Model Based Conformance Testing for Extensible Internet Protocols

Nikolay Pakulin
ISP RAS
npak@ispras.ru

Anastasia Tugaenko
ISP RAS
tugaeko@ispras.ru

***Abstract*—Many contemporary Internet protocols are extensible. Extensions may introduce new functionality, alter the format of protocol messages, affect basic functionality or even modify the protocol modus operandi. Model based testing of extensible protocols faces a number of problems, the most challenging one is that extensions altering basic functionality require changes in the protocol model. It is highly desirable to have a flexible protocol's model which would let test developers to extend the model without rewriting existing parts of the model. The article presents the method for model based conformance testing for extensible Internet protocols which satisfies this requirement. Each extension is specified in a separate unit, the method presents facility to combine those units into entire model for testing the protocol's implementations. The method uses Java language to formalize the requirements. As an example of the method application article presents test suite development for a number of SMTP extensions.**

***Index Terms*—model based testing, conformance testing, extensible protocols.**

I. Introduction

A lot of protocols for different applications are functioning in the contemporary Internet. Many protocols were developed more than a decade ago. Often while developing a protocol specification it is not easy to foresee all variants of protocol's usage in the future. Protocol application reveals new tasks, makes new demands for the utilizing protocol. The simplest way to update the protocol for new tasks is to develop and publish a new version of the protocol's standard. However for popular protocols this solution may results in necessity of frequent publications of standards' revisions. The high frequency of protocol updates requires much effort for redaction and agreement of the standard's text; it is fraught with injection of indeliberate mistakes and may violate the protocol stability. Also it may hamper the developers of protocols implementations.

To solve the problem of frequent renewal of the protocol specifications nowadays protocol designers follow the pattern that one might call as "extensible protocol". The pattern implies that new protocol features emerging after publication of the protocol standard are specified in separate documents as "extensions". That is, the description structure of extensible protocols consists of the following parts:

- the basic protocol functionality which must be supported by all implementations is specified in a fixed number of RFC[1] documents, and
- new functions (extensions) are specified in separate RFC documents. For consistent work of extensions with the basic standards developers publish the special RFCs describing general extensions methods.

The history of SMTP protocol [1], [2], [3] illustrates this pattern. Since its inception in 1982 the SMTP protocol went very well with its tasks on sending messages between hosts in computer networks. But the protocol restrictions became apparent in 1990-ies. Then SMTP developers decided to improve some SMTP protocol functions instead of replacing the standard with a new one. It was considered to keep the basic SMTP specifications "as is" and specify new functions in new RFCs. To accomplish this goal in 1995 the RFC 1869 "SMTP Service Extensions" [4] was published. The document specified the method for extending SMTP capabilities.

The SMTP extension mechanism proved to be successful, and the SMTP community agreed to integrate it into the main SMTP specification. In 2001 the revision RFC 2821 [2] was published, it includes the specification of basic functionality and also the specification of extensions mechanisms. Since then all new features, even the fundamental ones – security and authentication, are published in separate documents, leaving SMTP specification intact. The SMTP specification in effect, RFC 5321 [3], contains editorial and clarification changes to RFC 2821 and maintains the extensible architecture of the protocol.

In general protocol extensions may be classified as follows:

- extensions specifying new functionality (e.g., new commands, new types of messages and responses);
- extensions altering the format of protocol messages;
- extensions altering the basic protocol functionality or operations of other extensions;
- extensions altering *modus operandi* of the protocol.

The classification of the extensions provided above shows, that extensions shouldn't be considered as independent – they may affect each other or the basic protocol. This observation leads to a conclusion that a model of an extensible protocol is not a plain composition of the basic protocol model and models of extensions. Modeling of extensible protocols requires a specialized composition that takes into consideration

[1]RFC – Request for Comments – the normative document for the Internet standards

dependencies between extensions.

Another important aspect of protocol extensibility is optional support of extensions in various implementations. Implementors are not obliged to support all published extensions; as a result, implementations of the same protocol might provide different sets of protocol extensions. This observation and the fact that extensions may change the basic protocol's functionality hampers testing of extensible protocols. To test extensible protocols test developer must know the exact set of extensions supported by Implementation Under Test (IUT, also we will call it "target implementation") and take into account the profile of those extensions to assign the right verdict concerning IUT conformance to the protocol specification.

