Steps towards model-based, inference-driven SOA Testing

Ariele P. Maesano [1] [2], Fabio De Rosa [2], Libero Maesano [2], Perre-Henri Wuillemin [1]
{Ariele.Maesano, Pierre-Henri.Wuillemin}@lip6.fr
{ariele.maesano, fabio.de-rosa, libero.maesano}@simple-eng.com
[1] Laboratoire d'Informatique de Paris 6 UPMC, Paris, France
[2] SIMPLE ENGINEERING, 31-35, rue St. Ambroise, 75011 Paris, France

Abstract: This paper reports the progress of a project whose goal is to develop a model-driven, contract-based SOA testing environment in which grey-box testing strategies for large services architectures are driven by stochastic inference. In order to achieve this goal, a Bayesian network is built directly by mapping and model transformation from the services architecture design and test models. Because services architectures are complex and large systems, the generated BN and its associated probability tables may also be very large and the inference process may be excessively slow. Nevertheless, the structures of the services architecture design and test models let techniques of compilation improve the inference task, enabling the use of BN inference as a sustainable tool for driving failure discovering and troubleshooting strategies in large scale services architectures.

Keywords: Arithmetic Circuit, Bayesian Network, BN4SAT, contract-based, inference, model-driven, SOA, testing, troubleshooting, TTCN-3.

1. INTRODUCTION AND BACKGROUND

In the *contract-based* [19], *model-driven* [21] approach, Service Oriented Architecture (SOA) is a design and implementation style that has three main characteristics: (i) the collaboration among systems is carried out through *the exchange of services regulated by contracts*; (ii) *service contracts* are *formal models* of the service *functions* and of the provider and consumer *interfaces* and *external behaviors* (including *security* and *quality of service* aspects); (iii) service contracts do not include any information about internals of the systems that comply with them. A *services architecture* [19] is a network of participant systems, playing roles bound by *service contracts*, that collaborate in order to achieve business goals.
In this conceptual framework, the well-known validation question, i.e.: "Are you building the *right* services architecture?", is split into two related sub-questions: (i) a more specific validation question: "Are you formalizing the *right* service contracts?" and (ii) a verification question: "Are you implementing participant systems *right*, i.e. in compliance with the validated service contracts?".
The model-driven approach [21] [25] [16] simplifies the validation and verification task. Once the service contract model has been validated, a large part of the overall validation task is converted in the verification of the correct application of the model mapping and transformation until implementation. But the last verification question (about the compliance between the implementations and the contracts) stays open, because, in the SOA approach, system implementations are *black boxes*, *hidden* and *private* to the participants. Hence SOA Architects and Integrators can employ neither program static check methods nor white-box testing. The only available verification methods are *black-box testing* of individual participants and *grey-box testing* of interactions among participants, where they are observable.
SOA testing is perceived as a complex and huge task [5] [29]. The ongoing research spans three main areas: (i) the automated generation of test cases from contract models [3] [4], (ii) the automation of the test execution and evaluation environments [10] [27] [28] [31] and (iii) inference-based methods and tools in order to help Testers to efficiently manage test campaigns. To the best of our knowledge, there are not yet applications of *probabilistic inference* to the management of SOA testing campaigns. Our research focuses on this topic.
This paper reports the progress of a project (Bayesian Networks for Services Architecture Testing - BN4SAT) centered on the development of a Bayesian network (BN) inference approach and technology to services architecture testing management. The project is conducted in cooperation between the Laboratoire d'Informatique de Paris VI (LIP6 - Université Pierre et Marie Curie - Paris VI - France), the Centre National de la Recherche Scientifique (CNRS - France) and Simple Engineering, a European group specialized in the design of services architectures. The project is partially funded by the Association Nationale de la Recherche et de la Technologie (ANRT - France).
The remainder of the paper is organized as follows. Section 2 presents related work on the application of stochastic inference to system testing and troubleshooting. Section 3 describes the architecture of the testing environment. Section 4 aims to clarify the peculiar characteristic of the application of the model-driven approach to services architecture testing. Section 5 presents a Bayesian Network (BN) architecture specially designed for

stochastic inference upon SOA functional and interaction testing. A short discussion, some hints on future work and some conclusions are drawn in section 6.

