Towards the Modelling of Adaptation Rules and Histories for Multi-Cloud Applications

Kyriakos Kritikos[1], Chrysostomos Zeginis[1], Eleni Politaki[2] and Dimitris Plexousakis[1,2]
[1]*ICS-FORTH, Heraklion, Crete, Greece*
[2]*Department of Computer Science, University of Crete, Heraklion, Crete, Greece*

Keywords: Adaptation, Execution, History, Rule, Meta-model, DSL.

Abstract: Currently, there is a move towards adopting multi-clouds due to their main benefits, including vendor lock-in avoidance and optimal application realisation via different cloud services. However, such multi-cloud applications face a new challenge related to the dynamicity and uncertainty that even a single cloud environment exhibits. As such, they cannot deliver a suitable service level to their customers, resulting in SLA penalty costs and application provider reputation reduction. To this end, we have previously proposed a cross-level and multi-cloud application adaptation architecture. Towards realising this architecture, this paper proposes two extensions of the CAMEL language allowing to specify advanced adaptation rules and histories. Such extensions not only enable to cover cross-level application adaptation by executing adaptation workflows but also to progress such an adaptation to address both the application and exploited cloud services evolution.

1 INTRODUCTION

Cloud computing has revolutionized the way applications are developed, deployed and provisioned. Its wide adoption has led to a proliferation of cloud applications and services. Such applications were either migrated to the Cloud or were developed from scratch by adopting existing or new technologies. The success of this computing paradigm is mainly due to the benefits it delivers, including cost reduction, flexible resource management and resource elasticity allowing applications to scale on demand as needed.

Various providers have rushed to offer cloud services and platforms. The bigger from those providers have also attempted to lock-in cloud users by offering certain technologies, platforms as well as extra, added-value secondary services. This has led towards adapting multi-clouds both at the research and industry level. The application deployment at multiple clouds is quite promising as it enables to avoid the vendor lock-in while it also brings additional benefits, including the selection of the best cloud services to optimally realise an application according to its requirements and the increase of the application security level via adopting different security services.

However, multi-cloud applications face a novel challenge related to the dynamicity and uncertainty inherent even in a single cloud. This jeopardises an application's ability to keep up with its promises by delivering a suitable service level to its customers. Such a challenge is further hardened by the fact that an application can be realised by different cloud services types in different abstraction levels. Thus, an application is vulnerable to the quality level variation of such services. Further, the dynamicity aspect should not be neglected. Both services and application requirements can evolve over time. As such, even if a suitable way to adapt an application is discovered, it can be invalidated over time due to this dynamicity.

To close this gap, we have already proposed (Kritikos et al., 2017) an architecture of an advanced, cross-level and multi-cloud application adaptation framework. Such a framework will exhibit the following features: (a) semi-automatically infer new adaptation rules (Zeginis et al., 2015) by considering the application structure and the dependencies at the different abstraction levels; (b) dynamically transform such rules into adaptation workflows that can be enacted via workflow execution engines; (c) dynamically evolve adaptation rules based on their execution history, performance and successability in completely addressing an actual event ("problematic situation") that has caused their execution. All these features are currently being implemented to realise our vision.

This paper attempts to provide support to this ongoing realisation by proposing two meta-model ex-

Kritikos, K., Zeginis, C., Politaki, E. and Plexousakis, D.
Towards the Modelling of Adaptation Rules and Histories for Multi-Cloud Applications.
DOI: 10.5220/0007706503000307
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 300-307
ISBN: 978-989-758-365-0

tensions to the CAMEL state-of-the-art cloud modelling language (Rossini et al., 2015). The first extension focuses on the modelling of advanced adaptation rules. Such a modelling is quite rich as it re-uses elements from Complex Event Pattern (CEP) languages, like the Esper's[1] one, to specify complex event patterns as logical or time-based compositions of simpler events. Further, to complete the adaptation rule specification, it enables mapping such events patterns to workflow-language-independent adaptation workflows, including actions at any possible abstraction level (infrastructure, platform, software and workflow). To the best of our knowledge, no other meta-model or language (Song et al., 2013; Erbel et al., 2018; Marquezan et al., 2014; Lushpenko et al., 2015) has the right expressivity level to specify such adaptation rules. Our contribution benefits the first two features of the envisioned adaptation framework. Especially, the workflow language independence offers implementation flexibility as our adaptation framework could re-use any workflow engine, specialised in the use of a certain workflow language.

