A Novel Reconfigurable-by-Design Highly Distributed Applications Development Paradigm over Programmable Infrastructure

Panagiotis Gouvas, Constantinos Vassilakis, Eleni Fotopoulou, Anastasios Zafeiropoulos
R&D Department
Ubitech Ltd.
Athens, Greece.
{pgouvas, cvassilakis, efotopoulou, azafeiropoulos}@ubitech.eu

***Abstract*—Given the inability of Highly-Distributed-Application-Developers to foresee the changes as well as the heterogeneity on the underlying infrastructure, it is considerable crucial the design and development of novel software paradigms that facilitate application developers to take advantage of the emerging programmability of the underlying infrastructure and therefore develop Reconfigurable-by-Design applications. In parallel, it is crucial to design solutions that are scalable, support high performance, are resilient-to-failure and take into account the conditions of their runtime environment being able to adapt. Towards this direction, the ARCADIA project aims to design and validate a Novel Reconfigurable-By-Design Highly Distributed Applications (HDAs) Development Paradigm over Programmable Infrastructure. The proposed framework relies on the development of an extensible Context Model which will be used by developers to produce annotated source-code and generate HDAs as service chains of application tiers and network functions containing meaningful semantics. A Smart Controller responsible for on-boarding the HDAs will undertake the tasks of translating annotations to optimal infrastructural configuration. Such a controller will enforce an optimal configuration to the registered programmable resources and will pro-actively adjust the configuration plan based on the Infrastructural State and the Application State to meet objectives and apply policies. Driving a HDA through its entire lifetime proves highly beneficial for all stakeholders since the synergy of the introduced applications' reconfigurability and the underlying infrastructure's programmability, facilitates the development of new fine-grained strategies able to fulfil new and complex requirements.**

Keywords; highly distributed applications; reconfigurable by design applications; programmable infrastructure; distributed applications orchestration

I. INTRODUCTION

The generalized trend towards massive 'softwarization' of processes, devices, services and infrastructures has pointed out many limitations of the current practice to develop, deploy and run software applications. In particular, human resources are often overworked with an intrinsic infrastructural heterogeneity, namely a diversity of the programming frameworks and the execution environments that are utilized in the application lifecycle. A transition from the 'intelligent design' approach, which currently rules software engineering, to meta-design approaches as well as self-combining software systems has to be realized. To this aim, focus should be given on the design of software components that have the ability to collaborate in an autonomous and decentralized fashion. In particular, a set of considerations about quick development times, software re-utilization, data locality and so forth have brought the concept of Highly Distributed Applications (HDA), which run on a global heterogeneous infrastructure built on top of the "Future Internet".

Key drivers that boost this transition are emerging paradigms like virtualization and the availability of programmable infrastructures. However, the plethora of different solutions and approaches has led to a thicket of execution environments and their relative configuration artifacts, exacerbating the difficulty of software engineers to quickly adapting their systems to different run-time contexts.

Following the basic principles under the DevOps approach, an increasing interest has been devoted to include more "context-awareness" into the very same applications and services, by making them able to adapt and to adjust their behavior to different run-time environments, relying on the autonomic provisioning capability allowed by the large availability of programmable infrastructure. However, this must not turn into an overwhelming configuration burden for developers, rather it should be an opportunity to exploit the peculiarities of each execution environment and hence to optimize performance, availability, dependability, security, and cost of the applications.

Under this perspective, the vision of the ARCADIA project [1] is to provide a novel reconfigurable-by-design Highly Distributed Applications' development paradigm over programmable infrastructure. The approach relies on an extensible Context Model that will assist programmers to take into account the heterogeneity and peculiarity of the underlying infrastructure, and a Smart Controller that will undertake the tasks of optimal and dynamic deployment of applications over multiple domains starting from the instantiation of the Context Model.

