Accepted Manuscript

SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications

Polona Štefanič, Matej Cigale, Andrew C. Jones, Louise Knight, Ian Taylor, Cristiana Istrate, George Suciu, Alexandre Ulisses, Vlado Stankovski, Salman Taherizadeh, Guadalupe Flores Salado, Spiros Koulouzis, Paul Martin, Zhiming Zhao

PII: S0167-739X(18)31094-X
DOI: https://doi.org/10.1016/j.future.2019.04.008
Reference: FUTURE 4889

To appear in: *Future Generation Computer Systems*

Received date : 8 May 2018
Revised date : 2 March 2019
Accepted date : 3 April 2019

SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications

Polona Štefanič*[a], Matej Cigale[a], Andrew C. Jones[a], Louise Knight[a], Ian Taylor[a], Cristiana Istrate[b], George Suciu[b], Alexandre Ulisses[c], Vlado Stankovski[d], Salman Taherizadeh[d], Guadalupe Flores Salado[e], Spiros Koulouzis[f], Paul Martin[f], Zhiming Zhao[f]

[a] *Cardiff University, School of Computer Science and Informatics, UK*
[b] *BEIA Consult International, Romania*
[c] *MOG Technologies, Portugal*
[d] *University of Ljubljana, Slovenia*
[e] *Wellness Telecom, Spain*
[f] *University of Amsterdam, Netherlands*

Abstract

Time-critical applications, such as early warning systems or live event broadcasting, present particular challenges. They have hard limits on Quality of Service constraints that must be maintained, despite network fluctuations and varying peaks of load. Consequently, such applications must adapt elastically on-demand, and so must be capable of reconfiguring themselves, along with the underlying cloud infrastructure, to satisfy their constraints. Software engineering tools and methodologies currently do not support such a paradigm. In this paper, we describe a framework that has been designed to meet these objectives, as part of the EU SWITCH project. SWITCH offers a flexible co-programming architecture that provides an abstraction layer and an underlying infrastructure environment, which can help to both specify and support the life cycle of time-critical cloud native applications. We describe the architecture, design and implementation of the SWITCH components and describe how such tools are applied to three time-critical real-world use cases.

*Corresponding author
Email address: `StefanicP@cardiff.ac.uk` (Polona Štefanič*)

Keywords: Time-critical applications, Co-Programming model, Component-based software engineering, Quality of Service, Quality of Experience, Graphical service modelling

1. Introduction

Many industrial time-critical applications, such as disaster early warning systems, video conferencing, online gaming or live event broadcasting have highly time-critical requirements for their performance and present particular challenges for successful development, deployment and maintenance. They can only achieve their expected business value and outstanding social impact if they meet time-critical requirements, such as high performance, portability, availability, resilience and responsiveness. Furthermore, they must predict and cope with (unpredicted) peaks of load and offer rapid elasticity of on-demand computing resources and reconfigurability of underlying cloud infrastructure in order to meet the desired Quality of Service (QoS) (e.g. low response time and jitter) and Quality of Experience (QoE) (e.g. delivery of ultra-high definition television feeds) constraints.

Time-critical applications often involve distributed components and intensive data communication and may include remotely deployed field sensors in various geographical locations. However, the design, development and deployment of such applications are usually difficult and costly due to demanding requirements for the virtual runtime environment and sophisticated optimisation mechanisms needed for integrating the system components and provisioning the entire application. The cloud ecosystem provides elastic, controllable and on-demand services which can support complex time-critical applications. However, there is a lack of software engineering tools and methods for development, deployment and execution of such applications that would include programmability and controllability provided by the Clouds. Consequently, time-critical applications cannot get the full potential benefits from cloud-based technologies. Therefore, it is necessary to introduce novel software tools and approaches able to support fully the entire life cycle of time-critical applications for enhanced and optimised QoS by offering controllable and programmable features, such as (graphical) modelling of an application logic and workflow, infrastructure planning and provisioning, etc.

The aim of our research was therefore to assure self-adaptation, scalability, service availability and resilience by devising an application-infrastructure co-programming model and architecture that will provide a controllable and programmable environment for the creation of the application logic and workflow, enable reconfigurability of on-demand computing resources and underlying virtual runtime infrastructure, according to application needs.

The application-infrastructure co-programming model uses a unique architecture supported by three subsystems: SWITCH Interactive development Environment (SIDE), Dynamic Real-time Infrastructure Planner (DRIP) and Autonomous System Adaptation Platform (ASAP). SIDE provides a Graphical User Interface (GUI) for creation of software components and composition of an application's logic and workflow, and for monitoring and control of applications. Furthermore, it allows mapping application logic and workflow into TOSCA (OASIS Topology and Orchestration Specification for Cloud Applications) [1], direct manipulation of TOSCA fragments, and graphical modelling of docker compose files. The DRIP subsystem is responsible for infrastructure planning, provisioning, deployment and execution of applications in the virtual cloud infrastructure. ASAP provides monitoring services and facilitates scaling of applications, alarm triggers and self-adaptation.

The rest of this paper is organised as follows. Section 2 provides an overview of the related work. Section 3 presents the application-infrastructure co-programming model. In Section 4 we introduce the general SWITCH architecture with its subsystems, TOSCA orchestration standard and software engineering workflow in SWITCH. The example time-critical industrial cloud applications that implement SWITCH are described in section 5. We reveal the results of the evaluation in Section 6 and finally, we discuss future research options and conclude the paper with Section 7.

2. Related Work

SWITCH is not an isolated project; there are several other groups working on related problems, dealing with application composition, orchestration, deployment and adaptation of systems and workflows. However, SWITCH is unique since it is focused on time-critical applications, which are arguably the hardest to support in the current cloud ecosystem.

2.1. Cloud-based frameworks and methodologies

The ARCADIA methodology [2] offers deployment to multi-clouds and automatic real-time reconfiguration of applications. It relies on the modelling of software components in order to compose applications. Although the framework provides orchestration, Multi-Cloud deployment and a drag and drop service graph manager, it does not allow additional QoS properties to be attached to the components (e.g. QoS constraints, hardware requirements etc.); neither does it offer TOSCA manipulation.

Two service modelling tools exist for creation of Cloud applications and services. Juju [3] is a component-based graphical modelling tool for service-oriented architectures and application deployments, offering sets of predefined software assets and the relationships between them that contain knowledge of how to properly deploy and configure selected services in the Cloud. The other tool is Fabric8[1], a platform using Docker and Kubernetes as virtualisation and orchestration technologies respectively. It supports creation, deployment and continuous integration of microservices. However, these two service modelling tools do not have specific provisioning for time-critical applications, and do not offer infrastructure planning and provisioning.

On the other hand, the MODAClouds [4] methodology supports development of time-critical applications in the cloud but lacks support for software defined networking as a means of allowing programming and controlling the cloud infrastructure for performance optimisation; also it does not offer TOSCA manipulation and mapping. Finally, the CloudWave [5] and SSICLOPS [6] methodologies focus on tools for runtime monitoring of applications and services whereby Cloud services may accommodate changes in their requirements and context and meet their expected quality constraints. The CloudWave methodology proposes an architecture and implementation of Cloud benchmarking web services, however, it only measures and compares the disk speeds of different instances and storage types in Amazon EC2 and does not take into consideration the dynamic nature of the incoming data streams to deployed VMs or containers, which is one of the requirements of the SWITCH project.

Pegasus [7] encompasses an architecture and a set of technologies for execution of workflow-based applications in a variety of environments, such as clouds and grids, by automatically mapping pre-created high-level scientific workflows from the scientific domain to their execution environment. Similarly, Apache Airavata [8] enables composition, execution and monitoring of large-scale applications and workflows on distributed computing resources. It supports long running application workflows on distributed computational

[1] http://fabric8.io/guide/overview.html

resources. However, in terms of flexibility Pegasus and Airavata do not offer modification of any orchestration specification standards (e.g. TOSCA) nor do they support containerisation. The lack of support for programmability and controllabillity of application composition and the underlying architecture mean they are not suitable for time-critical applications.

The MiCADO [9] cloud orchestration framework investigates how automatic orchestration can be applied to cloud applications. As an orchestration standard TOSCA is used. However, this framework does not support mapping QoS notations into TOSCA, e.g. component-based hardware requirements, environment variables (an important requirement for SWITCH).

2.2. Cloud Infrastructure related provisioning approaches

Ensuring high QoS for real-time Cloud systems requires specialised infrastructure [10]. Infrastructure programmability and advanced virtualisation technologies, such as Software Defined Networking (SDN) [11] and Network Functions Virtualisation (NFV) [12], provide good flexibility in how infrastructure is managed and functions are deployed [13]. Time-critical requirements may be concerned simply with speed, e.g. minimising latency, or jitter, e.g. ensuring latency is kept consistent [14]. For custom infrastructure planning and optimisation, techniques such as multi-objective optimisation [15, 16] can map application requirements to infrastructure resources more effectively. This can then be used to identify violations of Service Level Agreements (SLA) [17]. For example, deadlines on the critical paths through media application workflows can be used to select virtual machines [18], automatically provisioning them even across multiple sites [19]. Transfer of application data can then be scheduled efficiently to the best sites [20, 21].

A taxonomy of (federated) Cloud computing environments is provided by Toosi *et al.* [22]. The semantic modelling of infrastructure and network may be needed for more intelligent infrastructure planning and monitoring. For example, MADL [23] uses an ontology to describe infrastructure for the storage, transportation and display of high-definition media; INDL provides ontologies for programmable network and infrastructure [24]. Such models might be used to extend cloud system specification standards such as TOSCA [25]. NDL-OWL [26] provides a Semantic Web model for networked cloud orchestration modelling network topologies, layers, utilities and technologies; it extends the Network Description Language upon which INDL is based and uses OWL. Efficient provisioning is crucial for the enhanced QoS of running applications. Therefore various optimisation approaches are highly desired, such as Multi-criteria optimisation approach for the management of Non-Functional Requirements [16].

2.3. Adaptation and monitoring related approaches

Most adaptation research has focused on finding solutions for systems that use heterogeneous infrastructure but homogeneous components. For instance, A. Llanes *et al.* [27] developed a system to balance ant colony optimisation tasks on heterogeneous infrastructure. P. Jamshidi *et al.* [28] presented a system based on fuzzy logic and the vPerfGuard [29] team developed a system that can predict performance based on low-level metrics.

Cloud applications are affected by more than just performance of the infrastructure. Network characteristics between subsystems also play a crucial role, as noted by D. Kliazovich *et al.* [30] with their CA-DAG model. In that work, the authors present Communication Aware Directed Acyclic Graphs (CA-DAG) used to model not only the performance of components but also the communications between the components.

2.4. Gap analysis

SWITCH focuses on application composition using modelling graphs and reconfiguration of underlying cloud infrastructure – by describing the functional and Non-Functional Requirements. The application-infrastructure Co-programming concept is predicated upon rapid infrastructure provisioning, deployment and reconfigurability according to network and cloud environment circumstances. In addition to application composition, an application must be monitored, adapting it according to criteria specified by the software developer. Although some elements of the SWITCH approach appear in prior work and systems, SWITCH brings these together and provides an integrated architecture and environment for application-infrastructure co-programming of an applications with time-critical constraints.

