A Cross-Layer BPaaS Adaptation Framework

Kyriakos Kritikos, Chrysostomos Zeginis
Information Systems Laboratory
ICS-FORTH, Heraklion, Crete, Greece
Email: kritikos@ics.forth.gr

Frank Griesinger, Daniel Seybold, Jörg Domaschka
Institute of Information Resource Management
Ulm University, Ulm, Germany
Email: {fname.lname}@uni-ulm.de

***Abstract*—The notion of a BPaaS is currently taking a momentum as many organisations attempt to move and offer their business processes (BPs) in the cloud. Such BPs need to be adaptively provisioned so as to sustain the service level promised in the respective SLA. However, current cloud-based adaptation frameworks cannot cover all possible abstraction levels and usually rely on simplistic adaptation rules. As such, this paper proposes a novel BPaaS adaptation framework able to orchestrate actions on different abstraction levels so as to better address the current problematic situation. This framework can support the dynamic generation of adaptation workflows as well as the recording of the adaptation history for analysis purposes. It is also coupled with the CAMEL language which has been extended to support the specification of cross-level adaptation workflows.**

I. Introduction

In order to survive in such a dynamic business world, organisations need to exploit service-based business processes (BPs) which can adapt to the current situation as well as offer suitable service levels to their customers. As such, the cloud can be considered as one of the most suitable medium to support such capabilities as apart from promising the supply of infinite and cheap commodity resources, it also enables organisations to focus mainly on their core business (as, e.g., technical administration can be outsourced). As such, we are currently observing a trend of BP migration in the cloud which leads to the notion of Business Process as a Service (BPaaS).

BPaaS indicates the delivery of BPs as on-demand services. Its paradigm enables well the role of a BPaaS broker, an entity able to either provide BPaaS products to potential organisations to realise their core or support BPs or supply consulting services to facilitate these organisations to move their business to the cloud. Such a broker, while might have great expertise in BP and workflow modelling, should be supported with tools or technical platforms able to cover the whole lifecycle [1] of the BPaaSes offered without requiring from the broker to dig into quite specific cloud technicalities.

The BPaaS life cycle ranges from the BPaaS design by business experts down to its technical operation on cloud resources. In the European project CloudSocket[1], the phases of this life cycle were defined as: (i) Design BPs at the highest level, (ii) Allocation as mapping from BPs to cloud services, (iii) Execution of the BPs based on the allocated cloud services, and (iv) Evaluation of the BPaaS instance (see Figure 1). To cater for this, CloudSocket developed an architecture built on this paradigm with one environment per life-cycle phase. The presented adaptation framework in this paper was integrated into the Execution Environment, as this is the phase were adaptation happens at run-time.

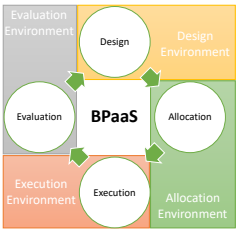

Fig. 1. BPaaS life cycle and the environments of the CloudSocket project.

By focusing on the execution lifecycle phase, which involves executing as well as dynamically provisioning a BPaaS, there is a great need to support the adaptation of a BPaaS to sustain the service level promised. Such an adaptation cannot be simply in the form of simple event to scaling actions, a current capability of existing cloud orchestration tools [2]. This form might be suitable in simple cases but is not sufficient to cover more sophisticated scenarios for BPaaS adaptation, subject to the fact that adaptation might require to occur in different abstraction levels, apart from the lower, infrastructure one, including the platform, service and workflow levels.

By considering the service-oriented computing literature, we can fortunately see some novel cross-level frameworks able to support more composite adaptation scenarios. However, such frameworks usually cover at most two levels and were developed before cloud computing came into real play. In addition, they usually employ level-specific adaptation mechanisms, triggered in an individual manner. In this way, the same situation could be sensed by these mechanisms which might attempt to react by performing respective adaptation actions. Without a central orhestration of such actions, there is a high risk that actions on one level might undo or interfere with actions on another level. This results in over-spending of resources as well as to vicious adaptation cycles.

To solve the above problem, this paper proposes a novel BPaaS adaptation framework able to orchestrate level-specific adaptation actions across levels and clouds. This framework builds upon a novel extension of the CAMEL[2] cloud application language [3], [4] able to support the specification of

[1] www.cloudsocket.eu

[2] www.camel-dsl.org

advanced adaptation rules that map event patterns to composite adaptation workflows which comprise individual adaptation actions on different levels. The proposed framework is an integration of existing adaptation frameworks: (a) a cross-layer service adaptation framework [5] and (b) a cross-cloud adaptation framework [6]. By considering that each framework focuses on different levels of abstraction, we can derive that that all possible levels in the BPaaS hierarchy are covered.

The combined framework exploits the individual ones in the form of web services that provide level-specific adaptation methods. By relying on a service-oriented architecture, the framework can support the dynamic concretisation of abstract adaptation workflows by considering the current adaptation capabilities of the BPaaS management system. It also includes an adaptation history record functionality on which analysis can be performed to derive interesting knowledge, including the successability of the adaptation rules, enabling the appropriate moderation of the adaptation system.