The second meta-model extends CAMEL's execution sub-DSL to capture not only application execution but also adaptation histories by explicating which adaptation actions were performed under which adaptation rule and how well they addressed the respective problematic event according to which performance level. This contribution benefits the 3rd adaptation framework feature. First, as it enables analysing the suitability of single and composite adaptation actions (i.e., workflows), thus allowing their prospective substitution when the need arises. It can also identify places for adaptation behaviour improvement by detecting situations where all automatically-generated alternative adaptation workflows for the same event are not so performant or suitable any more. As such, this meta-model well covers our goal to (semi-)automatically evolve the application adaptation behaviour over time to cope with permanent changes in both the services being exploited and the application requirements. This second paper contribution is novel as we are not aware of any approach able to record the cloud application adaptation history.

The rest of the paper is structured as follows. Next section introduces our envisioned adaptation framework architecture. Section 3 elaborates on the adaptation meta-model proposed. Section 4 explicates the extension performed over CAMEL's execution meta-model. Section 5 provides an example use case utilised to showcase the added-value of the two meta-models proposed. Finally, the last section concludes the paper and draws directions for further research.

[1] www.espertech.com/esper/

2 MULTI-CLOUD APPLICATION ADAPTATION FRAMEWORK

We have already proposed a holistic multi-cloud application adaptation framework, depicted in Fig. 1. This framework, currently under implementation, exhibits the following features: (a) enables inferring new from existing adaptation rules; (b) transforms such rules into adaptation workflows that can be executed by workflow engines; (c) dynamically changes adaptation workflows mapping to problematic events to better address them and thus evolve application adaptive behaviour; (d) enables editing adaptation rules, quite handy when automatically generated rules need adjustment or for rapidly dealing with cases not covered by the existing adaptation rule set; (e) enables browsing the adaptation history to check the successfulness of adaptation rules. All these five features are enabled by the paper's two contributions.

In the sequel, we shortly explain the adaptation framework components and their interactions. The *Adaptation UI* enables editing adaptation rules, enacting them (e.g., manual rules to rapidly react to unanticipated situations), and visualising both the application adaptation history and its analysis. The analysis results could then be approved by the expert to evolve the application adaptation behaviour, if needed.

Any adaptation rule kind, edited or automatically generated or enacted, passes via the *Transformer*, responsible for: (a) transforming the rules in CAMEL into the format expected by the *Rule Engine*; (b) transforming the adaptation workflow part of enacted rules into the language expected by the *Adaptation Engine*.

The *Rule Engine* enables the enactment path of rules. It takes as input monitoring events, retrieved from the *Monitoring Framework*, and checks which are the adaptation rule(s) triggered by them. Upon a rule triggering, the *Transformer* is informed to enact its adaptation workflow. Internally, the *Rule Engine* utilises a *Rule Base* for adaptation rule storage.

Once the enacted rules' workflow is transformed, it must be concretised. The main rationale is that the adaptation framework capabilities can be enriched over time such that a certain adaptation task could be realised by two or more alternatives. As such, the *Concretiser*, based on user requirements and preferences, attempts to solve a constraint optimisation problem to support the selection of the best possible adaptation task alternatives globally.

A concretised adaptation workflow can be then executed by the *Adaptation Engine*, taking the form of a workflow execution engine. This engine also stores information about the adaptation workflow execution in the *Model Repository* (a placeholder for all

