Derivation of Abstract Protocol Type Definitions for the Conformance Testing of Text-Based Protocols

Stephan Schulz

Nokia Research Center
P.O. Box 407
FIN-00045 NOKIA GROUP
Stephan.Schulz@nokia.com

Abstract. In today's specification and implementation of conformance test systems the derivation of a proper abstract protocol type definition for text-based protocols remains an open issue. Contrary to conventional telecommunication protocol standards, text-based protocol standards do not specify such an abstract protocol syntax but instead its transfer syntax, i.e., information about message content in conjunction with its encoding. We present in this paper an approach, which allows to extract an abstract protocol type definition from a text-based protocol transfer syntax specified using the Augmented Backus Naur Form (ABNF). In addition, we briefly discuss how a protocol independent codec system can be implemented based for such protocol type definitions.

1 Introduction

The specification of conformance test suites, which verify that a protocol implementation adheres to the protocol syntax and semantics as specified in a specification or standard, has only recently started to be practiced for text-based protocols. Arguably one reason for this late recognition is that the ideas and methodology for conformance testing originate from the telecom community [1] whereas most text-based protocols have been standardized in the internet community, which has a separate standardization body as well as a different testing philosophy. The lack of established conformance testing has led to the common situation that correct protocol syntax and semantics for text-based protocols are commonly specified by an implementation (of an influential software vendor) rather than the protocol standard.

When looking at text-based protocol syntax definitions and their standards from a telecommunication perspective we notice a number of differences: Firstly, protocol syntax is specified as a transfer syntax - most commonly using the Augmented Backus Naur Form (ABNF) [13] - instead of an abstract syntax, e.g., using the Abstract Syntax Notation One (ASN.1) [2], along with an encoding rule, e.g., Basic Encoding Rules (BER). A transfer syntax like ABNF mixes message content (i.e., its abstract syntax) with encoding information. Possibly this approach for protocol syntax definition originated from the fact that textual encoding, contrary to its binary counterpart, was initially readable for humans. Nevertheless some textual encoding can already today turn out to be barely human readable, e.g., large textually encoded Media Gateway Control (MEGACO ABNF) [3] messages. Secondly, multiple ways

R. Groz and R.M. Hierons (Eds.): TestCom 2004, LNCS 2978, pp. 177–192, 2004.

of encoding the same information are commonly acceptable, e.g., short and long forms for text tokens, or the appearance of multiple headers in encoded Session Initiation Protocol (SIP) [4] messages. Also, the concept of white space, which is a well defined sequence of characters contributing no information content to a message for a given protocol, is a previously unknown concept to the telecommunication protocol world.

Another possible reason for the late start of conformance testing of text-based protocols may be that classical conformance testing has not properly addressed the testing needs of such protocols. Only recently the best established testing language in conformance testing, the Testing and Test Control Notation (TTCN), was extended to better support the testing of text-based protocols in its third release, TTCN-3 [5], which includes for example numerous string manipulation functions as well as regular expressions for the matching of strings.

But even the best testing language alone is only a start to a good conformance test suite. In order to minimize the source of error from test case specification and to increase its coverage, it is highly desirable to provide an abstract protocol type definition to test engineers which relieves them from the task to assure correct encoding of message content. As text-based protocol standards do not provide an abstract syntax, protocol type definitions need to be derived from the transfer syntax. Tightly coupled to the abstract type definition is the implementation of a textual encoder and decoder in a test system which automates the transition from abstract syntax to transfer syntax and vice versa. This part is more commonly called the "codec" implementation. In our paper, we present the derivation of abstract type definitions from the ABNF definition of text-based protocols. We also outline the implementation of a protocol independent text codec based on our approach.

2 Related Research

The first TTCN-3 conformance test suite for text-based protocols was developed by the Telecommunication and Internet Protocol Harmonization Over Networks (TIPHON) workgroup at the European Telecommunication Standardization Institute (ETSI). This group wrote a SIP test suite for testing voice over IP call handling using SIP, which was released for the first time in the summer of 2002 [6]. A major TTCN-3 SIP test suite update was then released in the first quarter of 2003. In both test suites, the origin of the SIP TTCN-3 type definition is unclear. The best presentation of their efforts can be found in [15]. The paper leaves still a lot of questions open, e.g., the approach taken to derive the actual TTCN-3 SIP type specification. ETSI SIP test suite implementers have mentioned to the author that internal representation of SIP messages in existing open source SIP implementations probably had a significant impact on the design of the protocol type definition. Notice that this work only encompassed TTCN-3 code but not the implementation of the remaining parts of test system, e.g., a SIP codec, which is required to make this TTCN-3 test suite executable.

The first executable commercial TTCN-3 test system for SIP was released by the Berlin based company <testing_tech> [7]. Their latest release constituted in essence a commercial full TTCN-3 test system release for the second ETSI SIP test suite. Therefore their abstract protocol definition was the same as specified by ETSI. Their codec implementation is mostly based on an open source SIP parser implementation.

