Application-Infrastructure Co-Programming: managing the entire complex application lifecycle

Polona Štefanič*, Matej Cigale*, Andrew C. Jones* Louise Knight*, David Rogers*,
Francisco Quevedo Fernandez*, Ian Taylor*
*School of Computer Science and Informatics, Cardiff University,
Queen's Buildings, 5 The Parade, Roath, Cardiff CF24 3AA, UK
Email: [StefanicP | CigaleM | JonesAC | KnightL2 | RogersDM1 | QuevedoFernandezF | TaylorIJ1]@cardiff.ac.uk

***Abstract*—With an estimated 20 billion connected devices by 2020 generating enormous amounts of data, more data-centric ways of working are needed to cope with the dynamic load and reconfigurability of on-demand computing. There is a growing range of complex, specialised means by which this flexibility can be achieved, e.g. Software-defined networking (SDN). Specification of Quality of Service (QoS) constraints for time-critical characteristics, such as network availability and bandwidth, will be needed, in the same way that compute requirements can be specified in today's infrastructures. This is the motivation for SWITCH – an EU-funded H2020 project addressing the entire lifecycle of time-critical, self-adaptive cloud applications by developing new middleware and tools for interactive specification of such applications. This paper presents a user-facing perspective on SWITCH by discussing the SWITCH Interactive Development Environment (SIDE) Workbench. SIDE provides a programmable and dynamic graphical modeling environment for cloud applications that ensures efficient use of compute and network resources while satisfying time-critical QoS requirements. SIDE enables a user to specify the software components, properties and requirements, QoS parameters, machine requirements and their composition into a fully operational, multi-tier cloud application. In order to enable SIDE to represent the software and infrastructure constraints and to communicate them to other SWITCH components, we have defined a co-programming model using TOSCA that is capable of representing the application's state during the entire lifecycle of the application. We show how the SIDE Web GUI, along with TOSCA and the other subsystems, can support three use cases and provide a walk-through of one of these use cases to illustrate the power of such an approach.**

***Keywords*—Component-based Software Engineering, Cloud applications, Application-Infrastructure Co-Programming Model**

I. Introduction

With the ever-increasing amount of raw data and access to live streams, the reconfigurability of on-demand computing, such as Cloud computing, has become critical in order to service the increasing dynamic needs of applications. The lifecycle of such widely used cloud applications requires management techniques covering many aspects e.g. creation and execution of jobs, scheduling, monitoring, logging, provenance, etc. Such algorithms generally assume there is a reasonably stable infrastructure and take certain aspects of the infrastructure for granted, e.g. network availability and bandwidth. Consequently, until recently, application developers have often assumed their applications can function reliably throughout their lifecycle, based on known properties of anticipated demand, and have not needed to deal with Quality of Service (QoS) breaches for cloud-based networking demands.

Cloud environments provide elastic and controllable on-demand services that can support a wide range of applications. However, even though Software-Defined Networking (SDN) has been adopted widely to offer a separation of control and data planes, thus potentially easing complexity of dynamic network management, this control has yet to be integrated as part of these on-demand cloud services.

There is a growing number of industrial applications that would like to impose time-critical requirements in terms of Quality of Service (QoS) or Quality of Experience (QoE). Development of such applications is difficult and costly due to complex requirements that impact the runtime environment and the optimization needed during application development. Time-critical applications generally require rapid reconfiguration in response to time-sensitive events over a prolonged period. Without QoS for networking capabilities in cloud offerings, it is currently impossible for systems categorically to maintain a guaranteed level performance or service quality.

Support for time-critical applications in cloud environments is still at a very early stage, especially for applications requiring rapid self-adaptation in order to maintain a required level of system performance. Some businesses could profit greatly if such infrastructure were made available to them – e.g. businesses streaming media data, such as live sporting events, or those that deal with disaster early warning systems. In this paper, we describe a platform that aims to address these issues by offering software engineering support for the entire lifecycle of time-critical, self-adaptive cloud applications. We describe an interactive graphical interface that has been developed as part of the EU SWITCH[1] (Software Workbench for Interactive, Time Critical and Highly self-adaptive Cloud applications) project [1]. The SWITCH Interactive Development Environment (SIDE) allows a developer to specify and define the application logic and abstract runtime environment of time-critical applications, specifying QoS/QoE constraints.

