Chapter 7
Mining Lifecycle Event Logs for Enhancing Service-Based Applications

Schahram Dustdar
Vienna University of Technology, Austria

Philipp Leitner
Vienna University of Technology, Austria

Franco Maria Nardini
ISTI-CNR, Pisa, Italy

Fabrizio Silvestri
ISTI-CNR, Pisa, Italy

Gabriele Tolomei
ISTI-CNR, Pisa, Italy

ABSTRACT

Service-Oriented Architectures (SOAs), and traditional enterprise systems in general, record a variety of events (e.g., messages being sent and received between service components) to proper log files, i.e., event logs. These files constitute a huge and valuable source of knowledge that may be extracted through data mining techniques. To this end, process mining is increasingly gaining interest across the SOA community. The goal of process mining is to build models without a priori knowledge, i.e., to discover structured process models derived from specific patterns that are present in actual traces of service executions recorded in event logs. However, in this work, the authors focus on detecting frequent sequential patterns, thus considering process mining as a specific instance of the more general sequential pattern mining problem. Furthermore, they apply two sequential pattern mining algorithms to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, i.e., VRESCo. The obtained results show that the authors are able to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at design-time, when service-based application developers could be provided with service recommendation tools that are able to predict and thus to suggest next services that should be included in the current service composition.

DOI: 10.4018/978-1-4666-2089-6.ch007

INTRODUCTION

The vast majority of nowadays software-based systems, ranging from the simplest, i.e., small-scale, to the most complex, i.e., large-scale, record massive amounts of data in the form of logs. Such logs could either refer to the functioning of the system as well as keep trace of any possible software or human interaction with the system itself. For this reason, logs represent a valuable source of hidden knowledge that can be exploited in order to enhance the overall performances of any software-based system.

Well-known examples of systems that have started trying to improve their performances by analyzing event logs are surely Web Search Engines (SEs).

Roughly, SEs are increasingly exploiting past user behaviors recorded in query logs in order to better understand people search intents, thus, for providing users with better search experiences. Indeed, by accurately recognizing and predicting actual user information needs, SEs are now able to offer more sophisticated functionalities (e.g., query suggestion) as well as better relevant result sets in response to a specific query (e.g., query diversification).

Moreover, there are plenty of modern enterprise software systems that need to operate in highly dynamic and distributed environments in a standardized way. Such systems implement their business logic according to the Service-Oriented Architecture (SOA) principles, thus, assembling their business processes as the composition and orchestration of autonomous, protocol-independent, and distributed logic units, i.e., software services.

Service-based systems and applications (SBAs) require proper run-time environments where their composing services can be searched, bound, invoked, monitored and managed. Therefore, SBA's run-time support might keep track of what is going on during the whole application lifecycle by roughly recording all such events to log files, i.e., service event logs.

Analysis of such service event logs could reveal interesting patterns, which in turn might be exploited for improving the overall performances of SOA's run-time frameworks as well as supporting SBA designers during the whole application lifecycle.

The main contribution of this work concerns the application of data mining techniques to a real-life service event log collected by the VRESCo SOA run-time framework. Our aim is to analyze the historical events stored on VRESCo in order to discover software services that are frequently invoked and composed together, i.e., process mining.

Although traditional process mining refers to a set of techniques and methodologies whose aim is to distill a structured process description from a set of actual traces of executions recorded in event logs, here we treat it as an instance of the sequential pattern mining problem.

The remaining of the work is structured as follows. First, we start describing background concepts and past work that somehow concerns with service event log analysis. Section 1 describes the information collected by SOA lifecycle event logs, in particular focusing on the VRESCo run-time framework. In Section 2, we propose how VRESCo event log may be analyzed for approaching our research challenge. Therefore, Section 3 shows the experiments we conduct on a real VRESCo log data set. Finally, we summarize the contributions we provide in this work together with any further idea that could be better investigated as future work.

