Adequate Monitoring of Service Compositions

Antonia Bertolino
ISTI-CNR
via Moruzzi, 1
Pisa, Italy
antonia.bertolino@isti.cnr.it

Eda Marchetti
ISTI-CNR
via Moruzzi, 1
Pisa, Italy
eda.marchetti@isti.cnr.it

Andrea Morichetta
ISTI-CNR
via Moruzzi, 1
Pisa, Italy
andrea.morichetta@isti.cnr.it

ABSTRACT

Monitoring is essential to validate the runtime behaviour of dynamic distributed systems. However, monitors can inform of relevant events as they occur, but by their very nature they will not report about all those events that are not happening. In service-oriented applications it would be desirable to have means to assess the thoroughness of the interactions among the services that are being monitored. In case some events or message sequences or interaction patterns have not been observed for a while, in fact, one could timely check whether this happens because something is going wrong. In this paper, we introduce the novel notion of monitoring adequacy, which is generic and can be defined on different entities. We then define two adequacy criteria for service compositions and implement a proof-of-concept adequate monitoring framework. We validate the approach on two case studies, the Travel Reservation System and the Future Market choreographies.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and debugging—*monitors*

General Terms

Performance, Measurement, Management, Verification

Keywords

Adequacy criteria, Choreography, Operation coverage, Branch coverage, Monitoring

1. INTRODUCTION

Wisdom is knowing how little we know. Socrates

Services are pervasive and increasingly affect our everyday life. Services are discovered, accessed and bound dynamically from different devices. An essential instrument of service provision is the capability to monitor their behaviour and their offered Quality of Service (QoS). The two activities of service management and monitoring are together situated by Papazoglou and coauthors [21] at the top of the three planes of their research roadmap in service-oriented computing, above service composition and service foundations. In this paper we focus on monitoring of service compositions.

Service monitoring can come in many flavors, depending on several factors. Ghezzi and Guinea [12] classify monitors according to the type of monitored properties, the service composition paradigm, the method used to collect data, degree of invasiveness, and timeliness. For example, we could monitor functional or non-functional properties, of either service orchestrations or service choreographies, following an event-based or a state-based approach, etc. In fact many different approaches and frameworks have been recently proposed for service monitoring.

However, behind all monitoring approaches we can identify one common high-level goal, that is to timely detect potential problems occurring during service execution, and one common activity flow, consisting of: (*i*) dynamically extracting information about the interactions among services, (*ii*) collecting this information, and (*iii*) presenting it to users in useful formats [15]. Monitoring thus is inherently a passive activity: monitors by their very nature are limited to capturing and analysing what is going on.

In recent years, researchers have realized that standard monitors are not sufficient to validate system behaviours in the rapidly and continuously evolving context of service-oriented architecture, and have recognised the need to make them more powerful and self-aware. There have been several proposals, as we discuss in the related work section, for augmenting the analysis capability of a monitor, for example by making it capable to predict events before they occur or to extrapolate the allowed remaining margins of QoS values. However, we also would like to have ways to assess whether a monitoring session is *adequate*, i.e., if it is keeping track of all events of interest, or if instead the collected information misses some important event. By saying that the monitor does not report some event, here, we do not refer to the chance that the monitor lost it, but to the fact that the event did not happen.

Our proposal goes in the direction of making the monitor *wiser* in the sense of Socrates's saying, i.e., we would like to enable a monitor "to know that it does not know". To do this, we propose to explicitly define what are the relevant entities that we would expect to observe and hence to set an adequacy criterion on such entities, implying that the mon-

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2237-9/13/08
http://dx.doi.org/10.1145/2491411.2491441

itored information should normally report that such entities have been covered. As the standard monitoring activity proceeds, we can then progressively measure the percentage of observed entities, and thus become aware that some expected (simple or complex) events have not happened for some time.

Coverage assessment has been introduced in software testing decades ago, e.g. referring to coverage of entities in the program control-flow or data-flow [23], and nowadays constitutes a fundamental part of testing activity. We introduce here the similar and as important notion of *monitoring adequacy*. We highlight a fundamental difference between the traditional notion of testing adequacy and the novel notion of monitoring adequacy, namely the concept of the *observation window* along which coverage is measured: by defining the period in which monitor adequacy is assessed, the observation window adapts to monitoring the concept of a test session or test suite on which test adequacy is evaluated.

Presentation-wise we distinguish between the abstract definition of the monitor adequacy and its possible instantiations. The former is completely generic and can be applied to different entities and application domains; the latter are obtained after establishing a type of coverage requirements, i.e., what are the entities that will be counted. In this paper we provide contributions in both directions, as follows:

- we introduce the generic notion of a *monitor adequacy criterion*;
- we define and implement two specific instantiations of a monitor adequacy criterion for the monitoring of service compositions;
- we provide a proof-of-concept framework and its preliminary assessment on two case studies, showing the feasibility of the approach and interesting information that can be obtained.

The paper structure is as follows: Sections 2 and 3 illustrate the idea and propose a generic definition of adequate monitoring. Section 4 provides an instantiation of an adequate monitor for services choreographies and Section 5 describes the high-level architecture of a proof-of-concept adequate monitor. Section 6 introduces the case studies considered and Section 7 discusses some preliminary results. Finally, Section 8 puts our work in context of related work and Section 9 draws conclusions.

2. MOTIVATION

To illustrate our idea we consider a simplified example of a *home banking* application, which provides bank customers with different services performing the most common bank account operations, including: *withdraw*, *deposit*, *bank transfer*, *stock market*, *bank statement*, and so on. It is natural to suppose that the bank is interested in monitoring the banking activities performed by the users of home banking functionality, so to evaluate the QoS and to timely detect possible problems that could compromise the overall system security and usability.

However monitors typically allow to observe relevant events as they occur, but do not by default provide information that something is *not* happening. Now, suppose that for some time no home banking customers ask for stock market operations: this may be due simply to a temporary decrease of customers interest in stock market investment, or instead to a service malfunctioning. In either case, it could be very useful for the bank to be promptly informed that no invocation to the *stock market* service is being monitored, so that they can analyse the situation and implement the most suitable solution.

As another example, the bank can be interested to evaluate the sequence of interactions of its users and the respective responses, and in particular to check if their relative frequencies conform to those expected. Suppose that in a normal usage of the home banking platform, an operation profile assigns some respective percentages to every service invocation and response. A change of the expected usage statistics over time could happen for different reasons and is certainly an important symptom for the bank: if a drastic decrease of successful withdraw operations is evidenced, the bank could suspect of a malfunctioning of the service behavior and possibly activate the necessary testing and maintenance activity.