In addition to initial specification of application run-time environment and constraints, SIDE provides a unified interface with Switch's Dynamic Real-time Infrastructure Planner (DRIP) and Autonomous System Adaptation Platform (ASAP) systems. This provides the interactive ability to coordinate dynamic planning and application deployment onto cloud platforms and autonomously deploy, monitor and adapt the application and cloud environment (including network) in order to continue to satisfy QoS/QoE requirements. SIDE has

[1] http://www.switchproject.eu/

an interactive graphical modeling tool with which to drag and drop software components from SIDE's internal repository onto a graph canvas. The user builds up a graph representing software components, functional and non-functional requirements and network constraints that belong to the application in order to allow an application workflow to be defined.

The rest of this paper is organized as follows. The next section provides an overview of related work. Section III introduces the Co-Programming concept for specifying infrastructure requirements for the application and components, along with the general architecture of SWITCH and its subcomponents, focusing primarily on the SIDE user gateway. Section IV discusses the implementation of the SIDE workbench and Section VI describes how SWITCH is used in practice by describing how we applied SIDE to three use cases. We conclude the paper in Section VII and provide appropriate acknowledgements in Section VII.

II. Related work

In this section we present an overview of (i) cloud-based methodologies and frameworks that support orchestration and virtualization for the development of cloud applications and (ii) orchestration, automation and interoperability based specifications and languages.

A. Methodologies and frameworks

The ARCADIA [2] methodology provides a novel, reconfigurable-by-design Highly Distributed Applications development paradigm over programmable infrastructure that relies on modeling artefacts, concrete toolsets, and a well-established methodology. On the other hand, ARTIST [3] offers a model-driven approach for the migration of applications towards the Cloud. It is focused on cloud-compliant architecture issues at both the application and infrastructure levels, takes into account the impact of business model shift on the organisational processes, and includes business model issues that are strongly linked to the technical decisions that are made. The ALIGNED [4] methodology aims to develop models, methods and tools for engineering information systems based on co-evolving software and web data and model-driven software evolution based on Linked Data sources. At design time, ARCADIA, ARTIST and ALIGNED define a set of methodologies that make it easier to separate technology-independent from technology-specific aspects, allowing the creation of cloud applications that avoid vendor lock-in. In contrast, at the implementation stage, ASCETiC [5] provides tools that are able to generate code automatically from models created at design time. Furthermore, those design models can be translated into more formal models (e.g. queuing networks, Petri nets, fault trees, process algebras, etc.) over which it is possible to run quality analysis tools to verify if non-functional requirements can be satisfied before applications are deployed. Then, at deployment time, a set of tools can be used to select optimal configurations for the system and for automating deployment. The CloudWave [6] and SSICLOPS [7] methodologies focus on tools for the runtime monitoring of applications and services whereby cloud services should accommodate changes in their requirements and context and should meet their expected quality objectives. The MODA-Clouds [8] methodology offers the development of time-critical applications for clouds but it does not include software-defined networking as a means of allowing programmability of the environment of the cloud for performance optimization. NetIDE [9], on the other hand, provides a programming interface for the underlying network, but does not explicitly cover application development and runtime system adaptability. Service modeling tools such as Juju [10] and Fabric8 are used for the creation of cloud applications and services. Juju, as an open source universal component-based graphical modeling tool for service-oriented architectures and application deployments, offers sets of predefined software assets, and relationships and configurations among them, that come with a knowledge of how to properly deploy and configure selected services in the Cloud [10]. Fabric8 is an open source platform that uses Docker as its virtualisation technology and Kubernetes as its orchestration technology; it enables the creation, deployment, and continuous improvement of microservices.

B. Specifications and languages

In addition to methodologies and frameworks, there are various relevant standards and specifications that can be used for designing and modeling real-time applications, such as MARTE [11] and SysML, and also cloud orchestration standards such as TOSCA [12]. MARTE and SysML are modeling languages used for defining real-time applications, allowing the specification of QoS attributes. They both facilitate the formal verification of a system by transforming some of the models into Timed Petri Nets or Layered Queuing Networks over which various verification techniques can be run in order to verify the quality attributes. On the other hand, TOSCA [13] defines the interoperable description of applications, including their components, relationships, dependencies and requirements. TOSCA therefore enables portability and automated management across cloud providers regardless of underlying platform or infrastructure – thus expanding customer choice, improving reliability, and reducing cost and time-to-value.

