Towards a Method for end-to-end SDN App Development

Christian Stritzke, Claudia Priesterjahn
Fraunhofer Institute for Production Technology
Project Group Mechatronic Systems Design
33102 Paderborn, Germany
Email: {christian.stritzke, claudia.priesterjahn}@ipt.fraunhofer.de

Pedro A. Aranda Gutiérrez
Telefónica, Investigación y Desarrollo, S.A.U.
Network Innovation & Virtualisation
28006, Madrid
Email: pedroa.aranda@telefonica.com

Abstract—The ecosystem of software-defined networking (SDN) is still characterized by a multitude of different controller platforms, each with its own programming model, execution model, and capabilities. This creates a danger of a controller lock-in for both developers of SDN control applications and operators of SDN networks. Since no single controller platform appears to dominate the ecosystem for the foreseeable future, there is a need for a method that enables the unification of different controller technologies. This paper describes a preliminary version of the NetIDE method, whose goal is to deliver an integrated process for SDN development that allows developers to write, test and deploy applications independently from the underlying SDN technology.

I. INTRODUCTION

The SDN paradigm allows networking hardware to be made “malleable” and remotely manageable by so-called SDN-controllers. There are different closed-source SDN controller frameworks like the 'Big Controller' [1], etc. and open-source SDN controller frameworks like OpenDaylight [2], Floodlight [3], etc., each with their own programming model and development tools. However, developing an SDN App for a specific controller will tie the developer to the SDN controller framework of choice. i.e. normally, SDN Apps cannot be easily ported from one SDN controller framework to another, not to speak of the possibility of using a development tool (for example, a debugger) of one controller framework in another framework. Moreover, most of these approaches do not integrate SDN-specific tools such as debugging functionality for network states, compatibility checks, or support the composition of SDN Apps. Each of the aforementioned SDN-controllers also provides a different way to configure and deploy SDN Apps on it. This makes deploying SDN Apps on test or production environments complicated, in particular for network administrators without a programming background.

Within the NetIDE project, we aim at providing a one-stop solution for developing SDN Apps. The main ingredient is an Integrated Development Environment (IDE) that supports the whole development lifecycle for SDNs, from the implementation of SDN Apps to testing and deployment, and predicting or evaluating their performance. The IDE furthermore enables network administrators to configure and deploy SDN Apps independent from the applied controller framework.

In this paper we present a preliminary version of the NetIDE method, delivering an integrated process for SDN development using the IDE. A preliminary prototype \cite{2} of the NetIDE Developer Toolkit demonstrates parts of the method implemented as proof-of-concept. It will ultimately provide tool support to manage the whole lifecycle of an SDN App.

II. NETIDE METHOD

NetIDE supports developers and network administrators in five different phases of an SDN App life cycle: Development, Building, Testing, Deployment, and Execution on productive systems. The NetIDE-method is a process for developing SDNs using NetIDE, i.e. the activities and artefacts that are produced or processed in each phase. SDN App developers start by implementing the SDN App and annotating it with metadata which network administrators use for configuration and deployment. SDN developers then compile the SDN App and package it together with its metadata as a deployable package in the Build Phase. The package can be deployed on a simulation environment for testing. If the SDN App is considered stable, it is shipped to network administrators. They configure the SDN App based on the metadata and deploy and execute the SDN App. NetIDE also supports monitoring and profiling at runtime. In the remainder of this section, we explain the actions and artefacts of each life cycle phase.

A. Development Phase

Developers implement SDN Apps during the Development Phase producing SDN App code for a controller of their choice. Aside from implementing SDN Apps, developers create metadata in order to specify data that is used in the Deployment and Execution Phases later on in the process (cf. Secs. II-D and II-E). We explain this metadata in the following paragraph.

