The Dark Side of SOA Testing: Towards Testing Contemporary SOAs Based on Criticality Metrics

Philipp Leitner, Stefan Schulte, Schahram Dustdar
Distributed Systems Group,
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at

Ingo Pill, Marco Schulz, Franz Wotawa
Institute for Software Technology,
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria
{lastname}@ist.tugraz.at

***Abstract*—Service-Oriented Architectures (SOAs) have widely been accepted as the standard way of building large-scale, heterogeneous enterprise IT systems. In this paper, we explore the current limitations of testing contemporary SOAs, which are typically assemblies of various components, including services, message buses, business processes, and support components. We argue that, currently, SOA testing is too much concerned with testing single services or business processes, while there is little scientific literature on holistic testing of contemporary SOAs that includes all critical components and their mutual dependencies and interactions. In this paper, we detail the architecture of contemporary SOA, thoroughly assess the current state of research in respect of their testing, and introduce the notion of SOA criticality metrics as indicators for an individual component's criticality for the SOA as a whole. We enumerate an initial metric set for various component types and interactions, as well as discuss how these metrics can be used for testing contemporary SOA.**

***Index Terms*—service-based computing, contemporary SOA, SOA testing, metrics**

I. INTRODUCTION

In recent years, the principles of Service-Oriented Architectures (SOAs) have been receiving high attention, and are nowadays widely accepted in the software industry [1], [2]. The reasons for this trend origin in the advantages that SOAs offer with respect to communication interoperability, reusability and compatibility of services, as well as loose coupling between clients and servers. The evolution of SOAs entailed a variety of novel SOA-based technologies, including self-healing SOA systems [3], service runtime environments [4], enterprise service buses (ESBs) [5] and cloud computing [6].

However, one consequence of the fast evolution of SOAs is that the complexity of corresponding systems has been increasing steadily. Nowadays, SOA-based systems consist of more than just services and consumers. Instead, they comprise a multitude of Web services, registries, aggregators, mediators, message buses, monitoring and governance systems, and other components. Such SOAs are often referred to as contemporary SOAs [1]. The inherent complexity of contemporary SOAs result in a rather high error-proneness of corresponding systems.

While there is a large body of work assisting engineers with testing single services (e.g., [7], [8], [9]) or single business processes (e.g., [10], [11], [12]), we argue that, so far, the scientific community has failed to deliver adequate models to test contemporary SOAs in its entirety, both with regard to functional aspects, as well as with regard to non-functional system properties.

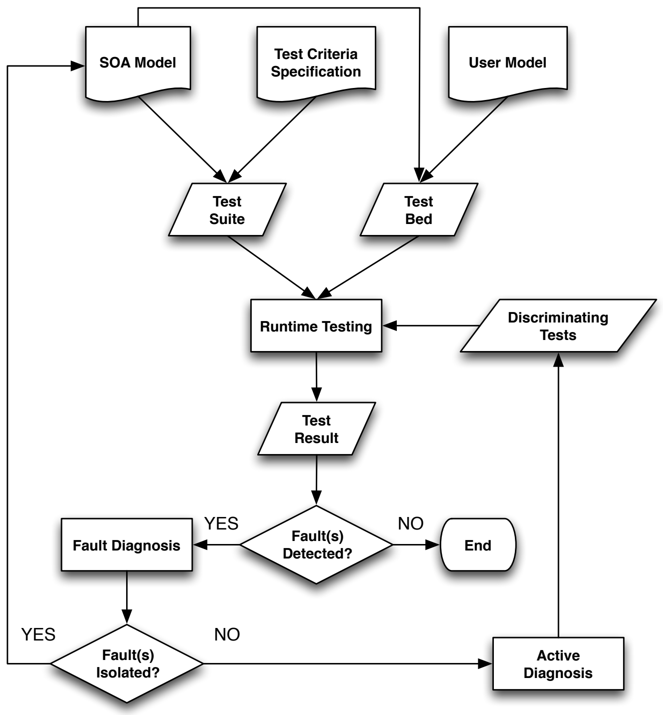

Fig. 1. High-Level Overview of SOA Testing

In Figure I, we give a high-level conceptual overview over the general field of SOA testing. This testing process underpins our work in the *Audit 4 SOAs* project[1], and also forms the basis of this paper. Essentially, SOA testing comprises design-time and runtime aspects. At design-time, models of the entire SOA (including services, compositions, message buses, mediators, etc.) need to be developed and validated. This requires a deep understanding of the SOA under test regarding which components are more error-prone, complex and critical than others, as well as with regard to how these components

[1]http://www.infosys.tuwien.ac.at/linksites/audit4soas/

interact and depend on each other. For instance, consider a business process that uses a given service. Obviously, the correct functioning of the business process depends upon the functioning of this service. Furthermore, both process and service depend on the availability and correct operation of the message bus. Such dependencies need to be made explicit and have to be taken into account when designing test cases. In general, the principle of the weakest element in the chain applies, i.e., overtesting, for instance, the business processes does not improve the stability of the system in total, if the underlying components are not sufficiently covered. Similarly, overtesting a service that is hardly used in the core processes of the SOA instead of a more critical one is also hardly optimal.