3. The concept of application-infrastructure Co-Programming

Several participants are involved in developing modern complex systems. The *component developer* is the person that creates or modifies application components, for instance a database. (S)he can add monitoring to these components in the form of prefabricated probes or special application-level metrics. Once a component has been created it can be added to the repository and its requirements and functionality described in SIDE. Note that a component developer can use a preexisting component and simply make it a SWITCH component by describing it in SIDE.

The *application developer* binds these prefabricated components together into an application while deciding different properties, such as what network they are part of or what port they are using, and sets up additional parameters such as the Alarm Trigger, etc. The *application user* uses the final application. (S)he can monitor the application and trigger adaptation, if the system was set up in this manner.

The application-infrastructure co-programming model (see Figure 1) provides abstractions and mechanisms to support QoS throughout the time-critical cloud application life cycle, by means of programmability and controllability of application logic and reconfigurability of the underlying infrastructure.

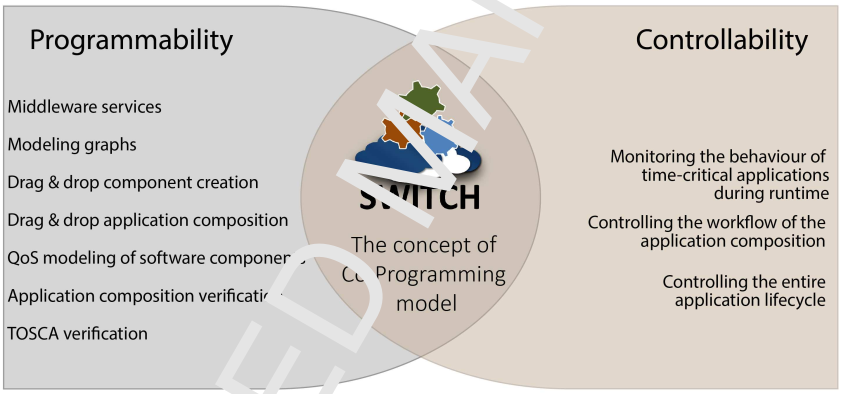


Figure 1: The concept of the application-infrastructure co-programming model which, through programmability and controllability, considers both the creation of the application logic and workflow and manipulation of the underlying (cloud) infrastructure.

Programmability of a system is its ability to be changed or manipulated, using instructions (e.g. from the software developer) that alter its behaviour. *Controllability* is a system property that denotes measuring its state, manipulating its outputs and monitoring/controlling its behaviour.

In the SWITCH workbench *programmability* is supported as follows: (1) application logic can be programmed using graphical modelling graphs with the consideration of an application's QoS parameters; (2) a virtual runtime environment for executing the application can be programmed using DRIP middleware services for the manipulation and reconfigurability of the underlying infrastructure (e.g. Software Defined Network) and on-demand resources; (3) programmable mechanisms are provided for deployment and adaptation of time-critical cloud applications; (4) QoS properties to be attached to components can be created programmatically as well. In contrast, *controllability* is achieved by (1) monitoring the behaviour of time-critical cloud applications and the underlying infrastructure at runtime (e.g. monitoring various metrics related to the application and its present state at runtime and offering possibilities to influence/reconfigure infrastructure properties if QoS is affected), and (2) controlling the workflow of component creation and application logic (e.g. by applying verification mechanisms to verify the correctness of application logic and also the correctness of TOSCA in which application logic is mapped). SWITCH checks that all constraints the component needs are provided and that the YAML description is valid.

3.1. Co-programming in comparison to DevOps and Software Defined Network

DevOps [31] is the combination of cultural philosophies, practices, and tools that increases the speed of application delivery. It automates the processes between software development and IT teams for faster building, testing, and releasing of software. The typical life cycle of an application in the DevOps process encompasses planning, building, continuous integration, deployment and operation. There are concrete tools and frameworks that support DevOps, such as Chef[2] and Jenkins[3]. On the other hand, *Software Defined Network (SDN)* offers abstraction of the network domain, and provides programmability of the network configuration. This means the network should therefore be more flexible and suitable for rapid changes. However, neither approach offers programmability of the application logic or workflow throughout the entire application life cycle.

The application infrastructure co-programming model, however, offers both programmability and controllability in the application logic design and development, and in the planning and provisioning of the virtual cloud infrastructure across the entire life cycle of time-critical applications. The unique abstraction of the co-programming model, supported by the SWITCH architecture, is designed to provide increased productivity of application design

[2] https://www.chef.io/
[3] https://jenkins.io/

and development, improved planning and provisioning and deployment efficiency, and improved QoS control efficiency. Co-programming gives the control over the application workflow and infrastructure to the developer, providing the ability to specify the constraints of the containerised microservice-based components or system during development, thus making sure that the developed components act in the manner they were intended. This minimises the chance of errors during creation, provisioning and deployment.

4. SWITCH architecture

In this section, the architecture of SWITCH and its three subsystems (SIDE, DRIP, ASAP) is presented (see Figure 2). The idea behind SWITCH is that the SWITCH subsystems are deployed in a shared environment for use in development of multiple applications.

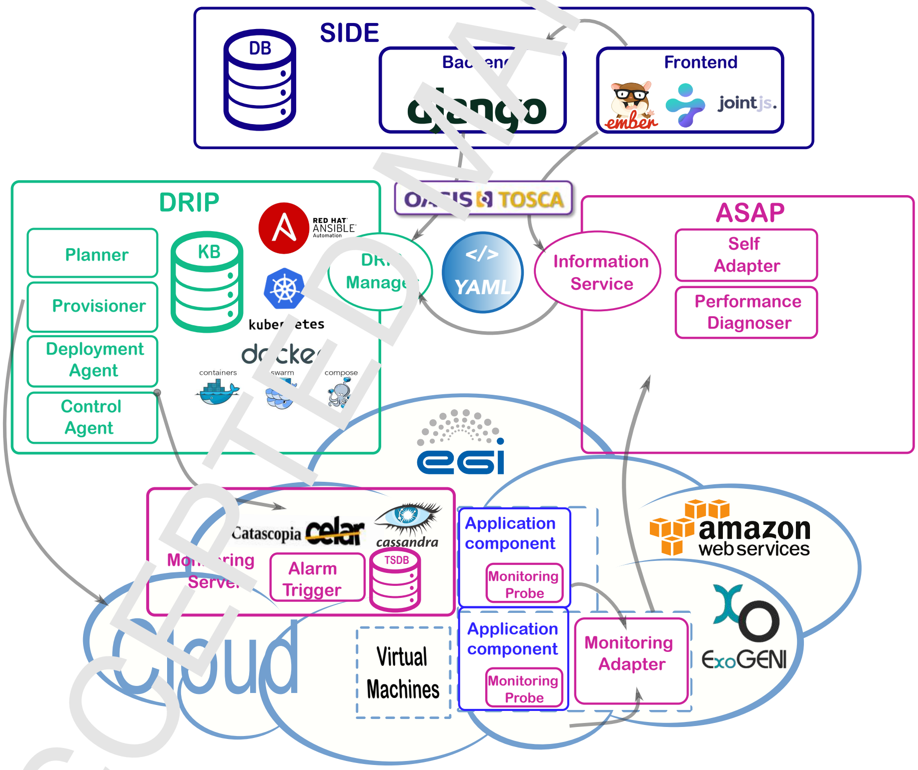


Figure 2: Overall architecture of SWITCH. The main components of the system, SIDE, DRIP and ASAP are colour-coded. Technologies used are identified by their icons or logos.

4.1. SWITCH Interactive Development Environment (SIDE)

SIDE is the interactive GUI of the SWITCH workbench. It offers interactive service modelling graphs for the design of the application workflow for containerised microservice-based cloud-native applications, and supports tasks including the following: component creation, application composition, application validation, infrastructure planning, provisioning, deployment and monitoring (see Figure 3). SIDE captures the application-infrastructure co-programming concept by giving the developer options for describing the requirements, constraints and underlying infrastructure of a system.

The SIDE frontend[4] is a GUI implemented using EmberJS technology which comprises several views, such as a component creation view and an application composition view. For the actual creation of modelling graphs the JointJS library is used and the Ember models are built on top of this.

The SIDE backend[5] uses the Django framework. It interacts with Ember Models, which provide the information on how to present the application modelling graphs. The description of an application's logic and workflow is mapped into TOSCA YAML format. The Django-based code validates the application as well. It checks if QoS parameters attached to the components are composed correctly, i.e. if they are generated into a valid YAML file, and if all mandatory parameters (e.g. hardware requirements, such as number of CPUs, amount of memory, etc.) for the specific component are defined. The backend communicates with other SWITCH subsystems by calling the APIs of DRIP and ASAP and provides generated TOSCA in which application logic and workflow are mapped. Furthermore, the SIDE backend receives the returned TOSCA for planning and provisioning and presents it to the software developer or DevOps engineer.

From the software developer's perspective, SIDE supports creation of the detailed specification of a system with dependency modelling graphs by dragging and dropping components (e.g. containerised software created from images pulled from DockerHub) onto a canvas, setting values for properties and linking them to specific components (see Figure 3). Moreover, using modelling graphs, created components can be suitably linked to one another to define the entire application logic and workflow and therefore describe the

[4]https://github.com/switch-project/SWITCH-version-2/tree/master/SIDE/side-ember

[5]https://github.com/switch-project/SWITCH-version-2/tree/master/SIDE/side_api

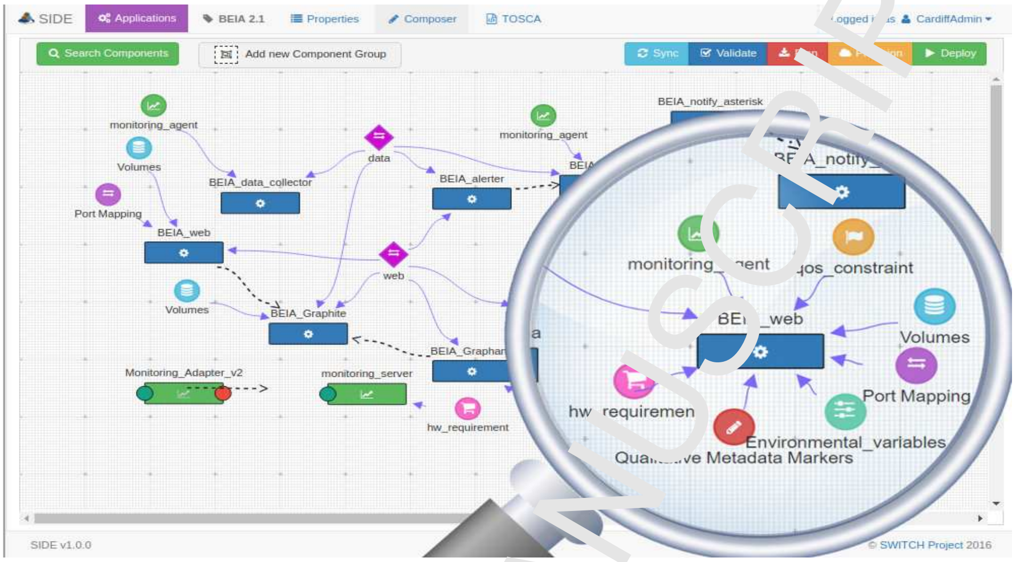


Figure 3: Example of an application composition in the SWITCH workbench. In the magnifying glass all properties that can be linked to the component and set as constraints are shown. The entire modelling graph presents the application composition of the BEIA use case. It is made up of various components to which properties are attached.

entire cloud native application. The specification of the system and underlying infrastructure can be added as well. Before mapping the application logic into TOSCA, application composition is verified and validated for errors (e.g. missing QoS parameters or incompatible components linked to one another).