As a consequence, model based testing (MBT) of extensible protocols faces a number of problems. First of all, the protocol model must be coherent with the IUT, i.e. the model must reflect the extensions, provided by the IUT, and leave all other extensions beyond the scope. Furthermore, the model must take into account that extensions may alter basic protocol functionality. Obviously, it is unwise to develop independent models for each thinkable set of protocol extensions. A more realistic scenario is to construct models of the protocol and its extensions separately and to combine them into the model of IUT depending on the actual extensions profile of that IUT. To implement this scenario the test development method should provide facilities for modular specification development, and facilities to combine the specification modules either statically or dynamically to match the list of supported extensions of arbitrary IUT. Despite the challenge with modular specification of extensible protocol it is highly desirable to have modular test specification, when test actions for extensions are specified in separate units and the actual test is composed from test specifications of extensions provided in IUT.

The article presents the method for model based conformance testing for extensible Internet protocols. Within the proposed method basic protocol functions and extensions are modeled as state machines with common state. Each state machine is specified in a separate unit, the whole model of IUT is composed from basic protocol specification and extensions before test execution starts. Authors has developed a composition facility that makes possible modular specification of extensions that introduce new functions, alter protocol specification and, to some extent, extensions that modify protocol message format. The proposed method does not allow specification of extensions that change modus operandi of a protocol (such as PIPELINING extension of SMTP [5]). The method supports modular development of test actions. Test is treated as a finite state machine and the method implies specification of test transitions for extensions separately. The method provides a facility that combines test specifications of different extensions into a single test state machine dynamically before test execution.

The paper is structures as follows. Section II presents existing approaches to model-based conformance testing and discusses their applicability to testing of extensible protocols. Section III describes our previous work on model based testing of electronic mail protocols. Section IV presents a new model based method for testing protocols with extensions. Section V consists of two parts, the first part describes SMTP protocol and a few extensions, and the second part presents an example of new method application – the development of a test suite prototype for SMTP protocol with extensions. Section VI discusses the applicability of a presented method. Section VII presents results of the work and describes directions for future work. And Section VIII is a conclusion.

II. Related Works

Internet protocols contain multitude of states, those states may be divided into groups with similar behavior inside each group. Manual test suite development for such kind of protocols seems very laborious and redundant because of similar checks in similar states. Tools for automated testing which allow re-use of a protocol model for verdict assignment become more frequent in use for developing test suites for Internet protocols.

In general, it is convenient to use tools possessing the following properties for testing Internet protocols:

- formal relation between requirements and tests;
- automated verdict assignment about the correctness of IUT behavior.
- automated tests generation depending on IUT responses;

Testing of protocol's extensions brings in supplementary requirements:

- the ability to easily change a protocol's model. Some extensions add new commands, new response codes, new states thereby changing the protocol's model. To develop test suites for protocols extensions test developer must have an opportunity to choose the right model: either basic or modified by a number of supported extensions;
- the ability to develop specifications and tests as separate units, one unit per extension, basic specification should be specified in separate units. This requirement is dictated by the fact that extensions are optional and not all implementations must support all extensions.

The majority of listed above requirements are satisfied by application of the model based approach and tools.

We don't consider the JUnit [7] and TTCN-3 [8] as they are not model based, the test suites written with these tools contain a lot of redundant code, test sequences must be constructed manually and all the verifications also must be specified manually. The whole process of test suite development for extensible protocols with these tools is very laborious because of a greater part of the basic test suite (test suite for testing the basic protocol's functionality) must be changed.

The UPPAAL [9] is a popular tool suite for model checking of real-time systems. It provides checking of models with hundreds of states, has some facilities to detect deadlocks. But testing of Internet protocols deals with black-box state machines, so test developers need a tool for automated test suites generator, and the UPPAAL doesn't provide such generator.

