Governing Regression Testing in Systems of Systems

Antonia Bertolino
ISTI–CNR
Pisa, Italy
antonia.bertolino@isti.cnr.it

Guglielmo De Angelis
IASI–CNR
Roma, Italy
guglielmo.deangelis@iasi.cnr.it

Francesca Lonetti
ISTI–CNR
Pisa, Italy
francesca.lonetti@isti.cnr.it

***Abstract*—Great advances in network technology and software engineering have triggered the development and spread of Systems of Systems (SoSs). The dynamic and evolvable nature of SoSs poses important challenges on the validation of such systems and in particular on their regression testing, aiming at assessing that run-time changes and evolutions do not introduce regression in SoS behavior. This paper outlines issues and challenges of regression testing of SoSs, identifying the main kinds of evolution that can impact on their regression testing activity. Furthermore, it presents a conceptual framework for governing the regression testing of SoSs. The proposed framework leverages the concept of an orchestration graph that describes the flow of test cases and sketches a solution for deriving a regression test plan according to test cases dependencies.**

***Index Terms*—System of Systems, Regression Testing, Governance, Test Cases Orchestration**

I. Introduction

Already more than two decades ago, in his highly-referenced taxonomy of architecting principles for Systems-of-Systems, Maier [1] referred to the latter as an *emergent class of systems*. In that paper he attempted a definition of an SoS as "*an assemblage of components which individually may be regarded as systems*", and which yields both operational and managerial independence. In the following years until present time, a growing literature has addressed the many and difficult challenges arising in SoS engineering, and several other definitions have been proposed, sometime even misleading or contradictory [2].

This difficulty in finding one comprehensive definition probably descends from the many different ways and purposes by which the constituent systems can collaborate, as well as from the inherent complexity of SoSs. A classification useful to clarify the field is due to the US Department of Defense and is reported by Nielsen and coauthors in their recent systematic review [3]. Four different SoS categories are defined: *Directed*, which are assembled and centrally managed solutions conceived in order to satisfy a specific purpose; *Collaborative*, which also foresee a shared management for the system-as-a-whole, but this is not compelling and is accepted in collaborative way; *Acknowledged*, which do not rely on a centralized management authority, and keep managerial, operational and technical independence at the constituent system level. The interactions among the constituent systems take place by abiding by the role foreseen in the collaboration; and *Virtual*, which do not rely on any managerial control, nor even on any explicit shared purpose among the constituent systems. Often in Virtual SoSs the interactions emerge due to the combined usage of several independent systems that are linked together by a final-user, and the SoS only exists with respect to his/her perspective.

Nonetheless, and whatever its category, as for the engineering of any other software systems, also SoSs should undergo an adequate testing campaign and in an earlier work [4] we investigate how and to what extent existing test techniques can be adapted to cope with SoS peculiarities.

One specific and prevalent type of testing is *regression testing* [5], which is conducted after introducing changes or evolution to ascertain that these have not caused undesired issues, or *regressions*. This is commonly acknowledged as the most demanding and costly part of the software testing process [6].

Even though a common definition for an SOS does not exist, all authors concur that *evolution* is a distinguishing characteristic. SoSs evolve dynamically and are integrated at run-time: Abbott [2] depicts the situation saying that SoSs are open at the top, i.e., new applications can be created at any time, and at the bottom, i.e., their underlying technology can change, and their composition evolves slowly but continuously. Therefore, in principle such continuous change and evolution require that a framework is established in which the test information (which test cases have been executed, when, and with what results) related to the SoS is kept, and a proper regression testing stage is established depending on the occurring modifications.

Surprisingly, while the need for regression testing of SoS (which we abbreviate as SoSRT) would seem obvious, we found very little or null attention in the literature to this specific problem. Clearly the way and capabilities for conducting the regression testing will also vary depending on the SoS category. For the case of a Directed SoS probably testing needs will not change much from the scenario of testing a component-based system [7], thanks to the presence of a centralized manager. On the opposite side, testing a Virtual SoS is probably feasible only in the field along the directions presented in [8] about service federations.