We do not say that a standard monitoring activity could not in principle detect the above mentioned situations. However, ultimately any monitor collects and analyses data following some predefined monitoring rules (for our scope it is indifferent how these rules are given, as probes, or assertions, or by other approaches). Therefore, to be informed that some interesting operation is not being observed, as in the missing invocations to *stock market* service example, or is happening with a frequency different from the expected one, as in the changed usage statistics example, the bank should in advance instruct the monitor with some time-out interval or some usage profile for each event they want to keep under control. Without being explicitly instructed by some specific rule, the monitor cannot spot potential critical situations. The reason behind such weakness is that monitors passively log what is happening, but lack any adequacy criterion to decide whether the collected data provide a complete snapshot of the system potential behaviours.

To address this need in a comprehensive way, we propose to enhance monitors with a notion of coverage, and make them capable to measure coverage of the relevant entities under a defined monitor adequacy criterion.

3. DEFINING MONITOR ADEQUACY

In this section we introduce the generic concept of a monitor adequacy criterion, without binding its definition to a specific coverage measure. In fact, the notion of monitor adequacy is neutral with respect to both the entities to be covered and the application domain.

To introduce the notion of monitor adequacy, it can be useful to first recall as a reference for comparison the well established notion of test adequacy. Since the seminal work of Goodenough and Gerhart [13], adequacy criteria are extensively investigated in software testing literature and considered an essential part of any testing method [23, 27]. Adequacy criteria are used to assess the thoroughness of test suites, by establishing a set of test requirements they should fulfill. More typically, in test coverage criteria these requirements are mapped onto a set of entities that must be covered when the test cases are executed, for example all statements or all branches in the program control-flow. If all entities are covered we say that the coverage criterion is satisfied; otherwise, we can use the percentage of covered entities as a quality measure of the test suite.

The intuitive motivation behind measuring test coverage is that if some entity has never been tested, it might contain undetected faults. Obviously, the converse reasoning does not apply: if we had covered all entities and detected no failure, this does not necessarily imply that the program is correct.

In a similar way, we propose here to assess the adequacy of the monitored executions by identifying what are the relevant entities to be covered and by assessing if all of them, or otherwise what percentage, have been observed. To do this, we can refer to the observed execution traces and measure coverage of the entities belonging to these traces. As for test adequacy, the motivation behind assessing monitoring adequacy is that if some entities are not covered, we cannot exclude that these might hide some problem.

Thus, the notion of monitoring adequacy is defined similarly to test adequacy, but it implies some significant differences. One first difference concerns the period along which we measure coverage. In fact, in the case of test adequacy, this is measured over a test session and the obtained measure refers to the executed test suite. In monitoring, there is not a similar notion of a test suite, since monitoring observes how the system spontaneously behaves. In principle, the monitor observation could last indefinitely, whereas for analysis purposes we need to set some limits upon the observed execution traces. Therefore, we introduce a notion similar to that of a test session by associating the monitoring coverage measure with the length of a considered observation period. Intuitively, we define a sliding observation window over a time measurement unit, which could be either continuous (e.g., we consider the execution traces collected in the last 120 sec) or discrete (e.g., we consider the most recent 1000 traces). Another difference concerns the fact that monitoring coverage is no longer a monotonically increasing function as it happens in a test session: as more traces are monitored, one entity that was covered before could not be covered anymore. A decrease of coverage testing could happen in regression testing, after some code modification has occurred. Monitoring coverage could normally increase or decrease, allowing for detecting possible dynamic modifications in the service interactions.

We now formally define monitoring coverage. Let us denote by $r_i \in R$ the ith entity to be covered, and by $\delta_i \in \Delta$ the length of its associated observation window. We could set different lengths δ_i for each r_i, or we could have one same δ interval for all entities. The monitor adequacy criterion C dynamically measures the coverage on R for a given Δ at each time unit t as follows:

$$C[R, \Delta](t) = \frac{\sum_{i=1}^{|R|} \lambda_i(t)}{|R|} \quad (1)$$

where for $r_i \in R$ and $\delta_i \in \Delta$

$$\lambda_i(t) = \begin{cases} 1 & \text{if } r_i \text{ is covered at least once in } [t - \delta_i, t] \\ 0 & \text{otherwise} \end{cases}$$

In summary our definition of monitoring adequacy introduces the following concepts:

- an "adequate monitor" is a monitor that is instructed to keep track of a set of requirements r_i to be covered in a window δ_i (this is similar to the instrumentation phase of coverage testing);
- a monitor coverage analyzer is a tool that, at every instant t, can provide a coverage measure as in Eq. 1 and, if this is less than 1, can provide a list of those entities that have not been covered;
- as for test coverage, an entity that is not covered is not necessarily the signal of a problem, however provides an indication that we have not "heard" for some time from that entity. We may want to program the coverage analyzer so to raise a warning message when some entity is not covered.

4. AN IMPLEMENTATION

We now refine the abstract concept of monitoring adequacy introduced in the previous section. With reference to Eq. 1, the instantiation of a monitoring coverage criterion requires: to specify the requirements R to be covered; to set the (continuous or discrete) time unit t, and consequently to define the respective observation windows Δ.

We take as the application domain for our implementation the case of service choreographies. A choreography specification (which could be expressed in various languages, e.g., WS-CDL [1] or BPMN [2]) can be seen as a contract defining the agreed ordering, conditions and constraints under which participating services interact by message exchange. This specification describes the externally observable behavior of all parties involved from a global viewpoint, differently from a service orchestration, in which participants interactions are regulated from a single point of control [21]. Strictly speaking, service choreographies are not executed, but they are *enacted*: this means that we cannot run some prescribed service interaction flow (like in a BPEL [3] orchestration), rather different behaviours "happen" as participant services autonomously play their roles.

Given such a loose context, it is important to verify whether the behaviours emerging in an enacted choreography are conforming to its specification. Runtime monitoring helps to address such need, and our notion of an adequacy criterion in choreography monitoring can be useful to help assess whether all (or how many of) the "interesting" behaviours are emerging. In this paper, as we motivate in Section 4.1, we focus on adequacy criteria that measure the operations invoked, either independently of the sequence of invocation (operation coverage), or within a specified sequence (we call this branch coverage).