C. Gap analysis

Considering these projects, there is still a lack of workbenches that would support the entire lifecycle – programmability and controllability of the development, provisioning and run-time adaptation of QoS of real-time cloud applications. Therefore, we propose an Application Infrastructure Co-programming model that allows in a continuous manner the design of the application logic and infrastructure planning and provisioning throughout the entire lifecycle of an application.

III. The concept of Co-Programming

The main idea of the Co-Programming concept is that the developer can specify the infrastructure and the requirements of the application and its components during development, iteratively refine and test, and subsequently deploy an adaptable application. By using a graphical front-end, users are able to create an application of a given type, assembling its workflow by adding components and connections and setting their properties and requirements. This composition is done by interacting with an information service in order to search and retrieve the component profiles available to the user. Finally, the definition of QoS attributes is also represented. This enables the developer to dynamically set the requirements

of the infrastructure and thus define the QoS for his or her application/component during development.

A. SWITCH general description; role of SIDE subsystem

In order to increase the productivity of developers, the reusability of software and enable greater cooperation, a new development paradigm based on microservices is receiving growing attention. In this paradigm applications are created from several services that are connected together and take advantage of preexisting systems such as databases, frameworks, and other services. SWITCH embraces this concept and extends it with the ability to specify the infrastructure where the application is running and modify it as necessary, based on the information gathered from it.

To achieve this, SWITCH comprises three major parts. The *DRIP subsystem* deals with planning, provisioning, and deploying the application. The result of this process, controlled by the developer, is a running application deployed on selected infrastructure. The *ASAP subsystem* collects monitoring data and uses this information to propose adaptation interventions to the infrastructure, and warns the user when certain events are triggered. Additionally SIDE can communicate with application components themselves via special dynamic variables that are also stored in TOSCA [12]. An application can have its internal parameters – for example video resolution – changed from inside SIDE. The *SIDE subsystem* is the part of the SWITCH system that enables the user to interact with the underlying systems. Its main job is enabling the developer to define the architecture and constraints of his or her application, and to deploy and monitor the application.

B. SIDE work flow description

The SIDE work flow heavily influences its requirements, see Figure 1. SIDE guides the developer through application creation, providing the tools to interact with the application. SIDE facilitates a rapid prototyping and modification approach to development where the developer can choose already existing components to be used. Once (s)he is satisfied with the architecture of the application (s)he can configure its parameters and in this manner fine tune the behaviour of the application. Further specific modification of the functionality of the system can be done inside the components or by creating custom components that can be reused.

Constraints can then be added to the application, mainly in the form of alarms for monitored metrics that are collected as part of the components, as well as constraints for the infrastructure, and can identify specific QoS constraints we want the system to control.

Once this first step is done the developer can deploy the application. Several steps need to be taken that enable any errors in the application to be found. In the first step, validation, SIDE checks if the requirements of the application are met and that the service requirements match the service provided. If met the planner can take all the infrastructure requirements of the system into account, and can attempt to find the infrastructure that meets the requirements, providing the proposed infrastructure to the developer. If the developer is satisfied, the infrastructure can then be provisioned, and the application can be deployed.

When the application is running the developer can monitor its performance and change some of the live variables, start new instances of the components or provision new VMs to the cluster. In the worst case scenario the developer can return to the originally-specified architecture, modify the parameters of the system and start the cycle again.

Communication between SIDE and its related subsystems is done with TOSCA; it is capable of representing the state of the application at all times. There are several APIs available that enable simpler integration of external components. These APIs generally use YAML.

