Event Pattern Discovery for Cross-Layer Adaptation of Multi-cloud Applications

Chrysostomos Zeginis, Kyriakos Kritikos, and Dimitris Plexousakis

ICS-FORTH
Heraklion GR-70013, Greece
{zegchris,kritikos,dp}@ics.forth.gr

Abstract. As Cloud computing becomes a widely accepted service delivery platform, developers usually resort in multi-cloud setups to optimize their application deployment. In such heterogeneous environments, during application execution, various events are produced by several layers (Cloud and SOA specific), leading to or indicating Service Level Objective (SLO) violations. To this end, this paper proposes a meta-model to describe the components of multi-cloud Service-based Applications (SBAs) and an event pattern discovery algorithm to discover valid event patterns causing specific SLO violations. The proposed approach is empirically evaluated based on a real-world application.

Keywords: Cloud computing, SOA, adaptation, modeling, pattern discovery.

1 Introduction

Cloud computing is a rapidly emerging paradigm offering virtualized resources for developing applications. Its adoption in the Service Oriented Architecture (SOA) world is increasing; enterprises acknowledge its flexibility and elasticity by choosing among various offerings at all Cloud layers (IaaS, PaaS and SaaS). In addition, as developers try to optimize their application deployment cost and performance, they may also deploy application parts on multiple VMs [1].

In this paper we focus on SBAs deployed on Clouds, which feature three main functional layers: the Business Process Management (BPM) layer, the Service Composition and Coordination (SCC) layer and the Service Infrastructure (SI) layer. In a Cloud environment, the SI layer maps to the PaaS and IaaS layers, while the SaaS layer includes the BPM and SCC layers. It is imperative that such distributed hosting environments, exhibit efficient cross-layer monitoring and adaptation mechanisms combining multi-layer monitored events and mapping them to suitable adaptation strategies. In [2] we have investigated the need for cross-layer adaptation, as current techniques are mainly fragmented by considering a single SBA layer, while the few cross-layer ones [3, 4] do not consider multi-cloud aspects. Concerning monitoring, in [5] we have presented a multi-cloud SBA framework. This paper goes a step further supporting multi-cloud SBA adaptation by focusing on an efficient method for processing the

M. Villari et al. (Eds.) : ESOCC 2014, LNCS 8745, pp. 138–147, 2014.

huge amount of monitored events and discovering frequent patterns leading to SLO violations. As such, a pattern discovery algorithm is introduced, exploiting a component meta-model whose instances describe SBA component dependencies. The discovered event patterns interrelate events leading to SLO violations and can be further exploited to enrich the scalability rules defined by experts.

The rest of the paper is structured as follows. Section 2 provides a motivating example, while Section 3 analyzes the multi-cloud monitoring and adaptation framework. In Section 4 we describe the component meta-model, exploited by the proposed pattern discovery algorithm (Section 5). Section 6 evaluates the algorithm's accuracy and performance. Finally, Section 7 reviews the related work, before Section 8 concludes and provides future work directions.

2 Motivating Example

A traffic management SBA deployed in a multi-cloud setup motivates our approach. In a normal traffic scanario, four tasks occur: environmental variable monitoring (T_M), public event and high traffic hours checking (T_C), current condition assessment (T_A) and device configuration (T_D). Task T_A requires high computation and storage capabilities, while the other three tasks require moderate storage capacity and low computational power and must be deployed geographically close to the municipal infrastructure. Thus, these tasks can operate on a private/municipal Cloud with their data periodically sent for processing to a central Cloud, hosting T_A. As illustrated in Fig. 1, various events are detected by the monitoring mechanisms in this multi-cloud setting, from lower level infrastructure events (e.g. low memory ($(w)e_1$), high CPU_load ($(c)e_2$), network uptime ($(c)e_4$), to higher level events (e.g. service execution time ($(w)e_3, (w)e_6$ or throughput ($(w)e_5$) violations. The main non-functional application goal is to capture warning (identified by we) or critical events (identified by ce) and interrelate them to discover event patterns leading to SLO violations.