BACKGROUND AND RELATED WORK

In this work, we present a use case for event log mining in service-based systems. This idea bears some resemblance to the established idea of business activity management (BAM) (Kochar, 2005). BAM considers the event-driven governance of business processes, and is, hence, mostly a term from the business domain. Technically, BAM is enabled by monitoring runtime of services and their interactions within company SOAs. To this end, event-based monitoring approaches (Zeng, Lei, & Chang, 2007, Baresi, Guinea, Pistore, & Trainotti, 2009, Wetzstein, Strauch, & Leymann, 2009) produce a steady stream of low-level lifecycle events, similarly to the lifecycle events discussed in Section 1.2 and to the event logs produced by VRESCo. These low-level events need to be aggregated so that real business information can be gained from them. Existing techniques to do this include SLA aggregation (Unger, Leymann, & Scheibler, 2008) or event-based SLA monitoring (Sahai, Machiraju, Sayal, van Moorsel, & Casati, 2002, Michlmayr, Rosenberg, Leitner, & Dustdar, 2009). Related to the ideas of BAM is research work by (Mulo, Zdun, & Dustdar, 2008), which considers event-based monitoring of business compliance. Our research, specifically mining for invocation sequences that lead to failed service invocations, is complementary to BAM. While BAM is mostly concerned with discovering failures, our research can be used to identify or predict them in advance.

In literature, events emitted by service-based applications have found various other uses. For instance, (Michlmayr, Rosenberg, Leitner, & Dustdar, 2008b) uses eventing information (along with various other data sources) to generate visualizations of the past behavior and quality of Web services, mostly to ease management and selection of services for other uses. This is related to our work, which identifies services which have in the past been used together, mostly to suggest suitable services for new uses. (Zeng, Lingenfelder, Lei, & Chang, 2008, Leitner, Wetzstein, Rosenberg, Michlmayr, Dustdar, & Leymann, 2009) have used SOA events to generate predictions of SLA violations. This is done by training machine learning models from the collected event data, and using runtime event information as input to those models.

Finally, our research is related to the idea of process mining (van der Aalst, 2004, van der Aalst, Weijters, & Maruster, 2004, van der Aalst, Dongen, Günther, Rozinat, Verbeek, & Weijters, 2009). Generally speaking, process mining considers discovering structures (mostly business processes) from traces of earlier executions of information systems. This also includes making implicit processes (that people are subconsciously following) explicit, so that they can be optimized using the techniques of business process reengineering. Our work is different to process mining in the sense that we do not suggest new processes from event logs. Instead, we rather give recommendations which combinations of services have in the past been used together (indicating that it might make sense to use them in combination).

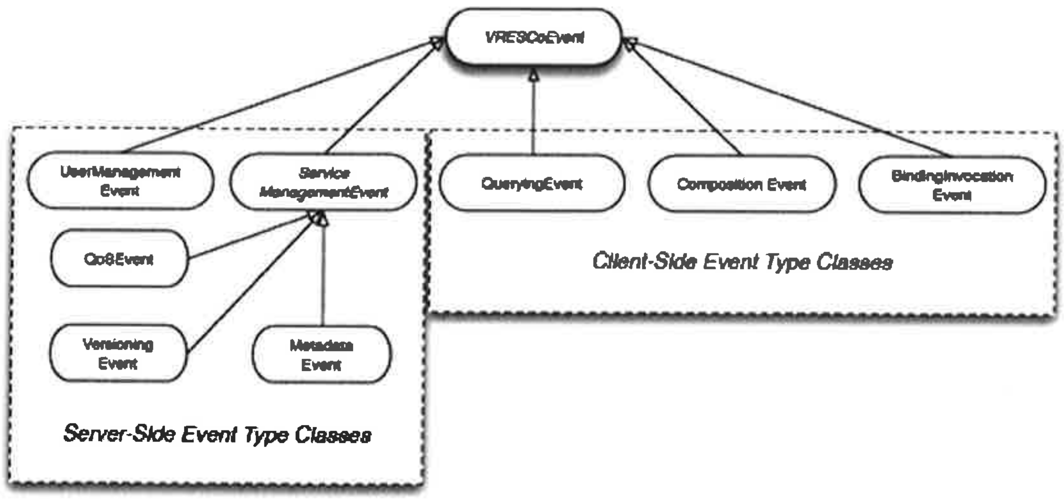
SOA LIFECYCLE EVENT LOGS

In the following, we will discuss SOA lifecycle event logs as they are used in this work. We then present the lifecycle events emitted by the VRESCo SOA runtime environment as a concrete example used in Section 2 and Section 3 of this work.