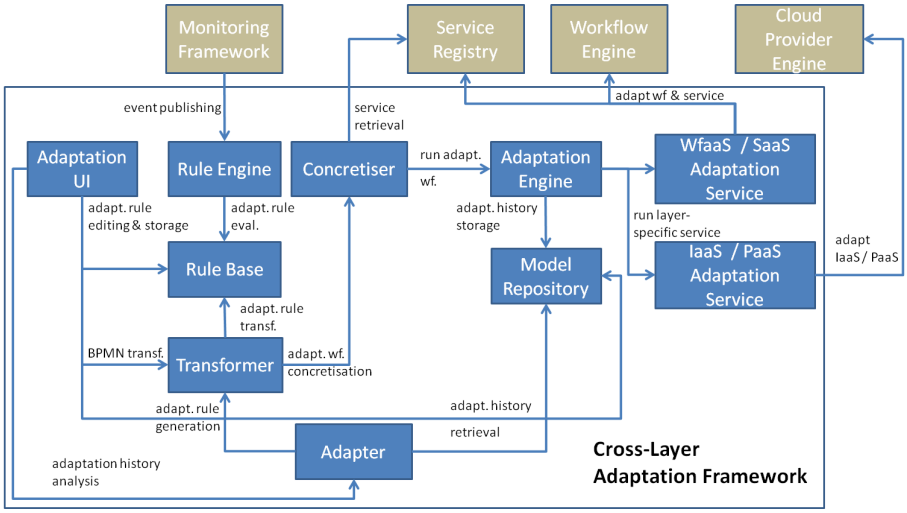

Figure 1: The envisioned adaptation framework architecture.

CAMEL models produced and manipulated) by using CAMEL's execution meta-model extension. The workflow execution relies on using level-specific services (e.g., WfaaS/SaaS adaptation service) representing service-based realisations of adaptation tasks.

The *Adapter* generates new or adapts existing adaptation rules by analysing the adaptation history. New rules are semi-automatically generated via our previous work (Zeginis et al., 2015), which combines the event patterns discovered with all possible combinations of adaptation tasks able to address them.

3 ADAPTATION META-MODEL

CAMEL originally covered the adaptation of multi-cloud applications via its scalability meta-model, dedicated to the capturing of scalability rules in form of mappings from events to scaling actions.

Events (see Fig. 2) could be simple or composite. Simple events were specified via metric conditions. Composite events were specified as event patterns by applying time-based (e.g., PRECEDES) and logical operators (e.g., AND) over simpler events, thus actually representing an event tree or hierarchy.

Scaling actions were distinguished into horizontal and vertical. Horizontal scaling actions were specified by indicating the number of instances to be added or removed for an application component. Vertical scaling actions were specifying the VM to be scaled and the respective update in the size or number of certain VM features like the number of cores.

To cover the whole adaptation possibilities of a multi-cloud application at different abstraction levels, CAMEL was extended (see Fig. 3) to transform its scalability meta-model to an adaptation one.

3.1 Adaptation Rules and Strategies

The meta-model's topmost element is *AdaptationModel*, acting as a container for all adaptation-related elements. The meta-model's key concept is *AdaptationRule*, mapping events to adaptation actions.

In essence, there can be multiple rules able to confront a problematic situation, i.e., an event; so, an adaptation system must choose the best from them. This choice can be reduced to selecting the rule with highest *priority*, i.e., a dynamically-modifiable attribute. This means that an already selected rule might be unsuitable for selection in the near future as it might not have a sufficient successability level or a better rule is discovered that more completely addresses the respective event. An *AdaptationStrategy* encapsulates all rules confronting such an event. The collection of all adaptation strategies then represents an application's current adaptive behaviour as well as the way such a behaviour can evolve over time.

3.2 Adaptation Tasks

Event specification has not been modified in the new CAMEL version. However, the scaling actions were subsumed by an adaptation task, that can be incorporated within adaptation workflows to confront problematic situations. Such a task can be simple or composite. A simple task maps to executing a level-specific action, while a composite task can be seen as an adaptation workflow, i.e., a composition of (simpler) adaptation tasks whose execution is controlled by well-known control flow constructs.