In this paper, we provide the following contributions: a definition of SoSRT, a discussion of related challenges, and

a preliminary outline of a framework that addresses this aim in the context of Collaborative and Acknowledged SoSs. Specifically, Section II overviews the status of current research on regression testing in the context of SoSs, while Section III presents our contribution to this topic. Concluding remarks are given in Section IV.

II. Regression Testing in SoSs

As said in the Introduction, regression testing aims to guarantee that the changes introduced in a program do not harm the behavior of the existing software. A simple approach for testing the modified software is to reuse the whole existing test suite (i.e., *retest-all*). This strategy becomes not affordable when the software evolves and the test suite size grows [5]. Many authors aim to make the regression testing activity more cost effective by proposing techniques for: i) test suite minimization to eliminate redundant test cases from the test suite [9], or ii) test case selection in order to re-run only tests related to the changed parts of the software [10], or iii) test case prioritization to identify an ordering of the test cases yielding early fault detection [11], [12].

Regression testing has been addressed for long time in the literature, considering a variety of domains such as Service-Oriented Architectures (SOAs) and configurable systems [13] among the most recent ones, as well as new development practices as Continuous Integration [14]. However, there is not so much recent literature about regression testing of SoSs.

Harrold and Orso [6] overview the state of the research and the state of the practice in regression testing, presenting an analysis of major issues and challenges. They note that despite significant research in regression testing, only a few of techniques and tools developed by researchers and practitioners have been applied on large-scale real-world systems and shown to be useful in practice. In many industrial contexts, test cases selection and maintenance remain manual on the basis of tester's experience. This rises the need of automated methods able to identify obsolete and redundant test cases. Among the major issues of regression testing they identified test suite maintenance and manipulation, while emerging research challenges include: i) building richer models representing complex testing information (e.g., relationships and dependencies among test cases); and ii) finding new manipulation techniques able to derive new test cases starting from existing ones.

This work contributes to both those research directions by leveraging the concept of *test case orchestration* [15], which refers to all those activities that dynamically organize the execution of tests. Test case orchestration enables the declaration of flows of test cases, and provides a means for running existing test cases in different operational contexts: previous test results can dynamically instantiate current parameters or modify the execution environment. We discuss further test case orchestration in the context of SoSRT in Sec. III,

Other model-based testing approaches not specifically tailored for regression testing advocate the *on-the-fly* generation of the next test case to execute [16]. These approaches and their underlying theory [17] are partially fitting within the context of SoSs. In fact, their main target usually is one black-box system. Difficulties can raise from the derivation of a behavioral model of the SoS or when the objective of the test is the interactions among several constituent systems.

The authors of [18] present a more recent perspective of regression testing challenges extracted from a set of empirical studies. They distinguish two major categories: *method related challenges* and *organization related challenges*. The former concerns handling failures, performance measurement, handling fault distribution, scalability of techniques, and tool support. The latter includes test suite maintenance, information availability, knowledge and skills of testers and management support. Both these challenges categories are common to many types of software systems including SoSs.

Other challenges are observed in an industrial case study of a highly complex SoS [19]. Although a separate regression testing phase is not performed in this study, the authors identify among the main issues influencing the maintenance of the regression test suite the high complexity and number of test cases as well as the lack of guidelines for testing.

However, in the context of SoSs, specific regression testing challenges arise because of their dynamic evolution and reconfiguration during operation time. In some cases, the constituent systems retain independent control and objectives: they may evolve according to internal needs, operational directions and managerial control; all these changes may be independent from the purposes of any SoS the constituent could be engaged in. This implies key challenges for regression testing. First, the updates on the constituent systems or on their interactions may lead to unintended emergent behaviours. Unpredictable scenarios become more frequent as the number or the variability of the constituent systems increase [20]. Another important challenge of SoSRT is related to their dimension and complexity. SoSs could be very large-scale systems and many regression testing methods developed for simple applications may not scale up to address the complexity of these systems.