The approach taken in both above cases, that is, to base the abstract protocol definition and its encoding for a TTCN-3 conformance test system on an open source parser implementation has two major drawbacks: First of all, the granularity of such a TTCN-3 type specification is likely to be driven by this parser implementation and not by the test purposes and syntax definition as it should be. Secondly, if there is a complete reliance on the codec implementation data structures, problems also arise when aspects of a protocol are to be tested which are not yet supported by any public parser implementations, e.g., 3GPP specific SIP extensions. Similarly, this solution is only suitable for protocols which have open source implementations available.

In other research, also at ETSI, S. Müller [8] has investigated the conformance testing of the text-based Open Settlement Protocol (OSP) using TTCN-3. In his documentation, he was the first to clearly and explicitly separate textual encoding from actual message content in a TTCN-3 data type definitions by isolating encoding information in TTCN-3 encoding attributes. Arguably the scope of his research was a little different than in the previous SIP cases. Contrary to SIP, OSP has a far simpler transfer protocol syntax and it is based on XML instead of ABNF.

Finally we want to mention that there have also been different approaches to test SIP, e.g., using XML [9] instead of TTCN. We feel a XML based test specification approach is not appropriate for conformance testing. Although XML is a widely accepted specification format it has not been designed nor standardized for conformance testing purposes. In essence, such an approach requires from its users to specify testing infrastructure, e.g., a rich type system, sophisticated matching mechanism, dynamic test configurations, test verdict management, which are already native in TTCN-3.

3 The Augmented Backus Naur Form

The Augmented Backus Naur Form (ABNF) is probably the best established method to specify protocol transfer syntax for text-based protocols in standardization. In particular, ABNF is used to specify the transfer syntax of many important text-based protocols which are to be used in future third generation telecommunication networks including Session Initiation Protocol (SIP) [4], Session Description Protocol (SDP) [10], the textual form of the Media Gateway Control protocol (MEGACO ABNF) [3], Real Time Streaming Protocol (RTSP) [11], and Hypertext Transfer Protocol (HTTP) [12]. This form of specification has been defined by the Internet Engineering Task Force (IETF) in their Request For Comments (RFC) 2234 [13]. Notice however that ABNF could also be used to described the syntax of text-based protocols which use currently another basis of specification, e.g., XML. In this section we provide a brief introduction into this specification format followed by an discussion of its limitations.

3.1 The Main Concepts

A ABNF based transfer syntax specification for a given protocol is generally specified by a set of ABNF rules, which first gradually define the structure of textually encoded messages and eventually specify a textual encoding for all possible atomic values. Such atomic values, i.e., rule elements which can not be further decomposed, are referred to as terminals. Each ABNF rule can be composed of one or

more elements which may be combined using the operators defined in [13]: concatenation, optionality, repetition, and alternation. Each element may either be fixed character string or single character, i.e., a terminal, or a reference to another rule. Finally, an ABNF specification may also include descriptive comments. We illustrate these main concepts with some small ABNF rule examples taken from the SIP standard [4].

The concatenation operator defines that the elements this rule definition must always follow each other in the specified order in a correctly encoded protocol message. The optional operator indicates that one or more elements may be but also may not be present at the location specified by the rule.

SIP-URI = "sip:" [userinfo] hostport uri-parameters [headers] (1)

Our first ABNF example rule (1) illustrates a combination of concatenation and optional operators. Here the rule <SIP-URI> defines that its correct encoding must have the terminal "sip:" followed by the result of resolving rules <hostport> and <uri-parameters>. The encoded strings which are the result of resolving rules <userinfo> and <headers>, however, may be omitted in a syntactically correct encoding of a message. If, however, one or both of them are present then they must occur in the specified positions, i.e., between the terminal "sip:" and the encoded element <hostport> and immediately following the encoded element <uri-parameters>, respectively.

Via = ("Via" / "v") HCOLON via-parm *(COMMA via-parm) (2)

Our second example rule (2) illustrates that ABNF rules may contain more than one and different ABNF operators. Firstly, the concatenation, which is shown as a sequence listing of rule elements, mandates that the result of encoding the first alternation operator must be followed by the result of resolving rules <HCOLON>, <via-parm>, and finally that of the repetition operator. The alternation operator, illustrated as a backslash, requires that either the terminal string either "Via" or "v" to be present in a valid encoded message. The repetition operator, which is shown with a star followed by element to be repeated within parenthesis, always indicates that zero or more (i.e., no upper limit) occurrences of the element may be repeated at this point of the encoded string. Numbers before and after the star can be used to specify lower and upper bounds on the allowed number of repetitions.

A common naming convention found in ABNF specifications is that capitalized rule identifiers identify rules which specify basic values, e.g., <HCOLON> used in example rule (2), which is in [4] defined as the string ":" which may be preceded and superceded by white space.