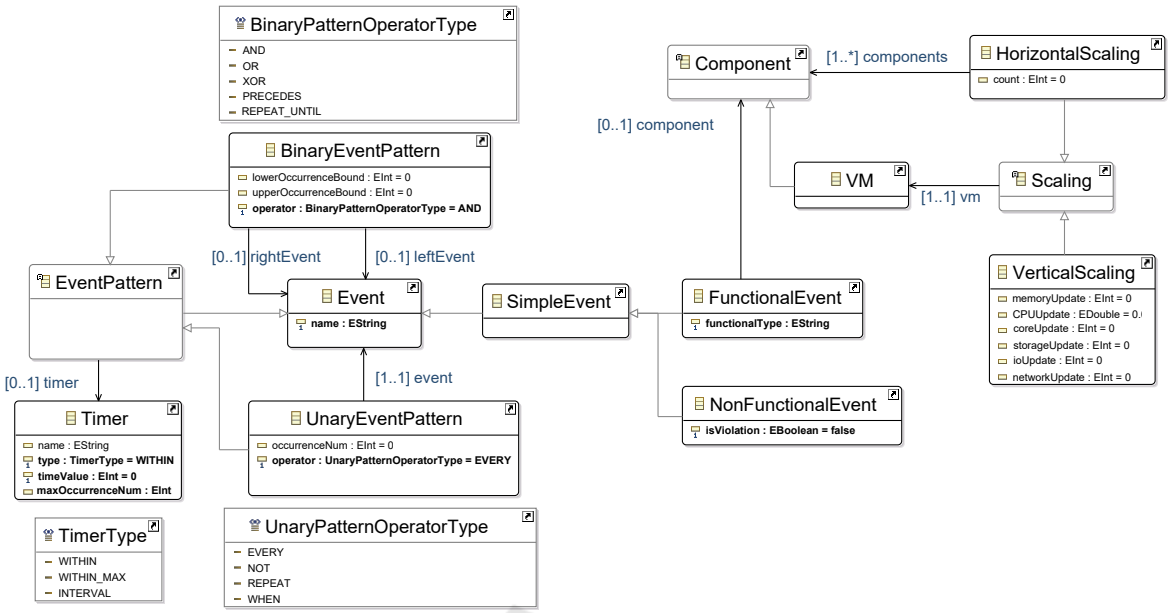

Figure 2: The representation of events in CAMEL.

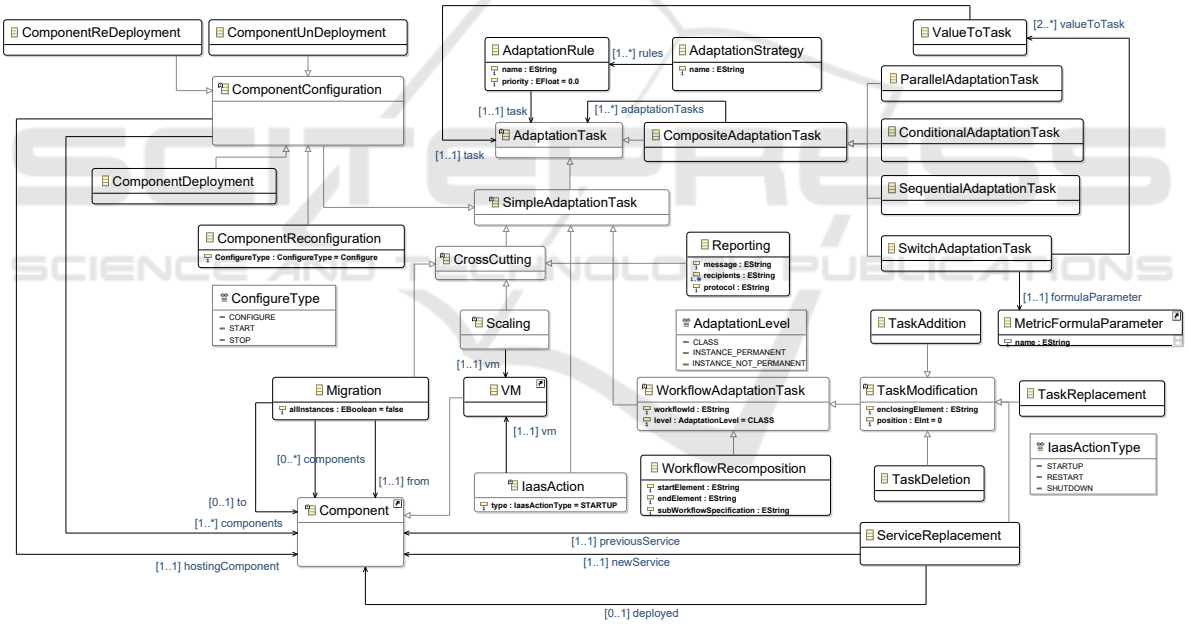

Figure 3: The adaptation meta-model in CAMEL.

3.2.1 Composite Adaptation Tasks

A composite task can be seen as a tree containing tasks with lower complexity. To simulate the four main types of basic control flows that can participate in any workflow kind, we have created respective subconcepts of the *CompositeAdaptationTask* concept.