Lifecycle Events

Events and complex event processing (CEP) (Luckham, 2001) are frequently used tools to document and track the lifecycle of applications in various domains. For instance, in the business domain the idea of business activity monitoring (BAM) (Kochar, 2005) uses events to monitor business process performance. Analogously, technical implementations of business processes on top of SOAs (service compositions) are often monitored using CEP. To this end, many service composition engines can be configured to track their current state in event logs. For instance, the Apache ODE WS-BPEL engine triggers a rich model of execution events (http://ode.apache.org/

Figure 1. VRESCo event type classes



ode-execution-events.html). Similarly, service compositions implemented using Windows Workflow Foundation can use the .NET tracking service to persist event logs (http://msdn.microsoft.com/en-us/library/ms735887(v=vs.85).aspx). However, tracking system state via event logs in SOA is not confined to composition engines. For instance, The Vienna Runtime Environment for Service-Oriented Computing (VRESCo) (Michlmayr, Rosenberg, Leitner, & Dustdar, 2010) uses events to track not only service compositions, but all entities and interactions in a SOA (services, users, compositions, metadata and interactions).

In its most general form, an event log E consists of a sequence of n recorded events, i.e., $E = <e_1, e_2, ..., e_n>$. Each event e_i in E usually contains at least a unique identifier, an event timestamp, the publisher of the event (e.g., the BPEL engine), the subject of the event (e.g., the composition instance that triggered the event), and the event type. Depending on the concrete event type, more detailed information is available. This type-specific information cannot be described generally, i.e., it is different from event type to event type as well as from system to system. In the following we describe the event types triggered by the VRESCo system as an example of the possibilities provided by event logs.

SOA Event Log: VRESCo

VRESCo is an experimental runtime environment developed at Vienna University of Technology. VRESCo is being developed under an open source license, and can be accessed via the project Web page (http://www.infosys.tuwien.ac.at/prototypes/VRESCo/). The project aims at solving some of the research problems identified in (Papazoglou, Traverso, Dustdar, & Leymann, 2007), e.g., dynamic selection of services based on Quality-of-Service (QoS), dynamic rebinding and service composition, service metadata and event-based services computing.

In the following we focus on the latter aspect. The foundations of event-based service-oriented computing have been discussed in (Michlmayr, Rosenberg, Leitner, & Dustdar, 2008a). In a nutshell, the goals of this earlier work were to track

Table 1. Client-triggered VRESCo events

Event Type Class	Event Type	Event Condition	Additional Event Information
BindingInvocation Event	ServiceInvokedEvent	Specific service is invoked	Message sent to service, Invoking user
	ServiceInvocationFailed Event	Service invocation failed	Message sent to service, Triggered fault
	ProxyRebindingEvent	Service proxy is (re-)bound to a new service	Old service, New service
QueryingEvent	RegistryQueriedEvent	Registry is queried	Query string
	ServiceFoundEvent	Specific service is found by a query	Query string, query results
	NoServiceFoundEvent	No services are found by a query	Query string

what is going on in a service-based application by constantly triggering events and using CEP to construct meaningful information from those events.

In a VRESCo system, events of various types are triggered. A simplified taxonomy of event type classes is depicted in Figure 1. As can be seen, events are triggered when services are queried, bound and invoked. Additionally, events indicate if the data or metadata about services changes (e.g., the QoS is changed, new operations are available). Each of the concrete event type classes (those with non-italic name) in turn contains a number of concrete event types that can be triggered. For full details on all events refer to (Michlmayr, Rosenberg, Leitner, & Dustdar, 2008a).

Events in VRESCo can be triggered either on client- or server-side. While events concerning metadata are triggered by the VRESCo server, all querying and invocation events are triggered by clients and only processed throughout the VRESCo event engine. These client-triggered events are listed in more detail in Table 1. In the table, we provide the condition that triggers each event along with the event type and event type class of the event. All of these events provide the basic information discussed above (sequence number, timestamp, etc.). In addition, events generally provide some type-specific additional information, which we also summarize in the table. For reasons of brevity, we have omitted composition events. Composition events in VRESCo are of comparable expressiveness as the events triggered by Apache ODE.

The VRESCo event engine stores triggered events in an event log. Therein, events are serialized as XML and can be accessed and analyzed via a RESTful service interface. In Figure 2, we provide an example event serialized to XML. Evidently, the event reflects a service invocation with a very simple input message (<order>) as payload.