A Highly Distributed Application (HDA) is defined as a reconfigurable-by-design distributed scalable structured system of software entities constructed to illustrate a network service when implemented to run over a programmable cloud infrastructure. A HDA is a multi-tier cloud application consisting of application's tiers chained with other software entities illustrating network functions applied to the network traffic towards/from and between application's tiers.

978-0-9883045-1-2/16 $31.00 © 2016 ITC

A HDA is reconfigurable-by-design. This definition implies that a HDA is designed to be context-aware, able to adapt its processes to the context and reconfigure its structure accordingly while share its context and enable programmability through exposing a programming interface. A HDA may expand or shrink by supporting horizontal scaling (out and in) for each software entity in the service chain while may reform (change its structure) by including or excluding software components from the chain and/or change routing among them as needed. A HDA may expose its context and state while it may provide a programmable interface for externally being adapted and configured.

A HDA runs over a programmable infrastructure. This definition implies that the infrastructure provides the ability to control and change its functions. Through programming interfaces the infrastructure enables configuration and control of its computing, network and storage resources along with flexible illustration of network functions and routing according to the needs of the instantiated HDAs.

The synergy of reconfigurability of each HDA and the programmability of the hosting infrastructure provides a twofold fine-grained adaptation potential as required for optimal achievement of objectives for both guest HDAs and host infrastructures.

Challenges addressed by ARCADIA include (i) the design of a novel software development paradigm that is going to facilitate software developers to develop highly distributed applications represented in the form of a service graph that can be deployable and orchestratable over programmable infrastructure, (ii) the design and implementation of an orchestrator (called Smart Controller in ARCADIA) able to undertake the developed HDA and proceed to optimal deployment and orchestration of the corresponding service/application and (iii) the design and implementation of a policy management framework that supports the definition of high and low level policies on behalf of a Services Provider to be associated with HDAs.

In Section II we briefly refer to the enabling technologies and related considerations, while in Section III the definition of a Highly Distributed Application is detailed. In Section IV the ARCADIA framework is presented in brief. Finally, in Section V challenges and concluding remarks are provided.

II. ENABLING TECHNOLOGIES AND RELATED CONSIDERATIONS

ARCADIA framework considers the lifetime of an application, starting from its development to cloud deployment, running and termination. The fact that the application is not as usual considered from the time it is a binary executable to be deployed, surely provides a strong advantage. However, several aspects have to be taken into account at each phase of the application's lifecycle and a synergy of diverse technologies to be properly employed in order to exploit the potential benefits of the approach.

A. Distributed Software Development Paradigms

The evolvement of new software development paradigms is following the need for development of applications that highly include the notion of modularity, distribution, scalability, elasticity and fault tolerance. Actually, we refer to an evolution from applications that are based on monolithic architectures to applications that can be represented as re-active systems composed by micro-services. Micro-services can be considered as the resulting set of services that arise from the process of decomposing an application into smaller pieces. Furthermore, we refer to applications that have to be deployed and executed over heterogeneous environments –in terms of underlying infrastructure and end users devices- as well as applications that have to take into account strict constraints in terms of performance.

Such an evolution is accompanied with the current trend in the design nature of applications that are consisted –following an increasing tendency- of distributed components that have to interact among each other and have to react based on events that are initiated in their environment. Such applications cannot be easily and effectively developed based on sequential programming paradigms, since the execution flow of the components, as well as the dynamicity in their instantiation and operation cannot be predicted or represented on a sequential flow. Furthermore, it is really hard to support stateful mechanisms in such cases, since in case of a state change all the associated components have to be informed at real time, without negative impact in the overall application performance.

In order to be able to develop efficient distributed applications with high notion of reactiveness, the reactive programming paradigm [2] has been recently proposed as a solution that is well-suited for developing event-driven applications. Reactive programming tackles issues posed by event-driven applications by providing abstractions to express programs as reactions to external events and having the language automatically manage the flow of time (by conceptually supporting simultaneity), and data and computation dependencies. Thus, programmers do not need to worry about the order of events and computation dependencies. The momentum for the adoption of reactive programming approaches has also been fortified by the need to transit from stateful to stateless approaches in order to increase the scalability of the provided services and applications. Nodes have to be able to be added or removed during runtime, independently if they are related with the same process or not, the same physical machine or not, or even if they are in a completely different point of presence (e.g. data center). Failures are also handled in an automated way.