A *SequentialAdaptationTask* represents a sequence of adaptation tasks, while a *ParallelAdaptationTask* a parallel execution of tasks. On the other hand, a *ConditionalAdaptationTask* relates to a conditional construct predicated over a certain event referenced. The occurrence of that event leads to executing the first task referenced by this composite task, while its non-occurrence the second referenced task. For a conditional branch with multiple alternatives, a *SwitchAdaptationTask* needs to be supplied, which relates to a certain *MetricFormulaParameter*, i.e., a metric or a metric formula. In each case, multiple values can be produced by monitoring this met-

ric or computing this formula that have to be mapped (see *ValueToTask* concept) to the respective adaptation tasks that need to be executed.

3.2.2 Simple Adaptation Tasks

A simple adaptation task is sub-classed, according to the (abstraction) level it pertains, to other concepts with the exception of the *Cross-Cutting* abstract concept, representing cross-cutting adaptation actions.

IaaSAction covers all possible actions (startup, shutdown and restart) at the IaaS level by referring to the VM to be adapted and the action type that can be performed on it (see *IaaSActionType* enumeration).

At the SaaS level, adaptation tasks are encapsulated by the abstract *ComponentConfiguration* concept which covers the configuration of one or more SaaS. This task can then be further distinguished into more concrete configuration tasks which include:

- *ComponentDeployment*: indicates that the SaaS components referenced must be deployed in a PaaS or IaaS component. Such an action could be useful in the context of deploying components, such as load balancers, only after the second instance of an application component is created.
- *ComponentUnDeployment*: indicates that the components referenced need to be undeployed from their hosting component, also referenced. Such a scenario could be useful in hybrid cloud bursting, where the public VMs for a certain application component are not needed any more.
- *ComponentReDeployment*: signifies that the components referenced must be redeployed. This could be useful for migrating the components to a new version, e.g., to address certain bugs.
- *ComponentReconfiguration*: signifies that the referenced components must be reconfigured by either stopping, starting (e.g., to bypass the potential transient error occurred) or redeploying them (e.g., when the error is not transient).

At the workflow level, *WorkflowAdaptationTask* encapsulates all possible adaptation tasks.A workflow adaptation task can be further distinguished into a *WorkflowRecomposition* and *TaskModication* one.

A *WorkflowRecomposition* signifies the need to recompose a workflow from the current execution point until the final workflow task by replacing its respective remaining content with a (sub-)workflow description. On the other hand, a *TaskModification* task signifies a single, low-level workflow modification at the task level by pointing to the respective workflow part (i.e., workflow task) where this change will be performed. Such a task can be further distinguished into:

- *TaskAddition*: signifies the addition of a workflow task (represented by an *ApplicationTask* encapsulating the task description) at the point referenced.
- *TaskDeletion*: indicates the deletion of the workflow task pointed.
- *TaskReplacement*: signifies the replacement of the workflow task pointed by another also referenced.

A change at the workflow level can be permanent or not. This is signified by the *level* attribute in *WorkflowAdaptationTask* to cover 3 possible cases: (a) *Class*: the change is permanent – case of workflow evolution; (b) *InstancePermanent*: the change is permanent at the current instance but does not affect the other workflow instances; (c) *InstanceNonPermanent*: this is a special case of an instance-level change where a task instance in a workflow loop is adapted just once in the context of the current loop repetition.

Finally, cross-cutting adaptation tasks include level-independent tasks or tasks realised in different or across levels. These can be further separated into:

- *EventCreation*: maps to creating an event that can be consumed by the adaptation system and possibly lead to an adaptation rule triggering. This is a nice mechanism enabling to, e.g., deal with the uncertainty in executing an adaptation rule. As such, it could check whether its execution was successful; in the opposite case, it could create an event to signify that. This can enable formulating more advanced adaptation rules, accounting the possibility of adaptation-related exceptions.
- *Reporting*: in some cases, it could be possible that the an adaptation rule triggering signifies the occurrence of a critical event of which a respective user, like an admin, should become aware. This could be useful, e.g., when the admin could complete the partial addressing of the current problem by the adaptation rule.
- *Scaling*: scaling is an adaptation task that can be alternatively performed in different levels (IaaS or PaaS). Its specification is already covered in the original CAMEL version.
- *Migration*: a component can be migrated into a different VM or environment of the same or different cloud provider. As such, this task is specified by referring to one or more components that need to be migrated from a certain hosting component to another one. The hosting components can be certain PaaSes or IaaSes, such that we could have different possible migration cases (e.g., from PaaS to IaaS or IaaS to PaaS). It can also be indicated whether all instances of the component(s) to be adapted must be migrated or only the affected

one. The former could be possible when the current hosting could be deemed problematic for all the component(s) instances and not a single one.

