Test suite development for conformance testing of email protocols

Nikolay Pakulin
ISP RAS
npak@ispras.ru

Anastasia Tugaenko
ISP RAS
tugaenko@ispras.ru

***Abstract*—The method for testing electronic mail protocols in the Internet to conform to the standards based on formal specifications is presented. The method is based on automated testing technology UniTESK in which functional requirements are formalized as pre- and postconditions and test sequence is generated on-the-fly from finite state machine (test state machine) traversal. The method is illustrated by the test suite development for SMTP and POP3 protocols using JavaTESK – a specification extension of Java language.**

***Keywords*-formal specifications; model based testing; protocols testing; conformance testing.**

I. Introduction

Emails are fundamental to modern communications between people. Hundreds of millions of emails float every day around the Internet. Reliability and correctness of the emailing infrastructure is vital to the modern information society. In this article we concern two aspects of these questions – reliability of (1) mail transfer in the Internet and (2) delivery of the email to recipient, typically a human being.

Most of emails in the Internet are transferred by means of SMTP – Simple Mail Transfer Protocol [1]. It is a text-based protocol with two parties: a client and a server. Client issues commands and server executes them, returning status code and other details if needed. SMTP has its own overlay network over Internet comprised by numerous mail servers and relay agents used to forward emails between various domains. A feature of SMTP is that each physical server could operate as both SMTP client and SMTP server: being a server it accepts an incoming email and becomes a client to forward it to a next hop.

SMTP is used to send messages, but when an SMTP implementation identifies that this is the final destination of an email it stops forwarding the mail and places it in an internal (implementation-specific) storage. To retrieve emails from the storage end-users utilize other protocols: POP3 (Post-Office Protocol, version 3 [2]) or IMAP4 (Internet Mail Access Protocol version 4 [3]). Both protocols are text-based with distinct roles of a client and a server. Clients access storage issuing protocol commands and servers provide required information in their replies. Typical POP3 and IMAP4 implementations support only one role at a time.

On its way from the originator to the recipient an email is processed by a number of intermediate servers. A typical case is that those servers come from different vendors thus having different implementations of the email protocols. The total reliability of emailing infrastructure substantially depends on the reliability of each server in the way and the compatibility between implementations.

Nowadays protocol conformance testing is the basic method of attesting implementations compatibility. That rationale for this statement bases on the suggestion of good protocol quality: if two implementations confirm to a protocol specification then they are compatible, they can correctly communicate with each other.

Despite of more than twenty-year history of mail protocol service and existence of dozens of SMTP/POP3/IMAP4 implementations there are no open and implementation-agnostic conformance test suites for those protocols. We believe that there are several reasons of this.

First, the simplicity of mail protocols is seeming. Let's explore the mail protocols features in the context of testing:

1) Mail protocols are underspecified: a large part of functionality is left to implementation developers; specifications prescribe several variants of possible system behavior;
2) mail protocols are nondeterministic, the standard allows various system behavior alternatives including refusal in mail message delivering or connection tear;
3) mail protocols requirements differ in the level of obligations (MUST, SHOULD, MAY);
4) protocol architecture is extensible, protocols implementation may use different extensions for supplemental functionality or even overlapping functionality.

Listed features demonstrate complexity of the task of conformance test for email protocols.

Second, developers has to focus testing on another big issue of mail server development: processing of a large number of settings necessary for practical use – parameters of routing, authentication, security, mail messages depository etc. We studied test suites developed by several open source implementations. The test suites turned out to be tightly coupled with the implementations under test (IUT) and non-portable to servers from other vendors, they use testing implementations features setting parameters, access to internal state, execution in the same process as the implementation. Tests designed for one implementation could not be applied to other implementations. Moreover, as shown by the analysis, these tests are inappropriate for conformance testing, they are oriented to checking implementation for numerous settings correctness that don't directly connected with SMTP and POP3 standards. The problem is that such approach to testing doesn't guarantee servers' compatibility to the standard. The situation is even worse – our conformance tests detected a serious functional defect in James server – cycling at certain conditions – that was overlooked by the James functional tests.