Once available the coverage information dynamically collected can be exploited in many ways. A choreography does not have any single point of control, but we can suppose a "choreography manager" stakeholder as a generic actor who will be in charge of choreography management and verification, and who is interested in the coverage assessment. The choreography manager could set a desired coverage measure and ask to be notified when the coverage decreases under that measure. Or, at a certain instant t, he/she could take a snapshot of coverage and check which entities are not covered. Or, a warning message could be issued if some entity has not been covered for a certain period.

It is worth notice that, as it happens with test coverage, monitoring adequacy can only give warnings. How to

[1] http://www.w3.org/TR/ws-cdl-10/
[2] http://www.omg.org/spec/BPMN/2.0/
[3] http://www.oasis-open.org/committees/wsbpel

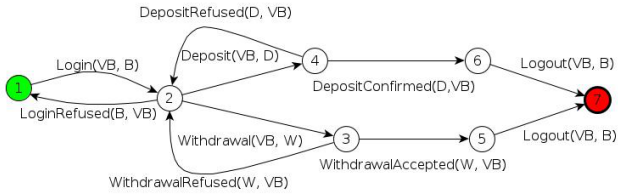

Figure 1: An excerpt of the FSM associated to the home banking choreography

best make use of those warnings and, where necessary, what could be effective counter-measures are two crucial consequent questions. We hint at some examples in Section 7, however further research is needed to answer them.

4.1 Entity Definition

Within a choreography the participating services interact by means of operation calls; thus, invocations of operations are the basic entities to be monitored. Operation coverage can be considered as a parallel to the standard test criterion of statement coverage. In particular, we define:

- *operation entity:* considering every service participating in a choreography, an operation entity is one of the operations specified in the service interface that is invoked inside the choreography at least once (a service could specify more operations that are not used within the considered choreography, which are not qualified as operation entities);

- *operation coverage domain:* considering a choreography, the operation coverage domain is the set of all the operation entities of all the services involved in the choreography;

- *percentage of operation coverage:* with reference to Eq. 1, the percentage of operation coverage at time t is given by 100*C, where R is the operation coverage domain (we discuss how the set Δ of the observation windows is defined in the next section).

Consequently, at a given instant t a choreography monitor is adequate with respect to the operation coverage criterion if the percentage of operations covered is 100% (or greater than an established threshold level).

As said, we see operation coverage as a basic criterion corresponding to all statement coverage. In testing, a minimum requirement is often considered coverage of all branches of the program control flow graph. We can establish a parallel for choreography monitoring considering a graph representation of the choreography behavioral model.

Whatever is the formalism used to represent the choreography specification, it is always possible to derive an abstract behavioral model, such as a Finite State Machine (FSM), in which: each transition represents a message exchange between two participants of the composition, and each node represents the evolution of composition states according to the exchange of messages. In Figure 1 we provide an excerpt of the behavioral model, represented by a FSM, of the home banking choreography example described in Section 2: *VB, B, D, W* indicate the *VirtualBank, Bank, Deposit, Withdraw* services involved in the choreography. We label each edge with the name of the involved operation and the sender and receiver services: for instance Login(VB,B) denotes the request sent from the VirtualBank to the Bank when a home banking user asks to login, and so on.

Over such a graph, we can introduce a concept similar to the program control flow branch. In particular:

- *branch entity:* considering the behavioral model graph of a choreography, a branch entity is an arc in this graph. It is indicated as `o(i,j)`, where `o` is operation, service `i` is the sender, and `j` is the receiver;

- *branch coverage domain:* considering a behavioral model graph of a choreography the branch coverage domain is the set of all the branch entities in the graph;

- *percentage of branch coverage:* with reference to Eq. 1, the percentage of branch coverage at time t is given by 100*C, where R is the branch coverage domain (for defining Δ see next section).

Consequently, at a given instant t a choreography monitor is adequate with respect to the branch coverage criterion if the percentage of branch coverage is 100% (or greater than an established threshold level).

4.2 Observation Windows

As defined in Eq. 1, the measure of (operation/branch) coverage depends on the length of the observation windows δ_i associated with the selected entities. How can we set the lengths of such observation windows? We provide below two practical methodologies based either on some prior knowledge (or assumptions) of the entities frequency, or on the data collected during a learning phase.

In case of operation entities, the operation coverage domain can be derived directly from the choreography specification. If some prior knowledge or expectation about how frequently such operations are invoked during the choreography enactment is available, this can be used to assign to each entity in the operation coverage domain some expected relative frequency.

In case of branches, we propose a procedure similar to that adopted in software reliability engineering for the definition of the implicit functional operation profile [20]. Considering the behavioral graph of the choreography specification, at each level, the arcs of the graph can be annotated with their relative frequencies so that the sum of the frequencies of all the arcs exiting from a node is equal to 1. In this case the absolute frequency of each branch entity can be calculated by multiplying the frequencies values of each branch entities in the path from the entity considered to the root node of the model. Each absolute frequency can then be used to calculate the length of the observation windows of the associated branch entity.

Above proposed methodologies require some prior knowledge of the entity frequencies that may not be always available. However in specific domains where a choreography is well established, it is thinkable that choreography managers or domain experts could have this knowledge or derive it from previously collected data set.

An alternative methodology we propose relies on a learning phase. The monitor is used in a first phase just to collect data useful for defining the frequency measures of interest (operation/branch). The data collection is performed for a certain period of time, under some supervision control for ensuring that the collected data are representative of regular behaviour. Afterwards the data are used to set the frequencies of each entity and to consequently estimate the associated observation windows. This methodology has the advantage that it does not require prior knowledge because data are directly obtained from the field. However the accuracy of the calculated entity frequencies is in relation with

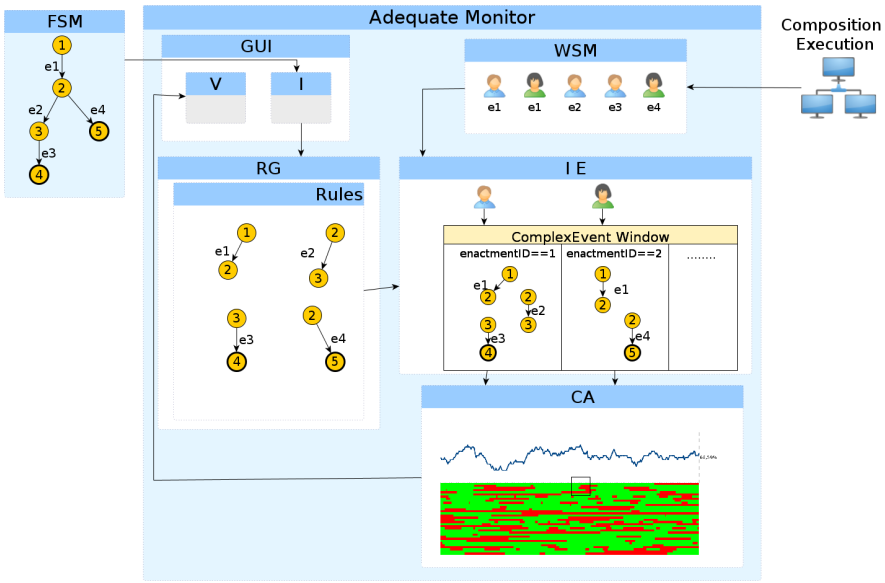

Figure 2: An instantiation of an adequate monitoring framework

the scheduled learning time and will rely on a defined level of precision. Consequently the derived observation windows have to be calculated considering a certain confidence level.