LIFECYCLE EVENT LOG MINING

In this work, we are interested in exploring how event log collected by the VRESCo framework, i.e., service event log, could be harnessed for better supporting service-based applications during their whole lifecycle.

Figure 2. Serialized invocation event

```
<ServiceInvokedEvent
   xmlns=" http://www.vitalab.tuwien.ac.at/vresco/usertypes">
  <Priority>0</Priority>
  <Publisher>guest</Publisher>
  <PublisherGroup>GuestGroup</PublisherGroup>
  <UserName>a007b09b-8c23-4fac-af30-0142a61f3795</UserName>
  <SeqNum>74006756-64f1-40cb-858e-565d4bc6a94c:24</SeqNum>
  <Timestamp>2010-11-09T09:58:49</Timestamp>
  <CurrentRevisionId>180</CurrentRevisionId>
  <CurrentRevisionWsdl>
    http://localhost:60000/AssemblingPlanningService?wsdl
  </CurrentRevisionWsdl>
  <FeatureName>GetPartFeature</FeatureName>
  <InvocationInfo>
    <service_input>
      <order><part1>text</part1></order>
    </service_input>
  </InvocationInfo>
</ServiceInvokedEvent>
```

Roughly, analysis of VRESCo event log data is finalized to the discovery of sequences of services that are frequently invoked together, thus, to detect processes, or part of those, as result of the compositions of highly co-invoked services.

This knowledge could be useful for improving the overall performances of the VRESCo run-time framework as well as supporting SBA designers during the whole application lifecycle. As an example, service-based application developers could be provided with service recommendation tools that are able to predict and, thus, to suggest next services that should be included in the current service composition at design-time.

Due to the huge amount of data collected in the VRESCo event log, data mining techniques represent a suitable approach for addressing our research challenge. In the following, we describe how processes may be mined from the VRESCo event log, namely how sequences of co-invoked services that frequently appear in the event log may be discovered and exploited.

Process Mining

According to van der Aalst *et al.* (van der Aalst, Weijters, & Maruster, 2004), the term process mining, also referred to as workflow mining, describes a set of techniques and methodologies whose aim is to distill a structured process description from a set of actual traces of executions recorded in event logs.

In our vision, a service log might be viewed as a database consisting of sequences of events that change with time, i.e., a time-series database (Han, & Kamber, 2006). Such kind of database records the valid time of each data set. For example, in a time-series database that records service invocation transactions, each transaction includes the unique identifier of the invoked service as well as an extra time-stamp attribute indicating when the event happened (Zhao, & Bhowmick, 2003).

Several kinds of patterns can be extracted from various types of time-series data. In this work, we are interested in finding sequences of services that are frequently invoked together in a specific order, i.e., sequential patterns (Agrawal, & Srikant, 1995).

Each process instance recorded on event logs might be expressed as an unrolled trace of invoked services. Thus, let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of services and let $S_j \subseteq S$ be an itemset of services invoked at the same time (or within a small time window), i.e., $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$. Therefore, a process $p_j = <S^j_1, S^j_2, \ldots, S^j_{|pj|}>$ has a unique identifier and represents a sequence of service itemsets, chronologically-ordered according to their time-stamps. Globally, a process database is a sequence database $P = \{p_1, p_2, \ldots, p_n\}$ of executed and recorded processes.

Roughly, a sequential pattern is a sequence of service itemsets that occur frequently in P according to a specific order, i.e., appear as subsequence of a large percentage of sequences of P. More formally, a sequence $p' = <S'_1, S'_2, \ldots, S'_u>$ is a subsequence of $p'' = <S''_1, S''_2, \ldots, S''_v>$, i.e., $p' \ll p''$ if there exists integers $1 \leq i_1 < \ldots < i_u \leq u$, such that $S'_x \subseteq S''_{ix}$ for each $1 \leq x \leq u$.

Then we may define the support $supp(p')$ of a sequence of service itemsets p' as the proportion of processes in the database P that contains p' as its subsequence, that is:

$$supp(p') = |\{p_j \mid p' \ll p_j\}| / |P| .$$

Therefore, sequential pattern mining is the process of extracting certain sequential patterns whose support exceed a predefined minimal support threshold *min_supp*.

