Conformance Testing Framework for Service Oriented Interconnection Technologies

A NoTA Case Study

Janne Keränen[1], Tomi Räty[2], Petri Jurmu[3], Matti Mäki[4], Olli-Pekka Puolitaival[5]

VTT Technical Research Centre of Finland, Kaitoväylä 1, Oulu, Finland

[1]janne.s.keranen@vtt.fi; [2]tomi.raty@vtt.fi; [3]petri.jurmu@vtt.fi; [4]matti.maki@vtt.fi; [5]olli-pekka.puolitaival@vtt.fi

***Abstract*-This paper presents a conformance testing framework for testing service oriented interconnection technologies. The framework is founded on previous research and elaborates the research track based on a market analysis. The conformance testing framework focuses on a case study, which concentrates on testing Network on Terminal Architecture (NoTA) NoTA is a service-oriented interconnection architecture that aims to facilitate communication between applications running on terminal devices. The conformance testing framework presented in the paper aims to ensure that the NoTA subsystems are interoperable with other NoTA subsystems. The conformance testing framework includes a testing process, testing tool chain framework, tool evaluations, and detailed descriptions for NoTA stack testing approaches. All these will be presented in the paper and are reflected to existing research literature. The research is based on the conceptual analysis of the related publications and technologies, and the results are derived by the presented testing framework with the constructive research.**

Keywords- Service-Oriented Architecture; Interconnection; Conformance Testing; Testing Framework

I. INTRODUCTION

An increasing number of communication technologies belonging to different providers are making networks complex and difficult to manage [1]. In the future, software will be used as a service running somewhere in remote on any convenient physical or virtual device [2]. These new Internet applications and services have increased in heterogeneity and complexity in service provisioning, and these services require interconnected, high-end resources [3]. This will put a high demand on information transport services by requiring reliable, ubiquitous, and seamless end-to-end connectivity [2]. Service-oriented design can be the answer to problems facing the providers of services for information transport [1]. This trend is expected to have a strong impact on the business models in the information and communication technologies value chain [1].

Service-oriented architectures (SOA) and solutions have been utilized in numerous different occasions and application domains to address these challenges. A service-oriented approach was used for interconnecting data-centric sensor networks with internet protocol (IP) networks based on device profiles for web services gateway because the newly developed wireless sensor network protocols are not compatible with common network protocols like IP [4]. In another research a service oriented intelligent middleware service framework (MSF) enables integrated e-logistics infrastructure and networks, and provides seamless dynamically created communication and interconnections among logistics participating middleware systems [5]. Service-oriented network architecture can also be used to bridge informational gap between user applications and optical networks providing technology-agnostic multi-granular optical network services for clouds [6]. SOA can also be enhanced with real-time capabilities for industrial automation addressing problems that arise in deploying a middleware layer for supporting SOAs in next-generation industrial automation platforms [7].

Service-oriented interconnection technologies are technologies that aim to enable and facilitate intercommunication of different devices, services, platforms and networks. Network on Terminal Architecture (NoTA) [8] is a service-oriented interconnection technology that aims to make service and application development independent of underlying physical transport layers by offering a common low-level interface for terminal devices. These service-oriented interconnection technologies try to help in interoperability and integration problems between intercommunicating components, but the sheer amount of vendors implementing these components create new interoperability problems.

Because a lack of trust prevents service computing’s mainstream adoption, a key issue is providing users and system integrators the means to build confidence that a service delivers the expected quality of service. The service provider can test a component only independently of the applications in which it will be used, and the system integrator is not able to access the source code to retest it. This calls for specific testing to guarantee the service-level agreements with consumers. [9]

These concerns apply to NoTA technology, which aims to facilitate multi-device and multi-platform communication between embedded devices. NoTA is a platform independent interconnect and therefore the NoTA stack will be implemented and optimized separately for many different platforms. Each device vendor develops their own version of the NoTA protocol stack to optimize the performance of their device. The developed device using NoTA must conform to the NoTA architecture and be interoperable with other NoTA devices and subsystems. Due to independent subsystem vendors and computing platforms in the NoTA ecosystem, the interoperability of NoTA subsystems is not self-evident. [10]

In this paper, we present an advanced version of the conformance testing framework originally presented in [10], and elaborate the approach based on market analysis information presented in Chapter III. The framework aims to facilitate NoTA subsystems interoperability, and their conformance with the NoTA architecture. Compared to the previous research [10], the conformance testing tool chain is lifted to a higher abstraction level, and the NoTA stack testing is presented with much more detail. There's also tool evaluations provided to give guidelines about appropriate tools to be used with the framework.

The paper is structured as follows. Chapter II gives an overview of related research concentrating in testing in service-oriented interconnection technologies. Chapter III presents the NoTA market analysis and conclusions which were the basis for development decisions. We depict our conformance testing frameworks testing process in Chapter IV. Conformance testing tool chain framework and corresponding tool evaluations are discussed in Chapter V. Different NoTA testing approaches including detailed descriptions of NoTA stack testing and other aspects to consider along with NoTA DIP presentation are discussed in Chapter VI. Discussion about the developed conformance testing framework is presented in Chapter VII.

II. RELATED RESEARCH

This chapter illustrates recent examples of testing in service-oriented architectures and interconnection technologies. The purpose is to provide a scope and basis for comparison by reviewing how testing is organized in other application domains. The comparison and conclusion is presented in final Chapter VII.