4 EXECUTION META-MODEL

4.1 Background

CAMEL's execution meta-model covers the modelling of historical information related to the execution of a multi-cloud application. It partitions this information into groups called *ExecutionContext*s, representing deployment episodes for a certain application. An *ExecutionModel* is then just a collection of deployment episodes over time for this application.

Covering historical information can be quite important. First, it can be used to check an application's performance capabilities. Second, it can be used to reason over the best deployments of an application or its components. Such a reasoning could enable to accelerate the deployment reasoning time (Kritikos et al., 2016) as certain application component-to-offer mappings can be fixed. Third, it can be utilised by, e.g., a Reinforcement Learning deployment reasoning algorithm (Horn, 2013) to avoid inspecting deployments that failed in the past (either leading to errors or SLO violations). Fourth, it can be used for traceability analysis to check which requirements have led to producing which application deployment models.

In this respect, CAMEL was designed to cover all necessary information to support deriving all above knowledge types. First, the execution context encapsulates information related to: (a) what was the application's execution period; (b) the overall cost for it; (c) which deployment model and requirement group drove application deployment and provisioning. Second, this execution context was used as a reference to all other history-related elements that can be modelled. These elements are now shortly explained.

*Measurement*s of specific *MetricInstance*s are covered by specifying their value and a reference to their generation timepoint. They can be also associated with SLOs to be assessed based on them. Depending on the type of the object being measured, we can distinguish between application, internal component, VM, PaaS and communication measurements.

An *SLOAssessment* covers an SLO assessment's outcome and the time point it was performed. SLO assessments enable to track over time which SLOs were violated, thus triggering the affected application's global or local reconfiguration.

Finally, a *RuleTrigger* represents a scalability rule's triggering, also associated with its occurrence time plus the respective instances of events causing it.

4.2 CAMEL Modifications

While CAMEL covered well execution history modelling, it still lacked the complete capturing of adaptation information.As such, we have decided to extend CAMEL to fully cover both adaptation rule triggering as well as measurability and successability aspects related to it. The main goal is to allow the adaptation system to assess the suitability of adaptation rules within an adaptation strategy and dynamically modify their priority according to the information recorded.

To this end, CAMEL's execution meta-model (see Fig. 4) was extended based on the following (3) ways. First, the *RuleTriggering* concept was extended to cover the triggering of adaptation rules as well as the realisation of adaptation tasks that took place in this triggering. This gave rise to modelling a new concept.

This concept is named *TaskRealisation*, covering details about how an adaptation task was realised in the context of an adaptation rule. Such details include when the task execution started and ended, to compute the task's execution time, and the execution result. The latter maps to the next *AdaptationResult* enumeration members:

- *SUCCESSFUL*: the adaptation task did achieve its main adaptation goal
- *UNSUCCESSFUL*: the task, while successfully executed, failed to achieved its adaptation goal
- *FAILED*: a certain error or failure occurred during the execution of the adaptation task.
- *UNAVAILABLE*: the adaptation task was not available or accessible at the time point of its execution

The first two members enable assessing an adaptation task's successability, which could be, e.g., measured by the percentage of times the task attained its goal. The third member enables assessing the task reliability, which could be measured by subtracting from one the percentage of times task execution has failed. Finally, the last member enables assessing task availability, which could be computed by subtracting from one the percentage of time this task was unavailable.

Apart from this information, a task's realisation refers to its parent's realisation. This could be handy for traceability reasons, i.e., check how many times one or more sub-tasks can be blamed for the failure of a parent task. This could enable discovering those parent adaptation tasks doomed to fail to address a "problematic" event due to the "bad" combination of included sub-tasks. For instance, the task combination is either overlapping or wrong which could always lead to the parent task's unsuccessfulness.