The rest of the paper is structured as follows. Section II focuses on the analysis of the related work. The adaptation-oriented extension of CAMEL is explained in Section III. The proposed framework is analysed in Section IV. A validation of the proposed framework according to a particular use case is supplied in Section V. Finally, the last section concludes the paper and draws directions for further research.

II. Related Work

This section focus on reviewing related work in service and cloud-based application adaptation, as both research areas are closely related to the proposed work.

A. Service Adaptation

1) Service Level Adaptation: Service adaptation work initially focused on covering just the service level. As such, approaches focusing on substituting services or on service re-composition were proposed which are analysed below.

An approach for BPEL process self-healing is presented in [7]. It relies on the Dynamo monitoring framework along with an AOP extension of ActiveBPEL and a monitoring and recovery subsystem using Drools Event-Condition-Action (ECA) rules. The problem of selecting alternative services and dealing with interface mismatches, when forwarding a request to an alternative endpoint, are not explicitly addressed. The recovery rules cannot also be changed dynamically, as they need to be compiled in an off-line manner.

The VieDAME environment [8] extends ActiveBPEL to enable BPEL process monitoring and partner service substitution based on various strategies, while service selection relies on defined selectors. It requires service registration to a repository, marking services to be monitored and eventually substituted as replaceable. It uses an engine adapter to extend ActiveBPEL functionality, but does not explicitly address fault handling.

An architecture and a DSL, named MONINA, are introduced in [9] to allow integrating functionality provided by different components and defining monitoring and adaptation functionality. Monitoring is carried out by complex-event processing queries, while adaptation is performed by condition action rules. This approach lacks cross-layer and multi-cloud features, and has not been validated.

2) Cross-Level Service Adaptation: As services operate under certain infrastructures and can be used to realise the functionality of BPs, it became apparent that additional levels had to be covered. This gave rise to cross-layer service adaptation frameworks which are now analysed in more detail.

A methodology for dynamic and flexible adaptation of multi-layer applications is proposed in [10]. This work advocates the use of templates, i.e., executable BPEL processes that encapsulate adaptation techniques, to handle specific mismatch types. Cross-layer service adaptation is achieved by directly (i.e., direct WSDL invocation) or indirectly linking adaptation templates (i.e., via event generation). Compared to our work, this approach does not handle all possible levels. The interlinking of different templates seems also to be a rather manual work, requiring high expertise and great effort from the expert. On the contrary, our work requires just the manual generation of simple rules by the expert while more complex ones are automatically produced by the system itself, allowing to cover more advanced adaptation scenarios.

An integrated multi-layer SBA monitoring and adaptation methodology was proposed in [11]. It comprises four main steps: (i) *Monitoring and Correlation*, where sensors capture run-time data about the software and infrastructural elements, (ii) *Analysis of Adaptation Needs*, where the anomalous situations and places for adaptation are identified, (iii) *Identification of Multi-layer Adaptation Strategies*, where existing adaptation capabilities within the system are used to define a multi-layer adaptation strategy as a set of software and infrastructure adaptation actions; and (iv) *Adaptation Enactment*, where adaptation engines at the software and infrastructure layer enact the multi-layer strategy's actions. The main approach drawbacks are that it does not feature proactive adaptation and does not detail how cross-layer monitoring is performed.

The CLAM holistic service management framework [12] can deal with cross- and multi- layer adaptation. This is achieved by first identifying the application capabilities affected by the adaptation actions and then by discovering an adaptation strategy able to solve the adaptation problem by properly coordinating a set of adaptation capabilities. The tree-based approach employed for defining adaptation paths seems very interesting but it can be time-consuming. During the ranking process of the adaptation branches, cost is also not taken into consideration. Cross-layer monitoring is also not elaborated. Finally, this approach does not exhibit proactive adaptation and does not deal with functional failures.

Our previous work includes a cross-layer service monitoring and adaptation framework [5] that follows a rule-based approach, where event patterns are mapped to adaptation strategies. Such strategies are mapped to adaptation workflows that can then be executed to address the current problematic situation. This framework supports pro-active adaptation by employing a logic-based mining approach [13] to mine pro-active adaptation rules from the service execution history. It is

also able to cover many levels but not all cloud-based ones.

B. Cloud-based Adaptation

Cloud-based application adaptation has focused mainly on horizontally scaling applications. A common approach followed by many commercial cloud platforms (e.g., Amazon EC2) is to support simple adaptation rules which trigger horizontal scaling actions when simple events, such as violations over CPU utilisation threshold, occur. The metrics mapping to such events mainly concern the infrastructure level.

A state of the art analysis of cloud orchestration tools was provided in [2]. The analysis outcome shows that the adaptation capabilities focus only on the IaaS level. Hence, an adaptation engine built on top of existing cloud orchestration tools can enable more sophisticated adaptation actions by integrating the PaaS and SaaS levels.

By considering hybrid and multi-cloud deployments, promising to optimise application performance and avoid lock-in issues, specific multi-cloud orchestration tools were proposed for application re-configuration. One such sophisticated framework [6] was developed in the PaaSage project [14]. This framework can support two adaptation types: (a) local or cloud-based adaptation based on expressive adaptation rules mapping to complex event patterns; (b) global adaptation aiming to globally reconfigure the application to still sustain its service level delivered. However, even in such a sophisticated framework, adaptation still remains at the IaaS level.