Researchers in [11, 12] present a framework for testing and validation of both functional and non-functional behaviour of service-based applications. The framework is based on SOA principles. The proposed framework comprises a set of tools that can be used together with existing service development environments. The framework aims at automating the testing process and currently framework consists of five tools composed as integrated services. The framework provides end-to-end testing of three layers: 1) service layer, 2) service composition and coordination, and 3) business process. [11, 12]

The research team in [13] presents a framework for testing the I/O behaviour in SOA environment. The framework derives minimal testable I/O pairs from a service component's behaviour specifications. These minimal testable I/O pairs are mapped to reusable primitives and then synthesized into test models in the discrete event system specification formalism to meet different test objectives. The framework supports automatically constructing the test models and composing them for test scenarios. The framework is developed in the context of testing service collaborations on net-centric implementations of service oriented architectures. [13]

The paper [14] proposes a ConfigTest mechanism to support testing dynamic reconfiguration. Dynamic reconfiguration in SOA means that testing need to be adaptive to the changes of the service-oriented applications at runtime. The ConfigTest approach enables the online change of test organization, test scheduling, test deployment, test case binding, and service binding. The paper also presents test broker architecture framework which supports runtime testing with collaborative agents in a coordinated and distributed environment. The test broker decouples test case definition from its implementation and usage. [14]

The research team in [15] presents a performance evaluation method for evaluating the feasibility of new NoTA applications on multi-core based mobile device platforms. NoTA application components were refined to encompass processes and platform services and functions to form layered application architecture. The NoTA application architecture modelling methodology was linked with a workload modelling approach used with a performance simulation approach. [15]

Service-Oriented Architecture (SOA) services often use traditional validation approaches. This can be expensive, because each service and all possible service combinations must be tested separately. Tsai et al. address the problem with an open testing framework which uses group testing technique, and eliminates test cases with overlapping coverage. The aim is to reduce testing effort while retaining effectiveness by using an adaptive testing process. [16]

Namli et al. describe an automated test execution framework for Health Level Seven (HL7) based systems, which utilize a variety of transport protocols and message choreographies. The framework provides a test description language with high-level constructs allowing dynamic set up of test scenarios, and can accommodate different HL7 protocols with messaging adaptors. [17]

III. NOTA MARKET ANALYSIS

After our previous research concerning conformance testing in service oriented interconnection technologies in NoTA [10], we decided to delve deeper into the current market situation in the field of NoTA. The aim was to gather knowledge of the current NoTA ecosystem and testing markets. This knowledge was used to draw analysis of the current NoTA market situation, which in turn was used as a basis for future NoTA testing platform requirements and design decisions. According to our queries among industry professionals, following aspects were found about the current NoTA markets situation.

NoTA is not yet enough popular and visible to potential subsystem vendors. NoTA is perceived as a potential concept, but the business case is not clear. The main problem with NoTA subsystem markets is that there are not enough system integrators, who buy the NoTA subsystems. Subsystem developers cannot make large investments without seeing clear enough businesses in NoTA, which means that more NoTA system integrators should involve in the NoTA community. NoTA has good potential, but the technological penetration would occur not until around 2015. The commercial boom would be around 2020. There are some industries where NoTA could penetrate faster, such as automotive industry.

Another viewpoint that arose was the assumption that the majority of the NoTA subsystem vendor companies want to do their testing solutions by themselves, and are not willing to subcontract any of the NoTA testing work. However, this issue would change over time if the NoTA markets start to develop more. For example, Digital Living Network Alliance (DLNA) was assessed as a reference technology to find out how testing services are arranged in other fields of technology. It became clear that there already exists a strong inner circle of big testing companies that dictate the testing markets in the field of DLNA.

Regarding the NoTA testing viewpoints, there is need for NoTA stack testing separately, and also for the whole NoTA subsystem testing. Currently, the emphasis is especially on the stack, since there are not yet many NoTA services to test. In the future, there will also be room for NoTA performance testing. Focus to operate as a testing framework provider between NoTA subsystems developers and system integrators was deemed reasonable. It also became clear that it is essential to develop the NoTA know-how. It can also be seen that NoTA is just one service-oriented interconnection technology among others, and the developed conformance testing framework could also be used with other technologies. At the moment, we are seeing NoTA as the core use case in our future conformance testing framework development, but with the modification that NoTA is just one technology among others. The focus of future development of the NoTA platform will be directed to a more generalized direction. The current NoTA testing approach is also brought to a more detailed level. The aim is to develop the current NoTA testing platform to a direction that would be most useful for the NoTA ecosystem and manufacturers.

IV. NoTA Conformance Testing Process

The conformance testing process aims to assure that the subsystem conforms to NoTA specification and architecture. The process is a high-level depiction of what to do when a NoTA subsystem vendors and integrators engage in testing activities. The reasoning for the process is that NoTA subsystem vendors develop their own version of the NoTA stack in order to optimize their hardware and software performance. However, the NoTA subsystem must conform to the NoTA stack specification in order to be interoperable with other NoTA subsystem modules. The conformance testing process helps subsystem developers to reduce subsystem investment risks by offering a way to test the custom NoTA stack of the subsystem developers against the NoTA specification, and to assure the interoperability of the whole subsystem with other NoTA subsystems. [10] The main objectives of the conformance testing process are to offer:

- added value for NoTA subsystem vendor: 1) proof that the NoTA subsystem conforms to the NoTA specification and architecture, and 2) proof that the NoTA subsystem services meets its functional requirements and conform to their specifications.
- added value for NoTA subsystem integrator: 1) confidence that the NoTA subsystem is interoperable with other NoTA subsystems, and 2) faster time-to-market due shortened integration & testing.