At runtime, previously defined test suites are executed using a test bed, which is an abstraction and staging environment for the real SOA. The results of these test runs are analyzed, and, if necessary, more discriminating test cases are designed. Oftentimes, this will also mean extending the test bed, i.e., taking more components into account. For implementing test beds, tools such as Genesis2 [13] can be used.

In this paper, we focus on the aspect of how to identify the most critical components in a contemporary SOA, and how to make explicit the dependencies between those components. We argue that this is required to steer the testing of contemporary SOAs, as well as to support the definition of useful testbeds (i.e., to build a useful staging environment, one first needs to identify the most critical components that need to be modelled). To this end, we propose to adopt complexity metrics [14] from the software engineering domain, in order to capture component dependencies and identify critical components that have to be tested with particular care. We refer to these metrics as criticality metrics. In this paper, we give an initial overview over some important criticality metrics (some of which are adapted from the current state of research), and showcase their relevance for, as well as integration with, SOA testing.

The remainder of this paper is structured as follows. In Section II, we explain the concept of contemporary SOAs in detail, and thoroughly discuss the current state of research with regard to testing contemporary SOAs. Arguing about their relevance and impact, we then introduce in Section III criticality metrics for SOA and give an initial set of SOA metrics useful for building suitable SOA testing strategies. Afterwards, we sketch how criticality metrics can be used to steer the concrete SOA testing in Section IV. Finally, Section V ends the paper with concluding remarks, and a brief outlook on open issues and future work.

II. State of Research in Contemporary SOA Testing

Early publications on service-oriented computing assumed that, in principle, SOAs orchestrate three role models: services, consumers, and one logical service registry that decouples consumers and services [15] (see Figure 2).

However, as indicated in Section I, as well as, for instance, in [2], today's service-based systems have come a long way from the simple publish-find-bind triangle assumed in the early days of SOA [15]. Nowadays, contemporary SOAs are an amalgam of *services*, *business processes*, *message buses*, *service registries*, *mediators*, *aggregators*, and *service monitors*. This more complex view on SOA is sketched in Figure 3, and is often referred to as the contemporary SOA model.

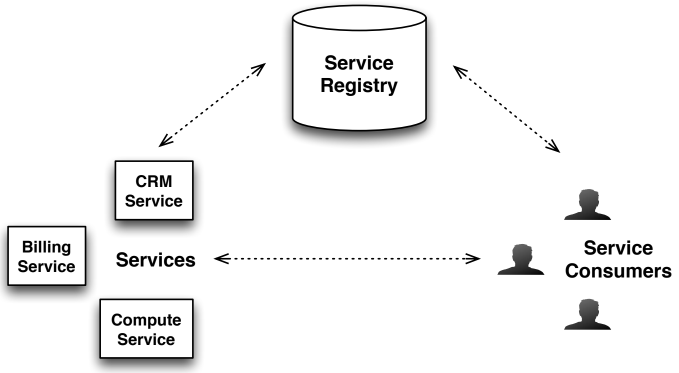

Fig. 2. Original SOA Model

In current literature, software testing is already a relatively well-covered research field [16], [17]. However, most research papers still mostly consider a world resembling Figure 2, where services are operating in isolation, and interactions between clients and services are point-to-point.

A. Testing Contemporary SOA Components

In the following, we iterate over the components of contemporary SOAs, as sketched in Figure 3, and discuss the current state of research regarding their testing.

Services: are the fundamental building blocks of SOAs. They are self-contained, reusable components that provide some domain functionality over a standardized interface [1]. Existing research deals with testing atomic services for robustness [8], conformance to a model [18], reliability [19], or generating test data used in service unit tests [20]. Among all the SOA components, *services* are certainly the ones whose testing is understood best today.

Business Processes: provide higher-level functionality as compositions of services [21]. Large enterprises typically make use of dozens, or even hundreds, of separate business processes. However, not all processes are equally important to the functioning of the enterprise. While *operational processes* are often mission-critical, some *support* and *management processes* are of more optional nature (and, hence, temporary failures in those processes are often preferred to failures in *operational processes*). Further, as show in Figure 3, business processes themselves often provide a service in the SOA, leading to complex, multi-level composition scenarios. The sum of all services and processes is often referred to as the *process landscape* of an enterprise. Testing of single business processes in isolation is well covered in today's scientific literature. This can take the form of generating test data for processes [22], BPEL conformance testing [11], composition