To handle these challenges the research on SoSRT must take into account the dynamic evolution and how such changes impact on the regression testing activity. Taking in mind that it is not possible to outline all the possible changes that can affect an SoS, we identify here some of the main kinds of evolution for SoSs deserving to be subject to regression testing:

I. *evolution of a constituent system within the SoS*: if one of the constituent systems is updated, because there is for instance a new release of one of its software components, or a new deployment or a new configuration of the constituent system is set up, it is needed to test that no regression has been brought in the SoS by the modified system;
II. *usage of additional functionalities of a constituent system*: functionalities of a constituent system which were not exercised in the initial configuration of the SoS may be required by components inserted at run-time. These functionalities need to be tested before being used and the existing test suites need to be augmented with new tests addressing these additional functionalities;

III. *interdependencies between constituent systems*: testing needs to assure the interdependencies between constituent systems remain stable and reliable when a new constituent system is integrated in the SoS or is disassembled from it.

IV. *operational independence of the constituent system*: constituent systems must be able to fulfill their operational purposes not involved in the SoS mission in their own right and when they are disassembled from the overall SoS. SoSRT here aims at checking that SoS mission did not impact its sub-systems.

Considering these main kinds of evolution for SoSs, we sketch a definition of SoSRT: Let S be an SoS validated on an existing test suite T, let S' be a modified version of S, let $CS_1, CS_2, \ldots, CS_n$ be the constituent systems engaged in S, SoSRT concerns the validation of S' on a test suite T' derived from T, when any (one or more) $CS_i \in \{CS_1, CS_2, \ldots, CS_n\}$ is affected by any of the evolution kinds I-IV described above.

As said, a *retest-all* strategy (i.e., $T' = T$), as well as traditional test selection or prioritization approaches, are hard to apply to SoSs, due to their dynamic and evolvable nature. This implies the need of governance approaches that are able to narrow the regression testing activities to those test cases that are more likely to spot inappropriate or unexpected behaviors. In this paper, we try to address this issue proposing a framework able to dynamically identify the next test case to execute in a given regression test suite, and at what time. We refer to next section for a description of the proposed approach.

III. Approach Proposal

Directed SoSs are often considered as a broader system that is built from the composition of the functionalities offered by each constituent part. The well-defined organization from both the managerial and the operational perspective leads to consider the engineering of these SoSs similar to the case of a stand-alone system entirely conceived internally to a single organization. In this sense, available approaches to regression testing can be straightforwardly adapted to the context of Directed SoSs. On the other hand, the complete lack of any managerial and operational coordination in Virtual SoSs hinders the identification of a systematic approach for supporting the regression testing activities. For this reason, the proposed approach does not address these classes of SoSs, but specifically focuses on Collaborative and Acknowledged categories. For both of them, it is reasonable to assume that each constituent system is equipped with some kind of on-purpose features (e.g., interfaces, communication channels, bulletin boards, etc.) suitable for notifying the others about evolution or changes on their status. In addition as Collaborative and Acknowledged SoSs foresee a managerial supervision of the constituents, it is also admissible to assume the presence of an underlying layer that can access this information and that is devoted to governing the regression testing activities on the SoS from both a managerial and an operational perspective (see Fig. 1).

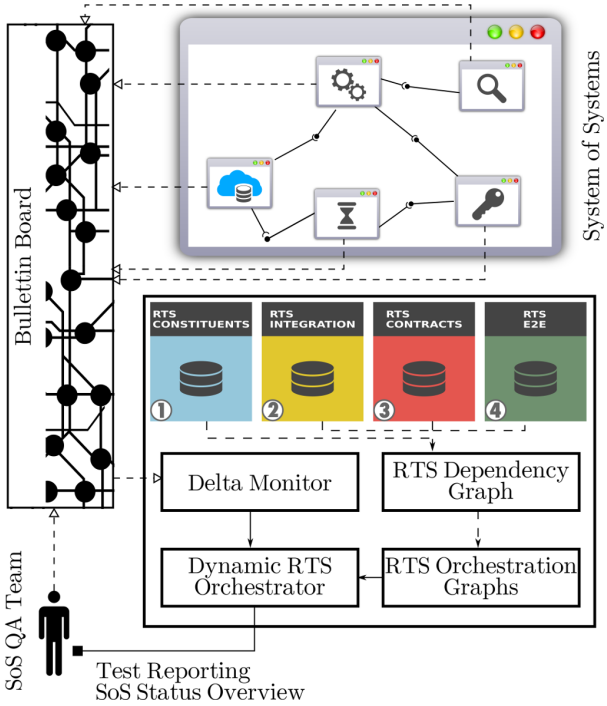

Fig. 1. A Framework for Governing SoS Regression Testing

In line with the literature on software testing, it is possible to reconsider the responsibility levels of each testing phase to the case SoSRT: from unit testing, to integration and contract testing, up to end-to-end testing. At each level, the degree of automation as well as the specific implementation of regression tests could be declined in terms of either a managerial or a technical perspective, depending on the cases.