Also it is necessary to note that there is a special tool developed in Apache Project – Mail Protocol Tester (MPT) – for verifying correctness of server replies. The input data for this tool is a script that specified server stimuli and expected server replies. The program uses regular expressions for comparison of IUTs replies and expected replies from defined scripts. If reply mismatches program stops and throws the mistake message. Apache James MPT does not support branching, cycles and parameters usage in tests.

Exact relationship between tests and standard's requirements allows hard confidence estimating in terms of external user of mail service. As the example with server James shows, the thoroughly testing of internal functions of implementation doesn't guarantee the functional quality of implementation in real environment.

Proceeding from the above arguments, we believe that the problem of conformance testing to the standards of mail protocols has significant practical importance. The ultimate goal of our research is to develop an opensource general-purpose conformance test suite for SMTP, POP3 and IMAP4. In this paper we present a model-based approach to conformance testing of email protocols. The presented approach focuses solely on conformance and does not consider other aspects of email infrastructure validation, such as interoperability testing, performance testing, reliability testing etc.

The paper is structured as follows: section II gives an overview of the existing approaches to protocol conformance testing and discusses why model-based testing is used in our approach. Section III provides a quick introduction to UniTESK technology that lays in the basis of our approach and Section IV introduces the proposed test development process. In section V we present the test suite for SMTP and POP3 developed so far and Section VI discusses pros and cons of the proposed approach. Section VII summarizes results achieved by now and highlights directions of future research.

II. Mail protocol testing

In the contemporary industry conformance testing of protocol implementations is mostly based on manual development of test suites consisting of independent test programs written in specialized or general-purpose programming language. Such programs are referred to as test cases; they implement stimulus test sequence generation, passing generated test inputs to the IUT, reading and analysis of observed outputs [4].

Let's consider requirements for test suite. Test suite for conformance testing must possess the following properties:

1) Requirements traceability. Tests must correlate with standard requirements. It must be clear for each requirement which test it is covered by.
2) Variety of settings for implementations features (MUST, SHOULD, MAY and others). There must be an option to define the set of requirements supported by an IUT and avoid requirements that IUT does not implement.
3) Completeness of test suite in terms of requirements coverage. Resulting test suite must cover at least all obligation requirements.

Test cases approach doesn't provide evident traceability of requirements. Requirements completeness in terms of coverage in such approach is also complicated. We rejected TTCN3 [5] and JUnit [6] because they don't provide formal connection between tests and requirements.

Besides that large number of tests presented as separate programs results in code redundancy or complex and complicated connections. It is necessary to use methods permitting decomposition of test suites and providing requirements traceability.

Required possibilities are given by tools based on formal method approach. Utilizing of formal specifications allows to:

1) define formal connections between requirements and tests; automatically backtrace quality of testing in terms of specification coverage;
2) using model repeatedly for checking correctness of implementations behavior;
3) generate test stimuli in terms of model and automatically filter redundant stimuli.

Also when choosing a method for generating test sequences it is necessary to take into account features of mail protocols. Particularly because of protocol behavior is nondeterministic and underspecified, one should choose approaches providing test sequences generation with a glance of IUT replies. As well when testing mail protocols it is complicated to make prediction for result or define equivalence of traces. Automatic verdict generation from specifications postconditions may solve the problem of verifying correctness of IUT behavior.

There are many instruments and approaches for testing. NModel [7] represents model system in C# language and provides basic facilities for on-the-fly testing and coverage maximization. But for on-the-fly testing test developer must write separate program describing complex traversal strategy. The current stable version of SpecExplorer [8] doesn't support on-the-fly testing.

Toolkits UniTESK [9] and Conformiq Qtronic [10] support formal specifications notation, automated on-the-fly test stimuli generator (code-level, there is no need to write separate program) and automated test results analysis. For this project the UniTESK was selected.

