A framework to support interoperability in IoT and facilitate the development and deployment of highly distributed cloud applications

Nikos Koutsouris, Apostolos Voulkidis, and Kostas Tsagkaris

WINGS ICT Solutions,
336 Syggrou Avenue, 17673 Athens, Greece
{nkouts,avoulkidis,ktsagk}@wings-ict-solutions.eu

Abstract. The constantly increased variety of available hardware and software solutions for the IoT sector is facilitating the development of novel applications, but at the same time the lack of standardized or widely accepted means of interaction, deployment and configuration is seriously hindering the IoT's potential. The ARCADIA framework is an application development paradigm that enables the cooperation between software components designed and implemented independently and using various technologies and platforms, so that they can form sophisticated, distributed, cloud applications.

Keywords: Highly distributed applications, microservice, cloud, unikernel, virtualization, devops, reconfiguration, SDN, annotations.

1 Introduction

The emerging era of cloud applications has already started and the concepts of IoT are ready to be introduced in the modern everyday life through the deployment of a large scope of novel applications, ranging from wearables, personal health and home automation, to smart city solutions, public safety and transportation. During the last years there is a proliferation of available hardware and software solutions related to IoT, which on one hand is definitely positive, but on the other hand, it has increased complexity in the IoT ecosystem, due to the low level of standardization and the lack of widely accepted means of interaction. This is something justifiable and actually expected, since manufacturers, as well as application providers, need to diversify, innovate and minimize time-to-market while developing their products. As for the caused heterogeneity, it can be compensated and hidden if a framework encompassing intelligent functions is used, like the one presented in the following.

The ARCADIA framework, which is developed in the ARCADIA project [1] and is funded by the H2020 EU programme, is a novel application development paradigm that enables the management of applications' configuration in a smart and dynamic way, allowing the combination of software components designed and implemented independently and using various technologies and platforms. The proposed framework addresses the challenge of interoperability by introducing the Smart

Controller, which incorporates several functionalities that can ensure the trustworthy interworking of components, based on an extensible context model that describes requirements and available options.

More information on the role and the modules of the Smart Controller is provided in section 2.2. Beforehand, sections 1.1 and 1.2 highlight in brief the main concepts and technologies behind ARCADIA, while section 2.1 introduces the basic parts of the ARCADIA ecosystem. Section 3 provides details on the steps required to develop a component, generate an application by chaining various components and then configure and deploy the application on the available infrastructure. Finally there are some conclusions on how the work should evolve in the mid and the long term. It has to be noted that this paper presents the current work in progress in the context of the ARCADIA project and all the described concepts will be elaborated and validated in a set of selected use cases before the first official release of the ARCADIA framework.

1.1 Virtualization and Cloudification

Virtualization refers to the act of creating a virtual (rather than actual) version of something, including, but not limited to, virtual computer hardware platforms, storage devices, and computer network resources. [2] It enables the optimized utilization of resources, as more applications and services can be packed onto the infrastructure. On the other hand, cloud computing offers through a broad network access, a pool of resources that can be assigned dynamically and on demand, while their usage can be monitored, controlled and optimized. To fully exploit the merits deriving from a virtualized cloud environment, it is required to go further than just porting applications and services from running on bare metal to running on Virtual Machines (VMs). Technologies such as containers and unikernels allow better resource and service management by further exploiting the concept of autonomous applications and micro-services. Unikernels are highly optimised, specialised machine images constructed by only using the minimum required set of operating system libraries to run an application. Their small footprint is an important feature for a cloud application as it reduces the cost of the deployment by using only minimal resources and increases the security of the application by shrinking the attack surface. Moreover, the lack of unnecessary operating system libraries allow unikernels to boot extremely fast making them ideal for mission critical or highly available applications.

One of the most difficult and expensive tasks on legacy, monolithic applications is scaling. Using design paradigms for cloud applications such as micro-service architecture, applications can be scaled up or down in a matter of seconds without extreme differences in cost. Modular applications consisting of several stateless micro-services are perfect for scaling operations and cost-effective deployment due to their autonomous nature. By separating data and functionality developers or service providers can easily scale out just a part of their application by deploying more instances of said micro services. In addition, stateless micro-services are more agile and fault tolerant which is vital requirement for highly distributed cloud applications.