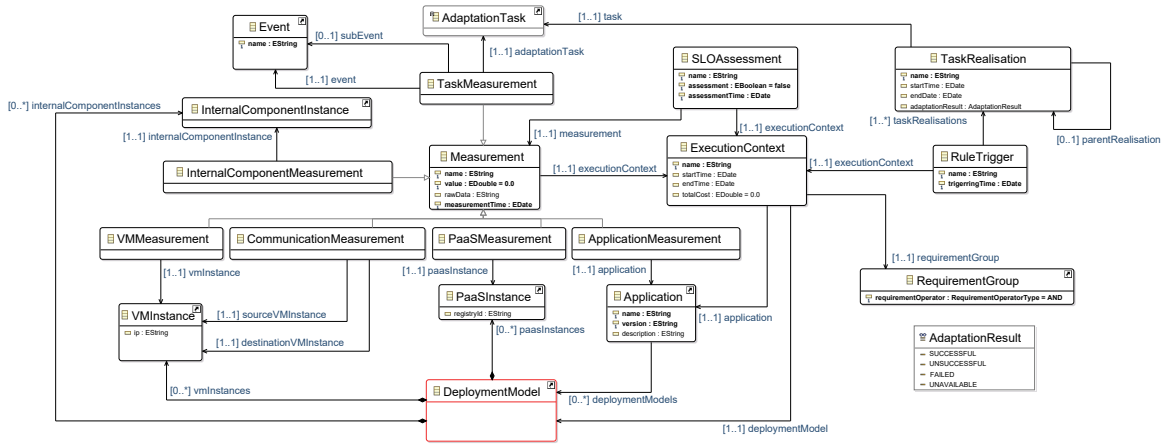

Figure 4: The extended execution meta-model in CAMEL.

From the raw information of task realisations, we can then deduce quality measurements for adaptation tasks, covering metrics associated with the task execution time, availability, reliability and successability. This could provide sufficient support for evaluating the adaptation rules' priority and thus their selection in the context of an adaptation strategy. For instance, this could be possible by aggregating such measurements and then applying an overall mathematical formula over them to calculate the adaptation rule priority. Thus, our vision for evolving a multi-cloud application's adaptation behaviour could be easily realised.

The adaptation task measurements also refer to the event being coped with. This ensures that a correct aggregation over them can be performed in the context of an adaptation strategy. Further, in case this is possible (i.e., we know how sub-events of event patterns map to adaptation sub-tasks), we could also record the sub-event of the overall event pattern which the respective adaptation sub-task is supposed to confront. This will supply an additional insight of, e.g., a certain task's successability across the multiple adaptation strategies in which it is potentially re-used.

5 RUNNING EXAMPLE

To demonstrate the two main paper contributions, we rely on a city traffic management use case relying on a sequential workflow repeatedly executed over time. This workflow comprises the following tasks:

- *monitor*: monitors a certain city area with respect to traffic and environment conditions and draws information concerning special events (e.g., concerts) that could take place in the city
- *analyse*: analyses the current situation, as sensed by *monitor*, and produces a traffic regulation plan
- *execute*: executes the regulation plan derived (by, e.g., controlling the traffic lights frequency in a problematic sub-area).

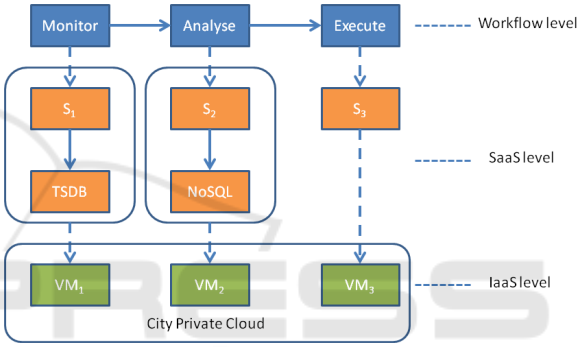

Figure 5: The architecture of the use-case system.

By following a micro-service architecture, these (3) tasks are mapped to micro-services, which are called as S_1, S_2, S_3 for short, respectively. The first service is deployed in VM VM_1, which also hosts an underlying time series data base (TSDB), for storing this service's measurements. The second service is deployed in VM VM_2, including also a NoSQL database for storing important knowledge required for traffic analysis. Finally, the last service is deployed in VM VM_3. All VMs have a different size (e.g., VM_2 is computationally and storage-wise wider than the other two) and were deployed in the city's private cloud. The overall system architecture is depicted in Figure 5.