The framework in [15] can deal with the specification, monitoring and control of cross-level elasticity policies specified in the SYBL DSL. A comparison between this DSL and CAMEL can be found in [16]. The *Elasticity Control Service* deals with the generation and enforcement of an elasticity plan. As such, it consults information from the monitoring service which indicates the possible result of an adaptation action in the plan. Currently, elasticity actions can be operated over the level of the cloud application, the service topologies of its service components and the topologies' service units. Thus, the workflow and platform levels are not covered. In addition, it is not explicated how the elasticity plans are generated.

C. Comparison

We now compare the aforementioned analysed adaptation approaches with our work according to the following criteria: (a) *cross-layer*: capability to enact adaptation strategies / workflows that deal with more than one level. Please note that a system might cover multiple levels but have single-level strategies; (b) *levels*: the levels covered mapping to the following evaluation values: "I" as infrastructure, "P" as platform, "S" as service and "W" as workflow; (c) *type*: the type of adaptation supported: pro-active denoted as "P", re-active denoted as "R", while "A" denotes all types; (d) *dynamic*: the capability to dynamically concretise the adaptation workflow at adaptation time based on the current system adaptation capabilities; (e) *history*: the capability to browse the adaptation history of a BPaaS / cloud-based application. Criteria (a), (d) and (e) take a yes ("Y") or no ("N") evaluation.

Table I depicts an overview of the comparison results, where rows map to the adaptation approaches and columns to the comparison criteria. As it can be seen, our work is the best according to all the criteria considered. All frameworks, apart from ours, do not cover all levels. When more than one level is covered, most of the approaches do deal with the execution of cross-level adaptation workflows. Pro-active adaptation is an aspect mostly neglected with the sole exception of our previous and new work. Dynamicity seems to be supported mainly by some cross-layer service adaptation frameworks and our current work. Finally, adaptation history browsing seems to be fully supported only by our work with the sole exception of PaaSage for which the respective modelling capability exists but has never been realised in terms of actual functionality.

III. CAMEL Extension

CAMEL is a multi-DSL based on Eclipse EMF that covers multiple aspects in multi-cloud application specification, such as deployment, requirement, metric, scalability and organisation. It has been built by exploiting existing DSLs, such as CloudML, and by developing new ones, such as the Scalability Rule Language (SRL)[16]. It includes OCL constraints to validate the models produced based on the domain semantics. CAMEL is also accompanied with a customised textual editor, following a particular textual syntax, which enables devops users to quickly model their cloud applications.

This section explains first the original SRL focus and then the extensions performed on CAMEL over this sub-DSL.

Original Version. SRL originally focused on specifying sophisticated scalability rules in the form of event patterns mapping to scalability actions (both horizontal and vertical). Event patterns were specified as composite events produced from combining simpler ones according to logical or time-based operators. This SRL part was motivated by event pattern languages proposed in Complex Event Processing (CEP) systems like Esper. Time-based operators included both unary (e.g., *repeat* – event to occur multiple times) and binary operators (e.g., *precedes*). Simple events were specified via conditions over non-functional metrics which included a comparison operator and a certain threshold, while they were also clarifying the condition context. The latter was explicating details like which application component is measured and what is the measurement schedule and window of the concerned metric. SRL was also able to specify all possible metric aspects, including sensor, computation formula and unit information. In this sense, SRL was deemed as a quite complete and rich language, beyond the state-of-the-art with respect to other scalability languages in the cloud.

New Version. To extend its scope to cover additional levels as well as adaptation action workflows, SRL is now extended to become a complete adaptation and not just a scalability language. The extensions focused on allowing the language to specify cross-level adaptation workflows independently of any workflow specification language. This enables a flexibility in the application of SRL and its realisation in respective cloud adaptation systems. They also focused on extending the

TABLE I
EVALUATION TABLE OVER SERVICE AND CLOUD ADAPTATION WORK.

Work	Cross-Layer	Levels	Type	Dynamic	History
[7]	N	S	R	N	N
[8]	N	S	R	N	N
[9]	N	S	R	N	N
[10]	N*	ISW	R	Y*	N
[11]	Y	ISW	R	Y	N
[12]	Y	ISW	R	Y	N
[5]	Y	ISW	A	N	N
Amazon EC2	N	I	R	N	N
PaaSage [6]	N	IP	R	N	N*
[15]	Y	IS	R	N	N
Our Framework	Y	IPSW	A	Y	Y

current set of adaptation actions that can be specified both on the infrastructural as well as higher abstraction levels. In the following, we analyse the main SRL meta-model, capturing its abstract syntax, depicted in Figure 2.

The *Adaptation Rule* concept represents an adaptation rule, mapping *Event*s (simple or event patterns) to an *Adaptation Task*. Adaptation tasks can be simple or composite. Simple tasks map to single, level-specific adaptation actions.