In the UniTESK technology formal specifications which formalize requirements as pre- and post- conditions are used for generation of test sequences. Also for test sequences generation must be given a certain finite state machine (test state machine). The test process in UniTESK is automatic traversal of test state machine in which IUT behavior is automatically verified by test oracles; test oracles are generated from formal specification. The utilizing of formal specifications allows automating verification of behavior correctness and estimation of hard confidence; presenting test as state machine makes possible to automatically generate long and various sequences of test events.

Authors used presented method for developing test suites for protocols SMTP and POP3. From tools implementing the UniTESK approach JavaTESK [11] was chosen. JavaTESK uses the programming language Java with a number of extensions for record formal specifications and specify tests.

III. UniTESK technology overview

The standard format for Internet protocol standardized documentation is defined by documents RFC (Request for Comment). Requirements in these documents are stated in English and correspond informal text that describes desirable system behavior. In the UniTESK technology (Fig. 1) specialized specification languages – extensions of Java and C – are used for record requirements. In this work was used Java extension JavaTESK.

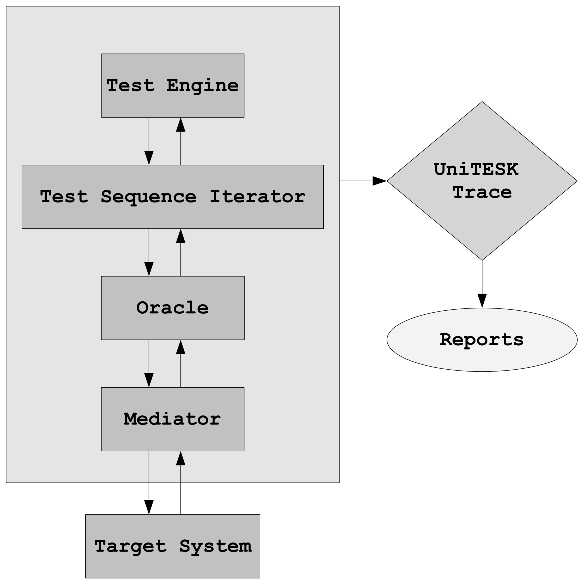

Fig.1. UniTESK Test Architecture

Recording the informal requirements of standardized documentation in formal language represents the protocol model. In UniTESK approach the formal model is constructed in terms of finite state machine. Transitions between states may be given in explicit or in implicit way. In case of explicit definition of transition the model contains algorithm for calculating next state and protocol reaction. Presentation of implicit transition is a predicate which defines restrictions on acceptable states and protocol reactions.

Specification in JavaTESK usually consists of one or few specification classes which describe states and transitions of modeling protocol. Protocol transitions are presented as special methods (specification methods). In addition there is a possibility to define restrictions on acceptable set of states by means of type invariants (type restrictions) and state variable invariants.

Definition of implicit transitions realized as pre- and postconditions. In preconditions there are restrictions on acceptable stimulus parameters values and on states from which stimuli may be given. The IUT may react on stimuli by changing state, giving a reaction or both. Postconditions define acceptability of demonstrated behavior.

For modeling IUT behavior one uses the set of data structures which referred to as abstract states. For verdict pronouncement about the correctness of IUT behavior UniTESK uses data from model abstract state.

In UniTESK both stimuli for the IUT and its reactions are described in terms of model; model is defined by the formal specifications. Correlation between the model and the IUT is established by an intermediary – mediator – which translates stimulus parameters from model form to protocol messages, IUT reactions to model representation and if necessarily transports changes from the IUT state to the abstract state.

Test scenario defines stimulus sequence applied to IUT. The metamodel of finite state machine is used as the theoretical basis for constructing scenarios. In JavaTESK test state machine is defined in scenario class which contains the procedure for calculating current state and iterator of test stimuli. The JavaTESK tool contains test engine for constructing test stimuli sequences from test state machine description.