2. RELATED WORK ON STOCHASTIC METHODS IN TESTING AND TROUBLESHOOTING

Stochastic methods are considered appropriate for failure detection and diagnosis of complex systems in cases where there is no complete knowledge of the system, i.e. the detection and diagnostic process is undertaken in presence of uncertainty and the evidence data domain is too much large to be completely analyzed [17].
Reference [30] describes a seminal work about the use of probabilistic inference based on a BN framework to understand which kind of input reveals software failures. Indeed, starting from a partitioning of the input domain, the BN allows the tester to highlight what partition or combination of partitions can be associated with a failure. Driven by the BN inference, the test system dynamically chooses the next test case in order to quickly determine the input associated with the faulty behaviour of the system.
Reference [1] proposes the use of *fault trees* for system troubleshooting. Fault trees allow locating the faulty components by means of *inquiries*. The construction of the fault tree for a complex system proceeds in a top-down fashion, from *events* to their *causes*, following the system decomposition, until elements revealing faults of basic components are reached. In [1] the authors use fault trees to build a BN, which is able, starting from a failure, to locate the component or the set of components with the highest fault probability, in particular the *minimal cut set* (minimal sets of components that need to be all defective to cause the system failure).
The use of probabilistic inference [22] appears to be adequate for discovery of failures and search and identification of faulty participants in a services architecture, where the only available information is the *match*/*mismatch* between the actual external behavior and the expected one.

3. SOA TESTING ENVIRONMENT

For the purpose of SOA testing, a services architecture (Services Architecture Under Test - SAUT) is a network of *systems* (System Under Test - SUT) connected by *channels* conveying *communicative actions* such as *messages, remote procedure calls (rpc), rpc replies*. Some or all of these channels are *observable* (*grey-box stance*). Conversely, the SUTs *internals* are *never observable* (*black-box stance*). A *test run* takes place when a Tester (a human being or a system) submits to a SUT a *stimulus* and compares the SAUT *actual response* - the exchange of actions between SUT's following the stimulus - with the *expected response* - the expected action exchange in a specified SAUT state. A test *matches* (*mismatches*) if the actual response corresponds to (differs from) the expected one. Black-box tests are run against individual SUTs whose internals are hidden and whose connections with other participants are not observed - only the SUT responses to stimuli are observed. Grey-box tests are run against a network of SUTs that are involved in an end-to-end interaction, for instance within a business process, where the channels among these SUTs are observed by some appropriate components of the testing environment.
The architecture of the considered testing environment is compliant with the OMG’s UML Testing Profile (UTP) [20], which is generally accepted as the reference architecture for black-box testing environments. The test environment for a SAUT is built of three component types:
(i) *Test Components*: specialized components, acting as: (a) *Stimulators* - that send stimuli to a SUT, in order to trigger a test run; (b) *Interceptors* - that sit transparently between two connected participants and perform different tasks; (c) *Emulators* - that emulate the behavior of SUTs. These components allow reaping SUTs actions, addressing actions to SUTs, comparing SUTs actions with the expected ones, formulating *local verdicts* (*pass*, *inconclusive*, *fail*, *error* - see below) and transmitting them with some complementary information to the Arbiter (see below).
(ii) *Arbiter* - the Arbiter collects the Test Components *local test verdicts* and produces *final test verdicts.*
(iii) *Scheduler* - the Scheduler drives the execution of the test runs. In order to start and stop a test run, the Scheduler selects and notifies the involved Test Components.
The verdict standard values are: (i) *pass* - the test matches and there is no evidence of failure; (ii) *fail* - there is evidence of a failure; (iii) *inconclusive* - a decision cannot be reached; (iv) *error* - a run-time error has occurred in the test environment.
We include an extra component in the test environment, the *Engine*, which is notified by the Arbiter with test verdicts and manages the Scheduler by handling the test strategy on the basis of probabilistic inference on the acquired test verdicts.
An interesting and mature architecture for test execution automation, compliant with the UML Testing Profile, is TTCN-3 [10]. TTCN-3 is a Testing and Test Control Notation that is specified, together with a TTC execution environment, by the European Telecommunications Standard Institute. Several authors have already recognized the suitability of TTCN-3 for Web service testing and also introduced the idea of automatically deriving test