5. ADEQUATE MONITOR ARCHITECTURE

With reference to Figure 2, we outline the components of the high-level architecture of the proof-of-concept adequate monitor for service compositions we have implemented:

- **GUI**: a web interface allows the user to upload the behavioral model of the service composition and the parameters useful for adequacy measurement. Through the GUI, the user can visualize the information produced by the Coverage Analyzer (see below), e.g., a curve of the evolution of the percentage of coverage measured, the coverage details for each considered entity, the derived monitoring statistics, possible warnings raised and the monitoring log.
- **WS_Monitor** (WSM): this is a web service that receives event messages from the services involved in the composition and forwards them to the Inferential Engine. Each *EventMessage* contains the following parameters: *CompositionID*, which is the unique identifier of the composition instantiation; *EnactmentID*, which identifies the user of the composition; *EventName*, which specifies the interaction performed; and *Data*, which is an optional parameter containing additional information.
- **Inferential Engine** (IE): this component dynamically evaluates the set of rules that match with the received events, and executes the actions associated to those that are satisfied. In the current implementation, we rely on Drools engine [11], that is a business rules management system with a forward chaining inference based engine.
- **Rules Generator** (RG): this component takes in input the behavioural model and the coverage parameters selected by the user through the GUI and derives the entities to be covered (by applying a graph traversal algorithm) and the length of the observation windows (according to the two approaches outlined in Section 4.2). For each entity, RG generates one corresponding rule for the employed Drools engine. A Drools rule is composed by a condition and an action, whereby the condition contains an ordered events sequence, and the action specifies the activities to be executed when the condition is satisfied. An example of a dummy rule as we specify them for the adequate monitor is depicted in Listing 1: the *when* clause is used to check if within the observation window (e.g., the last n *EventMessages*) the entity (e.g., an *EventMessage* with $EventName = eventx$) is not observed. If such clause is satisfied the corresponding action is performed (the *covered* attribute for that entity is set to false).
- **CoverageAnalyzer** (CA): this component captures all the actions executed by IE and elaborates them so to extract useful information for:
  - visualizing the monitored events and elaborating statistics about the overall percentage of coverage. The CA maintains a log of coverage for each entity, so at each instant it is possible to know the state of all entities and measure the total coverage or show the coverage trend along the time.
  - raising alert warnings for the entities that are not covered. A warning message indicates a condition that might need supervision and can be seen as a programming language exception. The warnings can be transmitted to the user in graphical form through the GUI or in textual form, such as an alert message. Moreover, a log file would report additional information like: the last time the entity was observed, the related sequence of events and parameters, etc.

– collecting the monitoring log for off-line analysis.

When in learning phase, CA simply logs the occurrence of entities to be covered and produces the statistics of the observed frequencies. These are referred for defining the observation windows in operation.

```
rule "Entity1"
  when
     list:ArrayList() from collect((EventMessage() over
          window:length(n)))
     not(EventMessage(EventName()=="eventx") from list)
  then
     Entity1.covered=false
  end
```

Listing 1: *Example rule*

Figure 2 shows the overall structure of the monitor and the data exchanged between the different components. The GUI is conceptually divided into two parts: Initialization and Visualization. The behavioral model is inserted by using the initialization part and passed to the RG. As an example in the figure we consider that the specification is provided as an FSM model and the criterion selected is branch coverage. The derived rules are sent to IE and in parallel WSM starts receiving the EventMessages. The icons drawn in the WSM component represent the *EnactmentID* while the labels *e1,..,e5* are the *EventName*s. For instance the male actor executes a sequence of activities in the composition that forces the events *e1, e2, e4*. These data are given to IE that recognizes the coverage of the corresponding branches, and notifies to CA the list of branches not exercised within their observation windows. This information is rendered to the users by means of the Visualization part in the GUI.

6. CASE STUDIES

In this section we introduce the two choreographies to which we applied our adequate monitor.

6.1 Travel Reservation System

The Travel Reservation System, as provided in the WSCI documentation [1], implements a business process for reserving and booking airline tickets and includes the following services: *Traveler*, which models the user of the Travel Reservation System; *Travel Agent*, which is in charge of managing the travel reservation process; *Airline*, which provides the functionalities for booking a selected flight. In order to have a more complete case study, we added the following services: *Hotel* and *Car Park*, which provide the functionalities for reserving and booking hotel rooms and parking spaces, respectively, and *Car Rental*, which allows to rent a car nearby the destination airport.

At very high level the considered business processes are the following: initially Traveler asks the Travel Agent for a flight itinerary, then the Travel Agent calculates the best solutions and asks the involved Airline to verify the seats availability. In case of positive response, the Travel Agent builds the itinerary and asks the Traveler to confirm the seats found or modify the original intention. In case of confirmation, the Travel Agent proceeds with the Airline. Once the flight is reserved the Traveler has different options: canceling the reservation, buying the flight ticket, or asking for additional services, i.e. rent a car, book a room or a parking space. When such negotiations are concluded Travel Agent notifies the selected trip to Traveler, who can again modify the solution or reserve it. When all the reservations are concluded the Traveler can call the Travel Agent that is in charge to confirm the requests with the involved services. These will then provide the Traveler with the appropriate responses. Finally the Traveler can cancel either the entire trip or only some of the required additional services by calling again the Travel Agent.

6.2 Future Market

The Future Market choreography is a real choreography, developed by the Distributed Systems Group at USP Computer Science Dpt. (http://gsd.ime.usp.br/), implementing a distributed on-line shopping service. As depicted in Figure 3, it consists of the following workflow: a customer provides in input a shopping list querying the price of each listed item to different on-line supermarkets, with the goal of finding the lowest price. Several services for subsequently purchasing and delivering the items from the multiple supermarkets are provided: the human customers that interact with the choreography through a Portal are represented as Customers; the Registry stores the list of services that implement the multiple supermarkets; in the version enacted, five different Supermarkets were registered that provide similar functionalities for searching and purchasing products; if a Supermarket is at shortage of some product, it can procure it either from a Supplier or directly from the Manufacturer; the Shipper is in charge of the delivery of acquired products to a postal address; and finally a Bank service supervises the on-line payments.