Tools NModel [10] and SpecExplorer [11] may be successfully used for developing test suites for simple small-scale protocols but they require development of considerable supplementary libraries to test large-scale protocols and extensions. Toolkit Conformiq Qtronic [12] doesn;t provide tests generation, as a result it generates TTCN-3 scripts.

III. Previous Work

In works [13], [14] the method for automated testing of e-mail protocols was presented. That method uses Java specification extension JavaTESK [15]. We used that method to develop test suites for basic functionality of implementations of mail protocols SMTP, POP3 and IMAP4 [14]. Developed test suites were applied to implementations and detected a number of noncompliances in well-known open-source mail servers.

But that method isn't suited for conformance testing of extensible protocols since it does not support modular models and tests and implied adaptation of the protocol model and test for each extension.

IV. Method for Extensible Protocols Testing

This section presents a method for model-based conformance testing of extensible Internet protocols' implementations. The primary target of the method is to provide test developers facilities for modular modeling and test specification. As stated in the introduction, protocol extensions are not independent. On the contrary, protocol extensions may affect each other; furthermore, extensions may even alter the basic functionality of the protocol. These observations lead to a conclusion, that a model of an extensible protocol can not be presented as a plain composition of separate models.

The method consists of two parts:

1) All the extensions are specified in separate units, in general one unit represents one extension. The structure of the model state is specified in the separate classes and all test units use this global shared state. Every unit may change the current state according to extension's specification. Basic functionality is also specified in one or several units (e.g. connect and disconnect may be specified in one unit and transaction commands – in the different unit). To model implementations with extensions the structure of general shared state may be changed by the following actions: addition of new symbols of the states introduced by the extensions; specifying new actions; specifying or changing precondition for actions from different states.
2) All test state machines for extensions are specified separately in different units. Whole test is constructed as a composition of test state machines for extensions supported by IUT. Note that this is not a true "composition" because of general test utilizes the shared state. Abstract test description for each extension is specified in a separate unit.

Application of this method defines the following actions to test extensible protocols. For adding new extension test developers add specification and test units into the project. If an extension adds new command then this command should be added to the commands (actions) register. Also the availability of this command in the set of states should be added to the states register. If extension changes the basic functionality then test developers should rebind old aspect with new specification unit. The comprehensive test is constructed as a composition of units specifying the extensions supported by the IUT.

The proposed method contains the following main aspects:

- simplification of the development and maintenance of model and tests for protocol's implementations. Specifications and tests for the protocol's extensions are defined in separated modules. This allows easily changing the protocol models for new protocol's extensions;
- construction of specification for IUT in accordance with set of extensions supported by target implementation;
- construction of general test as a composition of test modules for extensions supported by the target implementation.

The method utilizes the library for model based automated testing [16]. The protocol's model presented as a finite state machine. The implementation's requirements presented in contract specification notation: for each operation pre and postconditions are defined, precondition restricts the operation's availability in different states, postcondition specifies the required behavior. Also the library grants the following useful tools:

- test sequence iterator. The test sequence generates automatically from test model and contract specifications. Test model is presented as a finite state machine, each impact to the IUT contains precondition which specifies the acceptability of this impact in the current state;
- automated coverage calculation. States and transitions may be labeled with marks and branches, in this case they will be represented in the test report. Such labels allow defining the formal relation between requirements and tests;
- automated verdict assignment concerning the implementation under test behavior.

The presented method contains the following steps:

1) Creation of requirements catalogue for basic specification of the protocol. This catalogue contains only requirements from basic protocol specification and doesn't include the requirements from extensions'.
2) Creation of requirements catalogues for protocol extensions. These catalogues contain requirements from extensions specifications.
3) Designing of the extensible model for the basic specification. The extensibility of the model will be necessarily in the next steps;
4) Designing of units for extensions' specification. Generally, one unit represents one extension. These units may be easily included into protocol's model developed in the previous step;
5) Designing of formal specification for basic protocol and

extensions. The basic functionality specification is developed as self-sufficient. The extensions' specifications are developed as separate units which may be added into the basic specification.