An additional novelty that goes beyond the project's objectives is the notion of Qualitative Metadata Markers (QMM). These are suitable for modelling software components and were integrated into SIDE as a proof-of-concept. They give insight into which time-critical requirements have the greatest impact on the QoS. A QMM provides probabilities, showing which parameters have the greatest correlation with the QoS of a particular software component [32]. According to this information, time-critical requirements can be considered for further analysis since they are crucial for the application's QoS. The time-critical requirements with the greatest (positive or negative) influence on the QoS of the entire Cloud application can be exchanged between middleware services and are sent to a Multi-criteria decision making module. Time-critical requirements are usually mutually conflicting: altering one parameter usually has profound effects on the others. For example, increasing the availability of an application requires increased system redundancy, which can mean high operational costs. Selecting the most optimal trade-offs between multiple application runtime parameters can be a time-consuming and error-prone process, especially if considering complex Multi-Cloud environments. Our novel approach can help software engineers in the decision-making process to narrow down the number of virtual machine instances to an optimal number according to defined conflicting objectives (e.g. response time, monetary cost, etc.) [16]. Application components can then be deployed to these instances.

4.2. The Dynamic Real-time Infrastructure Planner (DRIP)

DRIP[6] is an open source service suite for automatically planning and provisioning networked virtual machines (VMs), deploying an application components and managing the resulting infrastructures at runtime. DRIP provides a holistic approach to the optimisation of resources and the satisfaction of application-level constraints such as deadlines or SLAs. DRIP can provision a virtual infrastructure across several cloud providers, and can be used to start, stop and resume execution of application components on demand. In particular, use of Open Cloud Computing Interface (OCCI) enables provisioning on multiple clouds and it supports various orchestration systems, such as Docker Swarm and Kubernetes. These functionalities are essential application-infrastructure co-programming, providing application developer with the ability to create systems that will meet their requirements.

The DRIP services (as shown in Figure 2) include:

- An infrastructure planner,
- An infrastructure provisioner,
- A deployment agent,
- Infrastructure control agents,
- A knowledge base,
- The DRIP manager,
- An internal message broker.

The *infrastructure planner* uses an adapted partial critical path algorithm to produce efficient infrastructure topologies based on application workflows and constraints by selecting cost-effective VMs [18], customising the network topology across VMs. The *infrastructure provisioner* can automate the provisioning of infrastructure plans produced by the planner onto the underlying infrastructure; it can decompose the infrastructure description and provision

[6] https://github.com/switch-project/SWITCH-version-2/tree/master/DRIP

it across multiple data centres (possibly from different providers) with transparent network configuration [19]. The *deployment agent* installs application components onto provisioned infrastructure. The deployment agent is able to schedule the deployment sequence taking network bottlenecks into account, and to maximise the fulfilment of deployment deadlines [21]. The *infrastructure control agents* are sets of APIs that DRIP provides to applications to control the scaling of containers or VMs and for adapting network flows. The *DRIP manager* is a Web service that allows DRIP functions to be invoked by external clients. Each request is directed to the appropriate component by the manager, which coordinates the components and scales them up if necessary. Resource information, credentials, performance profiles and application workflows are all internally managed via an internal *knowledge base*.

The provisioner's default provisioning interface is OCCI; it currently supports the Amazon EC2[7], EGI FedCloud[8] and ExoGeni[9] clouds. The deployment agent can deploy over Docker clusters (e.g. Docker Swarm, Kubernetes), and can deploy customised applications based on Ansible playbooks[10].

DRIP requires an application description from the software developer, identifying the specific components to be deployed along with their requirements, dependencies and constraints. This must be complemented by information about infrastructure resources (e.g. VM types and network bandwidth) obtained from the cloud providers. When a planning request arrives from SIDE (initiated by the software developer) the infrastructure planner generates a plan, which is sent from DRIP to SIDE and presented to the software developer for confirmation. A confirmed plan can then be given to the provisioner, along with necessary cloud credentials on behalf of the user if not already present in DRIP's knowledge base. DRIP provisions the planned infrastructure via interfaces offered by the selected cloud providers. The deployment agent then deploys all necessary application components onto the provisioned infrastructure from designated repositories and sets up control interfaces needed for runtime control of both application and infrastructure.

[7]https://aws.amazon.com/cn/ec2
[8]https://www.egi.eu/federation/egi-federated-cloud/
[9]http://www.exogeni.net/
[10]https://www.ansible.com/

4.3. Autonomous System Adaptation Platform (ASAP)

ASAP provides runtime adaptation and as such requires a stable and modifiable monitoring system that can be extended with additional functionalities enabling it to change system characteristics on the fly, by adding additional components, visualising system state and changing the infrastructure of the system. ASAP focuses on auto-scaling and allows for geographic orchestration (in multi-cloud environments), and multi-instance and multi-tenant operations. The ASAP subsystem (see Figure 2) comprises:

- Monitoring Probes,
- Monitoring Agents,
- Monitoring Server,
- Alarm Trigger,
- Time Series Database,
- Knowledge Base,
- Information Service,
- Performance Diagnoser,
- Self-Adapter, and
- Control Agent.

Figure 4 shows the adaptation sequence, from capturing the monitoring data on probes and agents to the final usage of this information.

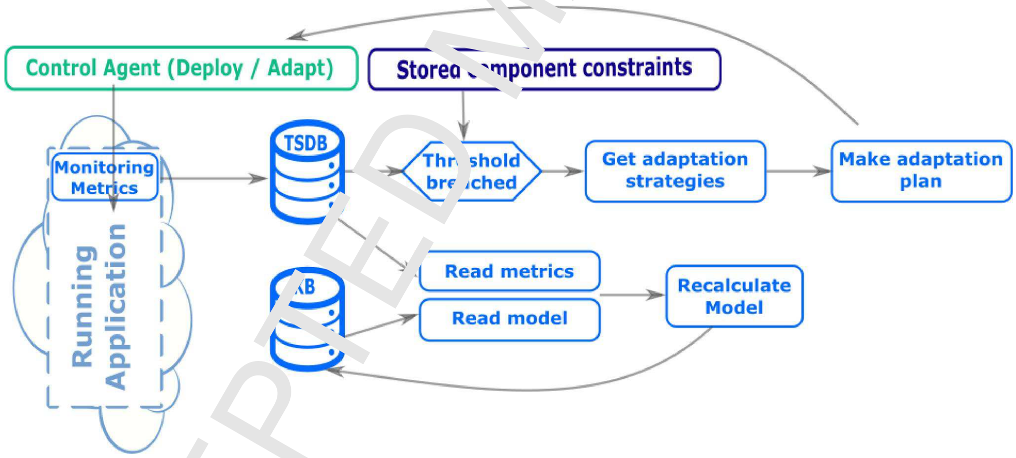


Figure 4: The dataflow of an ASAP adaptation solution.

At the first step, the purpose of *Monitoring Probes and Agents* is to collect the data that represents the current state of the application and infrastructure, and then aggregate and transfer the measured values to the *Monitoring Server* and the *Alarm Trigger*. The Monitoring Probes are lightweight, extensible and inherently decentralised. They have the ability to collect unstructured data from advanced probes, such as request process time through multiple components. The Monitoring Server receives the collected data and stores it in the *Time Series Database (TSDB)* to build a comprehensive representation of the system state. The *Performance Diagnoser* uses the information stored in the TSDB to construct a model for assessing the performance. This model is designed so that any problems that need corrective action can be identified. Concurrently, the Alarm Trigger investigates whether the measured values of monitored parameters exceed associated thresholds. When problems are detected, the *Self-Adapter* is invoked to propose suitable adaptation strategies. This component specifies a set of adaptation actions for the Control Agent allowing the transition of the whole system from its current state to the desired state. The *Control Agent*, which has the full control of application configurations and infrastructure resources (e.g. VMs/containers and network bandwidth), finally performs the adaptation actions [33]. These can be automated to restart a non-functioning component or set of components, adding a new instance of a component or moving the component to a different, potentially new VM.

In order to simplify development, an adapter was created to communicate the JCatascopia [34] messages to the Monitoring Server, without using the native Monitoring Agents. The adapter uses StatsD[11] to collect metrics from the infrastructure and feeds them to the JCatascopia Server. The infrastructure-level metrics are collected by ASAP and processed in the same way as application-level metrics. Information such as CPU, disk and memory usage is collected by the probes, and published in metric groups (e.g. 'CPUProbe', 'DiskStatsProbe' and 'MemoryProbe').

4.4. TOSCA as a SWITCH Co-Programming language

A range of data must be exchanged between the three SWITCH subsystems (SIDE, DRIP and ASAP) such as the user's specifications, application logic, time-critical constraints during an application's deployment, execution and runtime, etc. Therefore, SWITCH needs a suitable language to define and serialise such information concepts. The role of the TOSCA orchestration specification standard as an application-infrastructure co-programming language in SWITCH is to provide a format for storing programmable logic, such as dependency modelling graphs, along with the associated metadata, such as information on the quality constraints of applications, and require-

[11] https://www.librato.com/docs/kb/collect/collection_agents/stastd/

ments and dependencies among containerised software components.

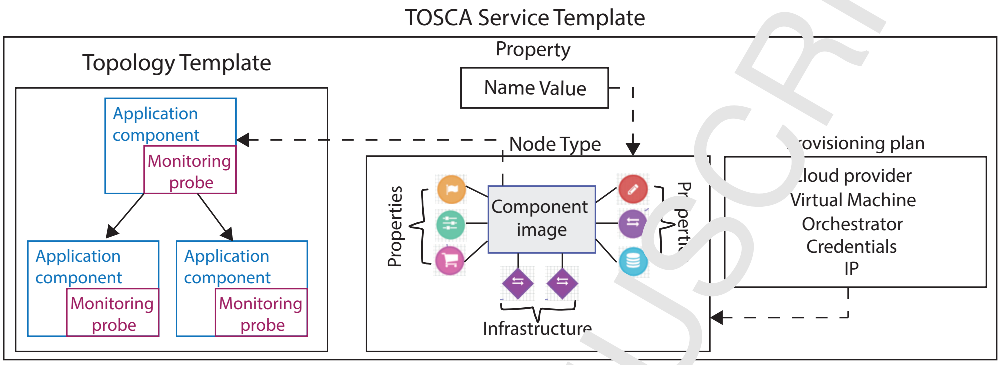


Figure 5: The TOSCA orchestration standard with its templates, application and provisioning plan description as they are mapped to TOSCA and used in the SWITCH workbench.