The adoption of such an approach facilitates the development of reconfigurable-by-design HDAs by providing for the illustration of the desired characteristics of scalability, dynamic structure, context-awareness, adaptability, and programmability.

B. Programmable Infrastructure

There are two complementary aspects stemming from different perspectives, needs and roles of the relevant actors. On the one hand, for developers infrastructure's programmability is the mean to create the proper execution environment independently of the underlying physical resources. They need both overarching resource abstractions at the design/development stage and convenient APIs at run-time, in order to implement their application in an environment-agnostic way and to dynamically tailor them to the actual (and usually changing) context. To this aim, the Programmable Infrastructure provides developers with a common and single point of access to all resources, hiding physical issues like resource nature, faults, maintenance operations, and so on. On the other hand, resource owners are mostly concerned with operation and maintenance of (usually) large pools of resources. They need handy tools to deal with typical management tasks like insertion, replacement, removal, upgrade, restoration and configuration with minimal service disruption and downtimes.

The synergy of different architectures, frameworks and implementations towards a fully programmable infrastructure is more and more evident in today's platforms. A programmable infrastructure's building blocks maybe considered to consist of the Application Execution Environment and the Application Networking Environment. The most common required characteristics regarding an application's running environment which partially led development in the virtualization area are: isolation of the application environment, resource isolation, low to zero performance loss compared to execution over a native operating system environment, easy sharing between virtualized hosts, easy management of application running environments, portability. Furthermore, resources management is considered quite crucial in order to efficiently handle capacity and meet applications requirements. Regarding the networking environment, a combination of Software Defined Networking (SDN) and Network function virtualization (NFV) seems ideal as it ensures not only initial deployment of an application but also the required flexibility in terms of reconfigurations during its runtime, triggered by scaling requirements, requirements related to achieving optimization objectives and sustaining continuity of operation under system's dynamics.

C. Applications Profiling, Deployment and Orchestration

Application deployment is the last set of actions before having the software up and running. In the case of Highly Distributed Applications, this can be a complex and hard task, which becomes even more complicated due to the need for coordination and management of the various service components of such an application.

An initial deployment of an application requires an initial estimation of the required resources to be acquired from the infrastructure along with several constraints to be met when the application is deployed. Embedding an HDA to a programmable infrastructure while satisfying constraints either coming from the application side or the provider's side is not a trivial task; efficient embedding algorithms have to be devised [3]. Furthermore, initially acquired resources may not be enough to serve workload during runtime and the need of more resources to be allocated may arise and drive a reconfiguration process, vertically up or horizontally out scaling the application in order to sustain application's uninterruptible execution and performance. On the other hand, underutilized resources already allocated for an application may also drive a reconfiguration process to release them in favor of efficient usage of available resources; scaling the application vertically down or horizontally in. Reconfigurations may be several as well during the runtime of an application in order to meet either application's or provider's objectives. Application profiling as a process to discover in detail an application's resource needs at different types and volumes of workload as well as the process of predicting workloads during an application's runtime, are considered crucial to provide necessary knowledge for driving initial application deployment and subsequent reconfigurations towards meeting objectives [4]. In such an environment all decisions and directed actions towards meeting goals and satisfying objectives while having a thorough view of each running application, the applications about to be deployed and the infrastructure resources and state, fall into the duties of the Orchestration process, a responsibility of the ARCADIA smart controller in our proposed framework.