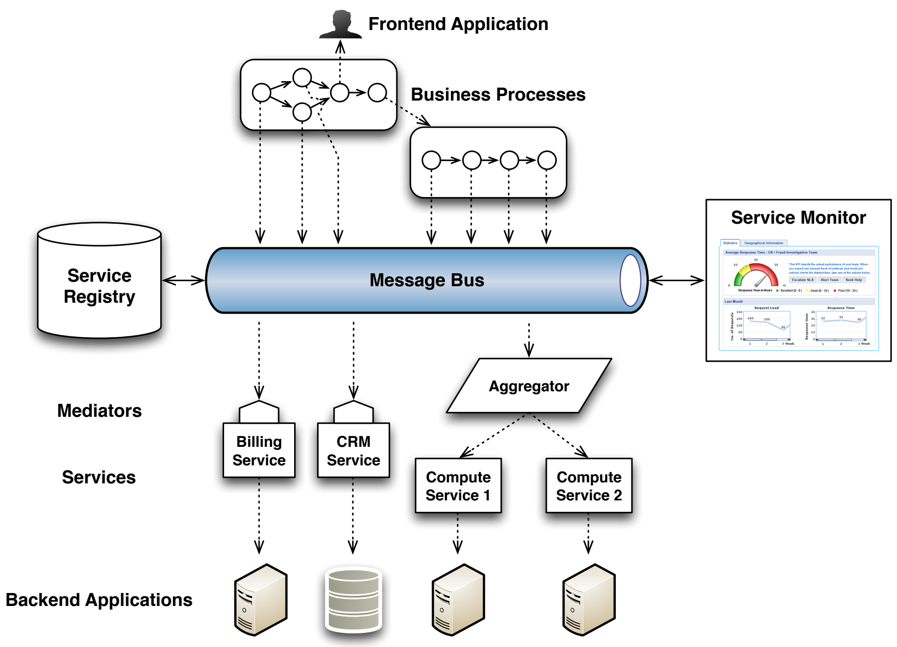

Fig. 3. Contemporary SOA Model

reliability testing [10], unit testing for BPEL [23], or data and flow coverage testing [12]. However, currently, there is little discussion regarding testing entire *process landscapes* as a whole.

Message Buses: have proven to be the invaluable glue that brings independent and heterogeneous services and processes together [5]. Buses provide message routing between services (decoupling consumers and services referentially as well as temporally), and, to some degree, message transformation that resolves interface and data differences between services (even though this task is often delegated to dedicated *mediators*, as discussed below). However, message buses are typically not, or not entirely, off-the-shelf components. For most contemporary SOAs, customizations and adaptations of off-the-shelf ESB products (such as, for instance, the open source product Apache ServiceMix[2]) are necessary. Hence, taking the bus into account during testing seems essential, as those custom deployments cannot be assumed to be error-free, especially in conjunction with the specific services, mediators and processes used. However, again, while there is some work on testing message buses in isolation (e.g., [24]), little research exists on testing the interplay between message buses and other components of contemporary SOAs.

Service Registries: are the only components in the "original" SOA model that have, in fact, failed to take off to the extent originally envisioned. Even today, services are rarely discovered dynamically, and connections between consumers and services are typically static. Still, service registries are oftentimes part of enterprise SOAs, either as part of the message bus [4] or as stand-alone applications. Interestingly enough, even though plenty of research exists on service registries and discovery (e.g., [25], [4], [26], [27]), very little work focuses on their testing, neither in isolation nor in conjunction with other SOA components.

Mediators: are the result of the observation, that, in spite of standardized interfaces and semantic Web services technology [28], communication of services built by different departments or vendors is still difficult. Hence, *mediators* are often used to translate between different data formats and protocols [29], [30]. Evidently, this transformation is often technically complex and brittle. Still, very few testing approaches for mediators are currently available in the literature.

Aggregators: are used to combine different services outside of the message bus. *Aggregators* typically appear as a single service to the bus, as well as to users of the service, and are sometimes used in load balancing or integration scenarios. Note that *aggregators* are not the same as service compositions. For testing purposes, some of the same methods used to test atomic services can be re-used. However, there is little work on how to test *aggregators* in conjunction with *message buses* and *business processes*.

Service Monitors: (as well as other components related to service governance) are the last puzzle pieces we need to take into account when testing contemporary SOAs. Unlike all other components discussed so far, *service monitors* are typically not critical to the correct basic functioning of the SOA (i.e., an outage of a *service monitor* does not necessarily mean the unavailability of any core business processes). However, today, business activity monitoring [31] has become such an essential tool for process owners and business analysts that an extended unavailability of *service monitors* is typically equally unacceptable as the unavailability

[2]http://servicemix.apache.org/

of core processes. Hence, *service monitors* also need to be part of a contemporary SOA testing strategy. Indeed, one large research thread considered by the community covers the testing of services with respect to their non-functional properties (typically referred to as Quality-of-Service, QoS). QoS testing comprises either passive monitoring of service performance [32], or active testing (for instance, using test invocations [33]). One paradigm that has lately been gaining traction for monitoring non-functional properties is the concept of event-based monitoring [34]. Event-based monitoring is a special case of passive testing, where data is extracted from low-level information by means of complex event processing techniques [35]. In earlier research, we used event-based monitoring in the context of business processes [36] and for applications deployed to the cloud [37], as well as for quality prediction in service compositions [38].

B. The Dark Side of SOA Testing

Generalizing, the inherent assumptions of existing SOA testing research are still rather limiting. Typically, the system under test is one single service or business process, along with the services used by this process. However, business processes and service-based applications in a contemporary SOA do not operate in a vacuum, and thus should not be tested in such artificial and barren environments. We refer to this as the dark (as in, the currently unexplored) side of SOA testing.