The core of the application logic description and workflow in TOSCA is the Service Template, which consists of a Topology Template and Management Plans, as can be seen in Figure 5. The topology template defines the structure of the application, whereas the management plans define the processes that are used to store the creation and termination information of the application during its runtime. The topology template is a directed graph containing node templates (vertices) and relationship templates (edges). Node templates contain descriptions of all (containerised) software components which are part of the application. Links, dependencies and relations between the node templates are defined by relationship templates. Node and relationship templates are typed by Node Types and Relationship Types, respectively. Types define the semantics of the templates, as well as their properties, their available management operations, and so on. As TOSCA is based on YAML, its types can be refined or extended easily. In SIDE the data is edited in a similar fashion, with the data mapping to the TOSCA (Figure 6 (B,C)). An example of QoS constraints that can be monitored and alarms set on them are presented in Figure 6 (A).

When mapping the application logic and workflow from modelling graphs into TOSCA, containerised software components with attached information on QoS parameters, such as hardware requirements (CPU, memory, . . .), QoS constraints (response time), port mapping, environment variables, etc. [35] are mapped into Node Types. Programmable and required QoS parameters

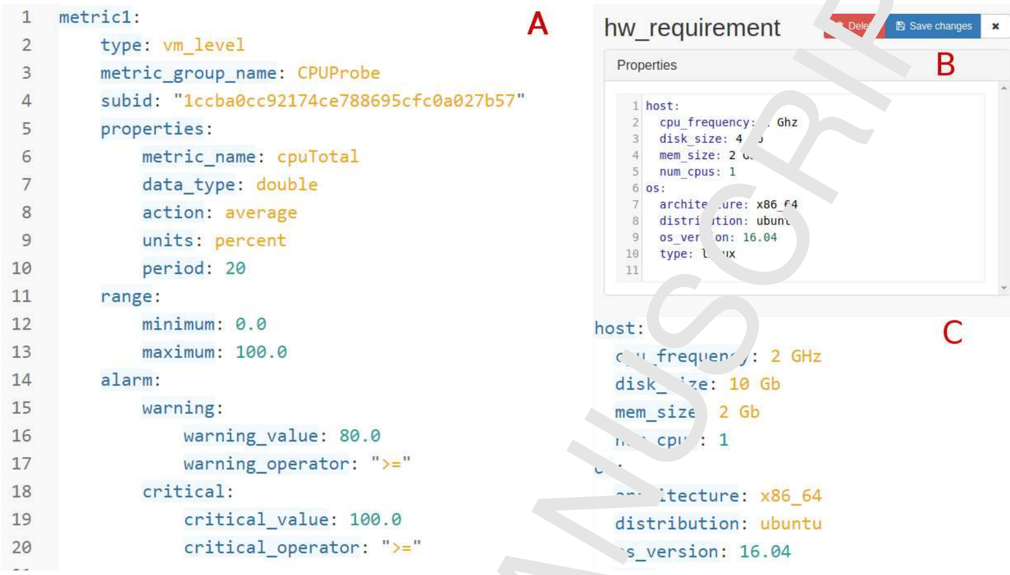


Figure 6: An example of TOSCA containing the Alarm trigger definition (A), an example of UI describing hardware requirements (B), and the corresponding entry in TOSCA (under TOSCA→ Node template→ Constraints (C)).

linked to specific components and dependencies among software components that build cloud-native applications are stored into Relationship Types.

Furthermore, the directed graph between the different node templates represented in the topology template alongside the properties and constraints (e.g. deadlines) defined for each node template, are used as input for the DRIP planner and provisioner to obtain the underlying virtual infrastructure on which the application is deployed. The specification planning and provisioning information, runtime characteristics and management of the application throughout its entire life cycle are defined using management plans.

4.5. Workflow in the SWITCH workbench

The sequence diagram in Figure 7 illustrates the workflow in SWITCH. After the user (e.g. software engineer) is successfully (1) registered and logged into the SWITCH workbench (s)he gets redirected to the dashboard where it is possible to choose between two main functionalities, such as component creation and application composition.

When creating the component, first (2) a docker image is pulled from DockerHub and stored into an internal SIDE repository (e.g. database). In

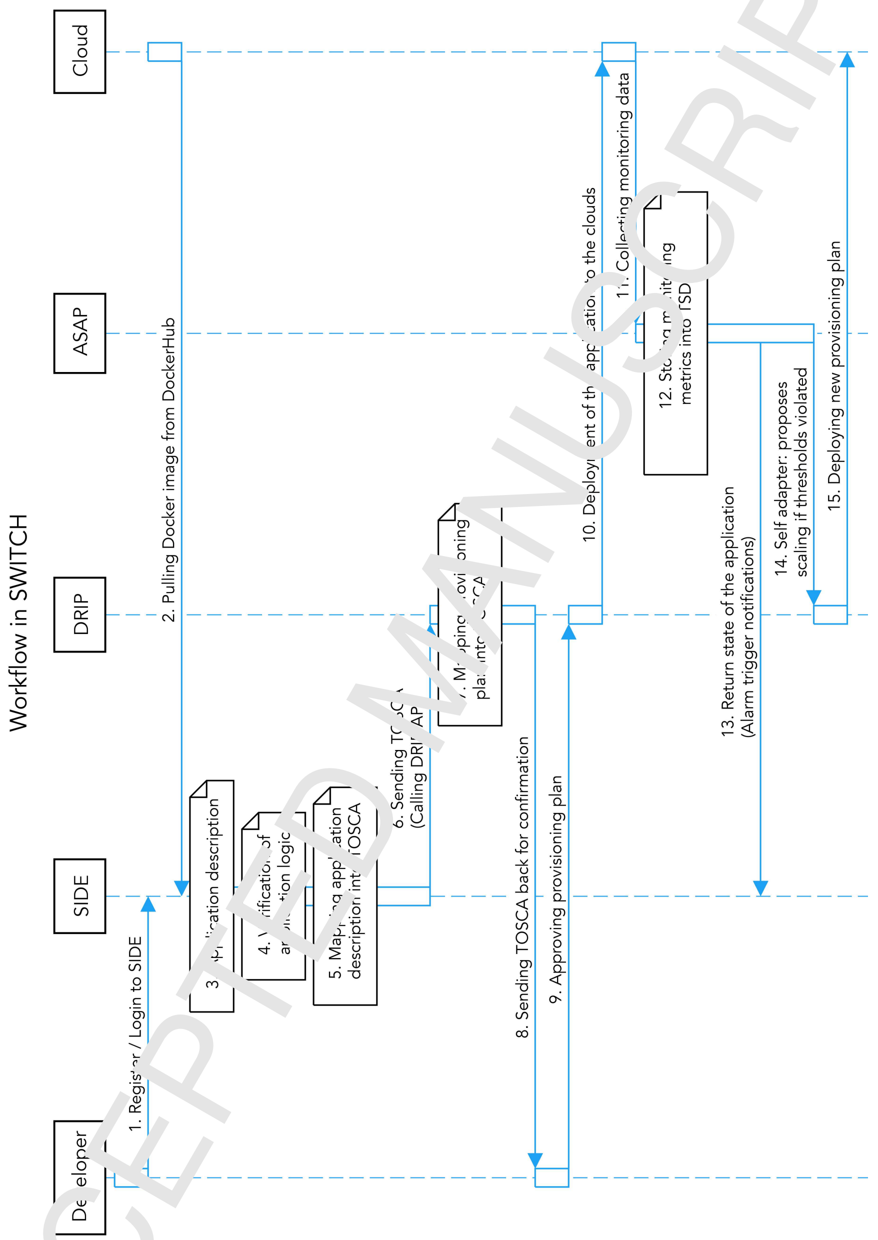


Figure [illegible]: The sequence diagram presents workflow in SWITCH workbench among all three subsystems (SIDE, DRIP, ASAP).

order to access advanced SWITCH functionalities a certain level of monitoring either through the Monitoring Adaptor or by including a JCatascopia probe must be added. Further on, (3) the application description is created using dynamic modelling graphs. Firstly, in the component creation phase a containerised component is taken from SIDE's internal repository and dragged and dropped onto the canvas. A unique and distinctive novelty in the SIDE workbench is the way additional properties (e.g. QoS constraints, hardware requirements, environmental variables, volumes, etc.) can be attached and manipulated to these containerised components using a component creation modelling graph. As can be seen in the magnified part of Figure 3 the component (*dark blue rectangle*) and various properties (*circles and diamonds*), which can be dragged onto the canvas as well, are linked to the component. With a right click on a specific property values can be set manually for that property which are mapped into TOSCA. A variety of properties can be attached to the component, such as (i) QoS constraints (e.g. response time, jitter), (ii) Hardware requirements (CPU frequency, memory etc.), (iii) Volume (enables a container to mount parts of the disk to persistent storage), (iv) Port mapping, (v) Environmental variables, (vi) Monitoring (monitoring components including the Monitoring Agent) [35].

The containerised component with linked properties is stored into the SIDE internal repository and can be (re)used and modified when composing larger multi-tier cloud native applications, via the application composition view [36]. After the application is composed (i.e. components with their properties are linked to one another and present a fully functional multi-tier microservice-based cloud-native application) (4) it is verified that all the components are correctly linked and the properties are set.

The entire application logic description is then mapped into the TOSCA orchestration standard that can be edited and manipulated in SIDE. Changing TOSCA directly also has an effect on modelling graphs. After creating TOSCA (5) it is verified for its correctness and (6) passed to DRIP via a RESTful API. According to the application description and set properties (e.g. constraints) DRIP calculates the size and amount of VMs needed for the optimal run of the application in the multi-cloud environment and (7) maps the provisioning plan into TOSCA which is (8) sent back to SIDE for confirmation. After a software engineer approves the proposed plan in SIDE (9), DRIP negotiates the SLAs of cloud providers and starts with the (10) deployment and execution of the entire application in the cloud environment. When the application is running the (11) monitoring metrics are being collected from ASAP and (12) stored into the TSDB for the Self adapter to analyse the data and monitoring server for monitoring metrics. During application runtime, (13) the Alarm trigger is returning the status of the application to SIDE. In case the thresholds for set constraints are violated (14) the Self adapter proposes scaling and sends the new plan to DRIP that (15) calculates and deploys the new provisioning plan.

5. Application to the SWITCH Use Cases

The SWITCH project was designed and tested on three industrial time-critical cloud applications. Each of these is supported by the SWITCH workbench in four ways: (1) defining the basic service components for the platform, e.g. setting up the proxy edge, the management server, the VoIP servers, the MCU Media mixer; (2) describing the application logic – sensor data collection, data storage, processing, activation of warning services, the properties for streaming services (input distributor and proxy transcoder); (3) describing the quality requirements at system, network, infrastructure and application levels, e.g. admissible percentage of packet loss or maximum latency, or defining the type of machines needed, etc.; (4) monitoring the runtime infrastructure and taking action if failure occurs (self-adaptation) or if additional resources are required to support an increased number of users. These four requirements map closely to the co-programming paradigm.