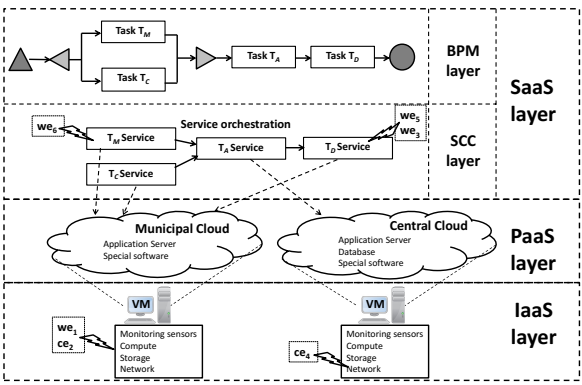


Fig. 1. Traffic management example

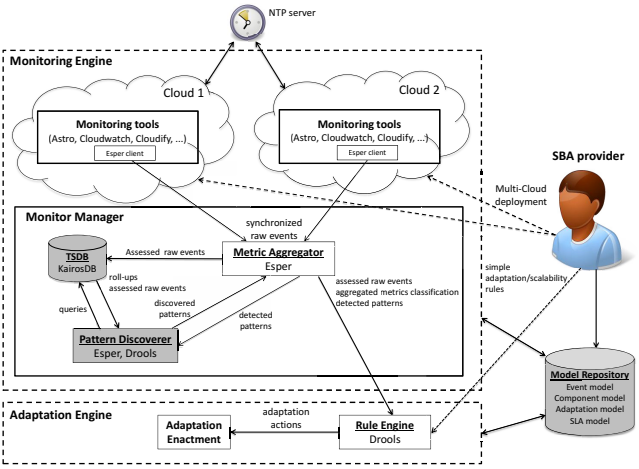


Fig. 2. Framework's architecture

3 Framework's Architecture

The architecture presented in our previous work [5] for cross-layer multi-cloud SBA monitoring is realized and enhanced with specific techniques and algorithms to support proactive adaptation. In this multi-cloud framework (Fig. 2 – the shadowed components indicate where this paper focuses), each of these Clouds exhibits monitoring components, which directly interact with the Monitor Manager, through a complex event processing (CEP) server-client mechanism. The *Metric Aggregator* component assesses and stores the monitored events to the time-series database (TSDB). The *Pattern Discoverer* component periodically queries the TSDB to get the assessed raw events for a specific time interval and identifies raw event patterns leading to SLO violations (mapping to specific aggregate metrics). The TSDB provides the aggregated metric values, necessary for pattern discovery. The discovered patterns are sent directly to the Metric Aggregator to detect them at runtime. Upon pattern detection, the Metric Aggregator urges the Rule Engine of the Adaptation Engine to fire the respective scalability rule (i.e. proactive adaptation) dictating the application of an adaptation strategy, realized by the Adaptation Enactment component (some adaptation actions are already realized, especially those mapping to scaling mechanisms provided by Cloud providers). Critical events are also passed to the Rule Engine to perform reactive adaptation.

4 Component Meta-model

A particular component meta-model (Fig. 3) was developed via UML to describe the source components for each event type which constitute the SBA system, as well as their dependencies. Its main benefits are that it is extensive to capture