The conformance testing process, its inputs and outputs, and its phases are illustrated in Fig. 1, and are described in the following chapters:

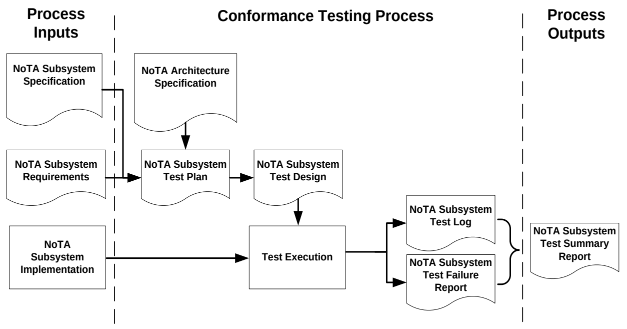
Process Inputs
Conformance Testing Process
Process Outputs
NoTA Subsystem Specification
NoTA Architecture Specification
NoTA Subsystem Requirements
NoTA Subsystem Test Plan
NoTA Subsystem Test Design
NoTA Subsystem Implementation
Test Execution
NoTA Subsystem Test Log
NoTA Subsystem Test Failure Report
NoTA Subsystem Test Summary Report

Fig. 1 Conformance testing process for NoTA [10]

A. Process Inputs

NoTA subsystem vendor offers subsystem's Spesifications, NoTA Subsystem Requirements and NoTA Subsystem Implementation to the party implementing the conformance testing process. The separate NoTA subsystem Requirements document captures the non-functional requirements of the subsystem. The NoTA Subsystem Implementation is the actual subsystem, which comprises of hardware and/or software. [10]

B. Process Outputs

The party implementing the conformance testing process produces a Test Summary Report, which includes all executed test cases, their results, and tracing to NoTA Subsystem Requirements. The Test Summary Report will include an overall report of the NoTA subsystems conformity and interoperability. The Test Summary Report is delivered to the NoTA subsystem vendor along with the NoTA subsystem. [10]

C. Conformance Testing Process Phases

This chapter describes the phases in the conformance testing process for NoTA. The party implementing the conformance testing process executes these phases. The phases were originally presented in [10].

- NoTA Architecture Specification (Device Interconnect Specification) provides the NoTA specification that sets the overall constraints for testing. The NoTA community provides the specification and is responsible for maintaining it.
- NoTA Subsystem Test Plan provides a high-level overview of the testing activities for the corresponding subsystem. It has NoTA Architecture Specification, NoTA Subsystem Specification, and NoTA Subsystem Requirements as it inputs. The test plan provides the basis for NoTA Subsystem Test Design. The NoTA subsystem vendor is responsible in delivering the required input documents.

- Test design defines the technical details of the test system setup, which must be addressed: the required hardware and software, testing tools, SUT adapters, test methods, and test cases. Test design phase receives NoTA Subsystem Test Plan as its input, and provides a framework for executing tests.

- Test execution tests the system for failures, according to the test design. Test execution phase receives NoTA Subsystem Test Design and NoTA Subsystem Implementation as its inputs, and creates NoTA Subsystem Test Log and NoTA Subsystem Test Failure Report as its outputs.

- Test reports phase summarizes and gives details of test results. The phase receives NoTA Subsystem Test Log and NoTA Subsystem Test Failure Report as its inputs, and creates NoTA Subsystem Test Summary Report as its output.

There exist a few possible problems in the NoTA testing process: 1) the NoTA technology must itself confirm the interoperability of NoTA architecture subsystems, but at the same time the specification must be flexible and not too rigid to allow development of the NoTA community. This raises the question that how to ensure consistency between NoTA specifications. One approach could be annual internal specifications update conferences between NoTA ecosystem members. 2) The purpose of NoTA is to enable flexible interconnectivity inside and between devices. This causes concern over unambiguity and generality of the NoTA interface, because subsystem internal functionalities may affect how you can use the interface. 3) Another question concerning the process is that are the actual testing activities performed in subsystem vendor premises and participated to vendor product development lifecycle, or are testing activities performed outside the vendor premises.

V. TESTING TOOL CHAIN FRAMEWORK

Due to conclusions drawn from NoTA market analysis, we are now seeing NoTA as one use case in our future framework development, where NoTA is just one technology among others. Therefore, the new focus of the tool chain development is to define a framework and requirements for a general purpose conformance testing tool chain that supports the testing process defined in the previous research [10].

Service-oriented technologies are a new area from viewpoint of testing and therefore traditional, formal testing methods and systems seem not to fit as such [9, 14]. Therefore we have outlined a novel testing tool framework that elaborates the types of tools needed in different phases of the testing process defined in the Chapter IV (Fig. 1). The main purpose is to support the transactions and deliverables described in the process with the conformance testing tool chain. However, since the focus of the conformance testing tool chain is shifted to a more general direction, the NoTA-specific details will be omitted. Since there is no specific target SUT technology, the main concern is to take into account the general level deliverables that are transferred between the testing partner and the potential customer, and how those deliverables affect to the process and the tool chain. In Fig. 2, rough, high-level transactions are illustrated. The customer must provide specifications about the system to be tested (the SUT), the actual SUT implementation for testing, and the customer must assist the testing partner in SUT specific technical issues. The testing partner produces and delivers a comprehensive test report to the customer. In a technology specific scenario there would be more transactions to be identified, and as a comparison, the NoTA conformance testing process (Fig. 1) identifies additional transactions: NoTA Acceptance Label and SIS specifications.

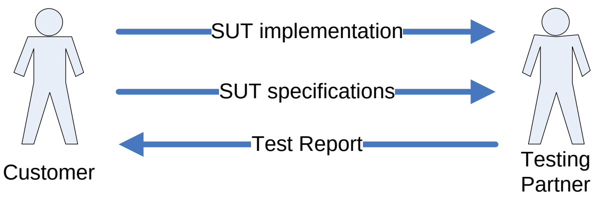

Fig. 2 Transactions between customer and testing partner

A. *High-Level Architecture of the Tool Chain Framework*

The diagram in Fig. 3 depicts the high-level architectural concept diagram of the tool chain framework. The yellow figures represent documents that flow between the tools, which are manifested in green box figures. The blue figures are data entities flowing between the tools and documents. The grey disk represents the SUT implementation.