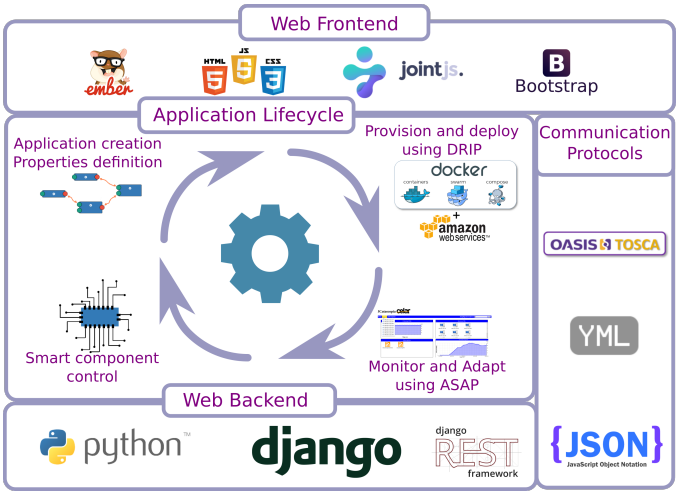

Fig. 1: SIDE work flow. SIDE supports continuous deployment and modification. The developer first creates the application and defines its properties and constraints. Then (s)he deploys the application and can monitor it and adapt using ASAP, and furthermore change application parameters directly in SIDE. If required the properties and even the application architecture can be changed.

C. Distributed Design: TOSCA as a distributed interoperability format

Given that we have a set of loosely-coupled components, a key requirement in the design of the system is to provide a so-called co-programming language that provides a distributed interchange format for expressing QoS/QoE constraints and interoperating between the various SWITCH subsystems.

In order to realize the application infrastructure co-programming model and to pass the data and information through the entire lifecycle of the application development, SWITCH needs a language capable of serializing that information and exchanging it among all three subsystems.

TOSCA [12] is a cloud specification standard developed by OASIS that was created specifically to automate the process of installation, deployment and management of applications, including monitoring, self-adaptation and auto-scaling in the cloud infrastructure. We have chosen TOSCA as the language for application modeling since it coves most of the aspects that SWITCH needs: it describes the application logic, it supports the ability to use VM and container images and implementation artifacts; it defines QoS through policies, and it manages the

entire application lifecycle including post-deployment aspects such as monitoring and adaptation. TOSCA uses the YAML data serialization language that uses objects to describe application components, their lifecycles and dependencies and other environment variables and settings.

IV. Application infrastructure and implementation

The role of the SWITCH co-programming language is to expose the semantic models for time-critical services and resources and provide a format for building dependency graphs, alongside the associated metadata required for planning and controlling execution of the application and its environment. This information is used by DRIP to plan and provision the infrastructure as well as to deploy the application; it is used by ASAP to control behaviour of the application at runtime.

A. SWITCH Workbench from developer's perspective

The SIDE GUI has four main phases/views: component creation view, application composition view, infrastructure planning and deployment view, and operation monitoring and adaptation view. Each of these phases/views has its own modeling graphs that provide to the user a consistent understanding of how the application, software components and cloud environment are configured with particular elements as part of a graph highlighted or manipulated in a way relevant to the current phase of development. However, each phase/view is defined with a set of sub-views that are focused on a small number of aspects and functionalities. In the component creation and application composition phase, non-functional requirements and other quality constraints are defined and assigned to each software component. In the infrastructure planning and deployment phase, monitoring metrics are gathered, the network and cloud infrastructure are configured, and proposed placement strategies are defined. The Operation monitoring and adaptation phase considers monitoring and adaptation strategies. This paper is dedicated mostly to the SIDE architecture and GUI, which is why it focuses on the component creation and application composition views only.

B. Component creation phase

In the component creation view, software developers can describe software components and microservices. He/she can create these components from scratch by packing them into container images, such as Docker or Singularity containers, or can gather software components from public repositories, such as Docker Hub[2] or App Hub[3] and import and store them in the SIDE components repository.

SIDE facilitates additional actions and manipulations on software components. QoS constraints and hardware requirements can be identified and defined for a specific component. Monitoring metrics and QoS constraints linked to those metrics can be defined. Component-specific parameters can be set in order to facilitate reusability. All these properties can be attached to a specific component by dragging and dropping them on the canvas and connecting them to the component. When placing software components on the canvas with its attached QoS constraints and hardware requirements TOSCA is generated and offered to the user. The newly designed software components with attached QoS constraints and hardware requirements are versioned and stored in the SIDE internal repository alongside the original software component imported from the repository.