5.1. SWITCH requirements

Before the SWITCH architecture was defined, we analysed three industrial time-critical applications: an elastic disaster early warning system[12] (BEIA use case); a cloud studio for directing and broadcasting live events[13] (MOG use case); and a collaborative real-time business communication platform[14] (WT use case). These three companies would be using SWITCH to implement their solutions. Based on this, we created a minimal list of requirements that should be satisfied by SWITCH, shown in Table 1. The table presents the collected requirements identified by developers and researchers in the field. Not all features are used by all the use cases, but all the use cases have their requirements met by SWITCH.

[12]BEIA Consult, Romania, http://www.beiaro.eu/
[13]MOG Technologies, Portugal. http://www.mog-technologies.com/
[14]Wellness Telecom, Spain, http://www.wtelecom.es/

Table 1: Critical requirements that SWITCH offers within the component creation and application composition phases for the WT, MOG and BEIA use cases.

Requirement	**WT**	**MOG**	**BEIA**
Component definition	√	√	√
Component composition	√	√	√
Component configuration	√	√	√
Scalability settings	√	√	
Network characteristics	√	√	√
Multicast definition		√	
Monitoring	√	√	√
Response to system state	√		√
Manual reconfiguration		√	√
Setting up proxy	√	√	
Management of VoIP Servers	√		

The first three requirements (*Component definition, composition and configuration*) are required by all applications. They are the pieces that enable the description of the application. The *Scalability settings* enable the system to define how each component is going to scale and what the requirements are for it. For instance one of the requirements could be that a certain number of ports are available on the VM the component is running on. The description of the *network characteristics* is also important for all the use case applications, as time-critical applications are heavily dependant on the network between the user and the application, and between each component. In order to meet the changing demands of the application and the changing environment, monitoring capability is required by all the applications. Most applications require some adaptation based on the *monitored system state* or *Manual reconfiguration* if certain services cannot be adapted on the fly as this would disrupt the normal functioning of the application. Additionally to these global requirements there were some special cases that also had to be met. MOG, due to the specifics of the system required the ability to *Multicast* the data from their components. BEIA required the ability to *reconfigure proxies* for their components. WT had requirements to manage their *VoIP servers* to a finer granularity deciding, for each component and deployment, which specific servers should be used.

5.2. Switch collaborative real-time business communication platform

The Unified Communication (UC) platform (WT Use Case) is a real-time, time-critical application for an enterprise business environment that embraces communication among two or more users. The platform offers presence detection, an instant messaging service (chat), message delivery service and audio and video calls. The architecture and interaction of SWITCH and the use case is illustrated in Figure 8. To provide the desired system, the developer needs control not only over the code that is running but also the underlying architecture – the core of co-programming.

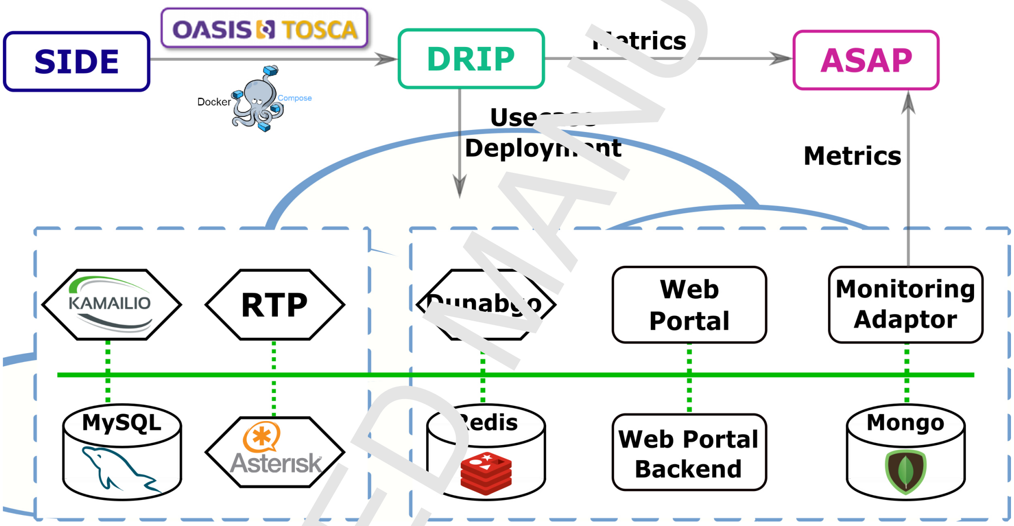


Figure 8: Architecture of the real-time UC platform.

The behaviour of the UC platform depends on the load demands of the system. In order to meet QoS requirements, the system is designed to automatically perform scaling if needed. Using SWITCH we can guarantee the traffic demand of the UC use case while maintaining the proper operation of the system no matter the workload (Figure 8). The SIDE subsystem allows developers to define the system at container level with their QoS requirements. DRIP checks the resources needed for the service before starting execution and deployment of the UC to different VMs. If the application must be scaled up, DRIP will provision new resources in a cloud environment while maintaining QoS. ASAP is responsible for monitoring and raising alarms when scaling is required.

In Table 2 time-critical requirements for the WT use case are presented. For the normal operation of Real-Time Protocol (RTP) Engine the most crucial time-critical constraints that must be satisfied are delay and jitter with 130ms and 100ms, respectively. Similarly, for Asterix PBX and Dubango WebRTC the most crucial is to satisfy jitter with threshold 150ms.

Table 2: QoS time-critical requirements in Unified Communication platform.

Component	**RTP Engine**	**Asterix PBX**	**Dubango WebRTC**
Delay (ms)	130	10	500
Jitter (ms)	100	150	150
Bandwidth (Mbps)	2	2	2
Loss Rate (%)	1	1	2
Error rate (%)	1	>1	>1

5.3. Switch elastic disaster early warning system

An elastic disaster early warning system enables people and authorities to save lives and property in case of disaster. In case of floods, a warning issued with enough time before the event will allow for reservoir operators to gradually reduce water levels, people to reinforce their homes, hospitals to be prepared to receive more patients, and authorities to prepare and provide help. The system uses advanced scaling techniques, combining VM provisioning and automatic SDN definitions to seamlessly increase the throughput of the operations during high demand and moves the location of the infrastructure in order to maintain functionality during cloud downtime. To do this the component and application performance must be monitored and maintained. In order to do this the QoS and the system requirements must be specified.

An early warning system collects data from real-time sensors, processes the information using predictive simulation tools and provides warning services for the public to obtain more information. The implementation of such a system faces several challenges, as the system must: (1) collect and process the sensor data in nearly real-time; (2) respond to urgent events rapidly; (3) predict the increase of load peaks in the network; (4) operate robustly and reliably; (5) be scalable when the amount of data increases.

A more dataflow-oriented representation is included in Figure 9. The *Data Collector* receives data from the *Remote Telemetry Station* through

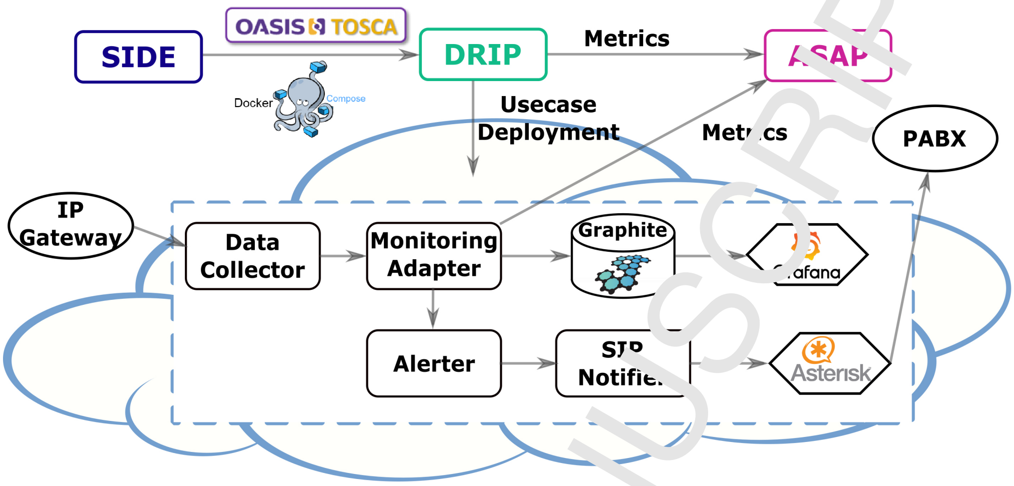


Figure 9: Functional diagram for an elastic disaster early warning system.

the *IP Gateway*. Collected data is stored in the *Graphite Database*. Data is sent to the Graphite Database through a *Monitoring Adapter*. Sending data this way is more efficient because it uses a simple protocol and a more scalable sampling. Data stored in Graphite is easily displayed in *Grafana dashboards*. When exceptional scenarios occur, the Data Collector sends a HTTP request to the *Alerter* for notifying the end-users. When the *Session Initiation Protocol (SIP) Notifier* receives a request from the *Alerter* it sends it to the *Asterisk* software which handles request and sends the notification through *PABX*.

Table 3: QoS in elastic disaster early warning system.

Component	**Graphite**	**SIP Notifier**	**IP Gateway**
Delay (ms)	10	10	500
Jitter (ms)	1	1	N/A
Bandwidth (Mbps)	40	400	>1
Loss Rate (%)	0.5	0.5	1.5
Error rate (%)	0.1	0.1	0.5

Table 3 contains the relevant metrics for the early warning system. Due to the nature of the system the SIP Notifier requires much higher bandwidth (400 Mbps) since it communicates with call centres, while Graphite requires less (40 Mbps) since it only stores the data from IP Gateways.

Dispatching the alerts to the final agents (e.g. citizens, authorities) is a time-critical component of this use case. Its elasticity mostly depends on the ability of the Notification System to handle a significant amount of call events. Each notification worker sends several application-level metrics (including the number of outgoing calls and the memory usage) to the ASAP subsystem through the Monitoring Adapter for an elastic provisioning level to be offered by DRIP by increasing/decreasing the number of workers. In order to meet these requirements the system must be described in concrete terms, specifying the values of the monitoring metrics and the actions that need to take place in order for the adaptation to occur so that it can be adapted when the number of final agents changes.

5.4. Switch cloud studio for directing and broadcasting live events

For the production of live TV events, a distributed cloud application has been developed within the SWITCH project, supported by the transmission of video over IP. Through a Web App it allows the director to perform actions such as changing the camera, selecting the number of input streams and choosing the output feed [37]. Since the cloud studio is expected to be an event-based service, i.e. it is started when it is needed and stopped when the broadcast stops, the program and the architecture that can service the system needs to be described, so that the deployment of the system can be done quickly with different starting parameters.

This is a prime example of the co-programming concept, as it enables the modification of the system - serving more cameras - and testing and maintaining performance for the system during run time.

Table 4: QoS metrics in Switch Cloud Studio.

Component	**Imput Distributor**	**Proxy Transcoder**	**Video Switcher**
Delay (ms)	30	30	30
Jitter (ms)	0.5	0.5	0.5
Bandwidth (Mbps)	130	130	130
Loss Rate (%)	>0.1	>0.1	>0.1
Error rate (%)	>0.1	>0.1	>0.1

Table 4, presents QoS metrics related to the MOG Use case. Jitter, and Loss and Error rates are of the greatest importance, while Delay is less important, as video can arrive late, as long as it arrives at the same rate.