To this end, as a first approach we apply one of the more efficient sequence pattern mining algorithm to the VRESCo event log, namely *PrefixSpan* (Pei, Han, Mortazavi-Asl, & Pinto, 2001).

Furthermore, we extend our first approach in order to exploit the temporal information associated with each service invocation in a different way. Indeed, in traditional sequential pattern mining, event time-stamps are only used for establishing the chronological order between service invocations, i.e., for simply stating that service s_i is invoked before service s_j. However, observing that s_i and s_j are invoked really closed to each other, e.g., within 5 seconds, rather than noting that s_i and s_j are farther away from each other, e.g. 5 minutes, could lead to different conclusions. Thus, we apply another sequential pattern mining algorithm, which is able to deal with this issue, i.e., *MiSTA* (Giannotti, Nanni, Pedreschi, & Pinelli, 2006).

Finally, in Section 3, we describe the different results we obtain on the VRESCo event log when using the two approaches described above.

EXPERIMENTATION

In order to test our claim about finding frequent sequential patterns inside service event logs, we use a real-life log of events collected by the VRESCo runtime framework.

This event log consists of 89 transactions. Each transaction is in turn composed of several events according to the ones described in Table 1 (e.g., *ServiceInvokedEvent*, *ServiceInvocationFailedEvent*, etc.).

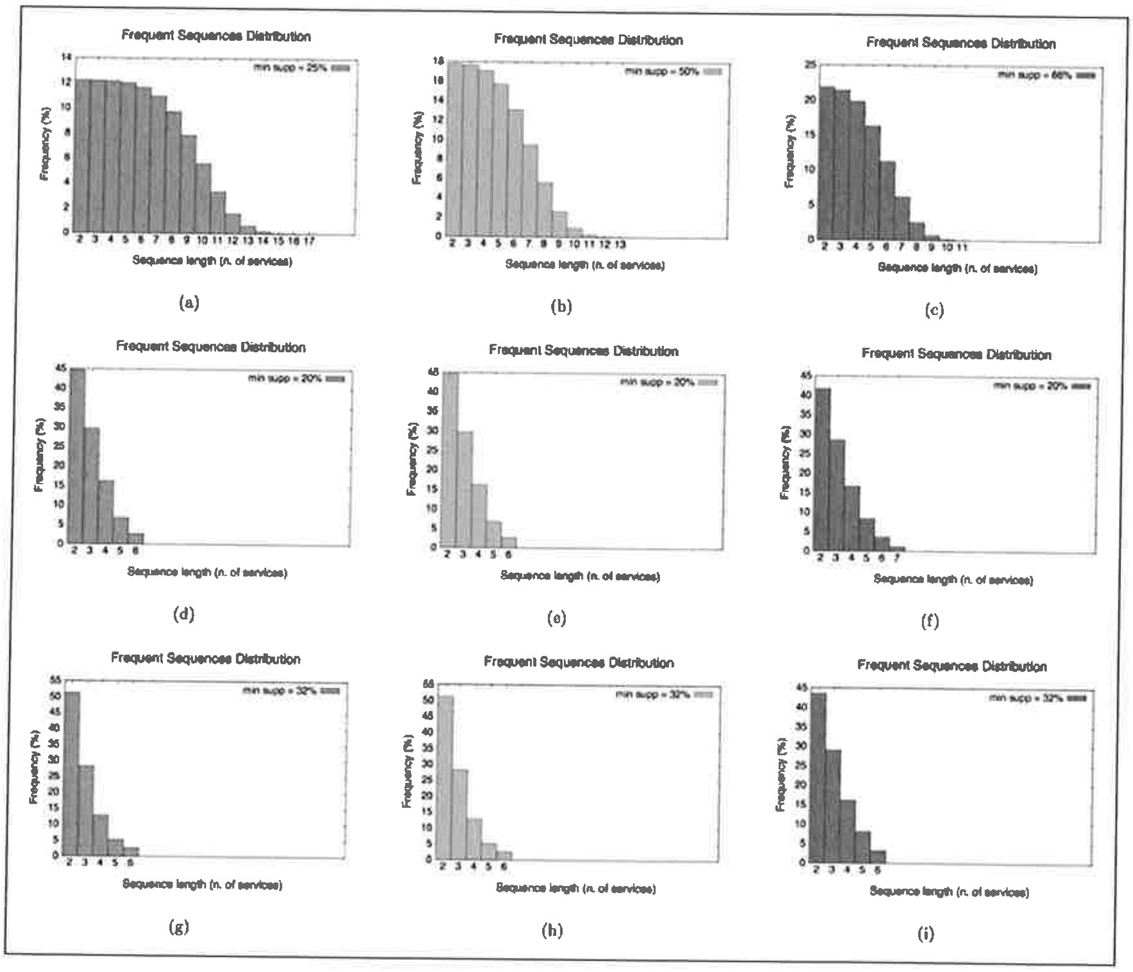
For the sake of our purposes, namely for discovering sequences of services that frequently are invoked together, we only consider the list of *ServiceInvokedEvent* for each transaction.