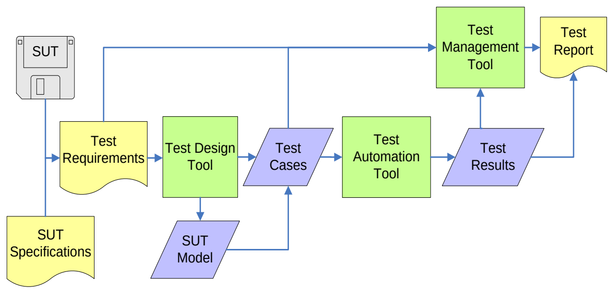

Fig. 3 A high-level architectural diagram of the tool chain framework

The transactions in Fig. 3 possess some features of the process flow in Fig. 1, but Fig. 3 also depicts the required tools, and their placements in the process. The framework states that there are three major tool requirements: 1) test design tool, 2) test automation tool, and 3) test management tool. It should be noted that these three major tool classifications may include many smaller testing tools, e.g., the test automation tool may possess separate test execution and test logging application. In addition, the test design tool could be, e.g., model-based testing, or some other method that may include a lot of manual labour, or scripting.

The testing tool chain framework is a high level architecture and it is aimed to provide a wide testing tool support in order to able to test SUTs thoroughly. The SUT specifications consist of any SUT specification documents or other resources that are used to ensure that the SUT conforms to its specifications (and to the functional requirements).

The tool chain framework provides an interface for exporting test reports and test logs. The tool chain components are to be integrated so that the actual testing process and activities can be executed without any extra development efforts. Extra development effort means that the interfaces of the tools must be customized. However, this does not include possible test harness and test adapter development.

The SUT specifications declare the structure, the functional behaviour, and the interfaces of the SUT software. The SUT specifications provide the basis for test requirements.

The test design tool provides an interface for entering test requirements and resources for designing tests. The test design is automated by using, e.g., model-based testing (SUT model), scripting, etc. The test design tool produces a comprehensive set of test cases that exercise the SUT thoroughly. These test cases contain the stimulus, and the test oracle, and they must be traceable back to SUT requirements.

The test cases have to clearly separate the data of different tests (precondition, expected results, test input data). Each test case has to contain a single event sequence, which performs a single task (for example, dial a number) and can be reused in other test sequences.

The test management tool provides interfaces for entering test requirements and test cases designed with the test design tool. The test management tool must be usable: simple, simple guidelines, visible, transparent, not many changing parts. The test management tool produces a test report. In the report, the test management tool must be able to: produce analytical and statistical data of the test cases, return to previous test results and compare results of different test runs, and have version control of test cases/scripts.

The test automation tool contains test preconditions to run the SUT and other equipment to appropriate state for the start of the real test. The test automation tool initialises test control, SUT, and other equipment to be controlled. The test automation tool provides comprehensive logging of the test execution and SUT actions during testing, a test harness/adapter in order to adapt to the SUT interfaces. The test automation tool must provide a mechanism to control the SUT and feed it with test input data, and provide log as a report of the test results. The test report must contain:

- All the executed tests and their traces to SUT requirements.
- A test failure report that analyses the reasons for the failed cases and their relation to the SUT requirements.
- Comprehensive graphs about test coverage and failed tests, differences of current and previous runs.
- Information, which is collected during testing: time, tester, failed and succeeded tests.
- The tested SUT software version and modifications, which have been tested.
- Information of used testware (documents and systems used in testing) such as test plans, test cases, test databases, test output, test documentation and test reports.

B. *Tool Evaluations*

This chapter presents tool evaluations made to assess possible tools to be used in the conformance testing tool chain framework. First, a general presentation of the tool is provided, and then the tool is assessed in the context of conformance testing tool chain framework.

1) *Topcased* [18]

Topcased is an eclipse-based open-source toolkit. It has features for creating diagrams using AADL, SysML and UML. In addition to modelling, Topcased provides model transformation and code generation features. Topcased is most often used to model UML, which is a standardized, general-purpose modelling language that includes graphical notation used to create an abstract model of a system. The purpose of the Topcased in the conformance testing tool chain would be to offer modelling services in cases where the tool chain would use model-based test generation. The Topcased modelling tool would enable the usage of third party UML models in MBT. Topcased can be used to design and convert third party UML models into a form that can be used in, e.g., Conformiq Qtronic. This would be a definite asset, if the potential client uses UML models in their SUT specifications. In these cases, the usage of Topcased would accelerate the model development. Obviously, the Topcased tool would be feasible only in testing scenarios where the MBT methodology is present.

2) *MaTeLo Usage Model Editor/Testor* [19]

The MaTeLo tools (Editor/Testor) are bound to each other, and one can’t be used without the other. The MaTeLo tools utilize Markov chain models, which can be created with MaTeLo Usage Model editor. Markov chain notation is a state diagram notation where transitions are related to SUT I/O and the activation frequency of states. MaTeLo Testor is a model-based test generator, which utilizes MaTeLo test models. MaTeLo Testor generates test cases according to user selected test criteria, and automatically generates the needed number of test scenarios conformed to the most probable usage of the system. Test suite export options are limited to following formats: XML/HTML, TTCN-3 and TestStand. MaTeLo tools are a reasonable option for tool chain scenarios where MBT methodology is used. MaTeLo tools are specialized in complex embedded and IT systems in various sectors and industries such as automotive, transportation, energy, and telecommunications. MaTeLo generates test suites using probabilities set to transitions of the test model. This leads to test cases where some transitions with high probability are selected frequently and other ones with low probability are selected rarely. In MaTeLo test generation, every test case is generated uniquely and therefore parts of the test model which are already covered in the previous test cases do not affect to later generated test cases. This leads to randomly generated test suites in which several test cases can contain the same input feeds multiple times and therefore, by using MaTeLo it is hard to generate test suites which would cover the whole test model with minimum (reasonable) amount of test cases.

