Keyword-Driven Testing with Message Sequence Charts

Boris Tyutin, Alexey Veselov, Vsevolod Kotlyarov
Saint Petersburg State Polytechnical University, Saint Petersburg, Russia
b.tyutin@ics2.ecd.spbstu.ru, veselov.alexey@gmail.com, vpk@ics2.ecd.spbstu.ru

***Abstract* — This paper overviews an approach to keyword-driven testing based on test cases created in Message Sequence Charts format. Main features and advantages of this idea are discussed. Last two chapters provide brief overview of current implementation of testing automation framework based on the presented approach.**

I. Introduction

Nowadays software development includes a wide range of techniques and strategies. During the past years they evolved from heavy and strict methodologies like waterfall life cycle to agile techniques and iterative approaches. All of them are now more or less standardized and are used in different areas of software engineering. The choice of the development model is driven by the characteristics of the particular projects.

In all kind of development processes we can find phases that have some particular goal and go one after another. In iterative approaches it is possible to get back to some of the previous step if something goes wrong. And testing is a reasonable approach for checking whether the whole work is done well.

It is obvious that different types of workflow activities require different types of testing. Being in a stage of requirement clarifying we can operate only a model of future software. At that moment it is impossible to do performance testing. Moreover, it is not required as the main goal of that stage is to reveal contradictions and gaps in the specification. But it is extremely expensive to maintain different testing processes for one product. It is required not only to integrate different technologies but also to maintain multiple test suites and keep them coherent with the requirements. To reduce costs testing should be scalable and allow applying the same approach on different life cycle stages.

Another problem of testing is that traditionally it is directed towards the engineers. Being unavailable for business people, information about performed checks and probations of software product becomes less useful for planning or marketing. And, vice versa, making test results more obvious for most of the stakeholders of developing process increases benefits of testing. So it can be used not only for finding bugs but also to collect and maintain that database of knowledge about the product. Test-driven development [1] is a good example of an attempt to achieve this goal.

Among the variety of existing testing approaches keyword-driven testing (KDT) is one of the most advanced [2]. It aims at simplifying the test suite development and maintenance and separates the logic of test procedures from the implementation. This paper describes an approach for automated testing based on Message Sequence Charts (MSC) [3] which implements KDT approach. Main concepts of the latter one are briefly overviewed and compared with the capabilities of MSC format. Basing on both concepts a testing approach is developed, and two main ways of its implementation are described. The article is concluded with the overview of current results and future plans.

II. Keyword-driven Approach

KDT is a third-generation approach for automated testing framework design. This means that it allows creating and executing structured scenarios with data being separated from control flow. Keyword-driven tests consist of a list of keywords and their parameters. Each keyword represents a predefined set of actions performed against the system under test (SUT). The scenario itself has tabular format, and can be edited as plain text or with special tools. Fig. 1 represents the high-level concepts of KDT approach.

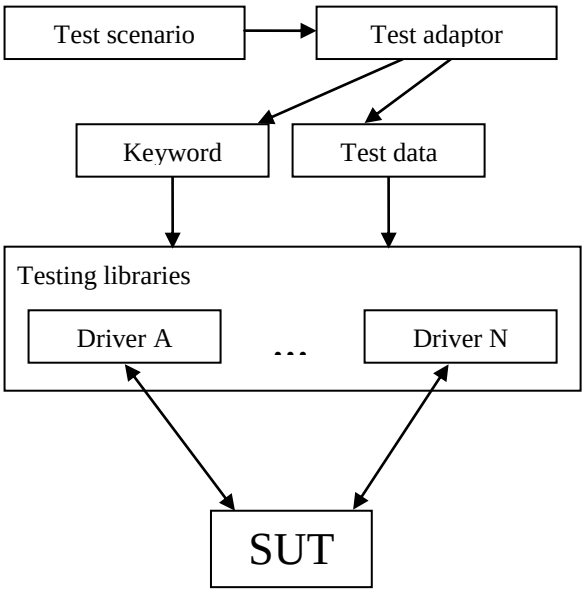

Figure 1. Keyword-driven testing.

Each keyword has clear and unambiguous definition. It can be implemented in code or written in document. Thereby KDT can be applied both for automated and manual testing. It is possible to develop keyword definitions separately from the tests, and, thus, testing can be divided into two independent flows – test design and test action specification. The former one can be done by someone without programming skills.