7. PRELIMINARY ASSESSMENT

In this section we report the results of applying the adequate monitoring to the two case studies. In particular, using the Travel Reservation System choreography we simulate various possible monitoring outcomes, whereas using the Future Market choreography we collect results from a real, even if less complex, application.

To assess an adequate monitor, we identified two critical concerns: the setting of the observation windows, since we foresee that these may have an important impact in the approach effectiveness, and understanding whether the observed variations in the coverage measure may be indeed useful to identify potential issues. Therefore, we used the case studies for investigating the following research questions:

RQ1: How does monitoring coverage vary depending on the observation windows?

RQ2: Can monitor coverage reveal potential issues?

7.1 Travel Reservation System

In the performed simulations we varied the combination of the following independent variables:

- *Coverage Criterion (CC)*: this parameter indicates the entities to be covered, i.e., CC$\in \{operations, branches\}$. The cardinality of operation coverage domain for the Travel Reservation System is 30, while that of branch coverage domain is 95;
- *Entity Frequencies (EF)*: we considered the possibility of manually annotating choreography model or to perform a learning phase, thus EF$\in \{annotation, learning\}$;

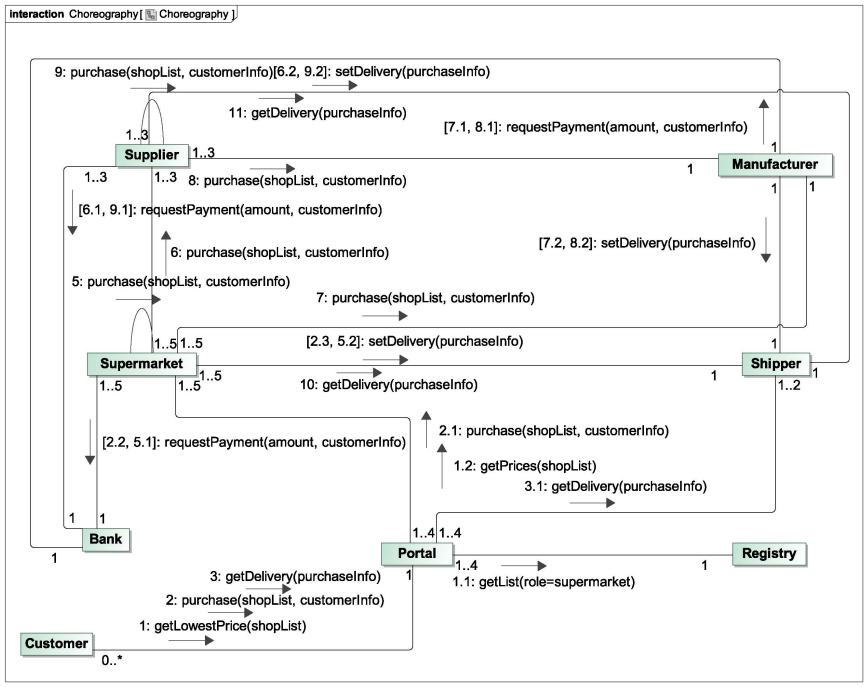

Figure 3: Future Market choreography

- *Confidence Level (CL)*: when EF = *learning*, we considered different confidence levels, with CL$\in \{90\%, 95\%, 99\%\}$;
- *Coverage Threshold (CT)*: this parameter indicates the coverage threshold to be considered adequate: we set CT$\in \{80\%, 85\%, 90\%\}$.

We launched various reservation requests from simulated clients, randomly choosing one possible complete path over the behavioural model of the Travel Reservation System. Thus considering a (discrete) time measurement unit, associated to the completion of a path in the behavioural model, we carried on the simulation for 50000 time units.

Observation Windows Analysis. Figure 4 summarizes in four frames the simulation results using different approaches for setting the entities windows. Each graph reports on the x-axis the time unit (t), and on the y-axis the percentage of cumulative entity coverage.

In particular, Figure 4 (a) (resp. (b)) shows a portion of the curves of the cumulative operation coverage considering the following setting of the independent variables: CC = operations (resp. branches), EF = learning, CL = 90% (95%, 99%), CT = 80% (85%, 90%). Since the lengths of each observation window δ_i have been obtained by statistical inference, the estimates are valid within a prefixed confidence level, as follows: the top level curves (for both operation and branch coverage) show an extract of the adequate monitoring considering CL=90%, those in the middle CL=95%, and those at the bottom CL=99%. In each graph, the dotted lines at 80, 85, and 90 correspond to the CT values.

The frame (c) (resp. (d)) in the same figure shows an extract of the curve of the cumulative operation (resp. branch) coverage considering the following setting of the independent variables: CC = operations (resp. branches), EF = annotation, CT = 80% (85%, 90%). Manually annotated frequencies should reflect some prior knowledge of the choreography behavior. However, we considered in this experiment the worst case of no prior knowledge and simply adopted a uniform distribution of the frequencies for the operations; for branches we considered as equally probable those at the same level in the graph.

From the comparison of the various curves in Figure 4, we can provide a preliminary answer to RQ1. A first, somewhat obvious, conclusion is that the more information we can use for setting the observation windows associated to the various entities, the better. Ideally we would like to observe a stable and high coverage throughout the choreography monitoring, except for problems, so having an oscillating coverage is not good. When EF = annotation (frames c and d) the coverage fluctuates rapidly, especially for operation coverage. This means that the provided annotations are based on wrong assumptions about the expected frequency of operation invocations, providing in the end a deceiving perception of possible choreography instabilities.

Concerning the case of EF = learning (frames a and b), the simulations with CL = 90% , 95%, 99% provide an increasing trend of both operation and branch coverage values. Considering for instance the frame (b) with CL=90% the branch coverage values are between 80% and 90%, with CL=95% the values are between 85% and 95%, when CL=99% the values are between 95% and 100%. This is in line with the higher confidence level adopted for setting the observation windows, which forces a consistent increase of the windows length. The consequence is that the adequate monitoring results less sensible to the choreography behaviors local variation and the perception provided by the monitor is that no anomalous situations are experienced.