the most common multi-cloud SBA components, related to functional and non-functional violations, and extensible to meet any SBA provider's needs, which may incorporate other layer-specific components utilized by its applications. Any adaptation manager can also exploit it to carefully design scalability rules based on the components' properties to stimulate the mapping from events to specific adaptation actions. Through capturing the component dependencies in a multi-cloud system, a root cause analysis for system faults can be performed. The Pattern Discovery Algorithm also exploits such dependencies to detect valid event patterns leading to critical events, where validity lies on causality by selecting events in the stream that drive the occurrence of other events in the pattern. Model-driven technologies of the Eclipse Modeling Framework (EMF) are exploited to create models complying to this meta-model. The core meta-model is graphically produced in ecore and then used to generate the base domain code. Next, the Connected Data Objects (CDO) technology is used, offering a model repository and a run-time persistence framework, accessible via querying mechanisms (SQL, HQL). For instance, the following SQL query returns the total number of active components included in the VM hosting the Monitor and DeviceConfig services (componentID=7001) and having ID 7026 or 7027 (i.e., the components producing e_1 and e_2 events), in order to identify if both components reside in the same VM and thus affect each other when a violation occurs.

```
SELECT COUNT(*)
FROM (SELECT VM_COMPUTE_LIST
     FROM VM WHERE ComponentID=7001) as computeList
WHERE componentID IN (7026,7027) AND (state = active);
```

5 Pattern Discovery

This section presents an offline algorithm for discovering event patterns leading to specific SLO violations, based on propositional logic [6]. The algorithm exploits component dependencies and contingency tables (Fig. 4) to identify association rules between events. These tables display frequency distributions of candidate patterns and their negations as antecedents and the specified metric event, as well as its negation as consequences. It mainly focuses on discovering patterns by considering SOA and Cloud layers, but it can also be applied on a single layer, discovering layer-specific patterns.

The algorithm starts by filtering the event stream (*line 3*) from events coming from different Cloud providers than those used by the SBA. Then, the event stream is split based on the aggregate metric's time interval (*line 4*). Intervals are characterized as critical, if there is an aggregate metric violation, or non-critical, otherwise (*line 5*). For critical intervals, the temporal ordered sets of the raw events subset's powerset are calculated, from the first interval event to that before the last critical raw measurement. All the sets of the interval powersets are filtered to discard the ones not interrelated with each other, based on the component model (*lines 6–11*), but the single events that might map to