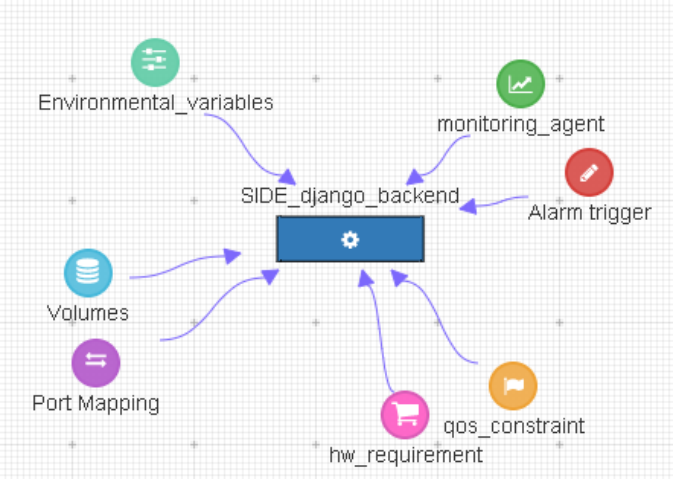

Fig. 2: Creation of the component with all possible properties that can be added to the component.

C. Application composition phase

In the application composition view, native Multi-tier cloud applications can be built on the canvas. Users can explore the internal SIDE repository and drag and drop previously created software components with their attached QoS constraints and hardware requirements and connect them together by setting the dependencies and environment variables. In order to make sure that the quality constraints are satisfied, it is essential to verify that those constraints are not violated so the applications will perform as expected once deployed.

Time-critical applications like the three SWITCH pilot use cases (see section VI) have strict performance requirements that must be satisfied at all times. In SWITCH we have identified three types of QoS metric that can be defined for those applications. These three types of QoS metric are:

- low-level metrics on individual components and on connections between a pair of components (e.g. latency < 20 ms, packet loss < 5);
- quality-based metrics (e.g. quality of the audio during a teleconference);
- multi-dimensional and application-wide constraints (e.g. resolution of video display).

V. Interaction between SWITCH subsystems

SWITCH (Figure 3) is composed of three main subsystems. Our focus is on SIDE, which enables the description and composition of components and applications. SIDE is composed of a front-end[4], a back-end[5] and a "SIDE centric" database.

[2]https://hub.docker.com/
[3]https://apphub.io/
[4]https://github.com/switch-project/side-ember
[5]https://github.com/switch-project/side_api

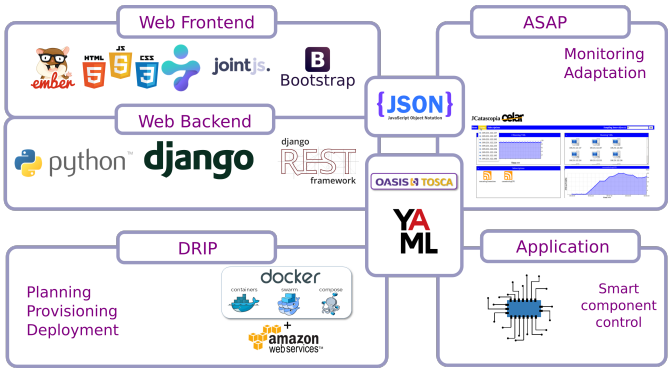

Fig. 3: SWITCH Architecture – SIDE Perspective. SWITCH contains three main components. SIDE is composed of the front-end and back-end that communicate internally with JSON. The back-end communicates with the rest of components using TOSCA. DRIP is responsible for planning, provisioning and deployment, while ASAP provides monitoring and run time adaptation. Applications can also communicate with SIDE using the TOSCA description.

The front-end is based on Ember. Graphs are generated with the help of JointJS, a JavaScript diagramming library. Most of the logic of SIDE is done in the back-end, which is built on the Django framework. The back-end is also responsible for generating TOSCA, and for interaction with the rest of the SWITCH components i.e. DRIP and ASAP.

A. *Infrastructure planning and deployment – SIDE/DRIP*

The basic dialogue between SIDE and DRIP for co-programming is conducted in three stages: 1) The application composed in SIDE is sent to DRIP for planning; 2) The infrastructure planned in DRIP for hosting the application is sent back to SIDE for approval; 3) Permission to provision the planned infrastructure is sent by SIDE to DRIP. This then continues into co-control, whereby control directives can be sent to the deployed application via DRIP from SIDE.