One approach to deal with the dark side of SOA testing is to take more and more components into account when designing tests. Ultimately, this leads to complex test environments (testbeds), where not only services and interactions are modelled, but entire environments and the various cross-references. Research regarding tools suitable for this task is currently at square one, with few notable exceptions (e.g., Genesis2 [13] or PUPPET [39]). However, using these tools to build actually useful testbeds that cover the most important properties of the SOA under test, is not trivial. In this paper, we introduce criticality metrics as heuristics to reflect the criticality of various components of the entire SOA, as well as capture interdependencies between components in a contemporary SOA. These simple metrics give us estimations for both, how error-prone and how important any given component is for the functioning of the system. We identify different criticality metrics for different types of components. Please note that the metrics we present here are not exhaustive. Thus, they are a mere starting point for future research.

III. Criticality Metrics for SOA Testing

The metrics we consider in this paper were inspired by metrics used in the software engineering domain to measure the complexity of computer programs [40]. The cyclomatic complexity, for instance, is defined via the amount of choices and cycles inside a program, where these choices and cycles correspond to conditionals and loops at programming language level. Accordingly, programs with high condition and loop counts suffer from a rather high cyclomatic complexity. The original claim in [40] was then that such program modules with a high cyclomatic complexity are error-prone, and thus should either be refactored or thoroughly tested.

Existing research [14] has already proposed a number of analogous complexity metrics for SOAs, including metrics such as the number of services, the number of versions per service, and the number of human tasks. However, these metrics do not aim at SOA testing. Much more, they are used to calculate aggregated complexity indices for the SOA in its entirety. Hence, in the following, we propose a set of metrics geared towards testing. We refer to those metrics as criticality metrics. It should be noted that, evidently, criticality metrics can only serve as heuristics aiming to objectively measure a component's actual criticality. For instance, it is indeed possible that metrics suggest a high criticality (and thus a higher risk level demanding for thorough testing) for a given business process, whereas, in practice, this process is so well-understood by domain experts that the actually associated risk level is much lower. Therefore, the intention of the metric model concept we propose is to serve as initial, sound, and objective fundament for reasoning about testing-critical components.

A. Introduction to Criticality Metrics

In general, each component in a contemporary SOA should be assigned one or more criticality metric value(s). Evidently, different metrics are used to capture the criticality of business processes, services, or service registries. Furthermore, the range of legal values is different for different metrics.

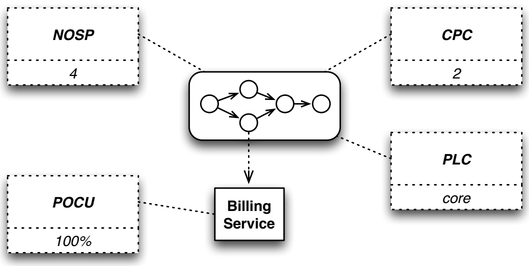

Fig. 4. Example Criticality Metrics on Different Levels

Some example is given in Figure 4 depicting a rather simple business process. The process is assigned values for three metrics, i.e., the number of services used in the process (`NOSP`), the cyclomatic process complexity (`CPC`), and the process landscape categorization (`PLC`). Furthermore, the process is using a service (the billing service), which is evaluated with another metric (`POCU`, the percentage of core process usages). `PLC` is an ordinal metric, with possible values `core` and `support`.

B. Initial Multi-Level SOA Criticality Metric Set

In the following, we propose an initial set of multi-level SOA criticality metrics. We focus on metrics for measuring services and business processes, mostly because these components are currently best understood by the research community. Hence, they are very useful for illustrating our general idea.

We summarize our initial metric set in Table I. More detailed discussions for each metric can be found in the following.

1) Service-Level Criticality Metrics: On service level, various metrics can be used to estimate how critical a specific service is to the SOA in general. That is, as a first step, a tester needs to have a look at the service in isolation, in order to estimate the complexity that a specific service induces on its own. Two simple ordinal metrics capturing this aspect are (1) the service statefulness (`SST`), i.e., whether the service's operations can be used in any order or whether a specific protocol is assumed, and (2) the whether the service requires a mediator for connecting to the message bus (`IMR`). Typically, stateful services are vastly more error-prone than stateless ones (see also [14]). Similarly, mediation is known to be complex and error-prone, hence, services that require mediation need to be tested with more care. Another relevant aspect considers the number of different service versions available (`NOSV`). This metric is motivated by the observation that real-life SOAs sometimes require multiple versions be kept online in parallel [41], which is formally known to affect reusability and testability.

Given that the primary goal of SOA services is to be used in business processes, it is not sufficient to just consider the complexity or criticality of services in isolation. In addition, we also need to take into account how often a service is used in processes, and how critical those processes are (i.e., services that are often used in core processes require more thorough testing). In our initial metric set, we capture this aspect with three related metrics. Firstly, the number of process usages (`NOPU`) defined as the number of business processes that refer to a given service as WS-BPEL partner link. Secondly, and based on `NOPU`, we define `POCU` as the percentage of times a service is used in core processes. `POCU` is defined in Equation 1. In the equation, `NOCU` is the number of times a service is used in a core business process in the SOA's process landscape. Note that `POCU` is related to the "coupling between services" metric discussed in [42]. Here, we have decided to not include this specific metric, as we assume that services in a contemporary SOA are not directly invoking each other.