3.2 Limitations of ABNF-Based Specification of Textual Encoding

Generally we find that an ABNF based specification textual encoding is not very concrete. Very commonly we find the concept of using repetition in combination with an alternation which essentially allows for information to occur in any given order and any number of times (including not at all). The problem here is that in practice the full capability of this loose specification is only rarely made use of but still the worst case needs to expected. Our example rule (3) from [14] specifies the list of parameters for HTTP digest computation. Based on a literal interpretation of this ABNF rule

multiple occurrences of every element are allowed. Multiple occurrences however only make sense in this case for the extension element <auth-param>. Notice that this example also uses two extensions to the ABNF notation which are not defined in [13]: the hash sign, a shorthand notation to specify that repeated encoded elements are to be separated by a comma, and the optional operator for an alternation element.

```
digest-challenge  = 1#( realm / [ domain ] / nonce / [ opaque ] / [ stale ] |      (3)
                        [ algorithm ] / [ qop-options ] / [ auth-param ] )
```

A considerable difference compared to conventional telecom encoding is that many text-based protocols allow multiple forms of encoding to represent the same information. One example for that is the compact form concept which is commonly used, e.g., in SIP [4] and MEGACO ABNF [5]. The compact form allows that encoded messages may either use an explicit or an abbreviated form of a string token to indicate the presence of some information. An example of a compact form specification is our rule (1) which allows "Via" or "v" to be used to indicate the presence of a “Via” header field. One problem with such multiple forms of encoding is that their equivalent information content is not apparent from the ABNF itself, but it is instead specified in the verbal sections of text-based protocol standards, e.g., Sections 7.3.3 in the SIP standard [4].

A second difference to binary encoding schemes is the concept of compressible white space. Such white space can be loosely defined as a set of possible characters - usually including extra space, tabulator, and line feed characters - which may appear in various combinations at well defined positions within an textually encoded message. It does not contribute any information about the content of the message or even its encoding. Although the concept of compressible white space is a general one, it is not consistently defined across different text protocol standards.

The biggest problem with ABNF based transfer syntax specifications, which may be a result of the weakness of the ABNF specification format itself, are in general significant restrictions specified upon message encoding in the *verbal* sections of text-based protocol standards or within ABNF comments. The SIP standard [4] in Section 20.10, for example, prohibits the appearance of URL parameters and headers in the plain addresses within “Contact”, “To” and “From” header fields which are not reflected in the definition of their ABNF rules. Another example is Table 2 in the SIP standard which specifies optional and mandatory headers for different SIP request kinds, whereas the SIP ABNF only defines one single SIP request kind which if literally followed allows the appearance of any header (in any number). The comment for the <serviceChangeParm> rule in the MEGACO ABNF specification [3] - an alternation - requires two of its alternation elements to be always present and only one of two other elements to be present at any time, which severely restrict this ABNF rule.

4 Derivation of Abstract Syntax from ABNF Specifications

Now that we have an idea on how textual encoding can be defined using ABNF, we want to present an approach which can be used to extract an abstract syntax from ABNF based transfer syntax specifications. More concretely we want to isolate the information from its encoding as much as feasible. We discuss here first a one-to-one

mapping of ABNF rules to type definitions. After that we will discuss how the results of this mapping can easily be improved further - especially in the context of the previously discussed limitations of ABNF based specifications.

We will use the TTCN-3 type system to illustrate our derived abstract syntax, i.e., our abstract protocol type definition. Notice that our derivation itself is also applicable to other Abstract Syntax Notation One (ASN.1) [2] like type systems. We still promote the use of TTCN-3 as in our opinion only this testing language currently offers the required language support, e.g., access, checking, and manipulation of character string values, to properly specify conformance test cases for text-based protocols.

4.1 Requirements for an Abstract Syntax Definition

First we would like to specify the requirements guide the derivation:

1. The abstract syntax shall follow the ABNF structure and naming as closely as possible. That means that based on the name of a data type the test engineer should be able to identify the location in the ABNF specification where its possible (encoded) values are defined.
2. The protocol type definition should consist of a minimum amount of data type definitions, i.e., its specification complexity should be minimized. It is in our opinion this is a critical requirement to derive protocol type definitions comprehensible to a test engineer.
3. When a textual encoding is much more intuitive than their more structured counterpart for an information element, e.g., IPv4 address specification, then the less structured representation should be chosen in the protocol type definition as long as our previous, second requirement is met.
4. The level of structuring imposed by the abstract protocol type definition must at least allow direct access to all information within a protocol message which are to be validated by a given test purposes or needed in order to communicate sensibly with the implementation under test, i.e., the test engineer shall not need to parse for information within an information element.
5. A protocol type definition must only be based on one single encoding of each information element. That format should be the more readable to the test engineer, e.g., long format instead of the compact format.
6. The derived data types shall enable a test engineer at least to send all possible valid message content. This requirement does not imply that all valid encodings are to be supported for sending purposes. Similarly, it does not exclude the possibility to accept protocol messages that arrive with different encoding than the selected one or with invalid content.