B. *Runtime control and adaptation – SIDE/ASAP*

By using the information service, the ASAP subsystem provides an API to the SIDE workbench for the retrieval of the runtime status of the deployed application and its monitoring data (historical and real time). The flow of information also goes in the other direction where components in the ASAP subsystem, such as the Alarm Trigger and the Self-adapter, can send notifications to the API provided by the SIDE backend, in order to notify the workbench when a certain threshold has been breached or to notify of the adaptation strategy selected to resolve a given situation.

VI. SWITCH Applications Integration Using SIDE

The SWITCH project includes three time-critical pilot cloud applications, for the purpose of supporting their development. These applications are: an elastic disaster early warning system (BEIA Consult)[6], a collaborative real-time business

[6]http://www.beiaro.eu/

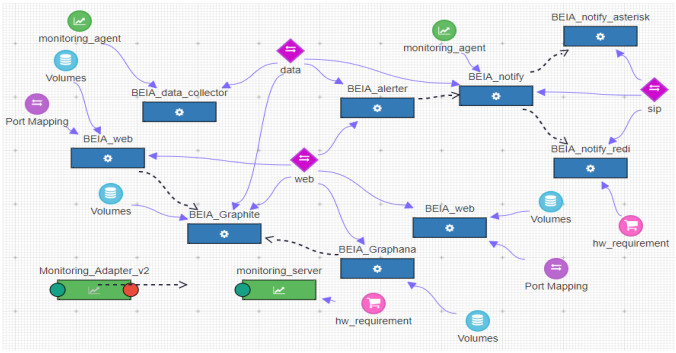

Fig. 4: The application composition of the BEIA use case (see Section VI).

communication platform application (Wellness Telecom – WT)[7] and a cloud-based broadcasting and directing studio for live events (MOG Technologies)[8]. Each of these applications comes with a set of multi-level requirements, such as system-level, network-based, application-level and performance-based requirements. Each has adaptation strategies that can be defined by the software developer while composing the application logic and setting the virtual environment. Once the application is deployed, the software developer will be able to program, control, and monitor the runtime status by getting information on the SWITCH IDE.

Using the SIDE subsystem GUI the software developer may define, control and program the following features for all three use cases: (i) describe the application logic for sensor data collection, data storage, processing, activation of warning services, virtual call center facilities, establishing the number of input cameras used for broadcasting the event, defining the properties for the streaming services; (ii) define quality requirements at system level, such as the admissible percentage of packet loss or the maximum latency and video and voice requirements to ensure that the QoE requirements are satisfied and controlled as well as establishing policies for how to scale the system. The critical requirements can also be validated using formal mechanisms; (iii) describe quality requirements for the runtime environment on the cloud, defining the type of machines needed for running the different services, (iv) monitor the runtime infrastructure and take action (automatically or manually) if failure occurs. Critical requirements for the specific use cases are summarized in Table I.

A. *Walk-through of an application composition using SIDE*

As can be seen from the Table I for the disaster early warning system (BEIA use case) it is crucial for the software developer to be able to set component definition, configuration and creation, add persistent storage, response to the system state, network characteristics and so on. All this can be done via SIDE. The BEIA use case comprises several components such as data collector, notification server, alerter, notify asterisk, web server; these components are presented within *dark blue rectangles* in the SIDE graphs (see Figure 4); data collector presents sensors that contain its own monitoring

[7]http://www.wtelecom.es/
[8]http://www.mog-technologies.com/

TABLE I: Critical requirements that SIDE 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		√	
Persistent storage	√		√
Monitoring	√	√	√
Response to system state	√		√
Manual reconfiguration		√	√
Setting up proxy		√	
Management of VoIP Servers	√		
MCU media mixer	√		
Setting up proxy edge, core	√		

(*green monitoring circle*); furthermore all components are linked to a network node (*purple diamonds*) – named web and data that relate to Docker Swarm. The *azure circles* are referred as Volumes and enable mapping data from a Docker container to the disk volume in order to maintain persistent data. Port mapping property (*light purple circle*) enables components to be accessed from the Internet and are linked to web servers that gather data collected from the field sensors. After application composition the validation process takes place. During this step monitoring components – monitoring adapter and monitoring server as part of the monitoring (marked as *green rectangles*) – are added to the application.