Related work is carried out in the INPUT project [3] which aims at designing a novel infrastructure and paradigm to support Future Internet personal cloud services in a more scalable and sustainable way. The INPUT technologies intend to enable next-generation cloud applications to go beyond classical service models, and even to replace physical Smart Devices, usually placed in users’ homes (e.g., set-top-boxes, etc.) or deployed around for monitoring purposes (e.g., sensors), with their virtual images, providing them to users “as a Service.”

1.2 Management and Orchestration

In the era of cloud applications and micro-services, the ability to deliver complex and agile applications is getting harder and harder. Such applications should be Reconfigurable-by-Design, infrastructure independent and at the same time, resilient to failures and easily scalable. To overcome these difficulties, management tools are trying to simplify the deployment and scaling process by automating different aspects of the work-flow. Service modeling tools, like Juju [4], enable developers and IT professionals to automate mundane tasks and reduce workloads, by undertaking a big part of the deployment process on a private or public cloud. Developers can use such tools to create the blueprint of their application called “service graph”, where they can define how micro-services are interacting with each other and have a general view of application data-flow. Moreover, DevOps environments are getting more and more difficult to manage due to the multiple Infrastructure as a Service (IaaS) providers.

Deploying a complex, service-based application on top of different infrastructures involves more complicated tasks, like network management, that require more sophisticated tools and frameworks. Apart from service modeling issues, different IaaS providers mean different network requirements and configurations as well as different policies. New development paradigms are trying to tackle such issues by leveraging the power of software defined networks and virtualized network functions. Application orchestration and network orchestration is an important requirement for cloud management tools and frameworks.

A related open-source system for automating deployment, scaling, and management of containerized applications is Kubernetes [5]. It groups containers that make up an application into logical units for easy management and discovery. It also supports self-healing of containers, service discovery and load balancing, horizontal scalability, batch execution and automated rollouts and rollbacks of application configurations. Finally, it is able to orchestrate storage and allow the seamless mounting of local storage, a public cloud provider or a network storage system.

2 The ARCADIA platform

The ARCADIA framework [6] is a novel reconfigurable-by-design Highly Distributed Applications (HDA) development paradigm. It takes care of multi-infrastructure deployment, high availability requirements and automatic real-time reconfiguration of applications. To solve such issues, ARCADIA applications are based on a micro-service model and are governed by a sophisticated policy manager. In other words, each ARCADIA application consists of several autonomous components, which can communicate with each other based on a service graph and policy rules defined by the developers. Each component can be stored in a public or private registry on the ARCADIA platform and it can be re-used on other applications. To create an ARCADIA component, developers can transform their legacy applications by either using specific JAVA annotations if applications are java-based, or by wrapping them using java interfaces.

JAVA annotations are used to provide meta-data to a java application, without affecting the execution of the application itself, although they can be used for that as well. They are pre-defined words preceded by the "@" symbol and they can be written in many different parts of the code depending on their configuration, for example whether they annotate methods, classes, fields, etc. Annotations are used during three stages of the application life-cycle determined by their defined retention policy; before compilation, during build time or on runtime. Most of the natively supported annotations are discarded during compilation stage; however, ARCADIA annotations are configured to stay past that stage and during runtime. Using the Reflection API, provided by JAVA, the ARCADIA Smart Controller can read those annotations and give instructions to the application.

The ARCADIA Smart Controller consists of several modules such as the unikernel generator, the deployment manager and the policy manager. The Smart Controller is the heart of the framework and its responsibilities include, among others, network management, policy enforcement and annotation processing.