Composite adaptation tasks are adaptation workflows, i.e., control flow-based combinations of other adaptation tasks. By focusing on simplicity and the satisfactory coverage of most control flows, we have considered capturing the most widely-known control flow constructs in workflow specification, including sequence, parallel, conditional and switch constructs. Respective sub-concepts of *CompositeAdaptationTask* have been created to denote the type of flow imposed. Sequence and parallel adaptation tasks just include a reference to the adaptation tasks being composed.

A conditional adaptation task also specifies an event. When this event occurs, then the first adaptation sub-task in the composition will be executed; otherwise, the second adaptation sub-task will be performed.

In case we need to map different evaluation values on a greater set of adaptation tasks, then a *SwitchAdaptationTask* can be used, mapping a set of dynamic variable values to respective adaptation sub-tasks. The semantics is that when the respective variable value is encountered, then only the corresponding adaptation task mapping to this value will be executed. This mapping is captured via the *ValueToTask* concept. A dynamic variable maps to referring to a *MetricFormulaParameter*, which covers both metrics as well as mathematical expressions over them.

Simple adaptation tasks have been sub-classed into concepts focusing on covering either level-specific actions (e.g., workflow modification) or a logical group of actions in a certain level (e.g., scaling). Currently, almost all levels are covered apart from the platform one, for which an extension is under-way. Workflow modifications are currently covered by task modifications and the *WorkflowRecomposition* action, which attempts to recompose a part in the (BPaaS) workflow specification, determined by a start and end element, by substituting it with a new sub-workflow specification.

Various task modification actions have been modelled. All such actions determine the control flow element in the BPaaS workflow which should include the affected task in a certain position. A task omit action declares that the affected task should not be executed, when its turn in the execution order arrives. A task addition action clarifies that a new task, whose specification is given, should be added after the affected task within the same enclosing control flow element. A task replacement indicates that a new task, whose specification is supplied, should replace the affected task.

The service / software level is currently covered by a service replacement action which indicates that a service realising a task in the BPaaS workflow should be replaced by a new one. It includes the specification of the following information: (a) a reference to a *Component* in the BPaaS deployment model, which can be an internal service component or an external SaaS, that has to be replaced; (b) attributes *newService* and *serviceSpecification* which should be supplied only when we need to explicitly specify the replacement service; otherwise, the adaptation system can decide itself which alternative service to use for the replacement. The first from the attributes denotes either the service's endpoint or its identifier in a service registry (part of the adaptation system). The second is used for supplying the specification for a service composition (as a service replacement) which does not take the form of a composite service with a certain endpoint; (c) the identifiers of the BPaaS tasks for which the replacement should occur.

At the infrastructure level, we have foreseen actions which either scale, migrate or manage deployments in the BPaaS deployment model. Scaling is covered via horizontal or vertical scaling actions and always refers to the VM to be scaled. A horizontal scaling action also indicates the components to be scaled and as their amount of instances to be created. As such, we also cover scenarios where not all components in a certain VM need to be scaled but only a sub-set of them. A vertical scaling action determines how the scaling is to be performed via identifying the modifications needed in the corresponding VM characteristics (e.g., number of cores or memory size).

A migration action explicates which components to migrate from an old to a new VM. In addition, it includes a boolean attribute which denotes the multiplicity of the migration. In particular, this attribute indicates whether we need to migrate