VII. Conclusions

In this paper, we presented the SIDE Workbench, which realises an application-infrastructure co-programming approach to the entire time-critical cloud application development lifecycle. It enables a developer to specify compute and network QoS for a multi-tier cloud application along with the adaptive control needed to reconfigure that application on the fly. To achieve this, the developer creates an outline of the application components (modelled as containers) and specifies QoS parameters on those components and for connections between them, along with alert thresholds that are used to trigger a reconfiguration. This process was described and is achieved in several stages. The Component creation phase enables description of the components that can be application specific or general. The developer then specifies the requirements of the application and strategies for dynamic monitoring, threshold setting and mitigation upon a breach of those settings.

We described how application components are wrapped as Docker containers. For example, a component could be a Python Django stack back-end that includes monitoring and response time of 30 ms. Such components are placed on a canvas to represent the application and adjusted so that parameters can gradually be made more concrete in line with application requirements. For instance the number of CPUs can be changed and the location of the service can be constrained to certain clouds. To illustrate this process, we have described three use cases specified in the SWITCH project and the high-level requirements of those. We then provided an overview of how one of those applications is specified using SIDE to provide a user walk-through of the process.

Acknowledgment

The research reported in this paper was funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 643963 (SWITCH project).

References

[1] Z. Zhao, P. Martin *et al.*, "Developing and operating time critical applications in clouds: The state of the art and the switch approach," *Procedia Computer Science*, vol. 68, no. Supplement C, pp. 17 – 28, 2015, 1st International Conference on Cloud Forward: From Distributed to Complete Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915030653

[2] J. Sterle, M. Rugelj *et al.*, "A novel approach to building a heterogeneous emergency response communication system," vol. 2015, 10 2015.

[3] A. Menychtas, C. Santzaridou *et al.*, "Artist methodology and framework: A novel approach for the migration of legacy software on the cloud. 2nd workshop on management of resources and services in cloud and sky computing," September 2013. [Online]. Available: https://hal.inria.fr/hal-00869276/file/ARTISTMethodologyFramework_SYNACS-MICAS2013.pdf

[4] O. Gavin, D. Kontokostas *et al.*, "The aligned project – aligned, quality-centric software and data engineering driven by semantics," in *Project Networking Session at ESWC 2016 THE SEMANTIC WEB. LATEST ADVANCES AND NEW DOMAINS*, ser. LNCS, H. Sack, E. Blomqvist *et al.*, Eds., vol. 9678. Springer, 2016. [Online]. Available: http://www.tara.tcd.ie/handle/2262/76242

[5] A. Juan Ferrer, D. Garcia *et al.*, "Ascetic: adapting service lifecycle towards efficient clouds," in *European Project Space: cases and examples*. SCITEPRESS, Apr 2014, pp. 89–106. [Online]. Available: http://hdl.handle.net/2117/105288;http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006183400890106

[6] D. Bruneo, T. Fritz *et al.*, "Cloudwave: Where adaptive cloud management meets devops," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, vol. Workshops, June 2014, pp. 1–6.

[7] M. Handley, C. Raiciu *et al.*, "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*, ser. SIGCOMM '17. ACM, 2017, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098825

[8] E. D. Nitto, M. A. A. da Silva *et al.*, "Supporting the development and operation of multi-cloud applications: The modaclouds approach," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Sept 2013, pp. 417–423.

[9] F. M. Facca, E. Salvadori *et al.*, "Netide: First steps towards an integrated development environment for portable network apps," in *2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 105–110.

[10] K. Baxley, J. D. la Rosa, and M. Wenning, "Deploying workloads with juju and maas in ubuntu 14.04 lts," May 2014, a Dell Technical White paper. [Online]. Available: https://linux.dell.com/files/whitepapers/Deploying_Workloads_With_Juju_And_MAAS-14.04LTS-Edition.pdf

[11] C. Andre, F. Mallet, and R. de Simone, "Time modeling in marte," November 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.2266&rep=rep1&type=pdf

[12] P. Hirmer, U. Breitenbücher *et al.*, "Automatic Topology Completion of TOSCA-based Cloud Applications," in *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, vol. 232. Bonn: Gesellschaft für Informatik e.V. (GI), September 2014, Workshop Paper, pp. 247–258. [Online]. Available: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-66&engl=1

[13] N. Ferry, A. Rossini *et al.*, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2013, pp. 887–894. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2013.133