$$\texttt{POCU} = 100\,\frac{\texttt{NOCU}}{\texttt{NOPU}} \tag{1}$$

2) Process-Level Criticality Metrics: In order to capture the complexity and criticality of business processes, the most basic metric is to simply count the number of service invocations in the process (e.g., count the number of "Invoke" activities in a WS-BPEL process). We refer to this metric as `NOAP`, the number of service invocation activities in a process. `NOAP` is closely related to two further metrics, (1) the number of distinct services that a process interacts with (`NOSP`), and (2) the number of sub-processes that are invoked (`NOPP`). In WS-BPEL, `NOAP` reflects the number of partner links that are used by the process. In contrast, `NOPP` cannot be calculated directly from the process definition, as sub-processes are not identifiable from the partner links in standard WS-BPEL. However, in our experience, invocations of sub-processes are particularly important to testing, as they establish "part-of" relationships between processes in the process landscape. Hence, for processes with $\texttt{NOPP} > 0$, it becomes crucial to take also special care of the sub-processes during testing. Some sophisticated contemporary SOAs make use of the principle of dynamic binding [43], i.e., they use a registry to find the most suitable concrete implementation of an abstract functionality at runtime, and invoke it dynamically using tools such as Daios [44]. In such SOAs, another important metric is `NODAP`, the number of dynamically bound service invocation activities in a process. Dynamic binding is inherently more complex and error-prone than static binding, and needs to be tested with care. One particularly tricky aspect of testing processes with $\texttt{NODAP} \geq 2$ is that, in order to achieve full test coverage, all possible combinations of concrete services need to be tested together. However, for non-trivial `NODAP` values, this leads to combinatorial explosion. Hence, in earlier research [12], we proposed heuristics on how to limit the problem space and still produce valid test results.

Moving on to more structure-related metrics, we adapt the original cyclomatic complexity metric as discussed before, in order to arrive at `CPC`, the cyclomatic process complexity of a WS-BPEL process defined in Equation 2. In this equation, e corresponds to the process' activity count (including all WS-BPEL activities, not only service invocations as in `NOAP`), n is the sum of all graphs in the process, and p is the count of the single control flow graphs.

$$\texttt{CPC} = e - n + 2p \tag{2}$$

An important aspect of a business process' complexity is not captured by `CPC`; the complexity of its message exchange pattern (MEP). Essentially, a MEP defines the message protocol required to correctly interact with the process. Some example MEPs for real-life processes are given in Figure 5. The simplest MEP is `in-only`. Such processes are started via an input message from the message bus, and do not write any answer back to the bus. Similarly well-understood are `in-out` MEPs, which are again started via an input message from the bus, and which write their result back to the bus using a single message. However, in addition to these simple MEPs, many processes use more complicated protocols, mixing various input and output messages, e.g., `in-multipleOut`. It goes without saying that such processes are inherently more complex, and, hence, require more testing.

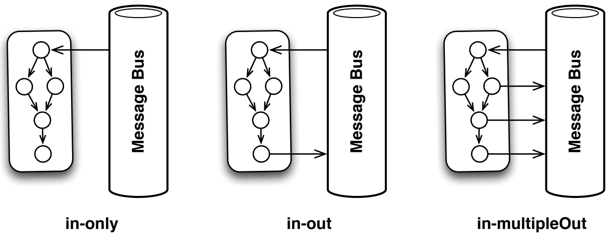

Fig. 5. Example MEPs of Business Processes

TABLE I
INITIAL CRITICALITY METRICS OVERVIEW

Component	Short Name	Full Name	Range
Service-Level Metrics			
	SSF	Service Statefulness	$SSF \in \{stateful, stateless\}$
	IMR	Interface Mediation Required	$IMR \in \{yes, no\}$
	NOSV	Number of Service Versions	$NOSV \in \mathbb{N}_1$
	NOPU	Number of Process Usages	$NOSV \in \mathbb{N}_0$
	POCU	Percentage of Core Process Usages	$POCU \in [0; 100]$
Process-Level Metrics			
	NOAP	Number of Service Invocation Activities in Process	$NOAP \in \mathbb{N}_0$
	NOSP	Number of Services in Process	$NOSP \in \mathbb{N}_0$
	NOPP	Number of Sub-Processes in Process	$NOPP \in \mathbb{N}_0$
	NODAP	Number of Dynamically Bound Service Invocation Activities in Process	$NODAP \in \mathbb{N}_0$
	CPC	Cyclomatic Process Complexity	$CPC \in \mathbb{N}_1$
	CMEP	Complexity of Message Exchange Pattern	$CMEP \in \mathbb{N}_1$
	PLC	Process Landscape Categorization	$PLC \in \{core, support\}$
Other Metrics			
	NRU	Number of Registry Users	$NRU \in \mathbb{N}_0$
	DIR	Dynamic Invocation Ratio	$DIR \in [0; 100]$

To the best of our knowledge, the current state of research has not yet identified a suitable metric for measuring the complexity of process MEPs. Hence, as a starting point, we propose the metric CMEP defined in Equation 3. Here, $|m_i|$ is the number of input messages that the process receives, and $|m_o|$ is the number of output messages it produces. In practice, CMEP can easily be calculated for WS-BPEL processes by counting the number of "Receive" and "Reply" activities in the process definition file. One thing to keep in mind with CMEP is that there is usually a significant increase of complexity between $\texttt{CMEP} = 2$ and $\texttt{CMEP} = 3$. The reason for this is that for $\texttt{CMEP} \geq 3$, issues of asynchrony and WS-BPEL message correlation can manifest themselves, especially if $|m_i| \geq 1$ (that is, if there is at least one non-initial "Receive" activity in the WS-BPEL definition).

$$\texttt{CMEP} = |m_i| + |m_o| \tag{3}$$

Finally, there is one additional metric that has a tremendous impact on the criticality of business process, the process landscape categorization (PLC). This metric measures whether the process is a core process in the SOA's process landscape, or whether it's a support or management process. This metric is ordinal and can have one of two values (core or support). We do not distinguish between support and management processes here. Typically, core processes require significantly more thorough testing, as their outage is less acceptable than the temporary unavailability of support processes.