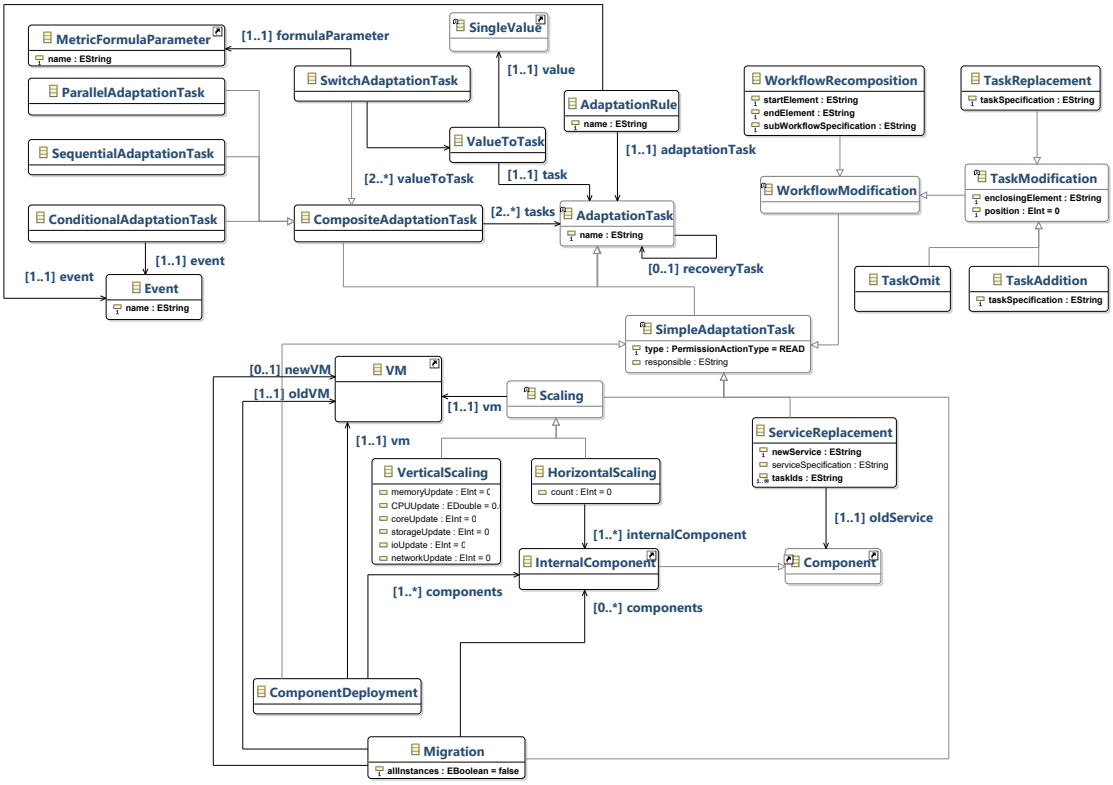

Fig. 2. The new SRL meta-model

all instances of the referred components from all instances of the old VM or just the component instances from the affected instance of the old VM.

Finally, we have specified two types of deployment management actions: component deployment and undeployment. Component deployment highlights that we need to deploy one or more instances of a component in a certain VM. This might be needed in cloud bursting (e.g., deploy a new component instance not in a private but a public VM) or load balancing scenarios (e.g., introduce load balancer over horizontally scaled instances of a component). Component undeployment is the opposite action of undeploying a component from a VM which could be useful in scenarios where, e.g., internal service components must be replaced by external SaaS services. In the latter scenario, the service component has first to be undeployed before being replaced by the external SaaS.

IV. Proposed Framework

As Section II highlighted, our framework advances the state-of-the-art by either exhibiting genuine features or combinations of features that cannot be supported by a single related adaptation framework. Moreover, the framework also caters for better supporting the expert according to the following novel features: (a) it allows the experts to edit adaptation rules, either those which have been automatically produced by the system or those which are newly modelled by them based on their experience and skills; (b) it enables executing manual adaptation workflows in case a problematic situation cannot be covered by the adaptation rule set modelled; (c) it allows retrieving the adaptation history for browsing and adaptation analysis reasons. For instance, some problematic adaptation rules can be detected, not able to confront the problems triggering them, which can be automatically replaced by other alternatives or can be manually edited by the expert.

Figure 3 depicts the architecture of the proposed framework, which comprises 9 main components. The *Transformer* component is responsible for transforming the CAMEL adaptation rules automatically derived from the BPaaS Evaluation Environment into the format needed by the *Rule Engine* as well as for transforming the rules' consequent part mapping to CAMEL adaptation workflows into BPMN for enabling their execution by the *Adaptation Engine*. The adaptation rules transformed cover both pro-active and re-active adaptation and are produced according to the approach in [13], which employs a logic-based mining method to derive event patterns mapping to SLO / KPI violations and a semi-automatic method to map the event patterns to adaptation workflows. The latter method relies on the initial knowledge derived from the expert in the form of simple event-to-adaptation action rules.

The *Rule Engine* is responsible for storing, in the *Rule Base*, and triggering adaptation rules. The triggering relies on monitoring facts, generated by the cross-layer *Monitoring Engine* via assessing Service Level Objective (SLOs) conditions, and fetched via a publish-subscribe mechanism, which enable the

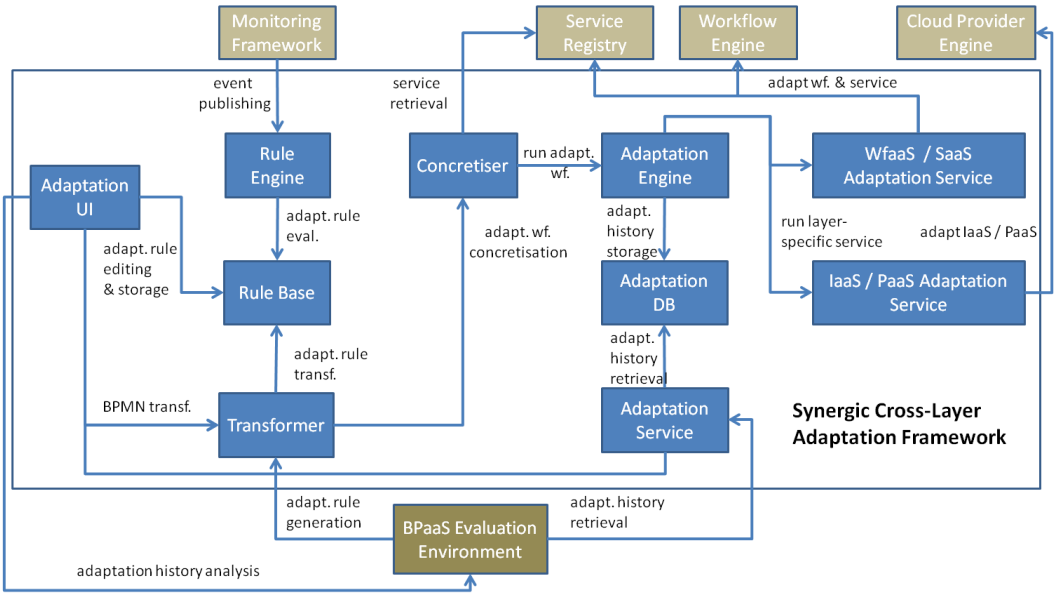

Fig. 3. The architecture of the cross-layer adaptation framework

adaptation rules firing.