Firstly, we run the *PrefixSpan* algorithm (Pei, Han, Mortazavi-Asl, & Pinto, 2001) on the VRESCo data set. We use several thresholds on the minimum support of the sequential patterns to be extracted, ranging from 20% to 100%. However, the maximum support for frequent sequences found in our data set is 66%.

The following table shows the distribution of the lengths of extracted sequence patterns, i.e., the number of invoked services that are inside a frequent sequence. In particular, Table 2 (a), (b), and (c) show the results obtained by using a minimum support threshold of 25%, 50%, and 66%, respectively.

As one would expect, independently from the minimum support chosen, 2-length sequences are

Table 2. Statistics about the ground-truth data



the most popular since for each k-length frequent sequences ($k > 2$) any i-length subsequence, i.e., $2 \leq i < k$, is also frequent by definition. Moreover, as the minimum support threshold increases, the maximum length of frequent sequences decreases as well from 17 to 11, and, also, most popular frequent sequences result to be globally shorter.

Finally, on average frequent sequences are composed of 5.86, 4.60, and 4.07 services for minimum support values of 25%, 50%, and 66%, respectively.

As a second step of our experimental phase, we also run the *MiSTA* algorithm (Giannotti, Nanni, Pedreschi, & Pinelli, 2006) on the VRESCo event log. This algorithm differs from classical sequential pattern mining algorithms like *PrefixSpan* because it also takes care of the time gaps between consecutive items in a sequence. In other words, *MiSTA* extracts frequent sequences by considering not only the *minimum support* but also another parameter, i.e., *tau*, which basically represents a threshold on the time-gap between pairs of consecutive items.

In our experiments, we use several combinations of such two parameters, namely *min_supp* and *tau*. In the following, we describe the distribution of frequent sequence size obtained by using two values for the minimum support, i.e., 20% and 32%, and three values for the time threshold *tau*, i.e., 5, 60, and 300 seconds, which are shown in Table 2 (d), (e), (f), (g), (h), and (i), respectively.

The total number of frequent sequences found when min supp = 20% is 33, both for *tau* = 5 and *tau* = 60, whereas it reaches the value of 35 when *tau* = 300. Moreover, when min supp = 32% the total amount of frequent sequences is 20, both for *tau* = 5 and *tau* = 60, whereas it reaches the value of 27 when *tau* = 300.

Finally, the maximum size of frequent sequences is 7, whereas the average ranges from a minimum of 2.80 to a maximum of 3.07. These results show that, by taking into account the time gap between service invocations using *MiSTA* instead of *PrefixSpan*, we are able to detect less and shorter frequent sequences on average.

We argue that *MiSTA* could provide better results if we previously analyze how time gaps are distributed across consecutive service invocations, and we leave this as a possible future work.

CONCLUSION AND FUTURE WORK

Service-Oriented Architectures (SOAs), and traditional enterprise systems in general, record a variety of events (e.g., messages being sent and received between service components) to proper log files, i.e., event logs. These les constitute a huge and valuable source of knowledge that may be extracted through data mining techniques.

In this work, we focus on process mining as a specific instance of the more general sequential pattern mining problem. Basically, our aim is to detect frequent sequential patterns that might be present in actual traces of service executions recorded in event logs. To this end, we apply two sequential pattern mining algorithms to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, i.e., VRESCo.