6) Formalization of requirements. At this step the requirements of basic specification and the requirements of extensions are formalized as pre and postconditions.
7) Developing of test scenarios for basic functionality.
8) Developing of units with test scenarios for protocol extensions.
9) Constructing of the comprehensive test for the target implementation. This test includes the scenarios for basic functionality and the scenarios for extensions supported by implementation under test.
10) Execution of test suites and analyzing the results. Test suite improvement when necessary.

Not all steps in the method are mandatory. Also note that this method may be used not only for extensible protocols and for extensions, it may be easily utilized for testing other kinds of Internet protocols. If in the last step not all requirements are covered by the constructed test than steps 6-10 may be repeated as many times as needed.

V. Method Case Study

This section presents the method case study on testing the extensible Simple Mail Transfer Protocol (SMTP). First subsection contains the description of SMTP protocol and its' extensions; second subsection presents the description of test suite development.

A. SMTP Protocol Extensions

Protocol SMTP is a text based protocol of the upper layer of the TCP/IP stack. The protocol consists of two parties: a client and a server. After establishing a connection the client issues commands to the server and the server executes them and returns responses to the client. The response depends on success of the command execution.

Simple mail transfer protocol is used to send messages. This protocol has a following feature: each physical server could operate as both SMTP server and SMTP client. Being a server it accepts incoming emails and then became a client to forward these received messages to the next hops. To forward messages between various domains SMTP uses its own overlay network over TCP/IP. When an SMTP implementation being a server identifies itself as the final destination of the message it stops forwarding the message and places it into internal implementation-specific storage. To retrieve emails from the storage end-users utilize other protocols: POP3 (Post-Office Protocol, version 3 [17]) or IMAP4 (Internet Mail Access Protocol version 4 [18]).

SMTP protocol is extensible. To identify what extensions are supported by a server implementation a client should issue the EHLO command. To this command the server replies with multiline response, lines provides information about supported extensions. Response lines include extension-specific keyword and also may contain any supplementary information.

The basic protocol model consists of the following states: *DISCONNECTED*, *CONNECTED*, *AFTER_HELLO* (after issuing EHLO or HELO commands), *AFTER_MAIL_FROM* (after issuing MAIL command), *AFTER_RCPT_TO* (after issuing RCPT command), *AFTER_DATA* (after issuing DATA command), *AFTER_DOT* (after issuing the sequence $\langle CRLF \rangle . \langle CRLF \rangle$ in the *AFTER_DATA* state). With respect to specification states *AFTER_EHLO* and *AFTER_DOT* are the same, we divide them in model to test implementations to conform to this requirement.

Lets consider a few examples of SMTP extensions and categorize them according to extensions classification proposed in Introduction I. The DSN extension described in RFC 3461 [19] specifies the Delivery Status Notifications (DSNs). For example, the client may specify that DSN should be generated under certain conditions (e.g. when the mail has reached the recipient) and sent to the initial client. To use this option initial client should add new parameters in the transaction commands. This extension is of type 1 – bringing in new functionality.

The AUTH extension specified in RFC 4954 [20] adding new state to the protocol model. If an implementation supports this extension then the basic set of commands would not be enough to send a message: server may require client's authentication. This extension also introduces new command AUTH, new parameters for MAIL command and new response codes. For example, a server may response to the transaction commands with new code 530. In this case such response means that authentication is required. The STARTTLS extension described in RFC 3207 [21] specifies the TLS usage which helps SMTP agents to protect all or few interactions from interceptions and attacks. This extension also adds new state into the protocol model, specifies new command STARTTLS and new response codes. If a server supports this extension the transaction commands are not allowed before the command STARTTLS. These extensions are of type 3 – altering the basic protocol's functionality.

The PIPELINING extension specified in RFC 2920 [5] provides a facility to group several commands to send them in one transfer operation. If a server supports this extension it may response to the group of commands as a whole instead of sending responses to separate commands. This extension changes the protocol's structure and is of type 4 – altering the protocol's modus operandi.

Protocol SMTP has a long history. Nowadays the basic specification [3] includes the mechanisms of extensions and provides different optional parameters in basic commands. So protocol SMTP has no extensions of type 2 (extensions altering the format of protocol's messages), all extensions which provides new parameters are fitted into the extensible messages format.