III. HIGHLY DISTRIBUTED APPLICATION (HDA)

A Highly Distributed Application (HDA) is defined as a reconfigurable-by-design distributed scalable structured system of software entities constructed to illustrate a network service when implemented to run over a programmable cloud infrastructure. A HDA is a multi-tier cloud application consisting of application's tiers chained with other software entities illustrating network functions applied to the network traffic towards/from and between application's tiers.

The term 'reconfigurable-by-design' may be considered as an extension of the term 'context aware adaptable' [5]. A HDA not only is designed to be context-aware, able to adapt its processes to the context but also is able to reconfigure its structure accordingly, share its context and enable programmability through exposing a programming interface.

Thus, a HDA may expand or shrink by supporting horizontal scaling (out and in) for each software entity in the service chain while may reform (change its structure) by including or excluding software components from the chain and/or change routing among them as needed. A HDA may expose its context and state while it may provide a programmable interface for communication and externally being adapted and configured or re-used as a component of other HDAs.

Each Application tier of a HDA is a distinct application-specific logical operation layer. Each other

software entity involved in the Application's chain is a Virtual Function (VF) specific logical operation layer. Each Application tier and other involved software entity provide/expose to each other a Binding Interface (BI) letting each other have access to provided functionalities and supporting communication. While developing an Application Tier, it is necessary to have knowledge of the BI of every other software component whose functionalities are required to be utilized within this Application Tier. Recursively, a component of a HDA chain may also be a chain itself; an already developed HDA or a VF chain exposing required functionalities. There is no need to communicate with every component of the nested chain but with a Service Binding Interface (SBI) providing for communication and access to exposed functionalities.

The developer of an application tier annotates its code with required qualitative and quantitative characteristics according to the context model. Annotations can be included within the source code and be properly translated before building the executable, as well as externally accompany the executable and be properly translated by other components of the architecture during executable's placement and runtime.

An indicative HDA is depicted in Figure 1(a) that corresponds to a graph containing a set of tiers along with a set of functions implemented in the form of Virtual Functions (VFs). It should be noted that in ARCADIA we are adopting the term Virtual Functions (VFs) instead of the term Virtual Networking Function (VNF) that is denoted in ETSI Network Function Virtualization (NFV) [6] since we do not only refer to networking functions but to generic functions. Each element in the graph has to be accompanied with a set of quantitative characteristics (e.g. set of metrics that can be monitored) and constraints (e.g. resource capacity constraints, dependencies). In Figure 1(b) an application tier (T4) of the example is shown to horizontally scale while the chain reconfigures itself by expanding and reforming, adding as well a Load Balancer component as now required.

In ARCADIA, the deployment and operation of the following HDA types is supported: (i) ARCADIA applications; applications that are going to be developed following the proposed software development paradigm, (ii) Legacy applications; existing applications already available in an executable form, and (iii) Hybrid applications; applications consisting of application tiers from both the aforementioned types.

Native ARCADIA applications will benefit from the full set of capabilities of the Framework. Regarding Legacy applications, a subset of offered capabilities would be possible to be available. In order a chain of a legacy application tiers to be deployable and benefit from the ARCADIA framework, proper interfacing should be built between application tiers while proper external annotations could accompany the executables. In fact, each legacy executable should be interfaced by developing an ARCADIA BI which will expose its functionalities and enable communication with other properly enhanced legacy executables in a common ARCADIA framework style/way. The same stands for hybrid applications as it concerns the legacy application tiers. Figure 2 shows an HDA formed as a chain of an ARCADIA Application Tier using an already developed hybrid application service chain.