To summarize, the ideal approach for setting the observation windows would be to exploit prior knowledge, when this

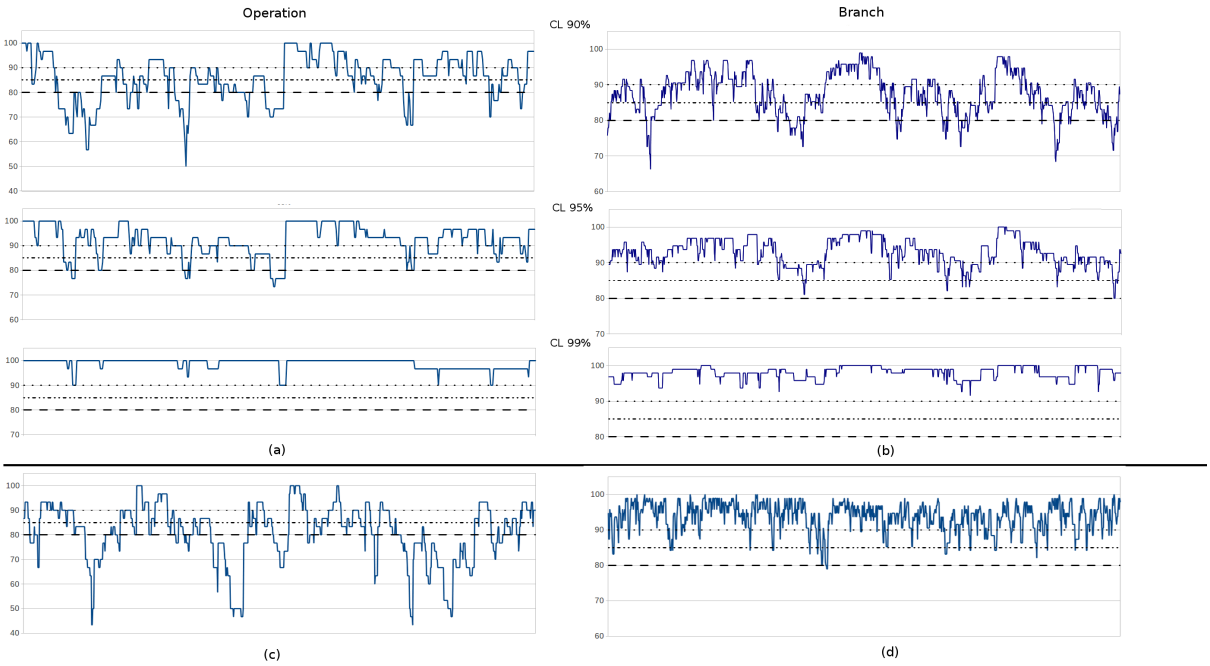

Figure 4: Coverage evaluation using learning data and prior knowledge (x axis is t and y axis is coverage)

faithfully represents the reality. When this knowledge is not available, a learning activity is a feasible compromise. The confidence level in learning phase directly influences the observation windows length: the higher is CL value, the longer will be the window; consequently, the higher is the chance to cover more entities but conversely the less timely will be the monitoring warnings.

Coverage Measure Analysis. Based on the above observations, the analysis of RQ2 has been carried out considering the following experimental setting: CC = operations (resp. branches), EF = learning, CL = 95%, CT = 90%.

Figure 5 (a) (resp. (b)) shows an excerpt of the graphical representations of the cumulative operation (resp. branch) coverage obtained using these settings. Focusing on the curves at the top, both on part (a) and part (b) the graph evidences several observation periods in which entity coverage decreases under the established boundary (90%).

For the choreography manager these variations of coverage measure could be a symptom of some anomalies that could require a deeper investigation. More importantly, the monitor not only gives these coverage cumulative value, but can also report detailed information on which entities are responsible for the coverage decrease. The bottom part of Figure 5 (a) (resp. (b)) shows pictorially the coverage of each entity in every choreography behavior execution. We draw a horizontal row for each entity, thus we have 30 rows for 30 operations in part (a) and 95 rows for 95 branches in part (b). Along each row, at instant t the point is green if the entity is covered, red otherwise[4]. Thus all red points are warnings of possible anomalous situations. The longer is a set of consecutive red points in a same row the more critical could be the problem. In detail a punctual analysis of the bottom parts of Figure 5 (a) (and (b)) shows that:

- There are entities that continuously alternate between green and red, e.g., the first lines of Figure 5 (b). This could evidence a situation in which the chosen observation windows are too short. In real contexts this could be due to several reasons: a not adequately long learning activity; a not accurate setting of the confidence level, or even some variations in the services behaviors.

- Conversely there are entities that are green almost all the period, e.g., the last few lines of Figure 5 (a). This could happen because the chosen observation windows are too long, however in general continuous green lines are not a problem.

- There are periods in which the coverage measure is under the established threshold (90%). The local decrease happens in situations in which several entities are not observed contemporaneously. This can be due to several reasons such for instance a modification in the expected users' behavior or a propagation of an anomaly to different services in the choreography. Whatever the causes, the local decrease of coverage can be an alarm of anomalous behavior in the choreography enactment.

- Even when the coverage measure is above the established threshold, some entities cannot be covered for long periods. These are evidenced by (long) red segments. The reasons of this situation could be in relation to serious malfunctioning of a services or important anomalies in the users' behaviors.

7.2 Future Market

In this section we report the results from adequate monitoring of Future Market. In this case study we considered both operation and branch coverage and set CT=90%. Due

[4]gray or black respectively in black&white printing

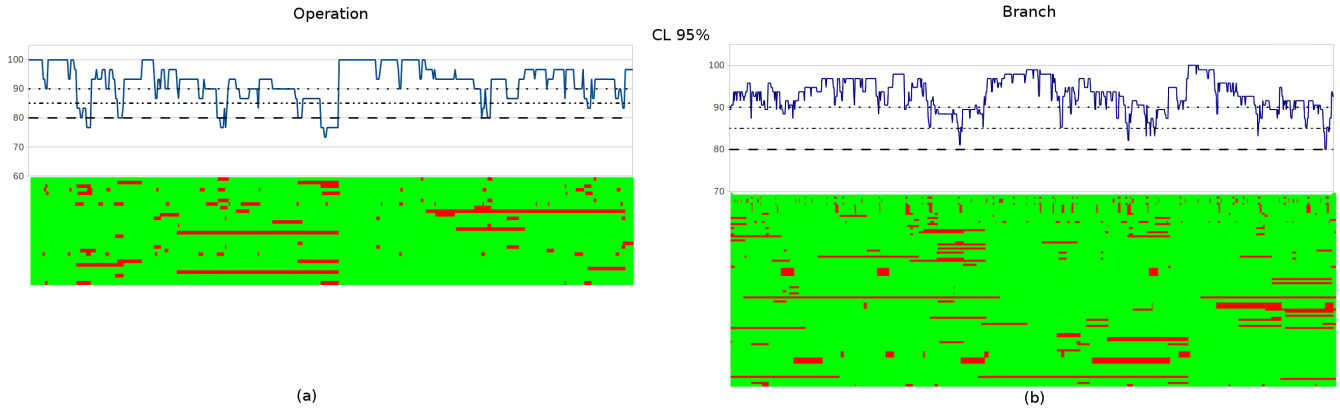

Figure 5: Adequate monitoring coverage of Travel Reservation System

to the simplicity of the choreography we opted for the prior knowledge approach to derive the observation windows, setting a uniform distribution of entity frequencies. We performed randomly exercised the choreography services.