interfaces from WSDL description [27] [28] [31]. Some TTC environments are available and we have chosen TTWorkbench [26] as the automated test execution environment for the BN4SAT Engine.

4. MODEL-DRIVEN TEST ENGINEERING

Following the MDA approach presented in [9] [25] that is coherent with recent SOA European initiatives (i.e., STREAM [23], ISO-IEC-FDIS-42010:2011 [12]) and other international standards (see [11] for an overview), a service contract as a set of functional, interface and behavioral requirements, can be described as a layered set of models. The starting point of a *testable* services architecture is in the *Platform Independent Model* (PIM), which is composed not only of a *Schema* (a UML model) of the services (composed of a structured collection of UML stereotyped elements [19]), but also of a collection of *samples*. A sample is a modeled, schema-compliant *occurrence of services interactions*, that coordinates either the *services deliveries* (*positive sample*), or the *services refusals*, when the delivery conditions are *not* satisfied (*negative sample*). A sample is composed of: (i) the *sample interaction* - a UML *Interaction* together with its *Message Types* instances - and (ii) the *sample context* - the collection of *domain objects* (instances) representing the states of the participants' resources in which the interactions take place. A good sample design practice is *input partitioning*, which is choosing samples as representatives of *input subdomains* within which it is assumed that the service provider (consumer) behaves the same, i.e. for every point within the subdomain the same rules apply.
The samples collection is the starting point of the test cases design and implementation. The transformation of the PIM Schema in the Platform Specific Model (PSM) Schema is accompanied by the transformation of the samples collection in a *test suite* (test cases collection), which is part of the PSM. Because of the black-box/grey-box stance, only the Interoperability PSM is relevant for SOA testing. The Web services (WS*) platform is the most popular interoperability platform that supports a number of widely accepted standards such as SOAP, XSD, WSDL, UDDI, WSBPEL and WS-Security. The mapping and mechanical transformation of the PIM service schema to the corresponding WS* PSM (WSDL, XSD, UDDI) is straightforward. The services architecture deployment is described by a UDDI V3 Data Structure [18] configuration file. Note that the basic UDDI V3 representation has been extended, through the standard UDDI extension mechanism, in order to represent *use links* between participants (represented as UDDI Business Services). For each participant are listed all the participants that accept actions (message sending, rpc, rpc reply) from it and the ports where these actions are directed. Each test case of the test suite includes: (i) the *test case oracle* - a XML *infoset* that represents the interchange of SOAP messages between SUTs (resulting from the transformation of the sample UML *Interaction* [15]), together with the collection of the corresponding SOAP *envelopes* (resulting from the transformation of the *Message Type* instances exchanged in the interaction); (ii) the *test case fact base* - a relational database resulting from the object/relation transformation of the sample context. Each *testable* SUT shall be able to implement a technical service whose main operation (load(factBase)) lets the test case context be internalized from the fact base.

5. THE BN4SAT GRAPH MODEL

BNs are direct acyclic graph models that represent stochastic variables and dependencies between these variables [22]. As a *failure discovering* driver, the purpose of BN inference is to get the next test case to run in order to reveal a failure. As a *troubleshooting* driver, the purpose of BN inference is to get the next test to run in order to locate faulty participants. The purpose of our research is to integrate failure discovering and troubleshooting. The engine is supposed to manage a well-adjusted dynamic test sequence that, with a minimum number of test case runs: (i) reveals a maximum number of failures and (ii) locates a maximum number of faulty participants.
The BN4SAT graph is built by the BN4SAT loader directly from: (i) the SAUT deployment UDDI files (ii) the WSDL files for all the involved service contracts and (iii) the test suite files.