The *Rule Engine* also offers an interface via which rules can be fetched and modified. This interface is exploited by the *Adaptation UI*, the main visual component enabling the expert to browse, edit, modify and store adaptation rules. This component also depicts the successability of the adaptation rules by processing the BPaaS adaptation history. This history is retrieved from the *Adaptation Service*, a service-based wrapper over the *Adaptation DB*. The *Adaptation UI* also connects to the UI component of the *Monitoring Engine* to enable the expert to observe the current BPaaS performance. In case a problematic situation occurs which is not handled by the adaptation rules injected into the system, the expert is allowed to specify a manual adaptation workflow, initially expressed in CAMEL and then mapped to BPMN by exploiting the *Transfomer*, and to execute it by invoking the corresponding method of the *Adaptation Service*.

The *Adaptation Service* does not actually execute the adaptation workflow but interacts with the *Adaptation Engine* for this execution: it first registers the workflow in the *Adaptation Engine* and then requests its execution. The latter component has been realised in the form of a normal workflow engine.

In case of normal, automatic triggering of adaptation rules, the consequent rule part maps to an adaptation workflow template in CAMEL. This template needs then to be concretised before being executed. As such, the workflow template passes from the *Rule Engine* to the *Workflow Concretiser* which takes care of mapping the workflow service tasks to respective adaptation services. Such a mapping / selection relies on the current system adaptation capabilities and on adaptation preferences or constraints that can be posed by the expert. The latter can be expressed either across all BPaaSes or for one BPaaS or even per adaptation rule / template basis. The selection is performed by considering our previous work in composite service concretisation [17], able to produce solutions even for over-constrained user requirements. Once the adaptation workflow is concretised, it is mapped to a BPMN workflow by the *Transformer* and then it is registered and executed by the *Adaptation Engine*.

The system adaptation capabilities are offered in the form of level-specific services, i.e., service-based libraries of adaptation actions able to confront specific problems in a certain level. Currently, such adaptation capabilities are covered by two services, mapping to 2 respective adaptation frameworks that constitute our previous work: (a) the service adaptation framework in [5] covering the workflow and service level; (b) the cross-cloud adaptation framework in [6] covering the platform and infrastructure level. The former framework can modify the workflow by, e.g., performing individual editing actions, such as, task skip, removal or addition, or even more sophisticated ones, such as workflow re-concretisation. It also includes well-known service adaptation capabilities, like service substitution and recomposition. That framework closely cooperates with the *Workflow Engine* in the *BPaaS Execution Environment* and the *Service Registry* to support the respective capabilities execution. The cross-cloud framework offers infrastructural adaptation capabilities mapping to horizontal scaling and migration as well as platform adaptation capabilities, such as platform service replacement. This framework closely cooperates with the *Cloud Provider Engine* in the *BPaaS Execution Environment*. More details about the *BPaaS Execution Environment*, i.e., the environment responsible for BPaaS deployment, execution, monitoring and adaptation, as well as the *Service Registry* can be found in [18].

Currently each new level-specific adaptation functionality ends up as a method of a respective adaptation service. We

are currently investigating ways via which such functionality could be automatically injected into the system without requiring to disrupt the adaptation service from executing.

Adaptation workflow execution history is maintained by the *Adaptation Engine* and stored in the *Adaptation DB*. This history, apart from detailing which service was executed, also clarifies some non-functional statistics over the performance of the whole adaptation workflow and its constituent services. It also explicates whether the workflow execution has been successful or not in dealing with the problematic situation that triggered it. Such adaptation history information is valuable for improving and optimising the BPaaS behaviour as it can enable reconfiguring the adaptation rules. As such, by wrapping this information via a service, we enable its retrieval for the following purposes: (a) to allow immediately an expert to browse the history parts and edit the problematic adaptation rules; (b) to enable performing more sophisticated analysis over this information via the *BPaaS Evaluation Environment*, which could unveil even more interesting added-value knowledge about the adaptation behaviour of a BPaaS or across BPaaSes. To this end, the latter environment has been configured to exploit the *Adaptation Service* to draw this information in regular time intervals. The expert will have the capability to either switch to the *BPaaS Evaluation Environment* to perform the analysis or to request to perform and browse the results of this analysis via the *Adaptation UI*.