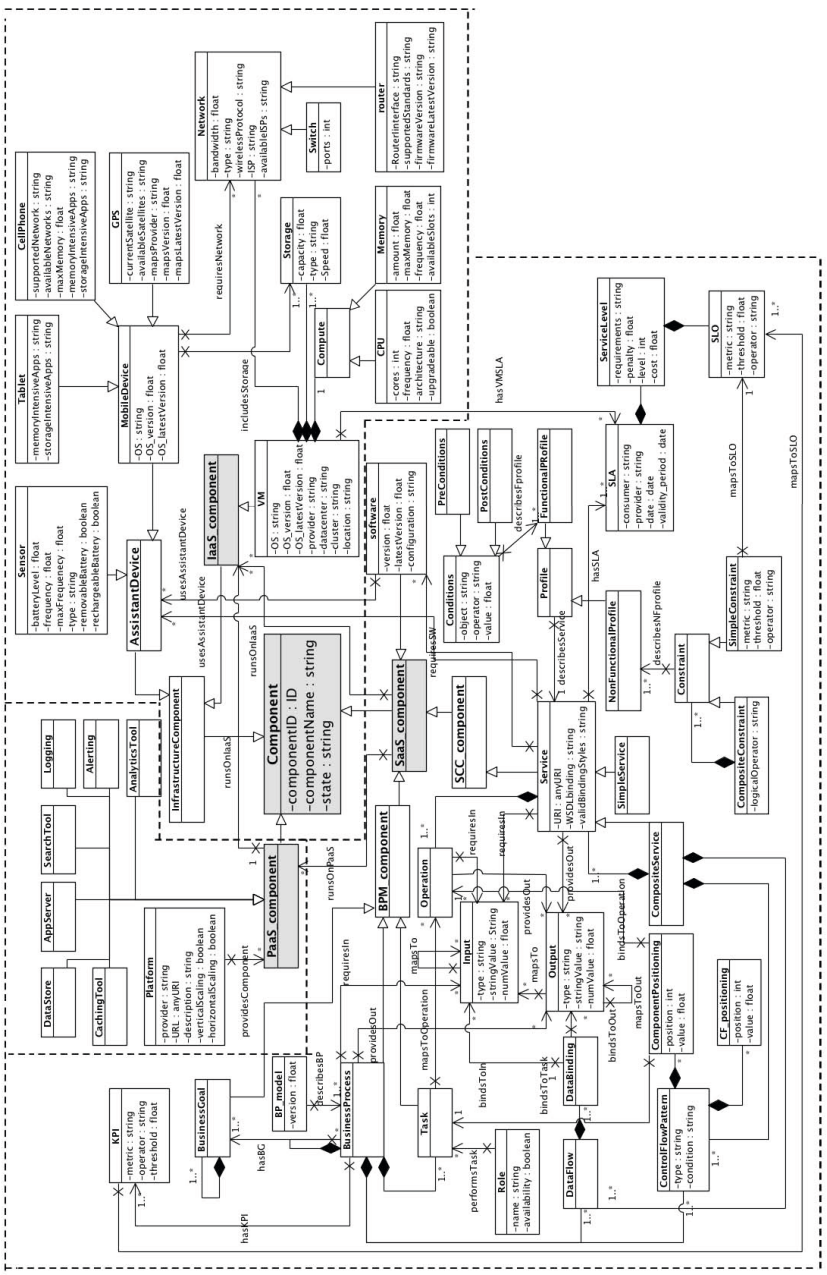


Fig. 3. The Cloud component meta-model

Algorithm 1. Event pattern discovery algorithm

```
1: Input: event stream, application, metric, interval size, component model
2: Output: discovered patterns, patterns ranking, association rules, ambiguous rules
3: filter raw events (ignore success /other applications'/other Cloud provider events)
4: divide event stream in event time intervals
5: define critical and non-critical intervals
6: while not end of event stream do
7:     A = events before the last critical raw event in this interval
8:     ℙ(A) → powerset of set A
9:     filter sets of ℙ(A) according to the component model
10:    update the powerset tree
11: end while
12: for i ← 1, treelevels do
13:    for j ← 1, treebranches do
14:        B_{i,j} = current branch
15:        while not end of event stream do
16:            A = events before the last critical raw event in this interval
17:            C = critical aggregate event for the specified metric
18:            compute S(B_{i,j}, C), S(B_{i,j}, ¬C), S(¬B_{i,j}, C), S(¬B_{i,j}, ¬C) in A
19:            update contingency table
20:        end while
21:        if (S(B_{i,j}, C) + S(¬B_{i,j}, ¬C)) > (S(¬B_{i,j}, C) + S(B_{i,j}, ¬C)) then
22:            create association rule (B_{i,j} → C)
23:            store B_{i,j} in pattern repository
24:        else
25:            discard B_{i,j}
26:        end if
27:    end for
28: end for
```

the critical violation are considered. Then, a level- and a branch-based traversal of the tree-based structure storing the candidate patterns are performed to calculate and store (*lines 15–20*) the frequencies for each considered set's contingency table. The powerset tree (Fig. 4) stores only unique sets and each node maps to a candidate pattern comprising all the events from the root to the current node. $B_{i,j}$ is the concerned sub-branch (i indicates the tree level and j the branch counter), C represents the aggregate metric violation, while the pair $S(B_{i,j}, C)$ is the frequency of $B_{i,j}$ set in critical intervals. A candidate set's negation means that either of the included events does not appear. For instance, for the pattern $\{e_1, e_2, e_4\}$, the following negation definition exists: $\neg\{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_4}\} \equiv \neg\mathbf{e_1} \vee \neg\mathbf{e_2} \vee \neg\mathbf{e_4}$. A negated event means that another or no event appears in the specific pattern's position. Frequencies $S(B_{i,j}, C)$ and $S(\neg B_{i,j}, \neg C)$ are used to determine an association rule. The latter invigorates the association rule under consideration, as the concurrent absence of a pattern and the considered violation event also interrelates the root and cause of the association rule. Consequently, to determine such a rule, the sum of frequencies