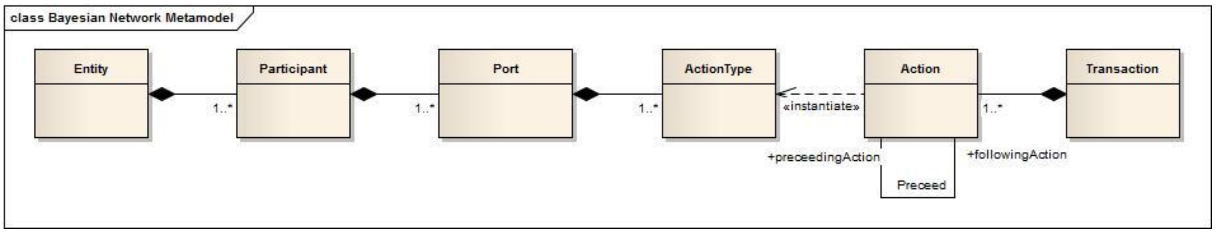

Figure 1. BN4SAT Metamodel.

The BN4SAT graph model has six Boolean stochastic variable types: *Entity*, *Participant*, *Port*, *ActionType*, *Action* and *Transaction* (Fig. 1):

(i) Entity: represents the probability distributions of the Boolean state {*notFaulty*, *faulty*} of a participating organization that is described as a Business Entity in the UDDI deployment file. The BN4SAT loader creates one entity variable for each Business Entity in the UDDI deployment files.
(ii) Participant: represents the probability distributions of the Boolean state {*notFaulty*, *faulty*} of a participant system that is described as a Business Service in the UDDI deployment files. The BN4SAT loader creates one participant variable for each Business Service and the link to the owing entity variable.
(iii) Port: represents the probability distribution of the Boolean state {*notFaulty*, *faulty*} of the *use* of a port by a participant. For each participant, the BN4SAT loader creates as many port variables as many ports it targets with its actions. The meaning of the variable value {*notFaulty*, *faulty*} is the correct/incorrect use of the port by the participant.
(iv) ActionType: represents the probability distribution of the Boolean state {*notFaulty*, *faulty*} of the action type that can be performed on the port by the participant. The BN4SAT loader creates one action type variable for each action type accepted by each port. The Boolean values represent the states in which there is no failure for the occurrences of the action type (*notFaulty*), or at least one failure (*faulty*).
(v) Action: represents the probability distribution of the Boolean state {*pass*, *fail*} of an action belonging to a test case. It represents the *evidence* that the action in the test run passes or fails. The BN4SAT loader creates one action variable for each action of each test case. Actions are arranged in a lattice representing the temporal precedence relationship between them in the test case interaction. An action variable has a link to its action type variable and a link to the transaction variable (see below) it belongs to.
(vi) Transaction: represents the probability distribution of the Boolean state {*pass*, *fail*} of the test case interaction (that organizes a number of actions). The BN4SAT loader creates one transaction variable for each test case in the test suite. The Boolean values represent the states in which all the actions of the transaction pass or at least one fails.
The Action variables are the *observed variables* of the BN4SAT graph architecture. Consequently, they are the BN nodes in which the *evidence* communicated by the Arbiter, which is a probability distribution {(*pass*,P),(*fail*,1-P)}, is inserted after the test case run. All the other BN4SAT variables are not observed and their probability distribution is computed by inference.