A. Implementation Details

All components were realised in Java. The *Adaptation UI* relies on a combination of angular.js[3] with Ajax. The *Transformer* was realised by exploiting the Epsilon[4] transformation framework of Eclipse and its encompassing ETL[5] language. The *Rule Engine* was realised via the *Drools* [6] rule engine. The Activiti[7] framework was used to realise the *Adaptation Engine*. The *Adaptation DB* is an extension of the H2 DB supported by Activiti. In the future, we plan to move to another relational DB like mysql. All service-based components were implemented in Jersey[8]. Finally, the *Concretiser* is just a small extension to our previous service concretisation work which enables it to communicate with the *Service Registry* to discover the current system adaptation capabilities.

V. Validation

The cross-layer adaptation framework validation concerns its application in a certain use case drawn from CloudSocket. The framework has been developed in the context of this project which deals with the cross-layer BPaaS management and includes several environments handling each BPaaS management life-cycle activity. Our framework is part of the BPaaS Execution Environment (cf. Figure 1). The use case applied relates to managing and adapting the *Send Invoice* BPaaS. This BPaaS concerns the realisation of sending invoice functionality, a support BP to an organisation. The *SendInvoice* BPaaS has been designed in the form of an extensive workflow [19], shown in simplified form in Figure 4. It includes using 2 main services: (a) an external CRM service called YMENS CRM, provided by the YMENS marketplace and service provider, to connect to the CRM of an organisation and retrieve required information for the invoicing; (b) an invoice management service enabling to create invoices and send them via email to the respective customers. The latter service is deployed in an internal VM, operating an ubuntu OS, belonging to the private cloud of the organisation purchasing the BPaaS and maps to the invoice ninja[9] service component. This service operates over an internal DB, also hosted in another VM in the same private cloud.

To initially adapt the BPaaS behaviour at runtime, two simple rules have been defined, now depicted in a logic-based form. The first rule indicates that if the average CPU utilisation becomes greater than 80% for the invoice ninja component, then this component will have to be horizontally scale. The second rule maps to re-running the service in the respective VM when it is detected to be down. Such simple rule maps to the general pattern that usually software bugs are temporary.

$$R1 : cpu\_viol(i\_ninja, send\_invoice) \Rightarrow hscale(i\_ninja)$$

$$R2 : down(i\_ninja, send\_invoice) \Rightarrow re-run(i\_ninja)$$

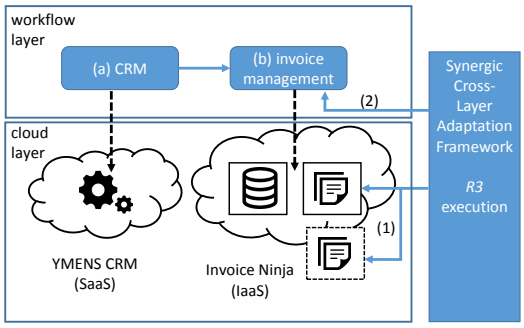

Fig. 4. Cross-layer adaptation scenario

While the BPaaS is running, the invoice ninja service component becomes unavailable and the second rule is fired. However, the component exhibits a permanent failure. As such, the *Adaptation UI* alerts the expert who immediately prepares an adaptation workflow by copying the consequent part from an adaptation rule of another BPaaS. She then issues the execution of this workflow to have the invoice ninja again up and running. The workflow logic is quite simple: the invoice ninja component has to be migrated to another VM (cf. step *(1)* in Figure 4) and the respective BPaaS workflow needs to be modified to include the new IP of the invoice ninja service (cf. step *(2)* in Figure 4). As such, this logic comprises executing 2 adaptation actions: a component migration and a service

[3]www.angularjs.org
[4]eclipse.org/epsilon/
[5]eclipse.org/epsilon/doc/etl
[6]https://www.drools.org
[7]www.activiti.org
[8]https://jersey.java.net/
[9]https://www.invoiceninja.com

substitution, offered by the two adaptation services in the architecture of the cross-layer BPaaS adaptation framework.

After executing the adaptation workflow, the expert observes that the invoice ninja service component is running again. As such, she has now the idea to express a new adaptation rule to cover the new situation encountered as follows:

$$R3 : down\,(i\_ninja, send\_invoice) \wedge failed\,(R2) \Rightarrow$$

$$seq\,(migrate\,(i\_ninja)\,, s\_replace\,(i\_ninja, send\_invoice))$$

Indeed, after incorporating this rule in the framework, she can actually observe from the adaptation history that after several executions of the *SendInvoice* BPaaS, the new rule was fired multiple times and was 100% successful.

VI. Conclusions and Future Work

This paper has presented a cross-layer, rule-based BPaaS adaptation framework that advances the state-of-the-art by covering all technical levels as well as both pro-active and re-active adaptation. This framework can dynamic concretise the adaptation workflows at runtime based on the current system adaptation capabilities. It also enables the expert to browse the adaptation history as well as edit, modify and store adaptation rules. In case of problematic situations that cannot be anticipated by the current adaptation rule set, the framework enables the expert to execute manually-defined adaptation workflows. The framework relies on a newly proposed extension of CAMEL able to specify advanced adaptation rules comprising composite cross-level adaptation workflows. It also relies on a *BPaaS Evaluation Environment* which, apart from being able to generate new adaptation rules and feed them into the system, will become capable to perform analysis over the adaptation history to discover places for optimal adaptation behaviour reconfiguration.