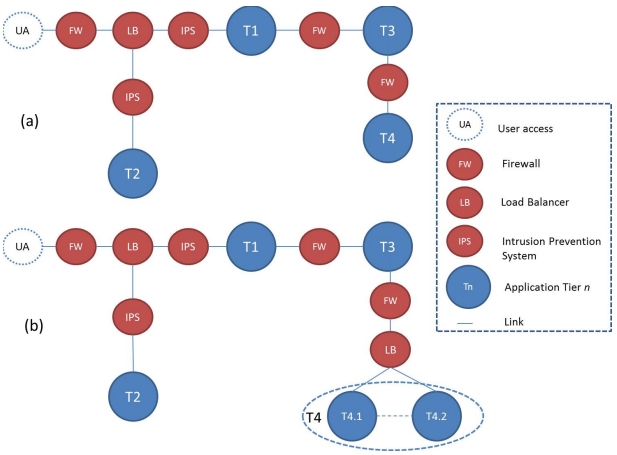

Figure 1. (a) HDA Indicative Breakdown, (b) HDA horizontal scaling.

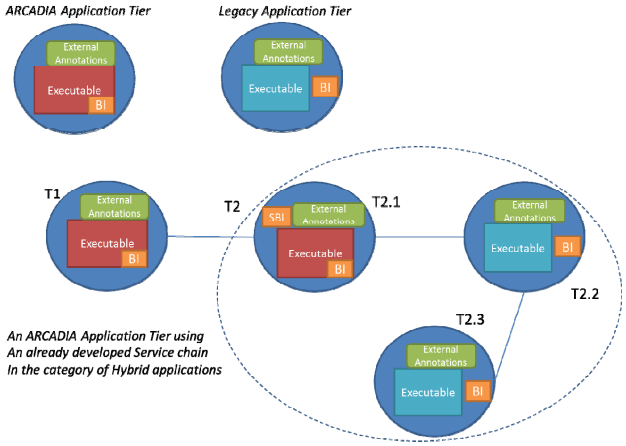

Figure 2: Example of an HDA formed as a chain of a new Application Tier and an already developed Hybrid application.

From an upper view, the lifecycle of an HDA starts from the development phase, followed by the deployment phase and operation phase and ends with its termination. Each phase requires several functional components of the architecture to work together to provide for and build an HDA, deploy it assigning resources from an infrastructure, run it while meeting objectives -developer wise and/or service provider wise- at all times and assure proper release of resources when terminates its operation. Deployment, Operation and Termination are supported by the Smart Controller as the intermediate between the applications and the infrastructure, while development is supported by several repositories providing easy access to reusable components and the defined context model providing access to the set of annotations and descriptions.

IV. ARCADIA FRAMEWORK

The vision of ARCADIA is to provide a novel reconfigurable-by-design Highly Distributed Applications (HDAs) development paradigm over programmable

infrastructure. The ARCADIA framework [7] consists of a set of components covering in a holistic way the development, deployment and management of applications in runtime over the available programmable infrastructure. A high level overview of the ARCADIA framework is provided in Figure 3 (including some implementation specific indications). In the upper level of the framework, a set of components are made available for designing, developing and deploying HDAs. The set of components are used by software developers towards the development of applications following the ARCADIA software development paradigm, as well as service providers towards the design of services graphs along with their mapping with policies. In the middle level of the framework, the ARCADIA Smart Controller deploys the applications over the available programmable infrastructure and manages the application during the execution time triggering re-configurations where required based on the defined optimization objectives and policies, on behalf of the application developer and the services provider. In the lower level of the framework, management of the available compute, storage and network resources is realized along with establishment of the required monitoring and signaling probes for the real-time management of the instantiated components and links.

A developer toolkit is made available which supports a set of views and roles targeted at the various phases of the application development process; application development, deployment script preparation, policies specification. Within the Toolkit, the software developer is able to develop native ARCADIA components and make them available –upon validation- to the Service Graph Repository, as well as make deployable service graphs based on native and/or reusable components or service graphs. The software developer is also able to use the Annotation framework for specifying annotations, while the implementation of component/service graph interfaces has to be based on the existing context model. Annotations are based on concepts represented in the ARCADIA Context Model [8] and can be interpreted during deployment targeting at providing hints towards the optimal deployment of the application. An ARCADIA service meta-model is followed during the design of ARCADIA applications targeting at adopting standardized ways of service graphs design and specification, including the implementation of common interfaces and the interconnection among the various components.