With KDT test suite becomes more stable. The cost of its maintenance is reduced because it is not necessary to fix test scenarios according to the changes in SUT, only keyword definitions. Test execution is more scalable [4]. Test cases themselves become easy to modify as they operate high-level abstractions and can reuse existing keywords. Due to the absence of low-level details test suite is readable by all stakeholders.

III. Message Sequence Charts Benefits

Message Sequence Charts are quite similar to UML Sequence Diagrams. Roughly speaking, they represent the interaction between a set of agents called instances by means of sending and receiving signals. More information about MSC can be found in ITU-T specifications [3]. Fig. 2 demonstrates an MSC for a part of SIP protocol.

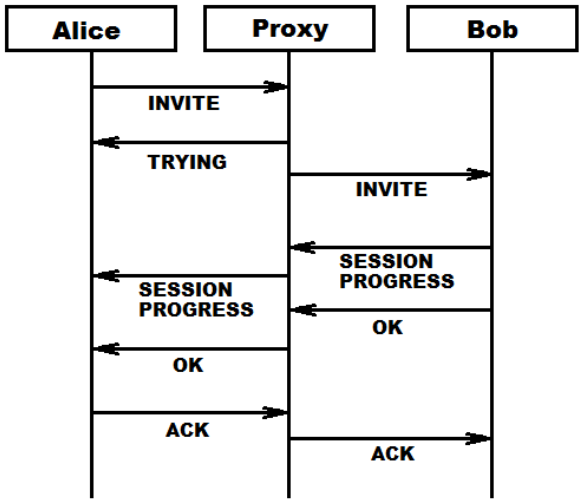

Figure 2. MSC for a part of SIP protocol

It is very native to use sequence diagrams to illustrate use-cases or even provide concrete specification for communication protocols or data exchange between software components. Phrase representation of MSC can be automatically parsed and analyzed. It means that diagrams may be used as test scenarios. In this context, the idea of signal in MSC is very close to the concept of keyword.

MSC as a format for keyword-driven tests has particular advantages. It is human-oriented and can be presented in graphical view [5]. Due to standardization it is possible to develop formal algorithms for MSC processing. Their implementation can rely on third-party tools and libraries. Some of the existing modeling and developer tools provide MSC data, which can be used both for documenting and testing the product.

IV. Technology Concept

Main elements of MSC diagrams are signals, actions and inline expressions. All of them can be used to implement keyword-driven testing concepts.

Signal exchange can be interpreted as a series of keywords, executed one after another. Testing data can be defined in signal parameters. In context of testing all instances present in sequence diagram refer to environment or SUT. Thus, signals sent from environment to SUT describe testing actions, while signals coming from SUT instances specify the reaction to the stimulus. In keyword-driven testing there is no difference between executed commands in scenario. The concept of sending and receiving signals has to be adopted in a way that makes tests more readable whilst clear and unambiguous.

Taking this into account we can consider signals sent from the environment as a pure keyword execution. Signals sent from SUT also have this meaning but should imply additional checks that affect the result of the test. In can be the interpretation of return code, for example. Another approach is to detect special states and events during testing and store them somewhere for future processing by means of signals sent from SUT. For automated keyword testing let's call it "passive driver".

Actions in MSC are used to describe internal events in instances such as data evaluation, or comments. As an element of testing scenario action can be used to increase readability of the test commenting what is happening inside the "black box". In manual testing it can give additional instructions of provide external references to documents. In automated testing actions can serve as a placeholder for executable code for test customization. In both cases actions can contain keywords.

Summing up what has been said it is possible to use MSC for keyword-driven testing and use diagram components to make tests more readable and vivid. But the role of each element must be clearly defined to keep scenarios easy to understand and provide unambiguous way of their creation.

MSC inline expressions allow specifying non-linear control flow. They describe repetitive actions (loops), optional or alternative behavior (opt and alt expression). Combined with condition events (as in Fig. 3), inline expressions can be used to represent if-else and loop statements in their traditional meaning. To enhance the conditions and data manipulations, variables can be declared using MSC text elements and then used in control flow management and testing data.