4.2 Handling of White Space and Multiple Encoding Representations

Since white space does not represent any information of value for conformance testing it does not get any representation in our abstract protocol definition. This in essence allows us to edit a given ABNF based protocol transfer syntax specification and substitute all occurrences white space in ABNF rules with its minimal

representation, i.e., no or a single space terminal. That also means that test engineers have no means to specify or test for valid extra white space occurrence in a message which is specified based on this abstract protocol type definition.

Another result of the requirements listed in the previous section is that only one of multiple encoding schemes has to be selected in prior to the derivation of the abstract data types. In this case, we advise to choose the long and more meaningful form over the compact form. Therefore, like in the case of white space handling, this could be interpreted as a replacement of all occurrences of compact form related alternation rules in the ABNF specification with the long form terminal only.

4.3 Delimiters

Before we start our discussion of our derivation we still have to introduce the concept of delimiters. Delimiters encode structural information in text-based protocol messages. In the context of ABNF, a delimiter is usually one but possibly multiple concatenated terminal strings in a ABNF rule definition which identify an encoded ABNF rule element, i.e., another ABNF rule definition or basic encoded value. Examples for multiple terminal strings constituting a single delimiter are <"Via" HCOLON> in our example rule (2) and <ErrorToken EQUAL> in our rule (4). Notice that both elements in these rules define only a single terminal or character string since short form as well as white space definitions have been removed in our ABNF specification as discussed in the previous section.

errorDescriptor = ErrorToken EQUAL ErrorCode
LBRKT [quotedString] RBRKT (4)

In delimiter kinds, we can firstly distinguish between element *pre-* and *post-* delimiters which may be specified for any element in a given ABNF rule. Element pre-delimiters have to precede in the encoded message the result of encoding a rule element, whereas post-delimiters must succeed the encoded form of an element. An example of a pre-delimiter for a rule element is the <COMMA> used in the ABNF rule definition in example rule (2).

In ABNF rules, which are defined using the concatenation operator, e.g., our rule (2), only optional elements allow a unique classification of a delimiter to be a pre or post-delimiter of that element and its preceding and following element. When two mandatory elements follow each other, however, a delimiter may be interpreted either as a post-delimiter of the first rule element or a pre-delimiter of the second element. Although from theoretical perspective the actual choice is irrelevant, from a practical perspective, i.e., the underlying codec implementation, choosing pre- over post-delimiter may be preferable to speed up the parsing process during decoding.

For ABNF rules, which are defined as a concatenation, there exists a similar concept of *rule* delimiters in addition to *element* delimiters. Rule pre- and post-delimiters follow the same principle as their element delimiter counterparts, only that they identify encoding information for the entire encoded rule instead of merely one element within that rule. Example rule (4) from [3] illustrates a rule post-delimiter <RBRKT>. The <RBRKT> rule can not be classified as a element post-delimiter of <quotedString> since its presence in an encoded message is only optional. Similarly like in the case of the element delimiters, a first element pre-delimiter may frequently

be interpreted either as rule or element pre-delimiters, e.g., <ErrorToken EQUAL> in example rule (4), and last element post-delimiters may be interpreted either as rule or element post-delimiters. Again the best classification of the rule versus element delimiter may depend on the underlying codec implementation.

Although delimiters are a key concept in transfer syntax specification for text-based protocols, they do not have to be specified in every rule for every element. Frequently many ABNF rules do not have any delimiters specified [3]. This only occurs for rules based exclusively consisting of rule reference elements which structure the element further. Here, the referenced rule usually provides a pre-delimiter for its first element. And example for this is the <uri-parameters> field in our example rule (1) which appears to have no delimiters with its preceding <hostport> field. The rule <uri-parameters>, however, is defined as a repetition with the element pre-delimiter <COMMA>.

4.4 Mapping ABNF Rules to Abstract and Basic Data Types

In this section, we define a direct mapping of ABNF rules to different abstract data type definitions. This mapping assumes that each ABNF rule is specified using only one concatenation, alternation or repetition operator. As shown in example rules (2) and (3), these operators frequently occur in combination. In these cases the ABNF rule must be split into one rule per operator which reference each other.

4.4.1 Rules Defining Basic Encoded Values

The first step of our mapping is an analysis of the ABNF specification which identifies all ABNF rules specifying a textual encoding of basic values other than a character string, i.e., real numbers, whole numbers, and hexadecimal values.

The encoding for whole numbers is generally specified in a repetition of the <DIGIT> rule, e.g., in the <CSeq> rule of the SIP ABNF. It may also be specified as an alternation of terminal strings which encode specific number. Such rules should mapped to, e.g., the TTCN-3 integer type. Range restrictions or value lists should be used in the particular integer type definition to reflect, e.g., the limits in the digit repetition.