3) Other Metrics: As discussed in detail in Section II, contemporary SOAs do not only consist of services and processes. Hence, we also introduce some metrics to measure the complexity and importance of service registries. Typically, registries in a contemporary SOA serve two purposes. Firstly, they are often used as repository of enterprise services, which is used by engineers at design-time to find existing services. In this case, the registry is not a core component of the running SOA. That is, short-time outages of a registry can typically be tolerated, as they have no negative bearing on the running processes. However, long-time outages are typically unacceptable, as they would hamper the ongoing development and improvement of processes. A most simple metric to measure the criticality of a registry in this respect, is the number of registry users NRU, that is, the number of developers or teams that require the registry for their work. Note that not every developer that has ever accessed the registry is counted towards NRU. Instead, we only count users that use the registry with a certain regularity. Typically, this metric can be extracted rather easily from corresponding access logs. $\texttt{NRU} = 0$ means that the registry is not critical for anybody, hence, testing of this component can be executed on a best-effort basis.

$$\texttt{DIR} = 100 \, \frac{\sum_{p \in P} \texttt{NODAP}_p}{\sum_{p \in P} \texttt{NOAP}_p} \tag{4}$$

However, if at least some services are invoked via dynamic binding, the involved service registry automatically becomes more critical. Naively, we can state that testing the service registry becomes the more important, the higher the percentage of dynamic bindings among all service invocations. We refer to this criticality metric as DIR, the dynamic invocation ratio, as defined in Equation 4. Here, P is the set of all processes in a SOA, and $\texttt{NODAP}_p$ and $\texttt{NOAP}_p$ are the corresponding NODAP and NOAP metric values for a given process p.

IV. DESIGNING SOA TEST SCENARIOS BASED ON CRITICALITY METRICS

Our discussion of criticality metrics for SOAs in the last section immediately raises the question of how to actually exploit them in testing. In particular, someone might be interested in using them for designing test suites that are well adapted to a given SOA-based system. In the course of answering this question, we first elaborate on the exploitation of metrics for testing in general. That is, within the testing community the use of coverage metrics and mutation scores for an evaluation but also a construction of effective test suites is well documented. In the case of SOAs [45], for example, used path coverage for the purpose of test-case generation. Hence, a straight forward use of metrics would be to adapt the test-case generation methods and algorithms for SOA testing.

In contrast to coverage metrics or similar measures used for assessing the quality of a test suite, our criticality metrics introduced for SOAs do not provide any direct means for generating test cases. This stems from the fact that our criticality metrics aim at determining risk levels for a SOA component, i.e., the corresponding effects of a faulty component for the SOA as a whole. Hence, the objective of criticality metrics is to estimate risks in the architecture of a certain SOA-based system. In order to exploit these data for generating test cases, we have to combine criticality metrics with measures for test-suite quality. A promising way to achieve this is to introduce a function that provides certain test-suite quality requirements for given criticality metrics values.

For example, it is reasonable to assume that a faulty core service poses a severe threat to a SOA, resulting in a higher risk level for this service. In case of service S having a `PLC` value *core*, we should spend more efforts on its testing. This can be ensured by stating that a test suite for S has to satisfy a 100 % MC/DC coverage, or alternatively that the mutation score for S should be larger than 80 %. Beware, that we assume these figures vary. Most likely, in practice a combination of several coverage criteria is desirable. Moreover, alternative measures like using the category-partitioning method might be applied in case of the requirement for white-box testing of services. Hence, a generally applicable method that is independent of the underlying SOA seems unlikely. As a consequence, we suggest to continuously improve the function (in its simplest form we assume a rule set from now on). That is, we start with a basic underlying rule set for mapping criticality metric data to test-suite quality criteria as used for test-case generation. Every time faults are detected (test cases fail), the rule set is evaluated and updated in order to integrate newly derived knowledge.

Concerning the desired learning effect, this continuous improvement process is likely to be individual to a certain SOA or even the company developing SOAs. Whether such a rule set can be generalized, or adapted to be used in different domains is an open research question. Moreover, within this process it is also important to impose additional constraints in respect of ressource limits (including testing time, budget etc.). If corresponding constraints are not defined, obviously the best and most likely most expensive testing method should be applied, i.e., testing all possible inputs, which is not feasible in practice.