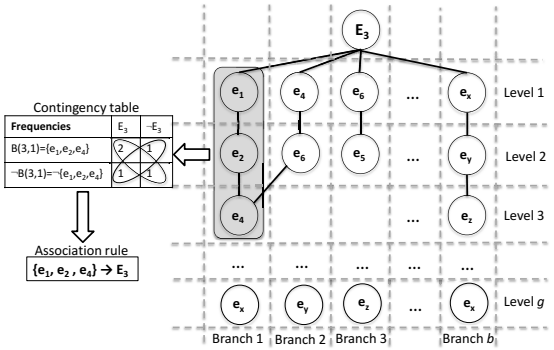


Fig. 4. Powerset tree

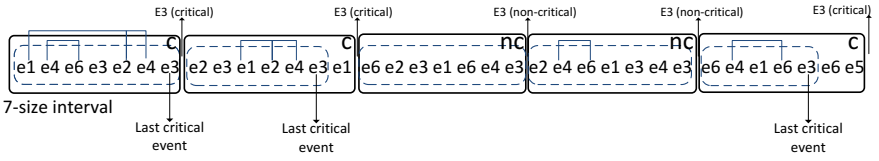


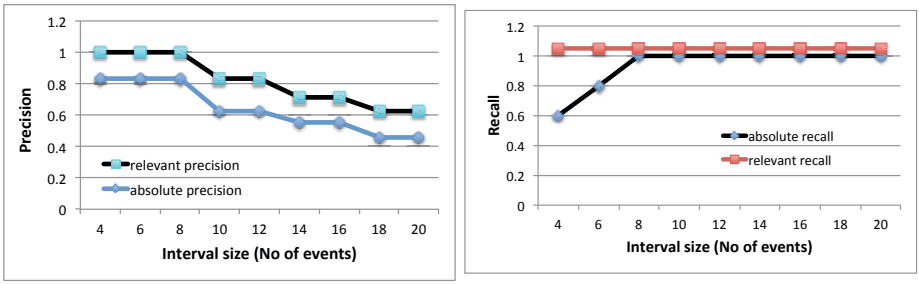
Fig. 5. Event stream split for pattern discovery algorithm

invigorating the association rule should be greater than the sum of frequencies weakening it (*lines 21–23*). The algorithm's time complexity depends on the event stream size n, the average tree level size g and the tree branches b. Thus, as the algorithm requires $g*b$ (i.e., approximately the cardinality of the powersets) iterations on the event stream, its complexity is $\mathcal{O}(n * b * g)$. This means that the powerset tree is extensively traversed to discover the association rules.

Fig. 5 clarifies the way intervals are processed to identify patterns for the average DeviceConfig average execution time violations (i.e., violation of E_3 event). The event stream comprises 35 events (after event filtering) of Section 2's 6 metrics. Each interval's powerset sets (figure's connected events) are processed to determine association rules. Thus, the event stream is split into 5 intervals: 3 critical (c mark on interval's upper right corner) and 2 non-critical (nc mark on interval's upper right corner). For critical intervals, the subset before the last raw critical DeviceConfig SaaS execution time event is considered, while for non-critical ones the whole interval. The algorithm discovers two patterns and extracts the association rules: (i) $\{e_1, e_2, e_4 \rightarrow E3\}$ and (ii) $\{e_4, e_6 \rightarrow E3\}$.