The obtained results show that we are able to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at design-time, when service-based application developers could be provided with service recommendation tools that are able to predict and thus to suggest next services that should be included in the current service composition.

Finally, as a future work, we aim at discovering possible sequences of invoked services, which frequently lead to failures or unexpected behaviors. This knowledge could be exploited either for preventing SBA designers to deploy possible faulty service compositions as well as for devising novel run-time adaptation mechanisms in response to undesired events.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *In Proceedings of the IEEE Conference on Data Engineering* (pp. 3-14).

Baresi, L., Guinea, S., Pistore, M., & Trainotti, M. (2009). Dynamo + Astro: An integrated approach for BPEL monitoring. In *Proceedings of the IEEE International Conference on Web Services* (pp. 230-237).

Giannotti, F., Nanni, M., Pedreschi, D., & Pinelli, F. (2006). Mining sequences with temporal annotations. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 593-597).

Han, J., & Kamber, M. (Jim Gray, Series Editor). (2006). *Data mining: Concepts and techniques*. Morgan Kaufmann Publishers.

Kochar, H. (2005). *Business activity monitoring and business intelligence*. Retrieved from http://www.ebizq.net/topics/bam/features/6596.html

Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., & Leymann, F. (2009). Runtime prediction of service level agreement violations for composite services. In *Proceedings of the Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing* (pp. 176-186).

Luckham, D. C. (2001). *The power of events: An introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc.

Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2008a). Advanced event processing and notifications in service runtime environments. In *Proceedings of the International Conference on Distributed Event-Based Systems* (pp. 115-125).

Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2008b). Selective service provenance in the VRESCo runtime. *International Journal of Web Services Research, 7*(2), 65–86.

Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2009). Comprehensive QoS monitoring of web services and event-based SLA violation detection. In *Proceedings of the International Workshop on Middleware for Service Oriented Computing* (pp. 1-6).

Michlmayr, A., Rosenberg, F., Leitner, P., & Dustdar, S. (2010). End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo. *IEEE Transactions on Service Computing, 3*, 193–205.

Mulo, E., Zdun, U., & Dustdar, S. (2008). Monitoring web service event trails for business compliance. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications* (pp. 1-8).

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

Pei, J., Han, J., Mortazavi-Asl, B., & Pinto, H. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of IEEE Conference on Data Engineering* (pp. 215-224).

Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A. P. A., & Casati, F. (2002). Automated SLA monitoring for web services. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management* (pp. 28-41).

Unger, T., Leymann, F., Leymann, F., & Scheibler, T. (2008). Aggregation of service level agreements in the context of business processes. In *Proceedings of the IEEE Enterprise Distributed Object Conference* (pp. 43-52).

van der Aalst, W. (2004). Process mining: A research agenda. *Computers in Industry, 53*(3), 231–244.

van der Aalst, W., Dongen, B. F. V., Gunther, C., Rozinat, A., Verbeek, H. M. W., & Weijters, A. J. M. M. (2009). ProM: The process mining toolkit. *Industrial Engineering (American Institute of Industrial Engineers), 489*, 1–4.

van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1128–1142.

Wetzstein, B., Strauch, S., & Leymann, F. (2009). Measuring performance metrics of WS-BPEL service compositions. In *Proceedings of the International Conference on Networking and Services* (pp. 49-56).

Zeng, L., Lei, H., & Chang, H. (2007). Monitoring the QoS for web services. In *Proceedings of the International Conference on Service-Oriented Computing* (pp. 132-144).

Zeng, L., Lingenfelder, C., Lei, H., & Chang, H. (2008). Event-driven quality of service prediction. In *Proceedings of the International Conference on Service-Oriented Computing* (pp. 147-161).

Zhao, Q., & Bhowmick, S. S. (2003). *Sequential pattern matching: A survey*. Retrieved from http://cs.nju.edu.cn/zhouzh/zhouzh.files/course/dm/reading/reading04/zhao_techrep03.pdf

KEY TERMS AND DEFINITIONS

Process Mining: Extract hidden knowledge from SOA event logs.

Sequential Pattern Mining: The general problem of extracting specific sequential patterns from datasets.

SOA Event Log: Log file recording messages sent and received between SOA components (i.e., services).

SOA: Service-Oriented Architecture.

Workflow Mining: Discover structured patterns of service interactions from SOA event logs.