IV. The proposed method for mail protocols conformance testing

Mail protocols may be in several states. When receiving certain stimulus they generate and send reactions and jump to another state or leave in current. With a glance of this fact on the basis of instrument UniTESK the method for testing mail protocols was developed. The developed method contains the following main steps:

1) Analysis of knowledge domain. Developing of examples and elementary tests. This step doesn't give any visible results but it is important for detailed protocol understanding and helps in implementation of next steps.
2) Creation of requirements catalogue. Requirements catalogue is a database or a table with description of requirements. Catalogue's record contains not only requirements description but also requirement identifier, type (syntax or functional), severity, link to the place in the RFC and maybe other attributes.
3) Designing of lite protocol model. Creation of experimental tests – test state machines with only one state. Lite protocol model includes information about commands and about possible reactions to these commands. Experimental test consists of

specification, mediator and scenario classes. Specification class on this step includes only signatures of methods; all verifications are making in scenario class. Such test referred to as linear test; applying stimuli and reading reactions are process in certain order defined in scenario class.

4) Designing of conceptual protocol model. Extracting states of basic protocol. Creation of test state machines with dedicated states. Expanding experimental test – addition the block which makes transitions between states. Conceptual model defines behavior of observing system as operations on some set of abstract components and composing objects. These components are used only for behavior modeling and may not conform to the model extraction. Addition of block for making system transitions between states turns test from linear to automatic test. In this test construction of stimulus sequence is made from test state machine traversal; applying stimuli and reading reactions are made only from acceptable states.
5) Requirements formalization. Relocation of verification of IUT responses into specification class. Checking completeness and consistency of requirements is made while formalizing requirements. The result of this step is the formal protocol specification written on one of the special program languages. In our case the Java extension JavaTESK was used.
6) Enhancement of scenario and specification for covering all requirements. In this step scenario classes contain only stimuli. The order of stimuli is formed from state machine traversal and depends on applying stimuli in certain states conditions. Usually one scenario class is responsible for the certain requirements section. For covering all formal requirements few scenario classes may be needed.
7) Execution of test suites and analyzing the results. Analysis may show that not all requirements are covered by generated test suite. If not all requirements are covered then step 6 must be repeated until covering all requirements from catalogue.

V. Method application for protocols SMTP and POP3 testing

The first step in writing tests for SMTP and POP3 implementations was analysis of knowledge domain, sending emails directly from server console. Then the requirements from RFCs were marked and categorized for types: commands and replies, routing, notifications, server settings, mail headers, mail body, etc. On this basis the lite protocol models were designed; test state machines consisted of only one state from which sent commands (for SMTP: EHLO, HELO, MAIL FROM, RCPT TO, DATA and others, for POP3: USER, PASS, LIST, STAT, RETR, DELE, TOP and others), received replies (for SMTP: three digit numeric code – reply code, for POP3: "+OK" or "-ERR" replies) and left at the same state. On this step implementations of test suites generation had a formal interface, specification classes was consisted of only methods' signatures; all IUTs behavior correctness verifications were made in scenario classes. Scenario classes consisted of methods applying (by means of mediator classes) stimuli to the IUTs, reading servers responses and returning verdicts about correctness of severs behavior. Mediator classes transformed the IUT stimuli format to model systems format and vice versa.

Then the basic protocol states were marked. On this step the new blocks responsible for making model system's transitions between states were added to the scenario classes. Specification classes were not changed.

In the next step blocks responsible for verification of IUT behavior correctness and blocks that make transitions between systems states were relocated from scenario classes to specification ones. Scenario classes solely applied stimuli to the IUT. From now certain commands could be passed only from acceptable states of state machines. The possibility of such verification is achieved by recording acceptable states in preconditions of specifications. Iterator every time checks the current state and whether the applying of the next command is allowed in current state. Also it gives the opportunity to check the fact that servers don't send commands from forbidden for such commands states.

VI. Discussion

While testing systems it always necessarily to know when testing may be considered as completed, what requirements have already been tested and what requirements are to be tested. Test cases testing cannot answer this question because correlation between tests and requirements is given informally in form of traceability matrix.

The utilizing of formal specifications allows formulation the exact unambiguous hard confidence criteria – testing may be completed when all elements of appropriate formal specifications are covered. UniTESK uses procedure of counting covered requirements and allows to define some selecting criterion for scenarios – if applying of certain scenario doesn't increase test coverage then system misses it and moves to the next scenario.