The encoding for real numbers is generally specified as a concatenation of a whole number followed by a "." and another whole number, e.g., in the <MIME-Version> rule of the SIP ABNF. This rule can be mapped to an TTCN-3 float type.

The encoding hexadecimal values is generally specified in a repetition of the <HEXDIG> rule, e.g., in the <SecurityParmIndex> rule of the MEGACO ABNF. Such rules should mapped to, e.g., the TTCN-3 hexstring type but also to the TTCN-3 octetstring type if applicable. Again range restrictions should be applied to reflect the ABNF repetition restrictions. Notice that encoding information, e.g., a pre-delimiter "0x", has to be contained in the abstract type definition either by creating hexstring alias type for this value or specified as field encoding information in its parent structured type(s) (see Section 4.4.3).

4.4.2 Rules Defining Alternations of Terminal Strings and Tokens

ABNF rules, which specify an alternation of terminal strings are mapped to a character string type, e.g., the TTCN-3 charstring or universal charstring. In this case, terminals are considered to not specify encoding information but a character string value. More specifically, the correct mapping of such an alternation is a character string type which is subtyped with a value list corresponding to the terminals used in the alternation.

The second use of character string type is to represent the commonly used <token> rule or its equivalent. This rule commonly acts as a placeholder in ABNF specifications for "any valid string value" or future protocol syntax extensions. Here, the use of subtyping is desirable but usually not practical.

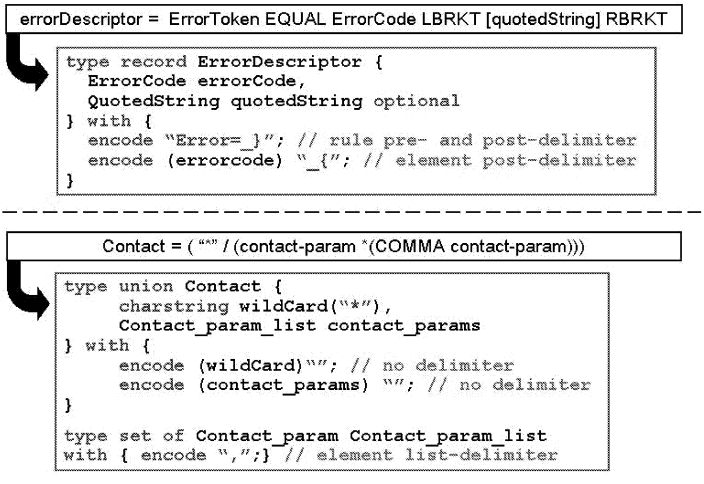

Fig. 1. Example mappings of ABNF rule (4) and <Contact> [4] to structured types

When fields in structured types (see following section) are of character string type without any subtyping restrictions, the underlying codec implementation should check for ABNF specified syntax for that particular part in the encoded message. In cases, where the specified transfer syntax is not checked by the codec, test engineers must check compliance of character string values to be sent or received in charstring type fields within a test case, e.g., by using regular expressions in the corresponding TTCN-3 receive templates. Similarly as for hexadecimal values, encoding information of character string values, i.e., pre- and post-delimiters, has to be either captured by defining a charstring type alias or within its parent structured type(s).

4.4.3 Other Rules

Rules which do not only specify a basic value are mapped to structured types: rules which are defined as concatenations to an ordered sequence type, e.g., a TTCN-3 record type, alternations to a choice type, e.g., a TTCN-3 union type, and repetitions to an unordered list type, e.g., a TTCN-3 set of type. Optional elements within a

concatenation get mapped to an optional field in the ordered sequence. Some mapping examples are shown in Figure 1. Notice that the concatenation in the second example <contact-param *(COMMA contact-param)> has been mapped to an unordered list based on our second simplification which we present later in Section 4.5.In the mapping, the name of the rule becomes the name of the structured type and the name of rule elements become field and field type names in the structured type definition. Notice again that information about delimiters used in these rules is not included in their structured type definition directly but may be reflected as encoding information. Our use of TTCN-3 encoding attributes will be discussed in more detail in Section 4.4.5.

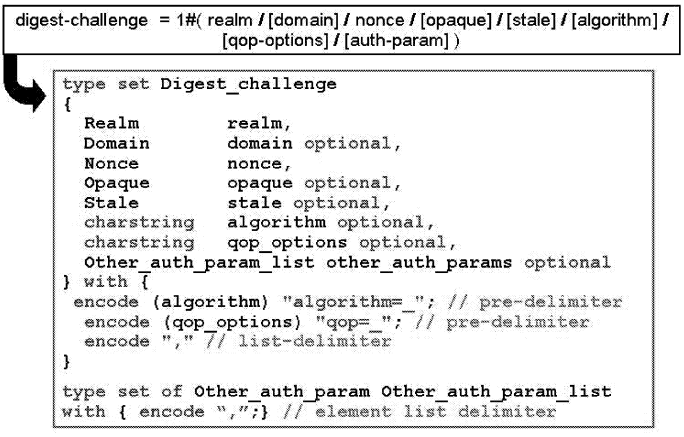

Fig. 2. Example Mapping of ABNF rule (3) to a TTCN-3 set type

4.4.4 Repeated Alternations

Repeated alternations, e.g., our rule (2), we treat as a special cases in our mapping. Instead of mapping them individually to an unordered list and a choice type, we map them to an unordered list type, e.g., the TTCN-3 set type, where in our first mapping each field is optional and of an unordered list type.