Figure 6 (a) shows an excerpt of the percentage of branch coverage observed. We omit the curve relative to operation coverage because it always reached 100%. The curve evidences that the coverage is below the set threshold, and from the detailed report we see there are two continuous red lines corresponding to two branches that are never covered. Following such results, an analysis of the choreography specification revealed that due to how the configuration parameters are set, every time the Supermarket does not have a product available and asks the Supplier, this always passes the request to the Manufacturer. Indeed the stock amount for some products for Supermarket and Supplier was set to the same values, thus every time the Supermarket was at shortage of some product the Supplier did too and it was forced to ask to the Manufacturer. Thus the branch representing the case that the Supplier returns the product without interacting with Manufacturer never occurs.

We notice that to have some comparison data about the effectiveness of the adequate monitor, in parallel we also set up a standard QoS monitoring tool (the Glimpse monitor [6]). The data collected by the standard monitor provided for each service interaction the time duration, but did not revel any anomalous behavior, as in any case the choreography enactments proceeded smoothly to their completion.

We modified the settings by differencing the stock amount for Supermarket and Supplier and repeated the measurement using the new configuration. As shown in Figure 6 (b), the continuous red lines disappeared, and an overall increasing of percentage of branch coverage was observed.

This outcome confirms our answer to RQ2 in the previous case study that the monitoring coverage provides useful indications of potential issues that may escape a traditional monitor.

7.3 Caveat

The two above case studies suffer from many limitations and threats: the first is a simulation and may not be representative of real situations, the second is a real (third-party) choreography, but quite simple. The results we show are quite preliminary and must be taken as a first attempt to show the feasibility of the approach and confirm its potential usefulness. However a more extensive experimentation of adequate monitoring is required for better understanding its strengths and limitations.

8. RELATED WORK

Monitoring is an essential asset for controlling and managing distributed systems and since early works such as [22, 15, 18] the related literature spans over more than three decades. A relatively recent survey [10] underlined a renewed interest in monitoring approaches and tools, due to the increasing complexity and pervasiveness of software systems. Rise of interest is even more evident with the advent of the service-oriented paradigm, whose characteristics such as stringent QoS requirements, decentralized management, loose coupling and dynamic binding of independent services make runtime monitoring at the same time indispensable and highly challenging.

For lack of space we do not survey here the broad related work on more "traditional" challenges posed by service compositions monitoring, which are briefly summarized in, e.g., [25] and [12]. Rather, we focus here on proposals of "smart" monitoring approaches, i.e., monitors enhanced beyond the passive observation of system executions, with the aim of preventing or anticipating potential risks. It is understood that such smart approaches cannot neglect any of the traditional challenges, and hence need to rely on top of solutions to address those ones as well.

As a trend, several researchers start to consider monitoring a useful instrument to observe the behavior of a service composition not only to report about problems that have already occurred but also to predict likely problems in the near future. Several proposals go in the direction of checking the execution traces against a model of correct executions, and of introducing enhanced analysis capabilities to try to foresee possible deviations. This type of approach has been realized in different flavors. In [2], for example, the authors propose a choreography monitor for early detection of runtime faults, intended as deviations from admissible message exchanges between the cooperating services. The monitor maintains a snapshot of the multi-party conversations and, in case a failure occurs in an instance of the choreographed services, it analyzes the specification and decides whether the choreography execution can be completed. If not, it no-

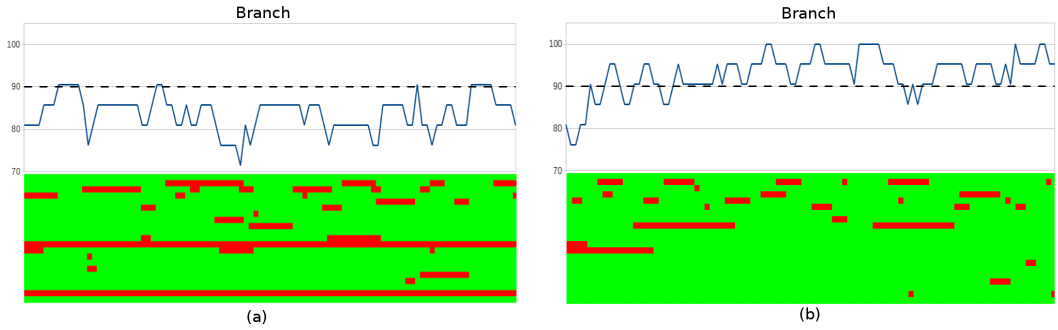

Figure 6: Adequate monitoring coverage of Future Market choreography

tifies the affected services. CASSANDRA [26] is a proactive monitor with the ability to map a services composition execution on a global interface automaton and to predict the future execution traces from the current state, thus identifying in advance the risk to reach erroneous states.

Other "smart" approaches focus on monitoring of non-functional properties to prevent QoS violations. The approach in [17] focuses on preventing QoS violations of explicitly defined SLAs. It uses machine learning to perform runtime prediction of SLA violations at defined checkpoints and, if necessary, triggers runtime adaptation. In both [3] and [14] the authors address the prediction of QoS violation problems in a service orchestration. In the ProAdapt framework [3], the monitor provides historical data of operations used in the instantiated execution models to an orchestrator that performs the runtime adaptation by selecting the concrete binding to services to be invoked in a composition. The approach in [14] focuses on cumulative QoS metrics and estimates the allowed QoS values for the remainder of the execution of the orchestration, starting from those cumulated QoS values. In [4], a monitoring approach is proposed for choreographies that calculates statically implicit QoS constraints and issues a warning when these are violated at runtime: as for an adequate monitor warning, a violation of an implicit QoS constraint could likely, but not necessarily, lead to the failure of a contractual constraint.