One of important advantages of this method is separating the class in which makes the pronouncement of verdict. The oracle which is generated from specification postconditions is responsible for verdict pronouncement. Due to this one hasn't to invent special functions for checking the correctness of IUT behavior.

To the lows of the method based on formal specifications one may attribute the absence of the quick test suites updating ability. When testing in terms of test cases new test results immediately from a simple test program. When testing in terms of formal specifications for developing new test it is necessary to thoroughly study requirements, formalize and classify them. Only after these preparations one may set out to write specification, mediator and scenario classes. Through this the period for new test development is increasing. But after specification, mediator and scenario classes have written one got not a single test but a set of tests responsible for corresponding requirements class.

VII. Results and further research

For protocol SMTP were marked 51 basic requirements, 43 of them are related to server commands and replies (11 of them are mandatory and 4 are optional), 8 related to routing (all are mandatory). For protocol POP3 were marked 58 requirements for all functionality, 5 of them are mandatory and 6 are optional. All marked requirements are covered. Developed test suites were applied for testing open source mail protocols implementations – Apache James, hMailServer, Postfix and Dovecot. In the course of testing the following disagreements between protocol implementations and standards [1, 2] were detected:

- absence of required commands supporting;
- protocol rules violation (passing commands from forbidden for such commands states);
- wrong reply codes to the protocol commands;
- cycling while redirecting mail.

The feature of mail protocols is that mail protocols are extensible. Certain extensions supplement existing functionality, i.e. add new requirements which are not in a contrast with requirements from main standard. But there are also such extensions which radically alter the protocol structure thereby discarding some requirements from the basic standard. For checking such extensions one should modify test suites in such a way that requirements that are amended by extensions have not been verified. Otherwise there will be no opportunity to reach 100% coverage of all obligatory requirements. In connection with many protocols be extensible there is a need for tools providing ability for generating test suites for testing different extensible protocols' implementations both supporting extensions and supporting only basic standard functionality.

VIII. Conclusion

The paper presents a new approach to mail protocol testing. The approach belongs to model-based testing domain, it uses contract specifications to formalize protocol specification and on-the-fly test sequence generation. The implementation of the approach is based on UniTESK technology. Distinctive features of this method are automated test sequences generation on basis of formal specifications, test coverage calculating which allows constructing stimuli in optimal way and also the presence of separate component – oracle – responsible for verdict about IUT behavior correctness returning.

Developed method was applied for testing of long used mail protocols implementations. In one of the tested implementations was found a critical defect – under specific circumstances while redirecting message the server is resending the mail to itself and the message never reaches the recipient.

References

[1] IETF RFC 5321. J. Klensin. Simple Mail Transfer Protocol. 2008.

[2] IETF RFC 1939. J. Myers, M. Rosem, Post Office Protocol – Version 3. 1996.

[3] IETF RFC 3501. M. Crispin. Internet Message Access Protocol – version 4rev1. 2003.

[4] ISO/IEC 9646. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts. Geneva: ISO, 1994.

[5] 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.

[6] Unit testing framework [URL] http://www.junit.org/.

[7] Jonathan Jacky, Margus Veanes, Colin Campbell, Wolfram Schulte. Model-based Software Testing and Analysis with C#. Cambridge University Press, 2008.

[8] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, Lev Nachmanson. Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer Microsoft Research, Redmond, 2007.

[9] A. Barantsev, I. Burdonov, A. Demakov, S. Zelenov, A. Kossatchev, V. Kuliamin, V. Omeltchenko, N. Pakoulin, A. Petrenko. UniTesK Approach to Test Development: achievements and Prospects. Proceedings of ISP RAS, No. 5, 2004.

[10] End-to-End Testing Automation in TTCN-3 environment using Conformiq Qtronic and Elvior MessageMagic. 2009

[11] JavaTESK: getting started (in Russian). Moscow, 2008.