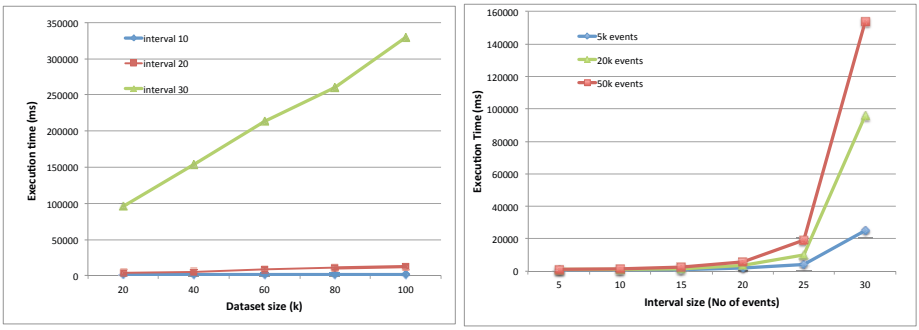
6 Evaluation

This section experimentally evaluates the algorithm's performance and accuracy, in order to optimize the definition of the aggregate metric (i.e., its optimal



(a) Precision **(b)** Recall

Fig. 6. Algorithm's accuracy



(a) Execution time / dataset size **(b)** Execution time / interval size

Fig. 7. Algorithm's performance

interval). An event dataset comprising 100k events from the traffic management app is used. During a pre-processing of the event stream, five periodic patterns are identified (two 2-size patterns, one 3-size pattern and two 4-size patterns). The main task is to discover patterns causing DeviceConfig SaaS execution time violations (i.e., violations of E_3). The experiments were performed on a machine with quad-core CPU 2.6GhZ, 8GB RAM and Mac OS X operating system.

The first experiment evaluates the algorithm's raw *relevant and absolute accuracy*. The former considers only the five known patterns, while the latter additionally considers their sub-patterns, as they can also drive proactive adaptation. The algorithm's *precision* and *recall* is measured while fluctuating the interval size from 4 to 20 events. Fig. 6a shows that *relevant precision* is 1 for small intervals and falls while increasing the interval size, while *absolute precision* fluctuates similarly at lower levels (as more irrelevant sub-patterns are discovered). The precision starts to fall over 8-size intervals, i.e., above the double of the maximum pattern (4 events). Fig. 6b shows that the algorithm's *absolute* recall is 1 for all considered interval sizes, except for 4-size and 6-size intervals, where it fails to discover two and one 4-size patterns respectively, due to interval

overlapping. Moreover, *relevant* recall is always 1, as the discovered sub-patterns compensate the "lost" relevant patterns (for 4- and 6-size intervals), as they also map to adaptation strategies addressing the whole pattern. Considering these accuracy results, the optimal definition of metric E_3 is to measure it in intervals containing in average 8 events. Thus, every aggregate metric's definition can be adjusted, enhancing the proactive adaptation of any SBA.

The second experiment evaluates the algorithm's execution time, based on the dataset and interval size. The results in Fig. 7 show that the algorithm's execution time linearly increases with an increasing dataset size, as expected. Larger intervals seem to hurt more the algorithm's performance, due to higher $b*g$ products. However, such execution time is acceptable, as this is an offline algorithm not affecting the overall framework's performance. Furthermore, the results in Fig. 7b reveal a changing relation between execution time and interval size; for larger intervals, it increases with a burst over 20-size intervals, due to rapid increase of $b*g$ (740 (b=148, g=5) for 30-size interval compared to 92 (b=23, g=4) for the 25-size and 48 (b=12, g=4) for the 20-size intervals), posed by the high increase of the considered unique sets.

7 Related Work

The mining of significant patterns within event stream areas have recently attracted many researchers. Most of these approaches are predominantly based on the *apriori* algorithm [7], producing an association rules set between items of large databases, based on a minimum support (*minsup*). Other approaches propose variations of these algorithms, focusing on performance [8] and accuracy [9] optimization. All such approaches suffer from many issues stemming from the difficulty in determining the optimal *minsup*. Contrarily, logic-based approaches exploit inferencing to discover patterns defining respective association rules. In [6] a pattern discovery approach is proposed mapping logical equivalences based on propositional logic. In particular, a rule mining framework is introduced, generating coherent rules for a given dataset that do not require setting an arbitrary *minsup*. [10] proposes an event calculus (EC) dialect for efficient run-time recognition that is scalable to large data streams.