6. DISCUSSION, FUTURE WORK AND CONCLUSION

For large services architectures, the computational complexity of the BN4SAT graph quickly grows, as a function of the number of nodes and arcs and of the size of the Conditional Probability Tables (CPT). In the progress of our research, we have implemented and assessed different classical inference methods such as *value elimination*, *lazy propagation*, *Shafer-Shenoy* [14] [24] and proved that they are not appropriate to cope in a sustainable way with BN4SAT graphs.
Afterwards, we have tried out other methods proposed by Chavira and Darwiche [6] [8], based on the use of *arithmetic circuits* (AC) as BN compiled representations. The multi-linear function characterizing the BN is factorized in the AC and used for faster inference. The AC can return a set of probabilities by a simple "bottom to top" and "top to bottom" run. The transformation of the BN into the AC is performed in the following steps: (i) the BN is encoded in a conjunctive normal form (CNF); (ii) the CNF is then used to create the deterministic decomposable negation normal form (d-DNNF); (iii) the normal form allows the AC generation. Darwiche [8] proposes a general method for CNF encoding, but, since the BN4SAT graph is composed of many logic (AND/OR) nodes, we have experimented with another method that produces a shorter equivalent CNF. Reference [7] documents the use of a hyper-graph partitioning technique to create a d-tree [13] - the d-DNNF is generated from the d-tree and then transformed into the AC. We have tried to appropriate this transformation to our BN4SAT graph model. Indeed, because of the specific characteristics of the BN4SAT graph architecture, we found a way to perform a very fast generation of reduced ACs whose structure increases the inference process speed, avoiding the difficulties of the classical methods. The technique has proved its efficiency, so the AC is now directly generated by mapping and model transformation from the Test PSM (UDDI, WSDL, Test Suite files).
The fault trees approach, mentioned in Section 2, could be applied when the dependencies between participants of a services architecture are known (we know that a system, in order to deliver a service, uses services delivered by other systems - the so-called *service composition*), but either the detail of the service contract are unknown or the channels between participants are not observable. We have started to investigate this issue and later in the project we will develop the experimentation on this point.
Our research highlights that model mapping and transformation can be applied not only to the generation of test cases but also to the automatic generation of BNs for probabilistic inference on test strategy. Moreover, model-driven transformation can support techniques of *inference by compilation*, allowing the use of the BN inference as a sustainable testing strategy driver of large scale services architectures. We intend to evaluate our results with realistic services architecture examples.

REFERENCES

[1] Bobbio, A., Portinale, L., Minichino, M., Ciancamerla, E.: Improving the analysis of dependable systems by mapping fault trees into bayesian networks. Reliability Engineering and System Safety, 71(3):249–260, 2001.

[2] Bruning, S., Weissleder, S., Malek, M.: A fault taxonomy for service-oriented architecture. In Proceedings of High-Assurance Systems Engineering, IEEE International Symposium, 0:367–368 (2007).

[3] Bruno, M., Canfora, G., Di Penta, M., Esposito, G., Mazza, V.: Using test cases as contract to ensure service compliance across releases. In Proceedings of Service-Oriented Computing - ICSOC 2005, Third International Conference, pages 87–100 (2005).

[4] Canfora, G., Di Penta, M.: Soa: Testing and self-checking. In Proceedings of International Workshop on Web Services- Modeling and Testing-WS-MaTE, pp. 3–12, (2006).

[5] Canfora, G., Di Penta, M.: Service Oriented Architectures Testing: A Survey. Software Engineering, Springer-Verlag, Berlin, Heidelberg (2009).

[6] Chavira, M., Darwiche, A., Jaeger, M.: Compiling relational Bayesian networks for exact inference. In International Journal of Approximate Reasoning, Volume 4, Issue 1-2, May, 2006.

[7] Darwiche A.: A compiler for deterministic, decomposable negation normal form. In Proc. Eighteenth National Conference on Artificial intelligence, Dechter, R., Kearns, M., Sutton, R. (Eds.). American Association for Artificial Intelligence, Menlo Park, CA, USA, 627-634.

[8] Darwiche D.: A differential approach to inference in Bayesian networks. In Journal of ACM, 50:280–305, May 2003.