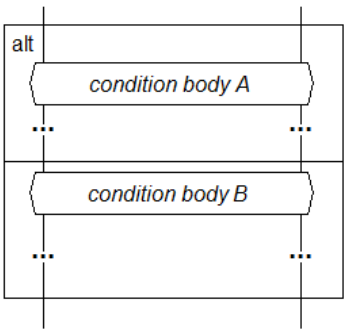

Figure 3. Conditional inline expression

Current research concentrates on automated testing rather than manual. It was mentioned earlier that MSC is standardized notation so it is possible to automatically produce human-readable tests based on it. Testing system can automatically run those tests by leveraging different underlying technologies with driver libraries. During the current research a testing automation framework was created according to the ideas presented above. Main concepts of the framework implementation are presented below with the information about current status of the research and plans for future development.

V. Automated Testing Framework Implementation

Main idea of the architecture corresponds to the concept of KDT approach. Keywords are extracted from MSC and processed using driver libraries. But instead of direct interpretation of commands tests are first translated to the program in target code. This approach is commonly used in testing with MSC [6]. Then it is build into executable called test unit. Its implementation is based on state machine approach which allows non-linear behavior and automatic analysis of execution trace. Main components of created framework are present in Fig. 4.

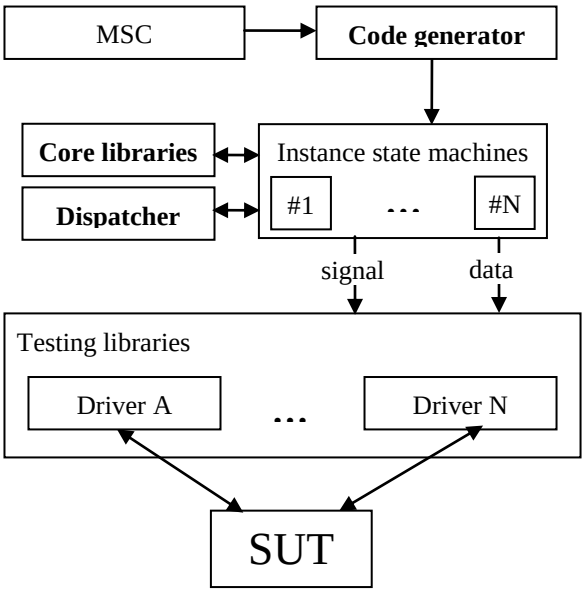

Figure 4. MSC-based testing automation framework

Each MSC environment instance is executed in separate thread. This requires introducing two types of variables – local and shared. Local variables may have different values in different instance treads. To provide thread-safe access to the set of shared variables called test context each test unit has special control module [7]. The latter one is also responsible for processing conditions for inline expressions with multiple instances involved. It tells which branch of alt should be used or whether instances need to execute another iteration of loop.

VI. Conclusion

This research is still in progress. Current implementation of framework include translator from MSC to C, core and driver libraries, test report generator. Proposed testing approach is now being approbated in testing of telecommunication software. Future plans include design and implementation of passive driver approach and integration with Robot framework testing libraries.

References

[1] K. Beck, Test-Driven Development by Example, Saint Petersburg: Piter, 2003.

[2] Faught, Danny R. Keyword-Driven Testing. Sticky Minds. Software Quality Engineering. http://www.stickyminds.com/article/keyword-driven-testing, 2004.

[3] ITU-T Recommendation Z.120: Message sequence chart (MSC). Geneva, Switzerland, October 1996, https://www.itu.int/rec/T-REC-Z.120-201102-I/en.

[4] Tiutin B., Veselov A., Kotlyarov V. Scaling of automated test execution // St. Petersburg State Polytechnical University Journal. № 3 (174). 2013. P. 118-122.

[5] W. Damm, D. Harel. LSCs: Breathing life into message sequence charts. Formal Methods in System Design, 19(1), 2001.

[6] Hu W., Sun X. Test Case Generation Based on MSC TTCN-3 // Proceedings of the International Conference on Information Engineering and Applications (IEA). London: Springer-Verlag, 2013. 888 p.

[7] Kaner C., Bach J., Pettichord B. Lessons Learned in Software Testing: A Context-Driven Approach. NY: John Wiley & Sons, Inc., 2001. 320 p.