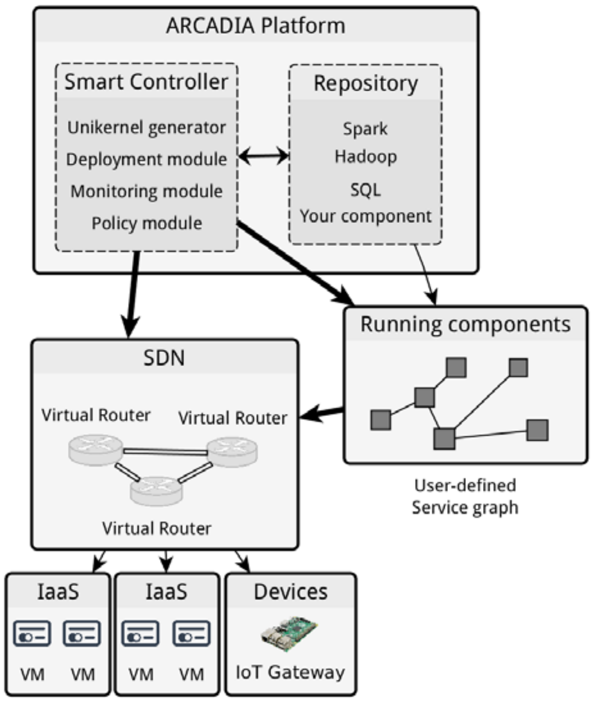


Fig. 1. Graphical representation of the ARCADIA ecosystem, illustrating also the main modules of the Smart Controller

2.1 Architecture of the ARCADIA ecosystem

As depicted in Fig. 1, the ARCADIA Smart Controller and the repository of the ARCADIA components form a platform that is managing the configuration, deployment, monitoring and potential reconfiguration of applications according to policies set by developers, application providers or IaaS providers. Developers create and push their components to the ARCADIA registry, where they can publish them with public or private access, according to the ARCADIA context model [7]. They can use all publicly available components to create a service graph for the application through an innovative web-based user interface. The deployment module creates the underlying Software Defined Network on top of different IaaS providers according to the policies and the requirements of the components that comprise the application.

2.2 The role of the Smart Controller

The Smart Controller (SC) is the most sophisticated module of the framework. It contains several sub-modules that are important for many aspects of the applications life-cycle from the development to monitoring and reconfiguration. Starting from the development, SC is responsible for interpretation of annotation usage in a component, finding and deploying the required dependencies of a component and finally generating the unikernel which is the purposed-build virtual machine image for cloud deployment. Smart Controller is infrastructure agnostic and can deploy applications on different infrastructure providers according to the policies defined by the developers. Moreover, by monitoring the components, Smart Controller is responsible for scaling and reconfiguring the application with complex optimization algorithms.

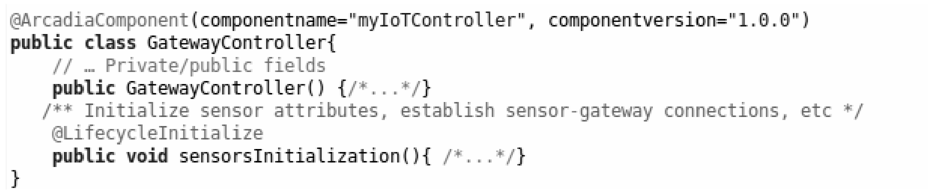
3 The ARCADIA development paradigm

ARCADIA framework doesn't force developers to re-write their existing applications from scratch, since by using specific JAVA annotations they can quickly convert a stand-alone application to an ARCADIA-compatible component.

3.1 Creating an ARCADIA component

In order to have a valid ARCADIA component, a minimum of four JAVA annotations have to be used in the application; "***@ArcadiaComponent***" that declares the name and the version of the component and three more that define the life-cycle management methods to be called by the Smart Controller; however, developers can use as many annotations as their application requires in order to offer metrics or configuration parameters. Moreover, developers can use annotations that define dependencies of the component, for example the requirement for a database or vice versa the definition of an interface for other components to depend on it.