In this paper we discuss SOA criticality metrics merely in a conceptual sense, and as suggested earlier, current research aims at extending our initial metric set. In the course of this work, an evaluation of the metrics themselves and corresponding amalgams will provide us with the necessary details to suggest actual strategies and implement a corresponding automatic or semi-automatic tool set for SOA testing. Unsaid so far, distinctive metrics might also suggest in which scope individual components or component subsets will have to be considered. That is, not every aspect will have to be considered in the general scope of system-wide run-time tests, so that the metrics will help also in deriving testing strategies that optimize testing efforts and thus allow to utilize defined budgets in an efficient manner.

V. Conclusions

In this paper, our goal was to motivate the necessity of an integrative, end-to-end testing of contemporary SOAs. We have shown that current research on SOA testing is too narrowly focussed on testing SOA components in isolation. In this work, we introduce SOA criticality metrics as a tool to steer the targeted and efficient testing of an SOA in its entirety, including services, business processes, mediators, aggregators and service registries. In order to provide a foundation for our discussion, we introduced an initial set of criticality metrics suitable for SOA, some of which are based on the results of related research. Furthermore, we illustrated how one can use this initial metric set to steer the testing of contemporary SOAs.

A. Future Research

As this paper only represents our first steps, there are of course a multitude of directions that require further research. Firstly, we need to extend and rework our initial criticality metric set. So far, we have been focusing mostly on metrics for services and business processes. In the future, we will extend this set with additional metrics for message buses, service registries, aggregators and service monitors. Additionally, the metrics we identified so far are mostly based on the study of related research and our own experience in building and testing SOAs. Hence, we will evaluate this metric set in the scope of real-life SOA, in order to assess whether these metrics actually provide the desired focus. Furthermore, while we gave an informal introduction in Section IV, we will elaborate on formalizing the process of testing SOAs by exploiting criticality metrics. Finally, we will continue our work on a semi-automated tool set that will simplify the calculation of diverse criticality metrics by parsing various artifacts, including WS-BPEL process definitions, WSDL contracts, message bus event logs and registry access logs.

Acknowledgements

We would like to thank Lukasz Juszczyk for his work on the initial ideas underlying this paper. The research leading to these results has received funding from the Austrian Science Fund (FWF) under project references P23313-N23 (Audit4SOAs) and P22959-N23 (MoDiaForTed), as well as the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 257483 (Indenica).

References

[1] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, November 2007.

[3] R. B. Halima, K. Drira, and M. Jmaiel, "A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services," in *Proceedings of the 2008 IEEE International Conference on Web Services (ICWS'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 104–111. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2008.113

[4] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo," *IEEE Transactions on Services Computing*, vol. 3, pp. 193–205, July 2010.

[5] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The Enterprise Service Bus: Making Service-Oriented Architecture Real," *IBM Systems Journal*, vol. 44, no. 4, 2005.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1721672

[7] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 205–212.

[8] E. Martin, S. Basu, and T. Xie, "WebSob: A Tool for Robustness Testing of Web Services," in *Companion to the proceedings of the 29th International Conference on Software Engineering (ICSE'07)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 65–66. [Online]. Available: http://dx.doi.org/10.1109/ICSECOMPANION.2007.84

[9] M. D. Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann, "Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 612–617. [Online]. Available: http://dx.doi.org/10.1109/DSN.2007.102

[10] Z. Ding and M. Jiang, "Port Based Reliability Computing for Service Composition," in *Proceedings of the 2009 IEEE International Conference on Services Computing (SCC'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 403–410. [Online]. Available: http://dx.doi.org/10.1109/SCC.2009.12

[11] A. J. Maâlej, M. Krichen, and M. Jmaiel, "Model-Based Conformance Testing of WS-BPEL Compositions," in *COMPSAC Workshops*, 2012, pp. 452–457.

[12] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar, "Test Coverage of Data-Centric Dynamic Compositions in Service-Based Systems," in *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation (ICST'11)*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 40–49. [Online]. Available: http://dx.doi.org/10.1109/ICST.2011.55

[13] L. Juszczyk and S. Dustdar, "Programmable Fault Injection Testbeds for Complex SOA," in *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10)*, 2010, pp. 411–425.

[14] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures," in *Service-Oriented Computing — ICSOC 2008 Workshops*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 41–52. [Online]. Available: http://dx.doi.org/10.1109/DSN.2007.102

[15] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 3–. [Online]. Available: http://dl.acm.org/citation.cfm?id=960322.960404

[16] G. Canfora and M. Penta, "Software engineering," A. Lucia and F. Ferrucci, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Service-Oriented Architectures Testing: A Survey, pp. 78–105.

[17] W. Tsai, X. Zhou, Y. Chen, and X. Bai, "On Testing and Evaluating Service-Oriented Software," *Computer*, vol. 41, no. 8, pp. 40 –46, aug. 2008.

[18] R. Heckel and L. Mariani, "Automatic Conformance Testing of Web Services," in *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE'05)*. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 34–48.

[19] S. Zhao, X. Lu, X. Zhou, and T. Zhang, "A Reliability Model for Web Services - From the Consumers' Perspective," in *Proceedings of the 2011 International Conference on Computer Science and Service System (CSSS)*, 2011.