3) *Qtronic*[20]

Conformiq Qtronic is a model-based testing tool which supports offline model-based test generation from models defined in QML, which is a UML/Java oriented modelling language developed by Qtronic. Qtronic supports third party UML model import, and provides model validation operations. The test reports can be exported in XML and HTML. Qtronic provides its own modelling tool, Qtronic modeller, along with the tool package delivered. Qtronic generates test suites according to user selected test criteria (e.g., coverage criteria or lookahead depth), and Qtronic Script adapter can be used in rendering test suites in e.g. TTCN-3 format. Qtronic is mostly used in protocol and communication systems testing, and is an efficient tool for designing tests. Therefore Qtronic is a reasonable option for test scenarios utilizing model-based testing. The QML modelling language is somewhat limited, which should be noted when designing test models. The SUT in question may possess qualities that can be hard to describe in QML. Also, just like in any MBT case, the MBT brings most benefits in cases where regression testing plays a big role in the overall SUT development process.

4) *TTworkbench* [21]

TTworkbench from Testing Technologies is an integrated test development and execution environment (IDE) for all kinds of test automation projects. TTworkbench is a TTCN-3 based tool and it can be deployed for testing scenarios in a wide range of different industries. The two main views in the TTworkbench working environment are the development perspective and execution management perspective. The development perspective comprises mainly of the CL Editor, which is a text editor that provides capabilities for editing TTCN-3 Core Language based test suites. Another major component of the Development perspective is TTthree, which provides the generation of Java sources from test suite specifications based on the TTCN-3 Code Language, as well as the compilation of Java sources into byte code class files and their packaging into a single jar archive file. The TTthree is used in developing adaptation and codec interfaces to adapt the TTCN-3 test system to the needs of the SUT. The execution management perspective, also known as the TTman, builds the whole execution management perspective used in executing tests against the SUT. TTworkbench is a flexible tool due to its modularity, and offers efficient test automation features. TTworkbench is a strong option for the test execution platform of choice when constituting the conformance testing tool chain.

5) *CUnit* [22]

CUnit is an open source unit testing framework for C language. CUnit is a lightweight system for writing, administering, and running unit tests. It provides a basic testing functionality with a flexible variety of user interfaces. CUnit proved to be too small (feature-wise) for the conformance testing tool chain. The CUnit tool is intended to be used in unit testing during early phases of development, and this often means that the developer oneself executes the unit tests. However, there could be some special cases where the CUnit could be utilized, such as detailed debugging after finding a bug from a SUT. The main problem with the CUnit is the lack of adequate test automation, management and logging. Another downside of the tool is that it is limited to development done in C language only.

6) *OpenTTCN Tester 2010* [23]

OpenTTCN Tester 2010 is a TTCN-3 testing tool developed by OpenTTCN. OpenTTCN Tester allows test editing, compilation, and execution as well as adapter development in ANSI C, Java and C#/Microsoft .NET Framework. OpenTTCN Tester resembles TTworkbench. OpenTTCN Tester recently changed their operating environment to Eclipse (as in TTworkbench), and the layout and major software component functionalities are quite similar to TTworkbench. OpenTTCN Tester 2010 lacks some features (e.g., test execution perspective and management, extensive textual logging, graphical test logging) found in TTworkbench, and is therefore limited compared to the TTworkbench. OpenTTCN Tester is a good test automation platform option, but offers no extra value.

7) *Testia Tarantula* [24]

Testia Tarantula is a test management tool developed by Prove Testia. The Tarantula test management tool enables to manage tests that have been previously designed. Tarantula includes requirement specifications management feature that allows generating a requirements matrix, which is a metric to know the functional coverage of the SUT. Tarantula can deliver metrics that will help in evaluating the quality of the SUT. Metrics are graphics and tables indicating success rates, progression/regression and much other data. Tarantula also possesses bug tracking features. Tarantula is a respectable option for the test management tool role in the conformance testing tool chain framework.

VI. NOTA TESTING APPROACHES

The following chapter present aspects, discussion and requirements for utilizing the conformance testing tool chain framework in NoTA testing. The NoTA use case aims to sketch and develop extensive NoTA testing framework and test cases for NoTA subsystems, DIP stack layers, and AN/SNs.

To understand NoTA testing, we first must present the NoTA architecture. NoTA was introduced in [25] and [26]. NoTA is an architecture for communication between system modules, which are called subsystems. The purpose of the NoTA architecture is to provide a unified way to define module interfaces in embedded devices. The NoTA architecture concept applies ideas from SOA world, but NoTA implement them in a device context. [25] The architecture concept and technologies of NoTA are operating system independent and portable to different platforms [8].

The logical NoTA architecture consists of two types of nodes: 1) Application Nodes (AN), and 2) Service Nodes (SN). These nodes are implemented with software, hardware or combination of both. The communication between nodes is managed using Device Interconnect Protocol (DIP). The DIP consists of two major interconnect layers: 1) High Interconnect (H_IN) and 2) Low Interconnect (L_IN), which is divided into two sub layers: 1) L_INup and 2) L_INdown. A full working NoTA system comprises of NoTA subsystems. A NoTA subsystem contains an implementation of the DIP stack and a certain set of nodes. [26, 8] The parts of the NoTA architecture are illustrated in Fig. 4.