There aren't any forced naming conventions and thus, there is no need for heavy code refactoring of existing applications. For example, as shown in Fig. 2, by annotating a method with "***@LifecycleInitialize***" the framework will know which method to call before starting the component. There are similar annotations for start and stop functions namely "***@LifecycleStart***" and "***@LifecycleStop***".



```
@ArcadiaComponent(componentname="myIoTController", componentversion="1.0.0")
public class GatewayController{
    // … Private/public fields
    public GatewayController() {/*...*/}
   /** Initialize sensor attributes, establish sensor-gateway connections, etc */
    @LifecycleInitialize
    public void sensorsInitialization(){ /*...*/}
}
```

Fig. 2. Use of annotations for managing the lifecycle of an ARCADIA component

The framework will validate the correct usage of annotations before generating the final component. Moreover, developers can pre-validate their application by using the ARCADIA plugin specifically developed for Eclipse Che web based IDE [8] during their development. Each component is bundled with an agent process responsible for controlling the component and communicating with the smart controller. The final component is either a purposed-build unikernel that can be run under a hypervisor on any cloud infrastructure or a simple application that can be run on bare-metal machines like a raspberry Pi, ideal for IoT usage Currently, ARCADIA supports virtual machines and unikernels but it can be easily extended to support bare-metal deployments. For the scope of this project, devices must be powerful enough to host a Java Virtual Machine (JVM).

3.2 Generating an application service graph

One of the main issues with IoT applications is the huge variety of hardware and software vendors, and more specifically the consumption of the different data types each of them produces. With the ARCADIA framework, a component running on an IoT gateway can consume data from different sensors, and then offer them as metrics to the ARCADIA platform or provide an interface for other components to access them, by using simple annotations.

Developers can then use these datasets in any way their application requires, for example store them in a database or create a pipeline for Big Data analysis. In addition, various public components will be available through the ARCADIA registry and can be easily added to the service graph of the application. Moreover, an advanced policy editor is part of the framework where developers can configure different aspects of their application and let the Smart Controller handle the requirements, like high-availability of a component.

3.2 Configuration and deployment process

Configuring how individual components will communicate and interact with each other is a challenging task, considering the different architectures of infrastructure providers. To solve such issues, ARCADIA creates a virtual, infrastructure independent network on top of Open Overlay Router and offers a juju-like, service graph manager. Through this manager, developers can visualize or reconfigure the service graph of their application, add or remove components with a simple “drag and drop” and create a workflow for their application. Moreover, they can create complex graphs required by many applications like Big Data clusters, and pass data through virtual functions. The Smart Controller is responsible for deploying each component, and its dependencies, according to the policies defined by either the developer or the infrastructure provider and report any possible graph error, like graph loops.

4 Conclusions

Internet of Things applications can take advantage of the different features of the ARCADIA framework. For example, by using policies, metrics and re-configuration parameters, developers can control IoT devices like actuators or motors through the gateway component. Moreover, the exploitation of the annotations for discovering in the ARCADIA repository components that are necessary for implementing an application, multiplies the available options for developers; the necessary adaptations for ensuring interoperability are responsibility of the Smart Controller, which will set the optimal configuration through the ARCADIA agent of the object under control.

Till the end of the project, quantitative information on the network overhead and on aspects related to networking, like the bandwidth or latency requirements, will be published.

Moreover, in the remaining duration of the project, the developed functionalities of the Smart Controller will be tested and evaluated. In addition they will be enriched with knowledge building capabilities so as to further improve their performance. The Policy Management and Service Chaining parts will also be finalized and a fully functional release is planned to be made available for download by the end of 2017.

Acknowledgment. The work described in this paper is being performed within the ARCADIA project and has received funding from the European Community's Horizon 2020 Programme under grant agreement n° 645372.

References

1. The ARCADIA Horizon 2020 Project, http://arcadia-framework.eu/
2. Wikipedia, http://en.wikipedia.org/wiki/Virtualization
3. The INPUT Horizon 2020 Project, http://www.input-project.eu/
4. Juju Orchestrator by Canonical Ltd, https://jujucharms.com/about
5. Kubernetes system, http://kubernetes.io/
6. ARCADIA Horizon2020 Project, deliverable D2.3 - Description of the ARCADIA Framework, http://www.arcadia-framework.eu/wp/documentation/deliverables/
7. ARCADIA Horizon2020 Project, deliverable D2.2 – Definition of the ARCADIA Context Model, http://www.arcadia-framework.eu/wp/documentation/deliverables/
8. Eclipse Che Next-Generation IDE, http://www.eclipse.org/che/