Each *Input Distributor* node is responsible for receiving an input stream, decompressing and delivering it, by multicast, generating the resulting media flows. In this case, the relevant nodes are the *Video Switcher* and the *Proxy Transcoder*. Each *Proxy Transcoder* is responsible for transcoding the pair of media flows it has subscribed to, generating a proxy version and making it available externally, for example for a Web Application. The *Video Switcher* must subscribe to the multicast addresses that the Input Distributors are providing, store the data it receives, and serve it by multicasting the Flow that the Business Logic determines [37].

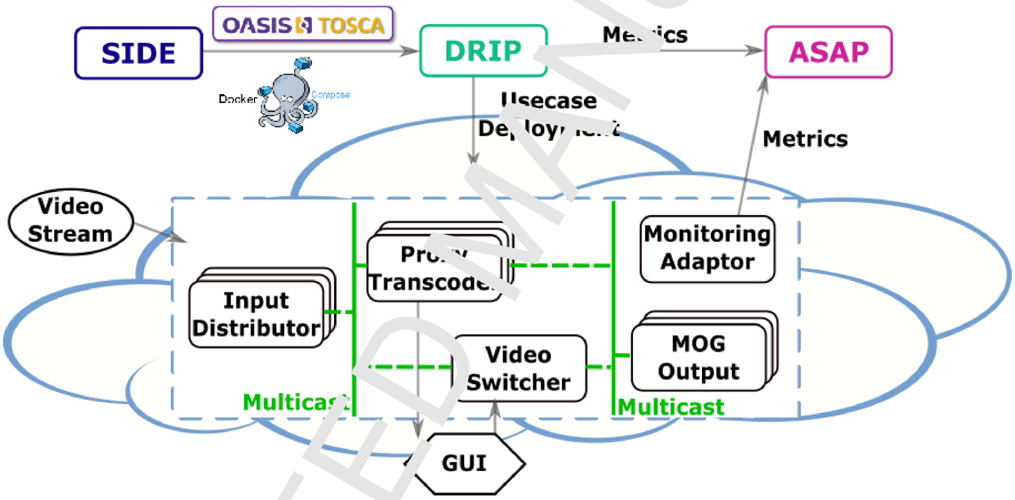


Figure 10: Live multi-stream switching in the cloud.

Each Output node receives, by multicast, video flow from the *Video Switcher* and delivers it abroad in a single stream. This means that there may be multiple Outputs, including, for example, an Output that delivers a stream with the same Input characteristics. Each component has specific properties that can be configured. The necessary connections and complexity can be added to build the desired scenario. The monitoring system (monitoring adapter and server) is added automatically if at least one of the components indicates that it has a monitoring agent attached.

6. Evaluation

Although our evaluation briefly tackles productivity, in this section we aim only to show that SWITCH IDE is capable of supporting software development of cloud-native applications with co-programming efficiently throughout their entire life-cycle. On the other hand, more evaluation, on real-world tasks and with control groups, would be needed in order to prove that productivity is improved by using SWITCH [38]. Most of productivity measurements focus on Lines of Code, which cannot be used in our case, as SIDE is closely related to graphic programming languages [39].

For the purpose of evaluation, we chose six academic researchers from the field of distributed cloud computing and DevOps engineering. For all the participants, we provided detailed instructions explaining how to create all three use cases with and without SWITCH. Participants were aware of our work and as experts in the field they are familiar with composing Docker-based cloud applications. Time was measured in minutes using stop watch and we were present the entire time of the experiment.

The participants were provided with instructions on how to use SIDE and on creating the TOSCA and Docker compose files. The instructions on how to create an application were provided to the test subjects, so that they only had to worry about how to describe the use case and not spend time on the use case architecture.

A clean install of SWITCH was used so that components could not be reused, but the participants were told that they are free to reuse components they create if they wish. During the creation of the application, time was kept for each stage of application creation (e.g. component creation, component modification (optional), application composition, and create the TOSCA and Docker compose files).

In the first stage of the experiment, participants were asked to describe all the containers (Figure 12) used in the application and the application itself. They were given all the information about the properties of the components (ports, docker image locations, volumes, variables etc.) and how they should be linked to one another. According to the time needed (measured in minutes) for software components description using SWITCH and creation of writing description of those components directly into TOSCA, we have calculated distribution (see Figure 12) that has revealed more consistency (a lot of users have similar times) when using SWITCH since it application logic and workflow are mapped into TOSCA fast and automatically.

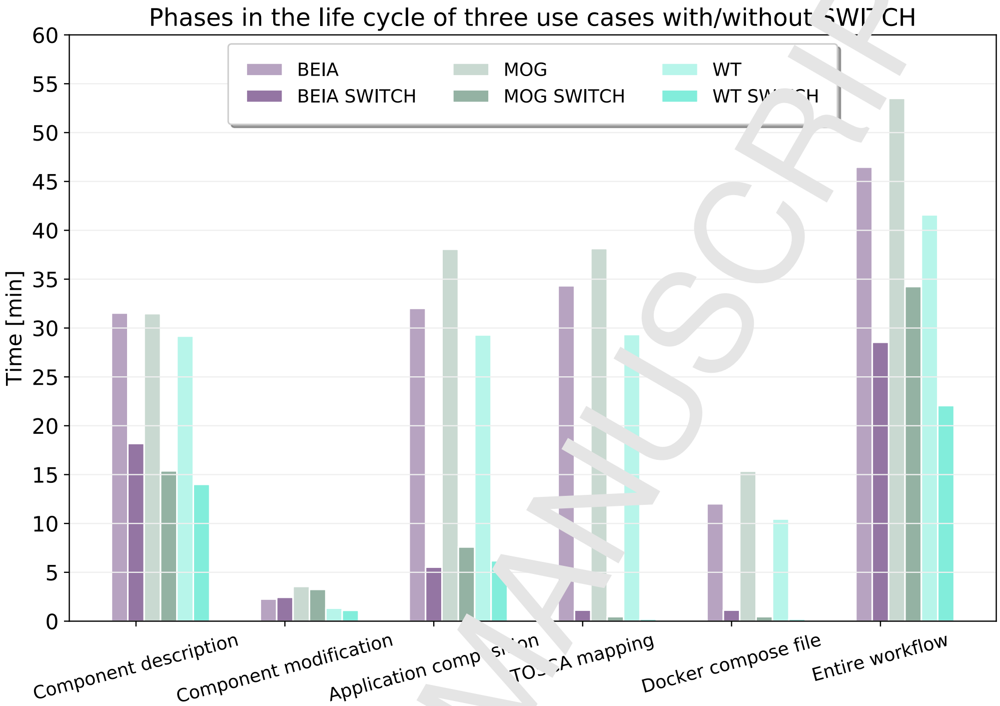


Figure 11: This bar chart illustrates the times (in minutes) needed to undertake various phases that are part of cloud application's life cycle using SWITCH and no SWITCH for all three beforehand described industrial use cases.

In the second phase, the participants were told to describe the same applications by creating the TOSCA and Docker Compose file for all three applications in Visual Studio Code. They were, again, provided with an example of the descriptions and expected to use code completion and copy paste to achieve their goals as fast as possible. At the end, the descriptions were checked in order to ascertain if they meet the TOSCA standards and that all the references were correct, but the descriptions were not used to deploy actual applications. The times needed to complete each phase in the life cycle of all three applications are presented in Figure 11. Values on the y axis present an average of all participants for each of the phases and for all three use cases.

According to the results, SWITCH IDE has obviously speeded up the implementation of all phases (and for all three use cases) that are part of the application's life cycle in comparison to the creation of components, TOSCA

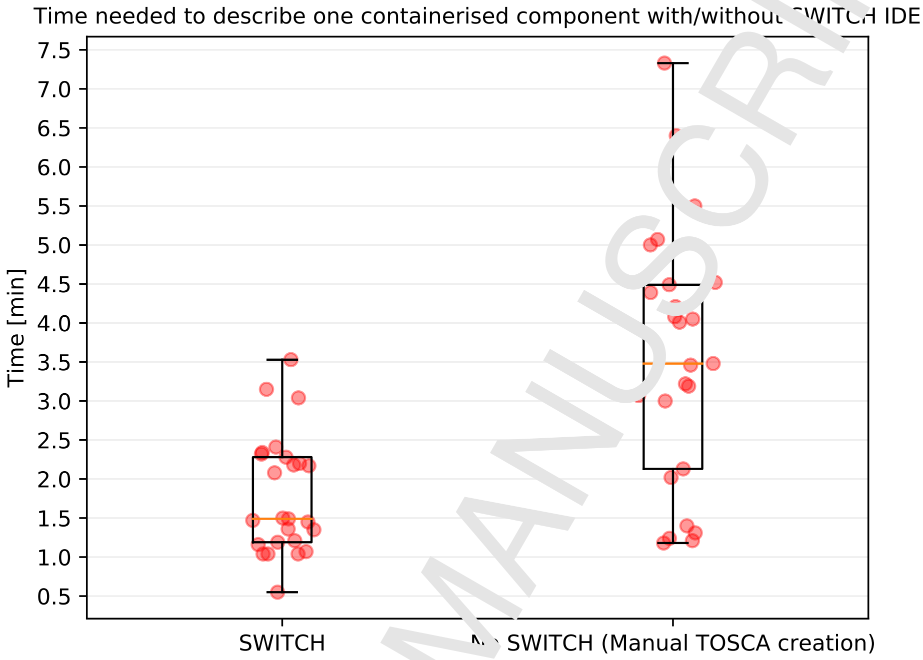


Figure 12: The plot presents the distribution of time it takes to describe software component in SIDE and its mapping into TOSCA vs. providing TOSCA description manually. Each red point on the chart presents average time needed to design each component for all three applications.

and Docker compose files manually. When comparing creation with/without SWITCH, it is clear that there exists significant decrease of the time needed for creation of any phase using SWITCH. Moreover, the most substantial difference can be seen with *TOSCA and Docker compose file* creation that is approximately more that 50 times faster on the average for all three use cases due to automatic TOSCA mapping and Docker compose file generation. The most relevant characteristic is the *Entire Workflow*, as it represents the actual time needed to create an application which is on average almost twice as faster as creating workflow manually.

7. Conclusion

In this paper, we have presented a new concept for engineering complex adaptable cloud systems with time-critical constraints: the application-infrastructure co-programming model. It offers programmability and controllability and reconfiguration of application logic composition and workflow and virtual environment and therefore offers application scalability, availability, resilience and self-adaptation. These are the essential QoS properties that are crucial for the QoE and present particular challenges specially for time-critical cloud applications.