B. Test Suite Development for SMTP Protocol Implementations with Extensions

The authors have developed a prototype of test suite for SMTP protocol using the presented method. For basic specifi-

cation we used the requirements catalogue from the previous works [14]. Also we made new requirements catalogues for two extensions: the AUTH extension [20] and the DSN [19] extension which cover two types of extensions (new functionality and altering basic functionality). For these extensions we develop new separate units with specifications (one unit for one extension) and new separate units with tests (one unit for one extension as well) .

The structure of the protocol model is organized as follows. We have a set of states and a set of actions. We define two maps from states to actions (we name them **allowed** and **denied**) which define policy which commands are allowed or denied in particular states. If an extension adds a new command we add this command to the actions set and define allowed/denied policy for this action. If an extension adds a new state we add this state to the states set and extend allowed/denied policy of the protocol commands for this state. Note, the pair state-action may be undefined in both **allowed** and **denied** maps. In this case we can provide two types of testing: the conformance testing, in which we consider undefined pair as denied; and the robustness testing when undefined pair is considered as allowed. In the latter case we try to send the command from pair in the state from this pair and looks whether the target implementation is down.

For testing the AUTH extension we defined a new state AFTER_AUTH and updated the allowed/denied policies for transaction commands (MAIL FROM, RCPT TO and DATA). If the implementation supports the AUTH extension the transaction commands may be issued only in authorized states. Also we added new command AUTH and the parameter AUTH for MAIL FROM command – the AUTH extension provides two authentications mechanisms. Then we updated maps **allowed** and **denied** to contain the information about allowed and denied transitions.

For testing the DSN extension we used methods for MAIL and RCPT commands with optional parameters. Since this extension adds only new parameters we didn't change the protocol model.

We defined a configuration file with a list of extensions supported by IUT. Then we used tool [16] to construct the whole model of IUT (from units specified above) and generate a test suite. Generated test allow detecting the following types of noncompliances:

- missing required commands;
- protocol rules violation, such as accepting commands in illegal states;
- wrong reply codes to the protocol commands.

VI. Discussion

Presented method is applicable for synchronous message based protocols. In such protocols clients send commands to the servers and servers executes them and returns the responses to each command. Responses contain the code which defines the success of the command execution.

Protocol model consists of few parts: basic part which represents the model of the basic protocol's specification and supplementary parts for protocol's extensions. Novelty of this method is the ability to easy altering the protocol's model and adding new tests.

The method was assayed by the development of test suites for SMTP protocol implementations with extensions. The SMTP extensions may add or alter the protocol's basic functionality, bring in new states, new commands and new response codes. The prototype of test suite for testing SMTP implementations with extensions shows the applicability of the method for testing extensible Internet protocols.

VII. Results and Future Work

The particular method for testing extensible Internet protocols is presented. The development is in progress, currently we have a method for testing a few types of extensible protocols. Protocol's extensions which we can test with developed method possess the following characteristics: they may add new commands, new responses, new model states but they must not alter the protocol's structure (modus operandi) and also they must not bring in new encodings of symbols of sending messages.

Using this method we have developed the prototype of test suite for testing SMTP implementations with a number of extensions. The current version of method isn't applicable for all types of extensions. For example, the extension PIPELINING [5] for protocol SMTP changes the structure of the protocol. If this extension is supported implementation from message-based became stream-based and requires other testing methods.

Most Internet protocols possess a command for identifying the list of supported extensions. The current version of presented method utilizes a configuration file to construct the modified model of IUT. In future versions we plan to add a feature of dynamic composing of the model and test state machine depending on implementation responses to the capabilities command.

After the tests has been executed test developers got a test trace. This trace contains the log of test execution, so it contains important information on what is wrong with implementation under test. Separately of this method we have a report generator, generated reports presents the test trace demonstrably but not obviously. Currently test developers should manually find the places in the test which shows the noncompliances with the specification. To operate with obtained information more easily we plan to improve the report generator.

VIII. Conclusion

The paper presents a method for automated model-based conformance testing of implementations of extensible Internet protocols. The modeling approach uses state machines to express functional specifications as a formal definition of textual requirements elicited from normative sources. Test is a traversal of some simplified (compared to the model) state machine; the sequence of test stimulus is generated depending on IUT responses.