[20] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based Testing Tool for Web Services," in *Proceedings of the 2009 International Conference on Software Testing Verification and Validation (ICST'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 326–335. [Online]. Available: http://dx.doi.org/10.1109/ICST.2009.28

[21] W. Van Der Aalst, A. H. M. T. Hofstede, and M. Weske, "Business Process Management: a Survey," in *Proceedings of the 2003 International Conference on Business Process Management (BPM'03)*. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 1–12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1761141.1761143

[22] Y. Ni, S.-S. Hou, L. Zhang, J. Zhu, Z. Li, Q. Lan, H. Mei, and J.-S. Sun, "Effective Message-Sequence Generation for Testing BPEL Programs," *IEEE Transactions on Services Computing (TSC)*, vol. 99, no. PrePrints, 2011.

[23] Z. Zakaria, R. Atan, A. A. A. Ghani, and N. F. M. Sani, "Unit Testing Approaches for BPEL: A Systematic Review," in *Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference (APSEC'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 316–322. [Online]. Available: http://dx.doi.org/10.1109/APSEC.2009.72

[24] K. Ueno and M. Tatsubori, "Early Capacity Testing of an Enterprise Service Bus." Washington, DC, USA: IEEE Computer Society, 2006, pp. 709–716. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2006.57

[25] D. N. Le, S. Goh, and A. Eck, "A Survey of Web Service Discovery Systems," *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 2, no. 2, pp. 65 – 80, 2007.

[26] C. Schmidt and M. Parashar, "A Peer-to-Peer Approach to Web Service Discovery," *World Wide Web*, vol. 7, no. 2, pp. 211–229, 2004.

[27] L.-H. Vu, M. Hauswirth, and K. Aberer, "Towards P2P-Based Semantic Web Service Discovery with QoS Support," in *Proceedings of the Third international conference on Business Process Management (BPM'05)*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 18–31. [Online]. Available: {http://dx.doi.org/10.1007/11678564_3}

[28] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001.

[29] P. Leitner, A. Michlmayr, and S. Dustdar, "Towards Flexible Interface Mediation for Dynamic Service Invocations," in *Proceedings of the 3rd Workshop on Emerging Web Services Technology, co-located with ECOWS'08*, 2008.

[30] S. K. Williams, S. A. Battle, and J. E. Cuadrado, "Protocol Mediation for Adaptation in Semantic Web Services," in *Proceedings of the European Semantic Web Conference (ESWC 2006)*, 2006, pp. 635–649.

[31] J.-P. Friedenstab, C. Janiesch, M. Matzner, and O. Muller, "Extending BPMN for Business Activity Monitoring," in *Proceedings of the 2012 45th Hawaii International Conference on System Sciences (HICSS'12)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 4158–4167. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2012.276

[32] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection," in *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC'09)*. New York, NY, USA: ACM, 2009, pp. 1–6.

[33] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards Pro-Active Adaptation With Confidence: Augmenting Service Monitoring With Online Testing," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*. New York, NY, USA: ACM, 2010, pp. 20–28. [Online]. Available: http://doi.acm.org/10.1145/1808984.1808987

[34] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 132–144.

[35] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, May 2002.

[36] B. Wetzstein, P. Leitner, F. Rosenberg, S. Dustdar, and F. Leymann, “Identifying influential factors of business process performance using dependency analysis,” *Enterprise Information Systems*, vol. 5, no. 1, pp. 79–98, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1080/17517575.2010.493956

[37] P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar, “Application-Level Performance Monitoring of Cloud Services Based on the Complex Event Processing Paradigm,” in *IEEE International Conference on Service-Oriented Computing and Applications (SOCA’12)*, 2012, p. (to appear).

[38] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, “Runtime Prediction of Service Level Agreement Violations for Composite Services,” in *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC’09)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 176–186. [Online]. Available: http://portal.acm.org/citation.cfm?id=1926618.1926639

[39] A. Bertolino, G. Angelis, L. Frantzen, and A. Polini, “Model-Based Generation of Testbeds for Web Services,” in *Proceedings of the 20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems (TestCom’08)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 266–282.

[40] T. J. McCabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Jul. 1976. [Online]. Available: http://dx.doi.org/10.1109/TSE.1976.233837

[41] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, “End-to-End Versioning Support for Web Services,” in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC’08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 59–66. [Online]. Available: http://portal.acm.org/citation.cfm?id=1447562.1447851

[42] H. Q. T. Pham Thi Quynh, “International journal of electrical and electronics engineering 3:5 2009,” 2009, ch. Dynamic Coupling Metrics for Service Oriented Software, pp. 282–87.

[43] M. D. Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. D. Nitto, “WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services,” in *Proceedings of the International Workshop on Service-Oriented Software Engineering (SOSE’06)*. New York, NY, USA: ACM, 2006, pp. 74–80. [Online]. Available: http://doi.acm.org/10.1145/1138486.1138502

[44] P. Leitner, F. Rosenberg, and S. Dustdar, “Daios: Efficient Dynamic Web Service Invocation,” *IEEE Internet Computing*, vol. 13, pp. 72–80, 2009.

[45] Y. Yuan, Z. Li, and W. Sun, “A graph-search based approach to bpel4ws test generation,” in *Proceedings of the International Conference on Software Engineering Advances*, ser. ICSEA ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 14–. [Online]. Available: http://dx.doi.org/10.1109/ICSEA.2006.6