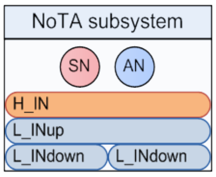

Fig. 4 NoTA architecture [25, 26]

NoTA DIP is intended to be a platform independent protocol and therefore it is ported for numerous different platforms. Different DIP stack versions may have unique optimizations or custom implementations in order to adapt to different platforms and processors. Conformance testing is needed to ensure the proper functionality of the optimized NoTA DIP stack, and the interoperability between all different stack implementations. The test requirements are drawn from NoTA DIP specification, and the test requirements provide the basis for test cases. In practice, the testing can also be implemented against an 'optimal' reference implementation of the DIP stack. It would be also interesting to gain comparative test data of the different implementations of the NoTA DIP stack. Possible deviations in the performance, such as in overhead/throughput would be of interest.

When considering NoTA testing, the first thing to take into account is the testing target. The testing target can be either whole DIP stack, all stack layers separately layer (H_IN, L_INup, L_INdown), ANs and SNs, and the whole subsystem (Fig. 4). The testing target affects directly if the testing can be done in a testing host or as remote testing. If the SUT NoTA DIP stack is optimised for SUT target platform, it cannot be installed to our testing host. If the NoTA subsystem is already compiled, or resides in a closed embedded device, it must be tested remotely. It should also be noted that there might be requirements for testing many simultaneous ANs and SNs in order to be able to assess the overall NoTA subsystem functionality. Because NoTA as a technology is intended to operate on multiple platforms, another major requirement for a feasible testing system is reusability for different platforms.

Functional testing of NoTA nodes requires that node specifications are received from NoTA vendor. Also, the test automation tool must be able to communicate with the SUT through NoTA stack using H_IN interface. In order to the test system to be reusable, all node-specific functionality must reside in the test cases, not in the SUT adapter. The NoTA testing platform must be able to simulate SUT dependent ANs/SNs.

NoTA subsystem interoperability testing requires that subsystems' specifications are received from NoTA vendors, including node specifications and possible modifications in the stacks. The test automation tool must adapt to environment with multiple NoTA subsystems, must be able to inspect communication between the subsystems, and simulate the functionality of one or more subsystems under testing.

A. *Negative and Performance Testing*

The main testing type for NoTA testing is functionality/conformance testing of all the testing targets. The testing of functionality can also be, e.g., negative testing. The purpose of the negative testing is to cause the stack to crash. A protocol stack is not allowed to crash under any circumstances. If the stack crashes, the test debug messages indicate in which part of the stack the error occurred. Negative testing for NoTA is test execution where: broken messages, too long messages, a broken message in between unbroken ones, sudden change in parameters are inserted in the NoTA message flow to inspect how the DIP stack operates in cases of anomalies. The negative testing can also be anomalies in connections: suspensions during connection initialization, many simultaneous connections, illegal connections, sudden connection creations and terminations in row, buffer overflows, and sudden disconnections.

Even though the conformance testing does not cover necessarily performance, a certain degree of performance testing is required to assure that the NoTA DIP stack has an adequate performance to operate with other NoTA DIP stack implementations. This kind of performance limit testing can cover: amount of time spent during discovery, amount of time spent in creating connections (e.g., versus TCP stack), bit rates with data with different packets sizes, differences between different L_INdown performances (TCP/IP, Bluetooth), time used when changing L_INdown on the fly, load/stress testing, and finding the performance limits of a stack. Another aspect of performance testing is drawing comparison between different stacks with throughput / round-trip delay testing. The performance testing is challenging because the NoTA stacks have different platforms and different host devices. Drawing objective comparable data is difficult. Security testing of the NoTA stack becomes topical in a context, where there might be many simultaneous NoTA resource managers.

B. *NoTA DIP Stack Testing Approaches*

In order to be able to test the NoTA DIP stack layers, there must be NoTA specifications available. Since there are no official specifications for the L_INdown layer, the testing of L_INdown layer requires specific interoperability testing arrangements. Regarding the other layers of NoTA, the specifications must provide finite state machines and APIs for each stack layer, and for each used socket. The specifications must also provide data type definitions of the input and output messages.

Stack layer testing is implemented a in a testing host PC, or in an embedded device with the aid of remote stubs. The stubs are platform specific, and may require a transport specific dongle. The inner state of stack layers must be monitored during testing, which requires test probes to enable the capture of messages from, and between stack layer interfaces. This requires a modified test version of the NoTA DIP stack.

1) Whole DIP Stack Testing Approach:

In the whole DIP stack testing approach (Fig. 5) the test execution platform sends input message via a communication medium and receives output via test node connection. Then the test executor compares real output value to the expected output value and makes decision of test case validity. Test executor can also send data to the node side and collect the return value from the other side. Therefore this approach makes it possible to test NoTA stack carefully and have good test coverage. This is a generic aspect of test automation and is very adaptable in conceptual level. Requirements for whole DIP stack testing: 1) test executor connection to H_IN and L_INdown layers, 2) node stub implementation, and 3) test executor connection to test node stub.

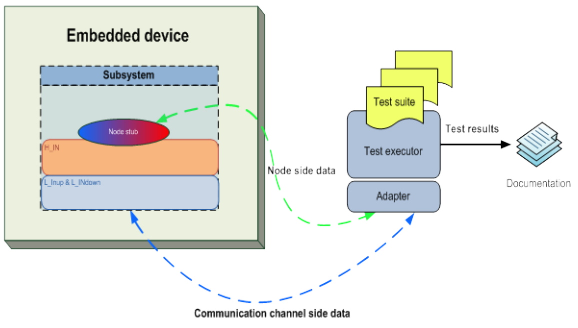

Fig. 5 Whole DIP stack testing approach

2) DIP Stack Layer Testing Approaches:

Layer testing is very close to stack testing in conceptual level. In practice however it is quite different because each layer of the stack has distinct interfaces. To test the DIP stack layers separately, they must be run as separate standalone layers. This also entails that the test automation tool must enable testing hooks that surround each part of the stack. This can be enabled e.g. via an adapter interface of the test automation tool. Interface requirements for L_INdown testing (Fig. 6): 1) testing of L_INdown requires a L_INdown-specific SUT adapter, 2) upper part of the testing hook uses L_INdown interface, and 3) lower part of the testing hook simulates various communication methods (TCP, Bluetooth, USB).

