It comprises a set of libraries, type definitions and tools that help in the task of IPv6 protocol test suite generation. Moreover, the toolkit itself is more than a library, because it addresses TTCN-3 gray areas in the transition from TTCN-3 Abstract Test Suites (ATS) to Executable Test Suites (ETS) and proposes a solution for problems related to development, reuse and maintenance. The toolkit main building element is the CoDec Generator, an IPv6 independent tool that addresses the problem of CoDec development.

The result is a set of free tools, either for IPv6 protocol testing or general protocol testing. CoDec development complexity was moved into the CoDec Generator, but it can be reused and maintained independently of the test suites. Test suite ”source code” now only contains TTCN-3 ATS and codets, which are platform language extensions helpers for the CoDec Generator. The main practical result is that these tools simplified our tasks of test development and maintenance.

The toolkit is not complete already. It is being re-engineered and re-factorized now that we are confident that it is useful and it will become part of our set of tools. Future plans include to extend the CoDec Generator to handle Java codets, so as to implement all TTCN-3 environments. We were also working on different ways for codet specification. A platform independent codet language is under analysis too.

References

1. ETSI. ES 201 873-1 Part 1: TTCN-3 Core Language, Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187301v030101p.pdf, 2005. [Online; accesed 19-April-2006].
2. ETSI. ES 201 873-1 Part 2: TTCN-3 Tabular presentation Format (TFT), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187302v030101p.pdf, 2005. [Online; accesed 28-April-2006].
3. ETSI. ES 201 873-1 Part 3: TTCN-3 Graphical presentation Format (GFT), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187303v030101p.pdf, 2005. [Online; accesed 28-April-2006].
4. ETSI. ES 201 873-1 Part 4: TTCN-3 Operational Semantics, Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187304v030101p.pdf, 2005. [Online; accesed 28-April-2006].
5. ETSI. ES 201 873-5 Part 5: TTCN-3 Runtime Interface (TRI), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187305v030101p.pdf, 2005. [Online; accesed 19-April-2006].

6. ETSI. ES 201 873-6 Part 6: TTCN-3 Control Interface (TCI), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187306v030101p.pdf, 2005. [Online; accesed 19-April-2006].
7. Jens Grabowski and Dieter Hogrefe. Towards the third edition of ttcn. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 19–30. Kluwer Academic Publishers, 1999.
8. Jens Grabowski, Anthony Wiles, Colin Willcock, and Dieter Hogrefe. On the design of the new testing language ttcn-3. In Hasan Ural, Robert L. Probert, and Gregor v. Bochmann, editors, *(TestCom 2000) Testing of Communicating Systems, Tools and Techniques, ISBN 0-7923-7921-7*, pages 161–176. Kluwer Academic Publishers, 2000.
9. Theofanis Vassiliou-Gioles, Ina Schieferdecker, Marc Born, Mario Winkler, and Mang Li. Configuration and execution support for distributed tests. In Gyula Csopaki, Sarlota Dibuz, and Katalin Tarnay, editors, *12th IFIP International Workshop ont Testing of Communicating Systems, Testing of Communicating Systems - Methods and Applications, ISBN 0-7923-8581-0*, pages 61–76. Kluwer Acdemic Publishers, 1999.
10. Jianping Wu, Whongjie Li, and Xia Yin. Towards Modeling and Testing of IP Routing Protocols. In Dieter Hogrefe and Anthony Wiles, editors, *Testing of Communicating Systems In 15th IFIP Testing of Communicating Systems, ISBN 3-540-40123-7*, pages 49–62. Springer, 2003.
11. A. Sabiguero and A. Baire and A. Floch and C. Viho. Using TTCN-3 in the internet Community: an experiment with the RIPng protocol. TTCN-3 User Conference 2005 - 6-8 June, Sophia-Antipolis, France (http://www.ttcn-3.org/TTCN3UC2005/program/Tuesday%207th%20June/Session1/02-UsingTTCN-3intheinternetcommunity.pdf), 2005.
12. Ethereal: A Network Protocol Analyzer. http://www.ethereal.com/, 2006. [Online; accesed 19-April-2006].
13. PDML Specification. http://analyzer.polito.it/docs/dissectors/PDMLSpec.htm, 2006. [Online; accesed 19-April-2006].
14. tcpdump/libpcap. http://www.tcpdump.org/, 2006. [Online; accesed 27-April-2006].
15. The Libnet Packet Construction Library. http://www.packetfactory.net/libnet/ (last ckecked 27/04/2006), 2006.
16. Ariel Sabiguero, Anthony Baire, and César Viho. Embeding traffic capturing and analysis extensions into TTCN-3 System Adaptor. In Winfried Dulz and Wolfgang Schröder-Preikschat, editor, *MMB Workshop Proceedings: Model Based Testing and Non-Functional Properties of Embedded Systems, ISBN 978-3-8007-2956-2*, pages 27–35. VDE Verlag, 2006.
17. Bison. http://www.gnu.org/software/bison/, 2006. [Online; accesed 22-April-2006].
18. Flex. http://www.gnu.org/software/flex/, 2006. [Online; accesed 22-April-2006].