However, all the above mentioned predictive approaches remain limited to the elaboration of the passively captured executions. In contrast, in our approach the monitor is empowered also to raise a warning of *not* having observed for some time interesting behaviours. To the best of our knowledge there are not similar approaches in literature.

Another branch of related work is passive testing, which refers to the observation of the input/output behavior of a system during normal operation for the purpose of detecting faults [16]. Every collected trace is matched to a model (most often a Finite-State Machine model, or also an Extended FSM [8]) to verify if it conforms to the specified behaviour. The conformance check is done either off-line or on-line. Proposed initially for the management of network protocols for which active testing was impractical due to limited controllability of inputs, research in passive testing has been recently relaunched for the management of web services, e.g. [9, 5]. Although the two branches of literature on monitoring and passive testing have little overlap, we can speculate that passive testing constitutes a sophisticated type of functional monitoring, in which the monitor is instructed to observe the traces permissible in the model. As in passive testing, we also compare the collected traces against a model, however our purpose is not (only) that of detecting deviations, rather we aim at measuring coverage over the considered model. Thus, our approach could be considered as an enhanced form of passive testing.

9. CONCLUSIONS AND FUTURE WORK

In this paper we introduced a novel notion of monitoring adequacy for service compositions and defined two specific instantiations of a monitor adequacy criterion: operations and branches. We also developed a proof-of-concept adequate monitoring and provided a preliminary assessment of the framework on two case studies.

The obtained results showed the feasibility of the approach and evidenced that adequate monitoring can effectively overcome the limitation of a standard monitor. Indeed, the last passively reports what is happening and is not able to decide whether the collected data provide a thorough snapshot of the system potential behaviours. Adequate monitoring of service composition enhances standard monitors with a notion of coverage, and thus makes them capable to measure coverage of the relevant entities under a defined monitor adequacy criterion. However an important lesson learned was that monitoring coverage can fluctuate depending on the observation windows, thus attention is required in the choice of entity frequencies. In the Future Market case study we manually analysed the choreography specification to locate the cause for the observed anomalies.

In future work we intend to further augment the monitoring framework with the capability to trigger on-line test cases for automating this analysis, following the early Audition approach [7]. Also in [19, 24] the authors propose to combine service monitoring with on-line testing, however those works augment on-line testing with the information coming from monitoring, whereas we propose conversely to augment monitoring with the capability to launch the not-observed executions. A more extensive assessment of the approach to evaluate its costs and benefits is needed.

10. ACKNOWLEDGMENTS

This work is partially supported by the European Project FP7 IP 257178: CHOReOS. We thank Fabio Kon and Carlos Eduardo Moreira dos Santos of University of Sao Paulo, Brazil for the Future Market choreography.

11. REFERENCES

[1] Web service choreography interface (WSCI) 1.0. http://www.w3.org/TR/wsci/.

[2] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan. Monitoring choreographed services. In *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pages 283–288. Springer Netherlands, 2007.

[3] R. Aschoff and A. Zisman. QoS-driven proactive adaptation of service composition. In *ICSOC*, pages 421–435, 2011.

[4] C. Bartolini, A. Bertolino, A. Ciancone, G. De Angelis, and R. Mirandola. Apprehensive QoS monitoring of service choreographies. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13. ACM, 2013.

[5] A. Benharref, R. Dssouli, M. Serhani, A. En-Nouaary, and R. Glitho. New approach for EFSM-based passive testing of web services. In A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, editors, *Testing of Software and Communicating Systems*, volume 4581 of *Lecture Notes in Computer Science*, pages 13–27. Springer Berlin Heidelberg, 2007.

[6] A. Bertolino, A. Calabrò, F. Lonetti, and A. Sabetta. Glimpse: a generic and flexible monitoring infrastructure. In *Proceedings of the 13th European Workshop on Dependable Computing*, EWDC '11, pages 73–78, New York, NY, USA, 2011. ACM.

[7] A. Bertolino and A. Polini. The audition framework for testing web services interoperability. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 134 – 142, aug.-3 sept. 2005.

[8] A. R. Cavalli, C. Gervy, and S. Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Information & Software Technology*, 45(12):837–852, 2003.

[9] G. Dai, X. Bai, Y. Wang, and F. Dai. Contract-based testing for web services. In *Proceedings of the 31st International Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 517 –526, July 2007.

[10] N. Delgado, A. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30(12):859 – 872, dec. 2004.

[11] Drools Fusion: Complex Event Processor. `http://www.jboss.org/drools/drools-fusion.html`.

[12] C. Ghezzi and S. Guinea. Run-time monitoring in service-oriented architectures. In L. Baresi and E. Di Nitto, editors, *Test and Analysis of Web Services*, pages 237–264. Springer, 2007.

[13] J. Goodenough and S. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, SE-1(2):156 –173, 1975.

[14] D. Ivanovic, M. Carro, and M. V. Hermenegildo. Constraint-based runtime prediction of sla violations in service orchestrations. In *ICSOC*, pages 62–76, 2011.

[15] J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, 1987.

[16] D. Lee, A. Netravali, K. Sabnani, B. Sugla, and A. John. Passive testing and applications to network management. In *Proceedings of the International Conference on Network Protocols*, pages 113 –122, 1997.

[17] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, prediction and prevention of sla violations in composite services. *Proceedings of the IEEE International Conference on Web Services*, 0:369–376, 2010.

[18] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. In *Network and distributed systems management*, pages 303–347. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[19] 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, pages 20–28, New York, NY, USA, 2010. ACM.

[20] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability*. McGraw-Hill New York, 1987.

[21] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[22] B. Plattner and J. Nievergelt. Special feature: Monitoring program execution: A survey. *Computer*, 14(11):76 –93, nov. 1981.

[23] S. Rapps and E. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-11(4):367–375, 1985.

[24] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl. Usage-based online testing for proactive adaptation of service-based applications. In *Proceedings of the IEEE 35th Annual Computer Software and Applications Conference (COMPSAC)*, pages 582–587, July 2011.

[25] M. Winkler, J. Cardoso, and G. Scheithauer. Challenges of business service monitoring in the internet of services. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 613–616, New York, NY, USA, 2008. ACM.

[26] P. Zhang, H. Muccini, A. Polini, and X. Li. Run-time systems failure prediction via proactive monitoring. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE'11, pages 484–487, Washington, DC, USA, 2011. IEEE Computer Society.

[27] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, pages 366–427, 1997.