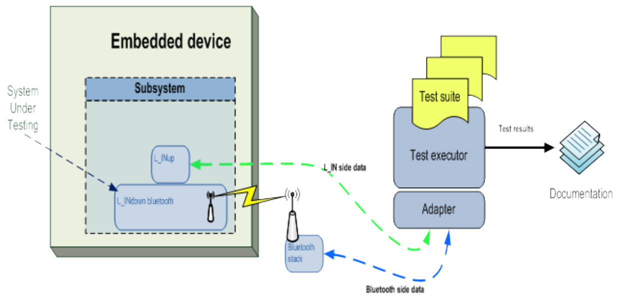

Fig. 6 L_INdown testing approach

Interface requirements for L_INup testing (Fig. 7): 1) testing of L_INup requires a L_INup-specific SUT adapter, 2) upper part of the testing hook uses L_INup interface, and 3) lower part of the testing hook provides L_INdown interface.

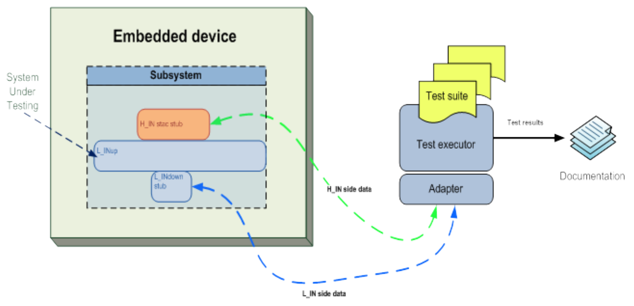

Fig. 7 L_INup testing approach

Interface requirements for H_IN testing (Fig. 8): 1) testing of H_IN requires a H_IN-specific SUT adapter, 2) upper part of the testing hook uses H_IN interface (node-test adapter might be reused), and 3) lower part of the testing hook provides L_INup interface.

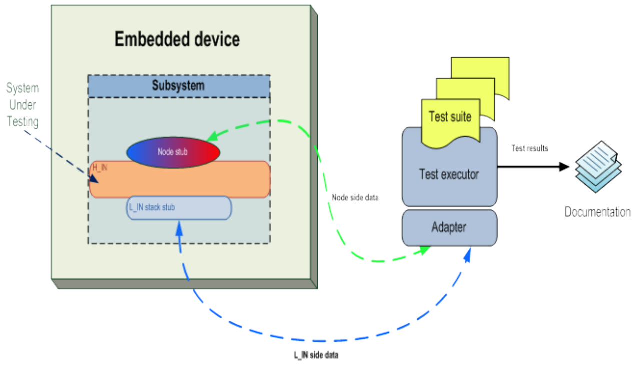

Fig. 8 H_IN testing approach

VII. CONCLUSIONS

We presented a conformance testing framework that included testing process, testing tool chain framework, tool evaluations and detailed approaches for NoTA stack testing. Service-oriented technologies are a new area from viewpoint of testing and therefore traditional, formal testing methods and systems seem not to fit as such [9, 14], and therefore we have outlined a novel testing tool framework that elaborates the types of tools needed in different phases of the testing process. The work presented in this paper elaborated previous research in [10] with a novel generalized testing tool chain framework and elaborated stack testing approaches that were engaged due to conclusions drawn from NoTA market analysis. The new paradigm of the conformance testing framework answers to the viewpoints raised by industry professionals.

These viewpoints include: 1) NoTA has good potential, but the technological penetration would occur not until around 2015, and NoTA subsystem vendors want to do their testing solutions by themselves. This is why we raised the abstraction level of the framework to include also other service-oriented technologies, and kept the NoTA is just one service-oriented interconnection technology among others.

The framework solution can also be utilized by the subsystem vendors. 2) It became clear that it is essential to develop the NoTA know-how. There is need for NoTA stack testing separately, and in the future also NoTA performance testing. To address these challenges, we elaborated the NoTA stack testing approaches, and also discussed the challenge of performance testing. 3) The focus of the testing framework to operate as a testing service provider between NoTA subsystems developers and system integrators was deemed reasonable. This viewpoint was retained.

When compared to existing research, the proposed solution in [11, 12] has an extensive set of tools, but the disadvantage in the solution is that the implementation of the framework is based on enterprise service bus, and it focuses on testing web services and service-based applications. The same applies for the [13], where the research concentrates on testing service collaborations on net-centric implementations of service oriented architectures, and [14] proposes a mechanism to support testing dynamic reconfiguration in web services testing. In turn our framework, which besides mere applications concerns also the service architecture implementation and its components. In addition our tool framework approach can be adapted to needs of different application domains. The solution in [13] bares resemblance to one presented in this paper since as it regards the SUTs I/O at behavioral level with testable I/O pairs. The work in [15] presents a performance evaluation method for evaluating the feasibility of NoTA applications, which is a far more developed performance testing concept than the discussion presented in this paper.

The testing framework in [16] does not consider any test execution methodologies, contrary to our work. The adaptive testing process in [16] relies on selecting the most efficient test cases, but does not define any roles or responsibilities associated in the testing process. The work in [17] does not present a testing process or guidelines to define e.g., what happens in testing between the HL7 parties and what data is exchanged. Our conformance testing process for NoTA considers the different actors and data flows associated in a testing process. Our process defined for NoTA is not directly applicable for other domain areas, but still provides a solid foundation.