The following future work directions are planned. First, the framework will be thoroughly validated based on real use cases and extensively tested. Second, a more sophisticated workflow concretisation algorithm will be devised able to consider the alternative templates that can confront the current situation as well as respective concretisation possibilities for these templates in conjunction with the performance and successability of the adaptation workflows used in the past based on the adaptation history captured. Third, additional adaptation capabilities will be developed and injected into the framework spanning different levels plus alternative implementations for existing ones with different cost and performance trade-offs. It will be investigated how the injection mechanism can become more automated to include such capabilities and implementations. Finally, various forms of adaptation history analysis will be realised in the *BPaaS Evaluation Environment*.

Acknowledgement

This research has received funding from the European Community's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

References

[1] D. Karagiannis, "Bpms: Business process management systems," *SIGOIS Bull.*, vol. 16, no. 1, pp. 10–13, August 1995.

[2] D. Baur, D. Seybold, F. Griesinger, A. Tsitsipas, C. B. Hauser, and J. Domaschka, "Cloud orchestration features: Are tools fit for purpose?" in *UCC*. IEEE, 2015, pp. 95–101.

[3] A. Rossini, K. Kritikos, N. Nikolov, J. Domaschka, F. Griesinger, D. Seybold, and D. Romero, "D2.1.3 – CloudML Implementation Documentation (Final version)," PaaSage project deliverable, October 2015.

[4] J. Domaschka, F. Griesinger, K. Kritikos, D. Seybold, R. S. G. (ATOS), and C. Zegkinis, *D3.4 – BPaaS Allocation and Execution Research Prototypes*, CloudSocket European Project, December 2016.

[5] C. Zeginis, K. Konsolaki, K. Kritikos, and D. Plexousakis, "ECMAF: an event-based cross-layer service monitoring and adaptation framework," in *ICSOC 2011 Workshops*, ser. Lecture Notes in Computer Science, vol. 7221. Paphos, Cyprus: Springer, 2011, pp. 147–161.

[6] J. Domaschka, D. Seybold, F. Griesinger, and D. Baur, "Axe: A novel approach for generic, flexible, and comprehensive monitoring and adaptation of cross-cloud applications," in *ESOCC Workshops*, ser. Communications in Computer and Information Science, vol. 567. Taormina, Italy: Springer, 2015, pp. 184–196.

[7] L. Baresi, S. Guinea, and L. Pasquale, "Self-healing bpel processes with dynamo and the jboss rule engine," in *International Workshop on Engineering of Software Services for Pervasive Environments: In Conjunction with the 6th ESEC/FSE Joint Meeting*, ser. ESSPE '07. New York, NY, USA: ACM, 2007, pp. 11–20.

[8] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for ws-bpel," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 815–824.

[9] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, and S. Dustdar, "Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems," *Software: Practice and Experience*, vol. 44, no. 7, pp. 805–822, 2014.

[10] R. Popescu, A. Staikopoulos, A. Brogi, P. Liu, and S. Clarke, "A formalized, taxonomy-driven approach to cross-layer application adaptation," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 1, pp. 7:1–7:30, May 2012.

[11] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein, "Multi-layered monitoring and adaptation," in *Proceedings of the 9th International Conference on Service-Oriented Computing*, ser. ICSOC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 359–373.

[12] A. Zengin, R. Kazhamiakin, and M. Pistore, "CLAM: cross-layer management of adaptation decisions for service-based applications," in *ICWS*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 698–699. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2011.76

[13] C. Zeginis, K. Kritikos, and D. Plexousakis, "Event Pattern Discovery in Multi-Cloud Service-Based Applications," *IJSSOE*, vol. 5, no. 4, pp. 78–103, 2015.

[14] L. Schubert, J. Domaschka, and P. Guisset, "PaaSage - making cloud usage easy," in *CloudScape*, 2016. [Online]. Available: http://www.cloudwatchhub.eu/paasage-making-cloud-usage-easy

[15] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "SYBL+MELA: Specifying, Monitoring, and Controlling Elasticity of Cloud Services," in *ICSOC*. Springer-Verlag, 2013.

[16] K. Kritikos, J. Domaschka, and A. Rossini, "SRL: A scalability rule language for multi-cloud environments," in *CloudCom*. IEEE Computer Society, 2014, pp. 1–9.

[17] K. Kritikos and D. Plexousakis, "Multi-cloud Application Design through Cloud Service Composition," in *CLOUD*. New York, NY, USA: IEEE Computer Society, 2015, pp. 686–693.

[18] R. Woitsch, M. Albayrak, H. Köhn, W. Utz, A. J. Ferrer, J. Iranzo, A. Leonforte, A. Gallo, V. Mihnea, R. Pacurar, C. Avasilcai, G. Arama, R. Boca, F. Griesinger, D. Seybold, J. Domaschka, K. Kritikos, and D. Plexousakis, *D4.1 – First CloudSocket Architecture*, CloudSocket European Project, August 2015.

[19] R. Woitsch, P. Martos, K. Hinkelmann, B. Lammel, and J. Jähnert, "D5.2 bpaas reference models," CloudSocket project deliverable, Tech. Rep., 2016.