The software developer is also able to adapt legacy applications transforming them to ARCADIA components, while the software developer and the services provider are able to make deployable service graphs out of reusable components or graphs as well as make available multi-tenant deployable service graphs (micro-services) for shared use. The latter are made available through the Micro-services repository. The services provider is also able to specify the set of policies to be applied as they are made available in the Policies Repository. Such policies can be high level policies or policies directly associated with an ARCADIA Service Graph. A Policies Editor is used to this end to facilitate the service provider to define a set of rules and actions taking into account the monitoring hooks/metrics available per Service Graph.

Following the creation of a deployment model, a deployment model instance is provided to the Smart Controller that undertakes the deployment and orchestration of the overall operation of the ARCADIA application. The Smart Controller is the application's on-boarding utility which undertakes the tasks of (i) translating deployment instances and annotations to optimal infrastructural configuration, (ii) initializing the optimal configuration to the registered programmable resources, (iii) supporting monitoring and analysis processes and (iv) reacting pro-actively and re-actively to the configuration plan based on the infrastructural state, the application's state and the applied policies.

The application's software components –as denoted in the corresponding service graph- are instantiated on demand by the Smart Controller. The defined monitoring hooks initiate a set of monitoring functionalities for specific performance metrics. The status of these metrics trigger re-configurations in the deployed application based on optimisation objectives (as denoted in the selected policies) along with a set of constraints that are considered during the application deployment and runtime. Resources reservation and release is realized on demand over the programmable infrastructure.

In more detail, the Smart Controller includes the following components: (i) the *Deployment Manager* that is responsible for the complex task of undertaking the deployment model instance and "translating" it into optimal deployment configuration taking under consideration the registered programmable resources, the current situation in the deployment ecosystem and the applied policies; (ii) the *Optimisation Engine* that pro-actively adjusts of the running configuration as well as re-actively triggers re-configurations in the deployment plan, based on measurements that derive from the monitoring components of the Smart Controller (Monitoring and Analysis Engine) and the existing policies as provided by the Policy Enforcement component. The ultimate goals of the Optimisation Engine are two: a) zero-service disruption and b) re-assurance of optimal configuration across time; (iii) the *Policy Enforcement* component which assures that the imposed policies on behalf of the Service Provider are adhered across the applications operational lifecycle; (iv) the *Execution Manager* that is responsible for the execution of the deployment plan based on the instantiation of the required components and the links among them, according to the denoted service graph in the deployment script. The Execution Manager is also responsible for implementing the monitoring mechanisms required per component and service graph for the collection of the information required by the denoted monitoring hooks. Such information is then provided to the Monitoring and Analysis Engine for further processing; (v) *the Programmable Resource Manager* that exposes a specific interface where programmable resources are registered and managed (reserved/released).