According to the analysis of functional and non-functional requirements of three time-critical industrial applications, we have discovered that programmable and controllable features can be best supported by having unique three-part SWITCH architecture. SWITCH Interactive development Environment (SIDE) that provides a GUI with service modelling tools of docker compose files for the creation of software components and the composition of an application's logic and workflow; Dynamic Real-time Infrastructure Planner (DRIP) is responsible for the infrastructure planning, provisioning, deployment and execution of applications to the virtual cloud infrastructure; Autonomous System Adaptation Platform (ASAP) provides monitoring services and deals with the scaling of applications, Alarm trigger and self-adaptation. In order to exchange data within all three subsystems application logic with all its constraints QoS parameters and application workflow are mapped into the OASIS TOSCA.

The novelty of the SWITCH system is the way that QoS parameters, such as NFR and network-, infrastructure- and application-level metrics can be visually presented, managed and linked to the components (e.g. containers) using graphical modelling. Furthermore, QoS parameters etc. are mapped into TOSCA and exchanged between the three subsystems.

As a result of the evaluation, using SWITCH for the creation of all three industrial applications with time-critical constraints through various phases in the life cycle of cloud-native applications (e.g. components and application creation, Docker compose file creation and TOSCA mapping) significantly decreases time due to the SWITCH co-programming properties. On the contrary, manually creating components and application, generating and mapping the entire application logic into TOSCA has proven to be considerably time consuming and process. The most significant difference among using SWITCH and manual creation was achieved in the process of TOSCA and Docker compose file generation for all three use cases and in the favour of SWITCH.

In addition to developing and demonstrating the effectiveness of the SWITCH architecture, we went beyond the project's objectives and also developed an Multi-Objective Optimisation approach for the trade off between conflicting Non-Functional Requirements in order to assure enhanced QoS. However, details of this latter approach are out of scope of the present paper and can be found elsewhere [16].

One thing that is still missing is a larger-scale trial with applications during their whole life-cycle, changing and updating the software in an iterative manner. This is only possible with a longer running successful application, something that will probably only be available in the next couple of years. During this time, SWITCH will not be abandoned. On the contrary, since graphical modelling of software components proved to be time saving for the creation of applications and reasonably easy to process. We are planning to create so called (1) Dynamic Metadata Documents Generating System that would be able to generate various types of documents, such as .yaml, .xml, Docker compose and similar based on application's QoS properties and (2) Applications Offline and Runtime State Snapshot Versioning System that would create and store a snapshot of created application's logic and workflow of a running state in the virtual infrastructure and be available from the internal SWITCH repository and reusable in other cloud environments. In general, we will follow state-of-the-art trends and strive towards novel ideas. Furthermore, extending TOSCA in order to support orchestration of applications that sent an enormous amount of (Big) data and run towards the fog and edge of the network will certainly be a challenge as well.

Acknowledgement

Funding: This work was supported by the European Union's Horizon 2020 research and innovation program [grant agreement No 643963 (SWITCH project)].

References

[1] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner, OpenTOSCA – A Runtime for TOSCA-Based Cloud

Applications, in: Proceedings of the 11th International Conference on Service-Oriented Computing – Volume 8274, ICSOC 2013, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 692–695. doi:10.1007/978-3-642-45005-1_62.

[2] N. Koutsouris, A. Voulkidis, K. Tsagkaris, A Framework to Support Interoperability in IoT and Facilitate the Development and Deployment of Highly Distributed Cloud Applications, in: N. Mitton, H. Chaouchi, T. Noel, T. Watteyne, A. Gabillon, P. Capolsini (Eds.), Interoperability, Safety and Security in IoT, Springer International Publishing, Cham, 2017, pp. 41–48. doi:10.1007/978-3-319-52727-7_6.

[3] K. Baxley, J. de la Rosa, M. Wenning, Deploying workloads with Juju and MAAS in Ubuntu 14.04 LTS, Tech. rep. (may 2014).
URL https://linux.dell.com/files/whitepapers/Deploying_Workloads_With_Juju_And_MAAS.pdf

[4] E. D. Nitto, M. A. A. d. Silva, D. A. G., Casale, C. D. Craciun, N. Ferry, V. Muntes, A. Solberg, Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach, in: 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE, Timisoara, Romania, 2013, pp. 417–423. doi:10.1109/SYNASC.2013.61.

[5] D. Bruneo, T. Fritz, S. Keidar-Barner, P. Leitner, F. Longo, C. Marquezan, A. Metzger, K. Pohl, A. Puliafito, D. Raz, A. Roth, E. Salant, I. Segall, M. Villari, Y. Wolfsthal, C. Woods, CloudWave: Where adaptive cloud management meets DevOps, in: IEEE Symposium on Computers and Communications (ISCC), Vol. Workshops, IEEE, 2014, pp. 1–6. doi:10.1109/ISCC.2014.6912638.

[6] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, M. Wójcik, Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17, ACM, New York, USA, 2017, pp. 29–42. doi:10.1145/3098822.3098825.

[7] E. Deelman, K. Vahi, M. Rynge, G. Juve, R. Mayani, R. F. da Silva, Pegasus in the Cloud: Science Automation through Workflow Technologies, IEEE Internet Computing 20 (1) (2016) 70–76. doi:10.1109/MIC.2016.15.

[8] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, A. Slominski, A. Douma, S. Perera, S. Weerawarana, Apache Airavata: A Framework for Distributed Applications and Computational Workflows, in: Proceedings of the 2011 ACM Workshop on Gateway Computing Environments, GCE '11, ACM, New York, USA, 2011, pp. 21–28. doi:10.1145/2110486.2110490.

[9] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, G. Terstyanszky, Micado-microservice-based cloud application-level dynamic orchestrator, Future Generation computer Systems (2017) 1–10.

[10] W. T. Tsai, Y.-H. Lee, Z. Cao, Y. Chen, B. Xiao, RTSOA: Real-Time Service-Oriented Architecture, in: 2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06), Shanghai, China, 2006, pp. 49–56. doi:10.1109/SOSE.2006.27.

[11] D. Meyer, The Software-Defined-Networking Research Group, IEEE Internet Computing 17 (6) (2013) 84–87. doi:10.1109/MIC.2013.122.

[12] M. Chiosi, D. Clarke, P. Willis, A. Reid, Network Functions Virtualisation, Tech. rep., Darmstadt-Germany (oct 2012).
URL http://portal.etsi.org/NFV/NFV_White_Paper.pdfurl

[13] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, Near optimal placement of virtual network functions, in: 2015 IEEE Conference on Computer Communications (INFOCOM), IEEE, 2015, pp. 1346–1354. doi:10.1109/INFOCOM.2015.7218511.

[14] Z. Zhao, P. Martin, C. de Laat, K. Jeffery, A. Jones, I. Taylor, A. Hardisty, M. Atkinson, A. Zuiderwijk, Y. Yin, Y. Chen, Time critical requirements and technical considerations for advanced support environments for data-intensive research, in: 2nd International workshop on Interoperable infrastructures for interdisciplinary big data sciences (IT4RIs 16), in the context of IEEE Real-time System Symposium (RTSS), Porto, Portugal, 2016, pp. 1–10. doi:10.5281/zenodo.204756.

[15] A. F. Antonescu, A.-M. Oprescu, Y. Demchenko, C. de Laat, T. Braun, Dynamic optimization of SLA-based services scaling rules, in: IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), , Vol. 1, IEEE, Bristol, UK, 2013, pp. 282–289. doi:10.1109/CloudCom.2013.44.

[16] P. Štefanič, D. Kimovski, G. Suciu, V. Stankovski, Non-functional requirements optimisation for multi-tier cloud applications: An early warning system case study, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2017, pp. 1–8. doi:10.1109/UIC-ATC.2017.8397637.

[17] C. Müller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortés, M. Rodríguez, Comprehensive Explanation of SLA Violations at Runtime, IEEE Transactions on Services Computing 7 (2) (2014) 168–183. doi:10.1109/TSC.2013.45.

[18] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, Z. Zhao, Planning virtual infrastructures for time critical applications with multiple deadline constraints, Future Generation Computer Systems 75 (2017) 365–375. doi:10.1016/j.future.2017.02.001.

[19] H. Zhou, Y. Hu, J. Wang, P. Martin, C. D. Laat, Z. Zhao, Fast and Dynamic Resource Provisioning for Quality Critical Cloud Applications, in: 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), IEEE, 2016, pp. 92–99. doi:10.1109/ISORC.2016.22.

[20] L. M. Vaquero, A. Celorio, F. Cuadrado, R. Cuevas, Deploying Large-Scale Datasets on-Demand in the Cloud: Treats and Tricks on Data Distribution, IEEE Transactions on Cloud Computing 3 (2) (2015) 132–144. doi:10.1109/TCC.2014.2360376.

[21] Y. Hu, J. Wang, H. Zhou, P. Martin, T. Arie, C. De Laat, Z. Zhao, Deadline-aware Deployment for Time Critical Applications in Clouds, in: Proceedings of Euro-Par 2017: Parallel Processing: 23rd International Conference on Parallel and Distributed Computing, 2017, 2017, pp. 345–357. doi:10.1007/978-3-319-64203-1_25.

[22] A. N. Toosi, R. N. Calheiros, R. Buyya, Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey, ACM Computing Surveys (CSUR) 47 (1) (2014) 7:1–7:47. doi:10.1145/2593512.

[23] O. D. Ntofon, D. K. Hunter, D. Simeonidou, Towards semantic modeling framework for future service oriented Networked Media Infrastructures, in: 2012 4th Computer Science and Electronic Engineering Conference (CEEC), 2012, pp. 200–205. doi:10.1109/CEEC.2012.6375405.

[24] M. Ghijsen, J. Van Der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, C. de Laat, A semantic-web approach for modeling computing infrastructures, Computers & Electrical Engineering 39 (8) (2013) 2553–2565. doi:https://doi.org/10.1016/j.compeleceng.2013.08.011.

[25] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, TOSCA: Portable Automated Deployment and Management of Cloud Applications, in: A. Bouguettaya, Q. Z. Sheng, F. Daniel (Eds.), Advanced Web Services, Springer New York, New York, USA, 2014, pp. 527–549. doi:10.1007/978-1-4614-7535-4_22.

[26] I. Baldine, Y. Xin, A. Mandal, C. H. Renci, U.-C. J. Chase, V. Marupadi, A. Yumerefendi, D. Irwin, Networked cloud orchestration: A geni perspective, in: 2010 IEEE Globecom Workshops, 2010, pp. 573–578. doi:10.1109/GLOCOMW.2010.5700385.

[27] A. Llanes, J. M. Cecilia, A. Sánchez, J. M. García, M. Amos, M. Ujaldón, Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization, Cluster Computing 19 (1) (2016) 1–11. doi:10.1007/s10586-016-0534-4.

[28] P. Jamshidi, C. Pahl, N. C. Mendonca, Managing Uncertainty in Autonomic Cloud Elasticity Controllers, IEEE Cloud Computing 3 (3) (2016) 50–60. doi:10.1109/MCC.2016.66.

[29] P. Xiong, C. Pu, X. Zhu, R. Griffith, vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments, in: Proceedings of the 4th ACM/SPEC international conference on International conference on performance engineering - ICPE '13, ACM Press, New York, USA, 2013, pp. 271–282. doi:10.1145/2479871.2479909.