The reason for mapping to an unordered sequence of unordered lists is that the latter has more expressive power than the alternative approach. In a unordered sequence of unordered lists, fields can be specified as optional or mandatory whereas an unordered list of choice does not allow such restriction. Also it allows more easily to place further restrictions on the number occurrences of individual alternation elements as we see in Figure 2. The problem which however remains is that the ABNF specifications itself do not necessarily contain the required information to make these restrictions, e.g., is a given alternation element mandatory. Such information is however frequently specified in the verbal section of text protocol standards. Therefore, in a second step we may then define the required presence of a particular unordered list field to mandatory.

Another simplification in this mapping is that commonly multiple occurrences of some alternation element(s) can not occur. In these cases, the type of the particular

unordered list field should be changed from its unordered list type to directly refer to type definition for the alternation element. If the alternation element however is an extension placeholder, its field must always be defined as an unordered list type as illustrated in Figure 2. Only the <auth-param> field is specified here as an unordered list since only it may occur multiple times within a HTTP digest challenge.

A strong benefit of this special handling is that the abstract type specification is tightened but also that the access of values for rule elements becomes significantly easier within test cases. The alternative - a mapping to an unordered list of choice – would require a search through the list of elements whereas the unordered sequence allows a direct access of a desired element via its field name.

A drawback of this mapping is that the order used in the order of field value encoding can no longer be influenced to the same extend in sending, e.g., digest challenge values, within a test case as it would be if an unordered list of choice type definition was used. In the latter case it would be possible to send of multiple values from different fields in an interleaved manner. We argue that it is acceptable to loose this capability since it does not change the *information conten*t of a message - merely its *encoding*. Notice that in the decoding of unordered sequence values in the underlying codec implementation must of course be able to handle such scenarios.

4.4.5 Retaining Encoding Information

As we discussed in the previous sections, delimiters, i.e., encoding information, are eliminated in our mapping to basic and structured data types. In principle, there are two ways to retain this information.

One approach is to use structured type names or type aliases for basic types to enable to an underlying codec implementation to perform a lookup of encoding information via the type name. This solution as the drawback that protocol specific information is split across two information sources, one being the type definition and the other being an external table which maps type names to encoding information.

Arguably a more elegant solution is to use encoding attributes in the abstract type definition, as proposed in [8], to capture such encoding information. Firstly, it allows test engineers to relate each abstract type better to an ABNF rule in the text protocol standard, but more importantly it keeps all the information pertaining to a specific abstract protocol definition in one place.

In the latter approach, rule delimiters would be specified as TTCN-3 type encoding attributes and element delimiters would be specified as field encoding attributes. This has been illustrated in Figure 1 and 2. In the special case of the repeated alternation mapping the type encoding attribute of the unordered sequence would contain the delimiters specified for the repetition element and its field delimiters would contain the delimiters specified for each alternation element. Secondly, unordered list types used for unordered sequence fields would also use the repetition delimiters.

Notice that in the specific case of TTCN-3, encoding attributes have the limitation of allowing only a specification of a single string per field or type which conflicts with our need to specify pre- and post-delimiters. We show here one possible way of solving this problem is by encoding pre- and post-delimiter information for a given field, as shown in the previous Figure 1 and 2 (where a string if is present before an underscore character specifies a pre-delimiter and if present after specifies a post-delimiter).

4.4.6 Observations

In our mapping, string terminals may be either classified as encoding information or message content. This classification is not uniquely possible when an ABNF rule element models a signal, e.g., MEGACO ABNF <KeepActiveToken> rule [3]. In that case, the terminal should be interpreted as message content, i.e., a character string value.

Another possibility than representing an alternation of terminal strings by subtyping the character string type is to map this construct instead to an enumeration type. This approach has been taken in the ETSI SIP protocol type definition in [6]. It is also valid but has the drawback that the testing language limits the choice in enumeration value names which makes it hard to create an intuitive relation between enumeration name and value when the terminal string contains special characters. A drawback of the character string mapping in practice is that not all TTCN-3 tools may offer access to subtyping information for codec implementations.