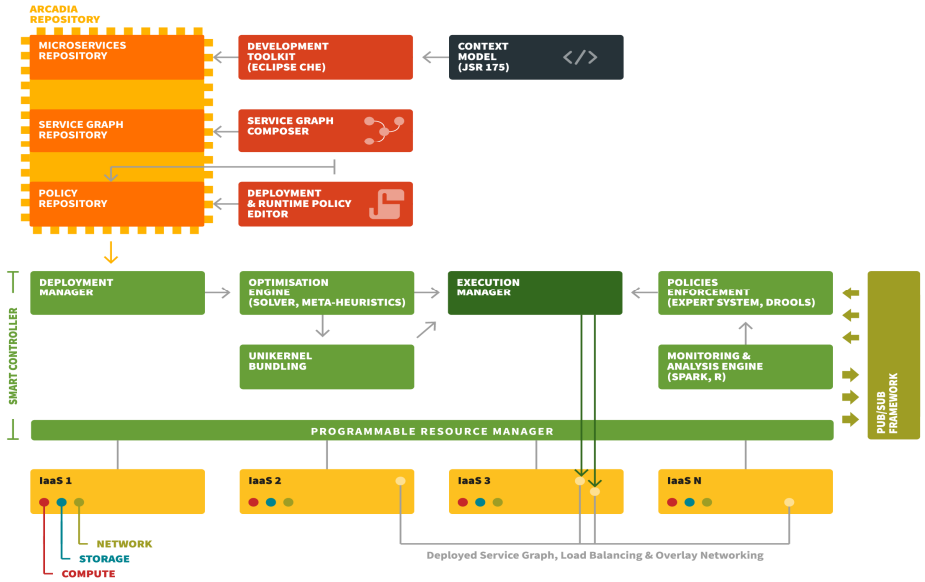

Figure 3: ARCADIA Framework High Level View

Programmable resources can span from configured IaaS frameworks, programmable physical switching/routing equipment, programmable firewalls, application servers, modularized software entities (databases, HTTP proxies etc.). Allocation/Release of resources is realised upon requests provided by the Deployment Manager; (vi) the *Monitoring and Analysis Engine* that is responsible for collecting the required information –as defined by the monitoring hooks per component and service graph- and supporting the extraction of insights and predictions upon analysis.

The considered software components per service graph are deployed in a multi-IaaS environment along with the associated mechanisms for supporting signaling and measurement feeds. Monitoring feeds to these mechanisms are provided based on information collected by the ARCADIA Agent that is included within each ARCADIA component.

V. Challenges and Concluding Remarks

Considering the application through its entire lifetime, starting from the development phase, gives the potential to drive the whole process so as to illustrate required characteristics and lead a desirable synergy of different technologies for the benefit of all stakeholders. Although there is a high potential to implement and satisfy new and complex requirements, there is as well an increased complexity in coordinating and relaxing the complexity of the various involved components which is only possible through a careful detailed design and exploitation of the full potential of current technologies and implementations.

The introduction of the Highly Distributed Application as a reconfigurable-by-design distributed scalable system that runs over a programmable infrastructure provides the maximum flexibility from both application wise and infrastructure wise sides, to be strategically exploited towards supporting current and emerging applications and environments.

The under development ARCADIA framework presented is expected to support and uncover the full potential of the approach. Currently devised algorithms as part of the functionality supported by the Smart Controller are about to exhibit the size of the foreseen gain for both applications and infrastructure in terms of performance, savings and offered capabilities. However, the benefits are not only operational as a developer through a simplified process will be able to have access to complex functionalities and technologies and devise new reusable applications and services at minimal development times.

Acknowledgment

This work has been co-funded by the ARCADIA project, a European Commission research program under Contract Number H2020-645372.

References

[1] The ARCADIA Horizon2020 Project. Available Online: http://arcadia-framework.eu/

[2] The Reactive Manifesto, Available Online: http://www.reactivemanifesto.org/

[3] Z. A. Mann, "Allocation of Virtual Machines in Cloud Data Centers – A Survey of Problem Models and Optimization Algorithms," ACM Computing Surveys, Vol. 48, Issue 1, September 2015

[4] R. Weingärtner, G. Beims Bräscher , C. Becker Westphall, "Cloud resource management: A survey on forecasting and profiling models," Journal of Network and Computer Applications, Vol. 47, January 2015, pp. 99–106

[5] M. Dalmau, P. Roose, S. Laplace, "Context Aware Adaptable Applications - A global approach," IJCSI International Journal of Computer Science Issues, Vol. 1, 2009

[6] ETSI, Network Function Virtualization, Online: http://www.etsi.org/technologies-clusters/technologies/nfv

[7] ARCADIA Horizon2020 Project, D2.3 - Description of the ARCADIA Framework, Available Online: http://www.arcadia-framework.eu/wp/documentation/deliverables/

[8] ARCADIA Horizon2020 Project, D2.2 – Definition of the ARCADIA Context Model, Available Online: http://www.arcadia-framework.eu/wp/documentation/deliverables/