Concerning IaaS modeling, some well-established approaches, such as the OASIS Cloud Application Management for Platforms (CAMP) specification (`www.oasis-open.org`), focus on modeling the most generic infrastructure components. At the PaaS layer, the mOSAIC EU Project's (`www.mosaic-cloud.eu`) entity ontology stands out as a solution for improving interoperability among existing Cloud solutions, platforms and services. Finally, SaaS layer modeling has been widely influenced by the service computing, such as the models introduced within the S-Cube EU project (`www.s-cube-network.eu`).

Compared to the related work, the main benefits of our approach are the following. First, we propose a component meta-model able to describe the components of multi-cloud SBA along with their interrelationships. The models complying to this meta-model assist in identifying correct event patterns by considering only events originating from interrelated components. Second, we

propose a logic-based algorithm for discovering event patterns leading to critical events within a monitoring event stream, enabling injecting proactiveness in an SBA system via mapping event patterns to respective adaptation strategies.

8 Conclusions and Future Work

This paper has presented an approach towards efficiently discovering detrimental event patterns causing critical violations during multi-cloud SBA execution. In particular, a component model is presented, describing SBA components and their interrelationships, that is exploited by a logic-based algorithm discovering event patterns leading to specific metric violations and can be further exploited by any adaptation engine to trigger suitable proactive adaptation actions when detected. For future work, we plan to optimize the pattern discovery algorithm's accuracy and performance and develop a scalability rule mechanism that could semi-automatically map discovered event patterns to suitable adaptation strategies. Finally, we are going to employ new techniques to infer new component dependencies from the current irrelevant discovered patterns.

Acknowledgements. We thankfully acknowledge the support of the PaaSage (FP7-317715) EU project.

References

1. Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., Papaioannou, A., Plexousakis, D., Zeginis, C.: Lifecycle Management of Service-based Applications on Multi-Clouds: A Research Roadmap. In: MultiCloud (2013)
2. Zeginis, C., Konsolaki, K., Kritikos, K., Plexousakis, D.: Towards proactive cross-layer service adaptation. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 704–711. Springer, Heidelberg (2012)
3. Zengin, A., Marconi, A., Pistore, M.: CLAM: Cross-layer Adaptation Manager for Service-Based Applications. In: QASBA 2011, pp. 21–27. ACM (2011)
4. Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., Clarke, S.: Taxonomy-driven Adaptation of Multi-Layer Applications using Templates. In: SASO (2010)
5. Zeginis, C., Kritikos, K., Garefalakis, P., Konsolaki, K., Magoutis, K., Plexousakis, D.: Towards cross-layer monitoring of multi-cloud service-based applications. In: Lau, K.-K., Lamersdorf, W., Pimentel, E. (eds.) ESOCC 2013. LNCS, vol. 8135, pp. 188–195. Springer, Heidelberg (2013)
6. Sim, A.T.H., Indrawan, M., Zutshi, S., Srinivasan, B.: Logic-based pattern discovery. IEEE Trans. Knowl. Data Eng. 22(6), 798–811 (2010)
7. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)
8. Bettini, C., Wang, X.S., Jajodia, S., Lin, J.L.: Discovering frequent event patterns with multiple granularities in time sequences. IEEE Trans. Knowl. Data Eng. 10(2), 222–237 (1998)
9. Hellerstein, J.L., Ma, S., Perng, C.S.: Discovering actionable patterns in event data. IBM Systems Journal 41(3), 475–493 (2002)
10. Artikis, A., Sergot, M.J., Paliouras, G.: Run-time composite event recognition. In: DEBS, pp. 69–80. ACM (2012)