In particular, regression tests aiming at validating the internal behavior of each constituent can be seen as unit-level checking of the SoS (i.e., ① in Fig. 1). Regression integration tests in SoSs are devoted to validate the communication paths and the interactions among the composed sub-systems (i.e., ② in Fig. 1). Typically their goal is to gain confidence that each sub-system can communicate with the others, rather than to fully test behavioral conformance. As such, regression test cases for integration are often considered as a fast feedback toward integration. Similarly, the responsibility of regression tests depending on contracts is to check interactions at the boundary of the constituents and to assert that each constituent system in the SoS meets its contract (either functional or not) while interacting with the others (i.e., ③ in Fig. 1). Finally, regression end-to-end tests aim to verify that the SoS achieves its system-as-a-whole goals. Usually, they are expected to focus on testing the messages between the sub-systems but also to validate that any extra network infrastructure such as firewalls, proxies or load-balancers are correctly configured (i.e., ④ in Fig. 1).

Some of the common issues discussed within the context of test automation [21] are also affecting SoSRT. Specifically, the costs of writing and maintaining regression tests increases

in the same way as the test objectives move from unit to end-to-end. As a consequence, testers remain largely responsible of properly identifying, selecting, and possibly combining regression test cases in order to assess if an SoS keeps working correctly even after changes have been introduced either in its constituent systems or its environment.

The approach for governing SoSRT introduced in Fig. 1 is grounded on the orchestrated execution of several testing modules yielding different granularity and focusing on different objectives. Its overall intent is to increase the quality, and the reusability of regression tests leveraging a test orchestration strategy enabling the creation of complex regression test suites as the composition of simple testing bundles. The specification of policies for structuring relations among test cases, as well as the set-up of policies governing their aggregation can be used by the members of the QA Team as the basis for driving regression testing activities and for supporting the root-cause analysis of spotted errors/issues. In detail, the proposed governance framework of regression testing in SoSs is structured around five main aspects:

- *Dependency Graphs*: Identify the dependencies among the available regression test cases. Dependencies could be established among test cases targeting the same testing objective or not;
- *Orchestration Graphs*: For each specific objective that has to be achieved, plan and declare flows of test case executions by taking into account the relations emerging from the dependency graphs;
- *Monitoring Rules*: Establish policies on the information, e.g., related to evolution, that should be notified by the constituents, and the criteria enabling the selection of some orchestration graph;
- *Tests Orchestration*: Launch the regression testing activities according to the statements in the orchestration graph: dynamically decide which is the next regression test case to execute. Report the outcomes to the QA Team;
- *Feedback*: Analyze the results of the testing session; in case of regressions in the SoS or in any of its constituent, plan and enforce a strategy for recovering.

In detail, the first aspect concerns building a dependency graph on top of the regression test suites available at any level. Dependencies among two regression test cases could be defined due to an explicit managerial assertion, but also by means of the formulation of a set of dependency criteria driving some (semi-)automatic classification procedure. For example a dependency between two regression integration test cases I_1, and I_2 (i.e., see ② in Fig. 1) could be established if the constituents referred by I_1 are a subset of the ones referred by I_2 or vice-versa; similarly I_1 could be marked with a dependency from all those regression test cases at sub-system level (i.e., see ① in Fig. 1) directly targeting any of the constituents referred by I_1 itself. In this last example, the rationale is that if any regression in communications is revealed by I_1, it could be meaningful to check if any regression occurred also at the level of those sub-systems involved in the communication. Similar criteria can be established by considering regression test cases about contracts or at end-to-end level.