One general problem which remains for our proposed mapping is the number of type definitions it generates, which can be significantly larger than the number of rules in the ABNF specification. The main reason for that is the necessity to have only one concatenation, alternation, and repetition per rule. Another issue is that the mapping only reflects aspects covered or expressible in the ABNF notation. There may however exist a number of restrictions in the informal part of a text protocol standard which could improve, i.e., tighten, the abstract protocol type definitions.

4.5 Further Improvements

As mentioned in our previous sections, text-based protocol standards frequently further restrict the valid textual encoding of protocol messages specified by ABNF in their informative sections. If these restrictions are not reflected in the protocol type definition, the protocol definition will allow the receipt of incorrect messages and leave the checking of their syntactical correctness to the test engineer. In most cases, these aspects can be compensated for within the test specification, e.g., checking that an omitted value is never received in a field which is declared optional by the mapping but mandatory in the informal section.

There is however one case where the abstract protocol type definition must be modified, which is when the arrival of list items is restricted to be ordered, e.g., via header field values in [4]. Here, the corresponding unordered list specifications must be changed an ordered list type, e.g., a TTCN-3 record of type, e.g., the derived “ViaParmList” set of type in Figure 1 must be modified into a record of type. Notice that there is no other alternative for a test engineer to reconstruct this particular information, i.e., the order of arrival.

All other improvements listed in this section are not mandatory but they help to solve the general problem of large number of type definitions or making abstract type definitions for text based protocols more readable:

1. Single field ordered sequences may be eliminated if that field is not optional. One reason to not remove such an ordered sequence may be that it carries some meaning, e.g., MEGACO ABNF [5] descriptors. In the case of a removal of the ordered sequence its delimiters or encoding information must be moved to its parent or child type definitions.

2. In ordered sequences where a field followed by a field list type of the same type, e.g., in the mapping of the <via-parm> and <via-parm> repetition in our example rule (2), can be simplified by removing that field from the sequence and placing lower length bound of 1 upon the list definition. This simplification also creates the need for a third element delimiter kind, the list-delimiter. A list delimiter must only appear between list elements but not before the first nor after the last element of a list value.
3. Structured type definitions, which have field types of the same type kind, may be combined into one definition, e.g., when a choice which defines the type of a field in another choice type they can be combined into a single choice. A condition for such a modification is that structured field type in question is only used in the definition of that structured type. In the specific case of choice types the parent choice also must not specify any element delimiters for the child choice type.
4. More aggressive subtyping restrictions may be defined for fields of basic type based on informal statements in the standard about the corresponding ABNF rule, e.g., MEGACO ABNF rule <UINT16> [3].
5. Finally, the character string type or aliases of it may be used is to abstract or replace unnecessarily complex type structures in appropriate places of a protocol type definition. This applies for example when a value is more intuitive to handle in its encoded form than its structured form, e.g., IP addresses or the SIP caller identification string. Secondly, this approach may be used if it is possible to ignore parts of protocol messages which are not of interest for a given test purpose, e.g., irrelevant SIP headers. As discussed in Section 4.4.2 and [15] this simplification in the protocol type definition may require test case engineers to check received values more closely for correctness.

5 A Generic Codec System Implementation

At Nokia the described derivation of protocol type definitions has been applied in a number of case studies in conformance testing for different text-based protocols, i.e., different SIP, SDP, and MEGACO ABNF test suites. These studies have shown that it is possible to implement one generic, protocol independent codec system based on the traversal of abstract TTCN-3 protocol type definitions, which can be used in real world text-protocol conformance testing. All protocol type definitions used in these studies have been derived from ABNF specifications as discussed in this paper. The implementation of the protocol type definition analysis by the codec implementation has been based on a TTCN-3 Control Interface (TCI) [16] like tool independent TTCN-3 type and value interface. In general the codec implementation can be split into three parts: an encoder, a pre-processor, and a decoder.

The purpose of the pre-processor is to examine and modify received encoded messages prior to passing them to the decoder, so that they conform to the encoding representation selected for the derivation of the TTCN-3 protocol type definition, e.g., to replace any occurrence of short with their corresponding long token forms or to combine multiple SIP header field occurrences into on header field value. Secondly, the pre-processor either removes or reduces white space in the places specified by the respective protocol standard.

The encoder traverses the TTCN-3 protocol type definition at run-time while building of the encoded message to be sent. From each structured type definition encoding information is read from TTCN-3 encoding attributes and added to the target encoded string. Basic values are converted into their corresponding string format and possibly also encoding information is added.