Focusing on S_2 that realises the analysis functionality, the next (2) adaptation rules were modelled:

$$down(S_2) \longrightarrow restart(S_2) \quad (1)$$

$$down(S_2) \longrightarrow reconfigure(S_2) \quad (2)$$

Table 1: Instance information for *TaskRealisation* class.

name	startTime	endTime	result	task
TR1	1545991912	1545991972	SUCCESSFUL	T1
TR2	1545995572	1545995632	UNSUCCESSFUL	T1
TR3	1545999232	1545999292	UNSUCCESSFUL	T1
TR4	1546002892	1546002952	UNSUCCESSFUL	T1
TR5	1546003012	1546003252	SUCCESSFUL	CT1
TR6	1546006852	1546007152	SUCCESSFUL	CT1

The first rule is required to overcome transient by restarting S_2. On the other hand, the second rule can be employed to overcome permanent errors by redeploying the component on the same VM. Via this reconfiguration, it might be possible that such errors are corrected by a new version of the respective code.

Both rules address the same event, so the first rule was initially selected as it has a better execution time than the second. However, such a rule was not always successful as Table 1 (indicating instances of the *TaskRealisation* class) highlights. As such, as that rule's successibility rate is below a certain threshold, the expert decides to replace it with the second rule. The outcome (depicted in the same table) is satisfactory as the second rule's successibility rate is 1. Thus, the adaptation history of the use case workflow guides the expert in making the right choices to modify the workflow's adaptation behaviour accordingly.

6 CONCLUSIONS

This paper has introduced two extensions to the CAMEL state-of-the-art cloud modelling language. The first extension concerns CAMEL's scalability sub-DSL, enhanced to specify sophisticated adaptation rules as a mapping between events (patterns) to adaptation workflows. Such workflows are specified in a language-independent manner, enabling their transformation into any workflow language, depending on the workflow engine used to execute them.

The second extension concerns CAMEL's execution sub-DSL, enhanced via the capability to record the adaptation history of a multi-cloud application, including details about the adaptation actions performed, like their start and end time and their outcome. Such information can be exploited to derive important knowledge about adaptation actions like their performance and successibility in terms of addressing a certain event. This can enable replacing adaptation workflows with alternative ones to better address the same "problematic" event.

Both extensions provide support to the three main features of our envisioned multi-cloud application adaptation framework. The first extension enables supporting any workflow execution engine that could be injected into our adaptation framework, while the second evolving the adaptive behaviour of a multi-cloud application which will be evident when the application context permanently changes. The appropriateness of the two extensions was demonstrated via the use of a certain use case.

ACKNOWLEDGEMENTS

The research leading to this survey paper has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731664.

REFERENCES

Erbel, J. M., Korte, F., and Grabowski, J. (2018). Comparison and Runtime Adaptation of Cloud Application Topologies based on OCCI. In *CLOSER*.

Horn, G. (2013). A vision for a stochastic reasoner for autonomic cloud deployment. In *NordiCloud*, pages 46–53. ACM.

Kritikos, K., Magoutis, K., and Plexousakis, D. (2016). Towards Knowledge-Based Assisted IaaS Selection. In *CloudCom*, pages 431–439. IEEE Computer Society.

Kritikos, K., Zeginis, C., Griesinger, F., Seybold, D., and Domaschka, J. (2017). A Cross-Layer BPaaS Adaptation Framework. In *FiCloud 2017*, pages 241–248, Prague, Czech Republic. IEEE Computer Society.

Lushpenko, M., Ferry, N., Song, H., Chauvel, F., and Solberg, A. (2015). Using adaptation plans to control the behavior of models@runtime. volume 1474, pages 11–20.

Marquezan, C. C., Wessling, F., Metzger, A., Pohl, K., Woods, C., and Wallbom, K. (2014). Towards exploiting the full adaptation potential of cloud applications. In *PESOS*.

Rossini, A., Kritikos, K., Nikolov, N., Domaschka, J., Griesinger, F., Seybold, D., and Romero, D. (2015). *D2.1.3 —- CAMEL Documentation*.

Song, H., Raj, A., Hajebi, S., Clarke, A., and Clarke, S. (2013). Model-based cross-layer monitoring and adaptation of multilayer systems. *SCIENCE CHINA Information Sciences*, 56(8):1–15.

Zeginis, C., Kritikos, K., and Plexousakis, D. (2015). Event pattern discovery in multi-cloud service-based applications. *IJSSOE*, 5(4):78–103.