The resulting dependencies among the regression test suites are the basis for constructing a set of orchestration graphs describing the (sequential or parallel) flows according to the regression test case that should be combined and launched against the SoS. It is important to clarify the term "test orchestration" refers to a concept that is different from both "test selection", and "test prioritization". Indeed test selection is typically an off-line activity that consists in choosing which are the more appropriate tests to execute. The selection could be driven by some software artifacts or models describing the SoS; it can also take into account some selection criteria (e.g., module coverage). Test prioritization can be seen as sorting activity on the tests (i.e., what to test first), which may have been previously selected. Test orchestration can be considered a superset of both: once a test case has been executed the choice of the next one is decided on-line taking into account either the observed test verdict (i.e. test passed or failed), or the output data produced by its execution.

As detailed in [15], verdict-driven orchestration is helpful in order to structure conditional chains of test cases executions by reasoning on the logical combination of the observed outcomes. In addition to such an aspect, data-driven orchestration also enables the possibility to interconnect test cases: the output data of each test can be used to feed the next test in the execution chain. However, test cases suitable for data-driven orchestration require more effort in the specification of the test cases: tests need to be designed to be interconnected as they should be composable. In general this requirement could be a threat to regression testing of SoSs, as the tests are supposed to bring models matching input/output values in addition to the usual test verdict.

Aspects such as *Dependency Graphs* and *Orchestration Graphs* concern off-line activities that are required during the creation or the maintenance of a regression test suite. The remaining aspects mainly refer to on-line activities. In particular, the bulletin board system is in charge of collecting the information exchanged by the constituents about their status or evolution. As introduced earlier, the detail of such information strongly depends on the specific instance of the SoS and their nature could span from detailed log entries at sub-system level up to managerial notification about the SoS [22]. The framework also includes a dedicated monitoring module which has the responsibility to dig such information looking for hints that deserve the activation of a new regression testing campaign (i.e., delta-monitoring in Fig. 1). The ways information are matched could be both explicit (e.g., due to the direct implementation of some managerial recommendations), or recognized by means of operative rules dynamically instantiated starting from a combination of higher-level meta-rules [23], [24]. When a change is detected, the delta-monitor triggers the orchestrator to dynamically select and launch the most appropriate regression testing strategy for the detected changes. More specifically, the envisioned approach is that the notification from the delta-monitor could enable the selection

of an entry point for one or more orchestration graphs as they were structured by the initial steps.

As introduced in [15], currently there are several Domain-specific Languages (DSL) that can be considered as technical alternatives for a system-level implementation of the proposed RTS Orchestrator. Among the others, an interesting solution comes from the Jenkins Pipeline[1]. Such a DSL easily supports sequencing and parallel execution of tests by including system-level operations such as checking out the project's source control, reporting, deploying, etc.

Looking at the results of the regression tests dynamically selected and launched by orchestrator, a QA Team (the approach refers to both Collaborative and Acknowledged SoSs, thus it is reasonable to assume such a team exists) can draw an outline of the status of the considered SoS. In this sense the QA Team can also inform the constituents about discovered regression errors or potential leaks by pushing such information inside the bulletin board system.

IV. Conclusions

Researchers investigated several approaches for checking if an evolution of a software brings undesired errors or side-effects. Nevertheless, the application of smart regression testing techniques still appears to be under-considered in the context of large-scale complex systems, and simpler *retest-all* strategies are often preferred. These practices appear to be even more frequent when the complexity of the regression test suite grows, like in the context of SoSs. Among the many possible explanations, we believe there is lack of structured methods conceived for regression test suites that can evolve over time in order to reflect SoS evolutions: by skipping redundant/obsolete test cases, or focusing only on a limited part of the SoS.

In this paper an approach has been presented to structure and automate the regression testing activities in Collaborative and Acknowledged SoSs. The work starts from an analysis of the objectives for each regression testing phase of SoSs; then it defines a proposal for a governance framework based on the concept of an orchestration graph. Specifically such a graph describes flows of regression tests cases and provides an executable abstraction for their automatic on-line composition. Future research will address an assessment of the proposed framework, and consideration of Virtual SoSs.

Acknowledgment

This paper has been partially supported by the Italian MIUR PRIN 2015 Project: GAUSS, and partially by the European Project H2020 731535: ElasTest

References

[1] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering: The Journal of the International Council on Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[2] R. Abbott, "Open at the top; open at the bottom; and continually (but slowly) evolving," in *Proc. of IEEE/SMC International Conference on System of Systems Engineering*, 2006, pp. 1–6.