The decoder receives a normalized encoded message from the pre-processor and then gradually converts it into a structured value by traversing again the TTCN-3 protocol type definition at run-time. Given a specific structured type in the protocol type definition and the current position in the encoded message, the decoder attempts to find encoding information for the current structured type definition, e.g., its delimiters, at the proper places in the encoded message. If successful, a corresponding value is constructed, the current position in the message is advanced, and depending on the value found the next type definition is selected and checked for presence. Depending on the optionality of a type in its parent type(s) a failure to find encoding information for a given field of a structured type may be acceptable or not. If the information is optional, then the analysis attempts to check for the presence of the next field or other feasible type information in the encoded string. In case of a failure for mandatory presence, a decoding failure is reported. Basic types are treated similarly. Here, the decoder also attempt to find the encoded value and converts that information at the current position in the encoded string into basic value upon success. Given that an optimization can in most cases detect the presence of optional fields based on the presence of pre-delimiters, this decoder implementation follows the philosophy of a single parsing approach of the normalized encoded message.

6 Conclusions

We presented in this paper a structured approach which extracts an abstract protocol type definition for text-based protocols whose transfer syntax is specified using ABNF. A mapping of ABNF rules to structured and basic types represents the key contribution. It separates textual encoding information from the message content and structure of text-based protocol messages. It assumes that definitions of white space occurrence in the ABNF specification have been removed and that one encoding has been selected for multiple variants prior to this mapping.

We also presented some additional transformations which can reduce the possibly high specification complexity of the resulting protocol type definition. These transformations also help to address the major shortcoming of text-based protocol standards which is their restriction of valid textual encoding and message content outside of the ABNF specification.

Finally, we outlined the implementation of a text-based protocol codec system based on the traversal of this abstract protocol type definition, but independent of the protocol to be tested or the concrete type definition itself. Contrary to textual codec implementations used in current TTCN-3 conformance test systems, the primary design criteria is not based on an already existing parser implementation. Currently at Nokia such a text codec implementation is used in practice for the conformance testing of numerous text-based protocol implementations.

We believe that our research results could help to turn the current art of conformance testing of ABNF specified text-based protocols into a science. In

addition, we hope that they could help to positively influence a future testing philosophy of the internet community (which is currently not much in favour of conformance testing) as well as the accuracy of the protocol syntax definitions in future text-based protocol standards. In addition, this derived abstract syntax could also be used to encode originally textually encoded information with a more efficient binary encoding instead.

Acknowledgements. The author would like to thank Martin Elger for his contributions to the implementation of text-based codec systems based on type information traversal. He has laid the foundation of our classification of different delimiters as well as their encoding in TTCN-3 encoding attributes. Also I would like to thank the members of the ETSI TIPHON work group for their helpful discussions and the reviewers of my paper for their helpful comments.

References

1. ISO/IEC 9646-1: "Information technology - Open Systems Interconnection – Conformance testing methodology and framework - Part 1: General Concepts". 2nd ed.. Geneva (December 1994).
2. ITU-T Recommendation X.680: "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".
3. ITU-T Recommendation Series H: "Audiovisual and Multimedia Systems; H.323-System Recommendations Implementors' Guide". Version 11 (2000).
4. J. Rosenberg et al: "SIP: Session Initiation Protocol". IETF RFC 3261 (June 2002).
5. ETSI ES 201 837-1 (V2.1.0): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language". Sophia Antipolis (February 2003).
6. ETSI DTS/TIPHON-06020-3 (V0.0.4): "Telecommunication and Internet Protocol Harmonization Over Networks (TIPHON); Conformance Testing for TIPHON Release 5; Session Initiation Protocol (SIP) Conformance Test Specifications; Part3: Abstract Test Suite (ATS) Specification". Sophia Antipolis (September 2001).
7. Testing Tech: TTsuite-SIP User Guide (2003). http://www.testingtech.de/products/TTsuite-SIP/index.html
8. ETSI DTS/TIPHON-6022-3 (V0.3.2): "Communication And Internet Protocol Harmonization Over Networks (TIPHON) Release 4; Technology Compliance Specifications; Open Settlement Protocol (OSP); Part 3: Abstract Test Suite (ATS) specification". Sophia Antipolis (January 2001).
9. M. Ranganathan, O. Deruelle, D. Montgomery: "Testing SIP Using XML Protocol Templates". Proceedings of Testcom Conference. Sophia Antipolis (May 2003).
10. M. Handley, V. Jacobson: "SDP: Session Description Protocol". IETF RFC 2327 (April 1998).
11. H. Schulzrinne, A. Rao, R. Lanphier: "Real Time Streaming Protocol (RTSP)", IETF RFC 2326 (April 1998)
12. R. Fielding et al.: "Hypertext Transfer Protocol – HTTP/1.1". IETF RFC 2616 (June 1999).
13. D. Crocker: "Augmented BNF for Syntax Specifications: ABNF". IETF RFC 2234 (November 1997).

14. J. Franks et al: “HTTP Authentication: Basic and Digest Access Authentication”. IETF RFC 2617 (June 1999).
15. A. Wiles et al: Experiences of Using TTCN-3 for Testing SIP and OSP. Proceedings of first ACATS ATS Conference. Athens (February 2002).
16. ETSI ES 201 837-6 (Draft V1.0.0): “Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface”. Sophia Antipolis (March 2003).