The future work and more detailed assessment of markets could involve the following challenges: 1) what kind of testing could be most useful from the stack development perspective, 2) should there be a defined stack testing process, and 3) are the stack layers tested first one by one or do they form bundles.

An issue to be noted when considering the utilization of the testing tool chain framework is that when considering the tool chain components for NoTA DIP stack testing, it should be thought through that is the dependency of proprietary testing tools necessary, and is there any interface conflicts between the tools.

REFERENCES

[1] C. Fortuna and M. Mohorcic, “Dynamic composition of services for end-to-end information transport,” Wireless Communications, IEEE, vol. 16, no.4, pp. 56-62, Aug. 2009.

[2] L. Siegele, “A Special Report on Corporate IT,” The Economist, vol. 389, no. 8603, Oct. 2008.

[3] C. E. Abosi, R. Nejabati, and D. Simeonidou, “A service plane architecture for future optical Internet,” International Conference on Optical Network Design and Modeling, ONDM 2009, 2009, pp. 1-6, 18-20 Feb. 2009.

[4] C. Hua and J. Chen, “Service-Oriented Transparent Interconnection between Data-Centric WSN and IP networks,” International Conference on Electrical and Control Engineering, ICECE, 2010, pp. 1884-1887, 25-27 June 2010.

[5] L. Zongwe, J. S. Li, C. J. Tan, F. C. H. Tong, A. Kwok, E. C. Wong, and H. B. Wang, “Intelligent Middleware Service Framework,” IEEE International Conference on Service Operations and Logistics, and Informatics, SOLI '06, 2006, pp. 1113-1118, 21-23 June 2006.

[6] G. Zervas, V. Martini, Y. Qin, E. Escalona, R. Nejabati, D. Simeonidou, F. Baroncelli, B. Martini, K. Torkmen, and P. Castoldi, “Service-Oriented Multigranular Optical Network Architecture for Clouds,” IEEE/OSA Journal of Optical Communications and Networking, vol. 2, no. 10, pp. 883-891, Oct. 2010.

[7] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, “A Real-Time Service-Oriented Architecture for Industrial Automation,” IEEE Transactions on Industrial Informatics, vol. 5, no. 3, pp. 267-277, Aug. 2009.

[8] (2011) NoTA - Nokia Developer. [Online]. Available: http://projects.developer.nokia.com/NoTA.

[9] G. Canfora and M. Di Penta, “Testing services and service-centric systems: challenges and opportunities,” IT Professional, vol. 8, no. 2, pp. 10-17, March-April 2006.

[10] J. Keranen, T. Raty, P. Jurmu, M. Maki, and O-P. Puolitaival, “Conformance Testing in Service-Oriented Interconnection Technologies in NoTA,” Eighth International Conference on Information Technology: New Generations, ITNG, 11-13 April 2011, pp. 304-309.

[11] V. Pavlov, B. Borisov, S. Ilieva, and D. Petrova-Antonova, “Framework for Testing Service Compositions,” 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, 23-26 Sept. 2010, pp. 557-560.

[12] S. Ilieva, V. Pavlov, and I. Manova, “A Composable Framework for Test Automation of Service-Based Applications,” Seventh International Conference on the Quality of Information and Communications Technology, QUATIC, 29 Sept. - 2 Oct. 2010, pp. 286-291.

[13] H. Xiaolin, B. P. Zeigler, H. H. Moon, and E. Mak, “DEVS Systems-Theory Framework for Reusable Testing of I/O Behaviors in Service Oriented Architectures,” IEEE International Conference on Information Reuse and Integration, IRI 2007, 13-15 Aug. 2007, pp. 394-399.

[14] B. Xiaoying, X. Dezheng, D. Guilan, T. Wei-Tek, and C. Yinong, “Dynamic Reconfigurable Testing of Service-Oriented Architecture,”31st Annual International Computer Software and Applications Conference, COMPSAC 2007, vol. 1, 24-27 July 2007, pp. 368-378.

[15] S. Khan, J. Saastamoinen, J. Huusko, and J. Nurmi, “Performance evaluation of distributed NoTA applications on multi-core platforms,” IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications, NESEA, 8-9 Dec. 2011, pp. 1-8.

[16] W. T. Tsai, Z. Xinyu, C. Yinong and B. Xiaoying, “On

Testing and Evaluating Service-Oriented Software," Computer, Vol. 41, No. 8, pp. 40-46. August 2008.

[17] T. Namli, G. Aluc and A. Dogac, "An Interoperability Test Framework for HL7-Based Systems," IEEE Transactions on Information Technology in Biomedicine, Vol. 13, No. 3, pp. 389-399, May 2009.

[18] (2011) Topcased - The Open-Source Toolkit for Critical Systems. [Online]. Available: http://www.topcased.org/.

[19] (2012) MaTeLo - ALL4TEC Model-based Testing Solutions. [Online]. Available: http://www.all4tec.net/index.php/All4tec/matelo-product.html.

[20] (2012) Conformiq Automated Test Design. [Online]. Available: http://www.conformiq.com/.

[21] (2012) Testing Technologies. [Online]. Available: http://www.testingtech.com/products/ttworkbench.php.

[22] (2010) A Unit Testing Framework for C. [Online]. Available: http://cunit.sourceforge.net/.

[23] (2012) OpenTTCN. [Online]. Available: http://www.openttcn.com/.

[24] (2012) Testia Tarantula. [Online]. Available: http://www.testia.fi/.

[25] K. Kronlöf, S. Kontinen, I. Oliver, and T. Eriksson, "A Method for Mobile Terminal Platform Architecture Development", Advances in Design and Specification Languages for Embedded Systems, Vol. 4, 2007, pp. 285 – 300.

[26] R. Suoranta, "New Directions in Mobile Device Architectures," 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, DSD 2006, October 2006.