[3] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 18:1–18:41, 2015.

[4] V. de Oliveira Neves, A. Bertolino, G. De Angelis, and L. Garces, "Do we need new strategies for testing systems-of-systems?" in *Proc. of SESoS*, 2018, pp. 29–32.

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[6] M. J. Harrold and A. Orso, "Retesting software during development and maintenance," in *Proc. of Frontiers of Software Maintenance*, 2008, pp. 99–108.

[7] M. Jaffar-ur Rehman, F. Jabeen, A. Bertolino, and A. Polini, "Testing software components for integration: a survey of issues and techniques," *Software Testing, Verification and Reliability*, vol. 17, no. 2, pp. 95–133, 2007.

[8] A. Bertolino, G. De Angelis, S. Kellomaki, and A. Polini, "Enhancing service federation trustworthiness through online testing," *IEEE Computer*, vol. 45, no. 1, pp. 66–72, 2012.

[9] A. Vahabzadeh, A. Stocco, and A. Mesbah, "Fine-grained test minimization," in *Proc. of ICSE*, 2018, pp. 210–221.

[10] R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, "Effective regression test case selection: A systematic literature review," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 29:1–29:32, May 2017.

[11] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74 – 93, 2018.

[12] D. Paterson, J. Campos, R. Abreu, G. M. Kapfhammer, G. Fraser, and P. McMinn, "An empirical study on the use of defect prediction for test case prioritization," in *Proc. of ICST*. IEEE, 2019, pp. 346–357.

[13] S. Souto and M. d'Amorim, "Time-space efficient regression testing for configurable systems," *Journal of Systems and Software*, vol. 137, pp. 733–746, 2018.

[14] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proc. of the 22nd International Symposium on Foundations of Software Engineering*, 2014, pp. 235–245.

[15] B. García, F. Lonetti, M. Gallego, B. Miranda, E. Jiménez, G. De Angelis, C. E. Moreira, and E. Marchetti, "A proposal to orchestrate test cases," in *Proc. of QUATIC*, 2018, pp. 38–46.

[16] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini, "Model-based Generation of Testbeds for Web Services," in *Proc. of the 20th IFIP Int. Conference on Testing of Communicating Systems (TESTCOM)*, ser. LNCS, vol. 5047. Springer, 2008, pp. 266–282.

[17] L. Frantzen and J. Tretmans, "Model-based testing of environmental conformance of components," in *Proc. of the 5th Int. Symposium on Formal Methods for Components and Objects (FMCO) – Revised Lectures*, ser. LNCS, vol. 4709. Springer, 2006, pp. 1–25.

[18] D. Brahneborg, W. Afzal, and A. Čauševič, "A pragmatic perspective on regression testing challenges," in *Proc. of Int. Conference on Software Quality, Reliability and Security Companion*, 2017, pp. 618–619.

[19] N. B. Ali, K. Petersen, and M. V. Mäntylä, "Testing highly complex system of systems: an industrial case study," in *Proc. of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 211–220.

[20] J. Dahmann, J. A. Lane, G. Rebovich, and R. Lowry, "Systems of systems test and evaluation challenges," in *Proc. of 5th International Conference on System of Systems Engineering*, 2010, pp. 1–6.

[21] D. Spinellis, "State-of-the-art software testing," *IEEE Software*, vol. 34, no. 5, pp. 4–6, 2017.

[22] A. Ben Hamida, A. Bertolino, A. Calabrò, G. De Angelis, N. Lago, and J. Lesbegueries, "Monitoring service choreographies from multiple sources," in *Proc. of the 4th Int. Workshop on Sw Engineering for Resilient Systems*, ser. LNCS, vol. 7527. Springer, 2012, pp. 134–149.

[23] A. Bertolino, A. Calabrò, and G. De Angelis, "A Generative Approach for the Adaptive Monitoring of SLA in Service Choreographies," in *Proc. of the 13th International Conference on Web Engineering*, ser. Lecture Notes in Computer Science, vol. 7977. Springer, 2013.

[24] ——, "Adaptive SLA Monitoring of Service Choreographies Enacted on the Cloud," in *Proc. of the 7th Int. Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, 2013, pp. 92–101.

[1] see at:https://jenkins.io/doc/book/pipeline/