The main idea of the method is modular approach to test suite development: both functional specifications (models) and test specifications of basic protocol functionality and each extension are developed in separate units. Models of extensions are expressed as state machines over common extensible set; the method provides facilities to combine such partial models into a complete state machine, depending on the exact set of extensions supported by a specific IUT. Test is constructed as composition of test state machines of the extensions supported by the specific IUT.

The characteristic feature of the proposed method is the choice of notations for models and test specification. We use programming language Java, pure, without any extensions (such as Java Modeling Language, JML [6]), while model composition is partially defined in XML. The selection of Java as the primary notation gives the full power of Java expressiveness, rich toolkits for model and test development and, potentially, has more chances to attract attention of industry since the method does not require experts in formal description languages.

Using this method the prototype of test suite for testing SMTP protocols with some extensions was developed. Test suite covers the following types of protocol's extensibility: adding new functionality and altering the basic protocol's functionality. The extensions which are altering the protocol's modus operandi have not been tested yet. The development of new method is ongoing project and extending this tool to test the extensions altering the protocol's modus operandi is one of the tasks to decide. Also we plan to improve the report generator and to extend the method and tool for testing more types of Internet protocols' extensions.

ACKNOWLEDGMENT

The authors would like to thank Victor Kuliamin for kindly provided Java library for automated model based testing.

REFERENCES

[1] IETF RFC 821. Jonathan B. Postel. Simple Mail Transfer Protocol. 1982.
[2] IETF RFC 2821. J. Klensin. Simple Mail Transfer Protocol. 2001.
[3] IETF RFC 1869. J. Klensin. Simple Mail Transfer Protocol. 2008.
[4] IETF RFC 1869. J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker. SMTP Service Extensions. 1995.
[5] IETF RFC 2920. N. Freed. SMTP Service Extension for Command Pipelining. 2000.
[6] Patrice Chalin, Joseph R. Kiniry, Gary T. Leavens, and Erik Poll. Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2. In Formal Methods for Components and Objects (FMCO) 2005, Revised Lectures, pages 342-363. Volume 4111 of Lecture Notes in Computer Science, Springer Verlag, 2006.
[7] Unit testing framework, http://www.junit.org.
[8] ETSI ES 201 873-1 V3.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. Sophia-Antipolis, France: ETSI (2009).
[9] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal 4.0. http://www.uppaal.com/
[10] Jacky, J., Veanes, M., Campbell, C., Schulte, W.: Model-based Software Testing and Analysis with C#. Cambridge University Press, Cambridge (2008).
[11] http://research.microsoft.com/pubs/77383/bookChapterOnSE.pdf http://research.microsoft.com/en-us/projects/specexplorer/
[12] End-to-End Testing Automation in TTCN-3 environment using Conformiq Qtronic and Elvior MessageMagic. 2009.
[13] A. Tugaenko, N. Pakulin. Test suite development for conformance testing of email protocols. // Proceedings of Spring/Summer Young Researchers' Colloquium on Software Engineering, pp. 87-91, Nizhniy Novgorod (2010).
[14] N. Pakulin, A. Tugaenko. Specification Based Conformance Testing for Email Protocols. // Proceedings of ISoLA 2010, pp.371-382. Heraclion, Greece, 2010.
[15] JavaTESK: getting started. Moscow, 2008.
[16] V. Kuliamin. Component architecture of model-based testing environment. Programming and Computer Software, 36(5):289-305, 2010.
[17] IETF RFC 1939. J. Myers, M. Rosem, Post Office Protocol – Version 3. 1996.
[18] IETF RFC 3501. M. Crispin. Internet Message Access Protocol – version 4rev1. 2003.
[19] IETF RFC 3461. K. Moore. Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs). 2003.
[20] IETF RFC 4954. R. Siemborski, A. Melnikov. SMTP Service Extension for Authentication. 2007.
[21] IETF RFC 3207. P. Hoffman. SMTP Service Extension for Secure SMTP over Transport Layer Security. 2002.