[30] D. Kliazovich, J. E. Pecero, A. Tchernykh, P. Bouvry, S. U. Khan, A. Y. Zomaya, CA-DAG: Communication-Aware Directed Acyclic Graphs for Modeling Cloud Computing Applications, in: 2013 IEEE Sixth International Conference on Cloud Computing, 2013, pp. 277–284. doi:10.1109/CLOUD.2013.40.

[31] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, IEEE Software 33 (3) (2016) 42–52. doi:10.1109/MS.2016.64.

[32] P. Štefanič, M. Cigale, A. Jones, V. Stankovski, Quality of Service Models for Microservices and Their Integration into the SWITCH IDE, in:

2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, Arizona, 2017, pp. 215–218. doi:10.1109/FAS-W.2017.150.

[33] S. Taherizadeh, I. Taylor, A. Jones, Z. Zhao, V. Stankovski, A network edge monitoring approach for real-time data streaming applications, in: International Conference on the Economics of Grids, Clouds, Systems, and Services, 2016, Springer International Publishing, Cham, 2016, pp. 293–303. doi:10.1007/978-3-319-61920-0_21

[34] D. Trihinas, G. Pallis, G. D. Dikaiakos, Jcatascopia: Monitoring elastically adaptive applications in the cloud, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2014, pp. 226–235. doi:10.1109/CCGrid.2014.41.

[35] P. Štefanič, M. Cigale, F. Q. Fernandez, D. Rogers, L. Knight, A. Jones, I. Taylor, TOSCA-based SWITCH Workbench for application composition and infrastructure planning of time-critical applications, in: The 3rd edition in the series of workshop on Interoperable infrastructures for interdisciplinary big data sciences (IT4RIs 18), Amsterdam, Netherlands, 2018, pp. 1–9. doi:10.5281/zenodo.1162872.

[36] P. Štefanič, M. Cigale, A. Jones, L. Knight, D. Rogers, F. Q. Fernandez, I. Taylor, Application-Infrastructure Co-Programming: managing the entire complex application life cycle, in: 10th International Workshop on Science Gateways (IWSG 2018), Edinburgh, UK, 2018.

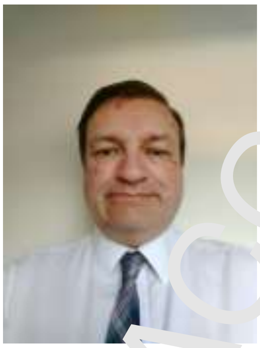
[37] M. Poeira, P. Santos, A. Ulisses, D. Costa, P. Ferreira, R. Amor, An architecture for time critical IP broadcasting in the cloud, in: Proceedings of 2nd International Workshop on Interoperable infrastructures for interdisciplinary big data sciences (IT4RIs 16) in the context of IEEE Real-time System Symposium (RTSS), Porto, Portugal, 2016, pp. 1–4. doi:10.5281/zenodo.204821.

[38] J. L. Krein, A. C. MacLean, C. D. Knutson, D. P. Delorey, D. L. Eggett, Impact of programming language fragmentation on developer productivity: a sourceforge empirical study, International Journal of Open Source Software and Processes (IJOSSP) 2 (2) (2010) 41–61.

[39] A. Idri, F. azzahra Amazal, A. Abran, Analogy-based software development effort estimation: A systematic mapping and review, Information and Software Technology 58 (2015) 206–230.

Polona Štefanič received her B.Sc. degree in computer science from University of Ljubljana, Slovenia in 2015. After finishing her studies, she worked as a Research Assistant on the European Horizon2020 project ENTICE at University of Ljubljana; following that post she got the Research Assistant position at Cardiff University where she worked on the SWITCH project. Currently she is Teacher at Cardiff University and is working towards her Ph.D. degree at University of Ljubljana. Her research interests include distributed cloud computing, software engineering of cloud applications and management of Non-Functional Requirements within the entire cloud application lifecycle. She works occasionally as a Freelance Full Stack Developer.

Matej Cigale received his BSc from University of Ljubljana, Faculty of Computer Science and Informatics. He started working in the industry until he enrolled to Ph.D. in 2015 in the field of Artificial Intelligence. He worked on the European H2020 SWITCH project first at University of Ljubljana and then at Cardiff University, School of Computer Science and Informatics. Currently he is employed at Jozef Stefan Institute on the INSENSION and CrowdHEALTH H2020 projects, researching in the smart algorithms related to health of people where he uses various machine learning algorithms to provide insight into their state and wellbeing.



Andrew Jones is a Senior Lecturer and Director of Learning & Teaching in the School of Computer Science & Informatics at Cardiff University (UK). Until recently his research has been mainly in the field of biodiversity informatics, particularly in the areas of scientific workflows and interoperability using distributed resources, and cross-mapping between scientific taxonomies. He has participated in a number of EC FP7 and UK Research Councils-funded projects. More recently he has focused on environments and techniques to support development of cloud-based software - in particular, in the EC H2020 SWITCH project.

Louise Knight received her BSc and PhD degrees in Computer Science, both from Cardiff University, in 2013 and 2018 respectively. Her PhD project concerned the parallelisation of Bioinformatics algorithms using CUDA-enabled graphics cards. Since completing her PhD, she has worked on the H2020 SWITCH project, investigating how SWITCH may support CUDA in the future, and she currently works as a Teacher at Cardiff University. Her research interests include GPGPUs, Cloud computing, and time-critical applications.

Ian Taylor is a Professor at the University of Notre Dame and a Reader at Cardiff University. He has a degree in Computing Science, a Ph.D. studying neural networks applied to musical pitch and he designed/implemented the data acquisition system and Triana workflow system for the GEO600 gravitational wave project. He now specializes in Blockchain, open data access, Web dashboards/APIs and workflows. Ian has published over 180 papers (h-index 41), 3 books and has won the Naval Research Lab best paper award in 2010, 2011 and 2015. Ian acts as general chair for the WORKS Workflow workshop yearly at Supercomputing.

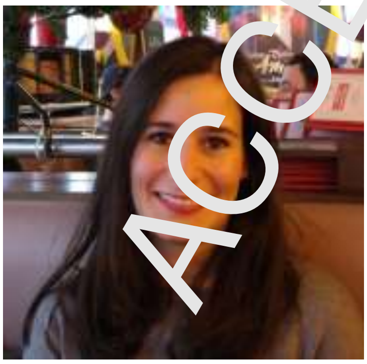
Cristiana Istrate is a student in the 4th year at Electronics, Telecommunications and Information Technology, specialization Technologies and Telecommunications Systems, UPB (University "Politehnica" of Bucharest) and a research and development assistant at BEIA Consult International. The articles in which she actively contributed were: "Real-Time Telemetry System for Emergency Situations using SWITCH", presented at the 3rd edition of IT4RIs, "Integrated Software Platform for Mobile Malware Analysis", presented at the 3rd edition of Fabulous. Currently she is involved in the following projects: E-STAR that requires the design of a PLM platform, SWITCH and ODSI that demands the design of new security models.

George Suciu holds a PhD in cloud communications from the University POLITEHNICA of Bucharest. Also, he holds a MBA in Informatics Project Management and IPR from the Faculty of Cybernetics, Statistics and Economic Informatics of the Academy of Economic Studies Bucharest, and currently, his post doc research work is focused on the field of cloud communications, blockchain, big data and IoT/M2M. George has experience as coordinator for over 30 R&D projects (FP7, H2020, Eureka / Eurostars, etc.) and is currently involved in 10 international and 5 national projects. He is the author or co-author of over 150 journal articles and scientific conference papers and holding over 5 patents. He is R&D and Innovation Manager at BEIA Consult International.

Alexandre Ulisses is currently the Innovation Director of MOG Technologies where he is coordinating several EU and national R&D projects. He worked for several years as a researcher in INESCPORTO where he was involved in various projects in broadcasting and video coding. He is a lecturer on multimedia and audiovisual topics at the Polytechnic Institute of Viana do Castelo. He was also a project manager at the Portuguese National Innovation Agency where he managed several innovation oriented projects. He was also chairman of the ICT Sector Group of the EEN, the largest innovation and technology transfer network worldwide.

Vlado Stankovski is an Associate Professor of Computer Science focusing on Distributed, Grid, Cloud and Edge Computing, employed at the Faculty of Civil and Geodetic Engineering, University of Ljubljana. He has been the technical manager of the FP6 DataMiningGrid project, one of the managers of the FP6 InteliGrid project and took part in the FP7 mOSAIC Cloud project. He is currently taking part in two H2020 SWITCH and ENTICE projects in software engineering for big data and advanced cloud computing.

Salman Taherizadeh has been working and taking active part in two H2020 projects called SWITCH (Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications) and ENTICE (dEcentralised repositories for traNsparent and efficienT vIrtual maChine opErations). Salman Taherizadeh is currently employed as PhD researcher at Jožef Stefan Institute (JSI), and he is currently taking part in the PrEstoCloud (Proactive Cloud Resources Management at the Edge for Efficient Real-Time Big Data Processing) project. His research is focused on highly adaptive time-critical cloud and edge computing applications. He has published works in the Computer Journal (Oxford University Press), Journal of Systems and Software (JSS), International Journal of Information Science and Management (IJISM), etc.



Guadalupe Flores Salado received her Master in Telecommunication Engineering in 2011, a Master in Electronics and Telecommunication in 2013 and her PhD in 2017. She has been working for 6 years in the Electronic Department in the Superior School of Engineering in sensors, microfabrication and MEMS. During this time, she developed her expertise in the area of microfluidics doing a PhD about Lab on Chips. Since 2017 she is working in Wellness Telecom in the R& D Department where she has been managing H2020 European Projects.

Spiros Koulouzis, has received an M.Sc. degree in Intelligent and Multi-Agent Systems conferred October 2006 by University of Westminster and M.Sc. in Grid Computing conferred March 2010 by University of Amsterdam. He received his Ph.D. in Computer Science from the University of Amsterdam in 2016. He is currently part of the Systems and Network Engineering research group and his research interests include distributed and parallel systems.

Paul Martin obtained his PhD in Informatics in 2011 from the University of Edinburgh with an interest in semantic modelling, distributed artificial intelligence and argumentation. After working within the Data Intensive Research group at the University of Edinburgh for four years, he joined the System and Network Engineering group at the University of Amsterdam in 2015. Since 2011 he has worked in a number of software and infrastructure-related EU projects: the FP7 projects ADMIRE, ENVRI and VERCE, and the H2020 projects SWITCH, ENVRIplus and VRE4EIC.

Zhiming Zhao obtained his Ph.D. in computer science in 2004 from University of Amsterdam (UvA). He is a senior researcher in the System and Network Engineering group at UvA. He is the scientific coordinator of the European H2020 SWTICH project and leads the Data for Science theme in the ENVRIPLUS project. He participated in VRE4EIC and several other EU projects. His research interests include software defined networking, cloud computing, time critical systems and big data management.

Highlights

The main objective is to address entire lifecycle of time-critical cloud applicaitons
SWITCH offers middleware services for infrastructure planning and provisioning
Interactive graphical modeling tools for specification of time-critical requirements
Self-adaptation of on-demand resources and reconfigurability of infrastructure
The concept of co-programming model to support programmability and controllability