[9] De Rosa, F., Maesano, A., Maesano, L.: Service Orientation, Requirement Definition and Requirement-based Testing: a Contract-based, Model-driven Approach. Simple Engineering, Internal Report R2011062502 (2011).

[10] European Telecommunications Standards Institute (ETSI): Testing & Test Control Notation (TTCN 3), http://www.ttcn-3.org/

[11] IFEAD / TOGAF Open Standards, Enterprise Architecture Standards Overview, http://www.enterprise-architecture.info/EA_Standards.htm (November 2010).

[12] ISO/IEC 42010:2007 – originally IEEE Std 1471:2000 – Recommended Practice for Architectural Description of Software-intensive Systems, Final Draft, http://www.iso-architecture.org/ieee-1471/ (July 2011).

[13] Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: application in VLSI domain, Proceedings of the 34th annual conference on Design automation, p.526-529, June 09-13, 1997, Anaheim, California, United States.

[14] Madsen, A.L., Jensen F.V.: Lazy propagation in junction trees. In Proc. 14th Conf. on Uncertainty in Artificial Intelligence, 1998.

[15] Maesano, A.P.: Stochastic Decision Aid for SOA Testing - UPMC/CNRS - LIP6 - Interim Report - Mars 2011, Paris (2011).

[16] Maesano L., De Rosa, F.: simpleSOAD® 2.0 - Architecture & Governance. In Proceedings of the 22nd International Conference on Software & Systems Engineering and their Applications (ICSSEA 2010) - Paris Dec 7-9, 2010.

[17] Nielsen, Th., Wuillemin, P.H., Jensen, F., Kjaerulf, U.: Using ROBDDs for Inference in Bayesian Networks with Troubleshooting as an Example. In Boutilier, C., Goldszmidt, M. (Eds.), Proceedings of the 16th conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 426–435 (2000).

[18] OASIS, Universal Description Discovery & Integration, V3.0.2 (UDDI) , http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[19] Object Management Group: Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS) - OMG Adopted Specification - FTF Beta 2, http://www.omg.org/spec/SoaML/20091101

[20] Object Management Group: UML Testing Profile, Version 1.0 (2005).

[21] Object Management Group: Model Driven Architecture, http://www.omg.org/mda/

[22] Pearl. J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman (1988).

[23] Schekkerman, J.: A Successful and Pragmatic “Managed Diversity” Enterprise Architecture Approach, Institute for Enterprise Architecture Developments and Logica Business Consulting, http://www.enterprise-architecture.info/Images/STREAM/White%20Paper%20STREAM%20%202010%20v1.2.pdf, (2010).

[24] Shafer, G.: Probabilistic Expert Systems. Society for Industrial and Applied Mathematics, Philadelphia, (1996).

[25] Simple Engineering: simpleSOAD® 2.0 Reference Manual & Pattern Library- R2011043001, Rome (2011).

[26] Testing Technologies, TTworkbench, http://www.testingtech.com/
[27] Vassiliou-Gioles, T.: Testing Web Services with TTCN-3, in Testing Experience, June 2008.
[28] Werner, W., Grabowski, J., Troschütz, S., Zeiss, B.: A TTCN-3-based Web Service Test Framework. Proceedings of Software Engineering (Workshops) 2008. pp.375~382 (2008).
[29] Wieczorek, S., Stefanescu, A., Grossman, J.: Enabling Model-Based testing for SOA Integration. In Proc. of 1st Workshop on Model-based Testing in Practice (Bauer, Th., Eichlert, H., Rennoch, A., eds.) MoTiP'08 (2008).
[30] Wooff, D.A., Goldstein, M., Coolen, F.P.A.: Bayesian graphical models for software testing. IEEE Transactions on Software Engineering, 28:510–525 (2002).
[31] Xiong, P., Probert, R.L., Stepien, B.: An Efficient Formal Testing Approach for Web Service with TTCN-3. Proceedings of the 13th International Conference on Software, Telecommunications and Computer Networks (SoftCOM) (2005).