System requirements specify a set of systems on which an SDN App can be deployed. This includes hardware as well as software requirements. A Parameter specification enables developers to specify an interface that allows network administrators to configure an SDN App before deploying it. Parameters can be for example firewall rules or a preferred routing algorithm for routing applications. Templates define...
how the parameters are reflected in the SDN App code. Topology requirements specify a set of topologies on which an SDN App can operate correctly. A Composition specification allows the interconnection of multiple SDN Apps. Developers can specify a composition of SDN Apps for different controller frameworks in order to form a single SDN App which combines the functionality of the SDN Apps it is composed of. A Composition Specification specifies how events from switches are propagated and how to handle flow mods coming back from the controllers in order to produce a composed SDN App.

B. Build Phase

The build phase includes generating composition code i.e. the transformation from a composition specification (cf. Sec. II-A) to runnable code. A transformer generates executable Composition Code which reflects the composition specification. If needed, the SDN App code and the composition code are compiled into binary and/or object code by the existing compilers for the respective languages. At the end of the build phase, the IDE creates a package containing all the artefacts named above. This package can be used for testing or passed on to network administrators for deployment.

C. Simulation / Test Phase

Before running an SDN App on a productive system, developers can test it in a virtual simulation environment (e.g. Virtualbox [4] running Mininet [5]). Developers have to set up a simulation environment, i.e., specify the hardware and software running on the target machine. They can also set up a virtual network with a graphical editor. The NetIDE Package is deployed on a respective virtual machine with a virtual network. Developers can now run the SDN Apps from the package and a network simulator on the machine. Testing involves the execution of SDN Apps and debugging, profiling, etc. (cf. Sec. II-E). The IDE supports to automatically set up the virtual machine with a network simulator and controllers.

D. Deployment Phase

It is the task of a network administrator to perform the steps for deployment on a productive system in this phase. Network administrators deploy a (composed) SDN App on a target system. The target system comprises a controller with a given hardware and software configuration as well as a physical network topology. With a NetIDE package described in Sec. II-A, network administrators and IDE go through the following steps before deploying the package.

The IDE checks whether the target system fits the requirements specified in the Development Phase and stored in the NetIDE Package. It obtains information about hardware and software configurations of the target system and verifies them against the system requirements. The IDE will issue warnings for unmet hardware requirements and suggest the installation of missing software.

Network administrators have to assign values to the specified parameters in the deployment phase. NetIDE displays an input form based on the parameter specification to be filled out by network administrators. Parameter values are then passed to the SDN Apps with a template engine.

The IDE also matches topology requirements. The network topology of the target system is automatically matched to the topology requirements in the NetIDE package. Concrete topologies are obtained by a topology discovery on the target controller. There can be three outcomes of that matching: a single matching, multiple matchings or no matchings at all. Depending on the outcome, network administrators can either deploy the NetIDE App directly, have to choose a concrete matching, or are unable to deploy the App. Network administrators have to choose between the proposed matchings before deployment. The switch IDs and properties of the concrete topology will then be passed to the SDN App with a template engine similar to passing parameters.

E. Execution Phase

NetIDE executes the (composed) SDN App on a simulation environment or productive system in the execution phase. Controllers are launched automatically with the according parameters via scripts executed on the target system. NetIDE interacts with the Network Engine [6] in this step. The Network Engine runs on the target system and manages the composition of SDN Apps according to the composition specification. It also provides an interface to load NetIDE packages and manages the execution of SDN Apps. Another module of the Network Engine monitors running SDN Apps and produces Debug and Profiling Information if chosen by administrators or developers. This information is passed to and visualized in the IDE. Developers also have the possibility to manipulate flow tables or, in virtual networks, change the topology in a simulation environment at runtime for debugging purposes.

III. Conclusion and Future Work

We have described a method for end-to-end SDN App development in this work. The method describes the implementation, building, simulation/test, deployment, and execution of SDN Apps. It furthermore enables developers to enrich SDN Apps with metadata such as parameters, requirements, and more. SDN Apps developed with that method can ideally be deployed through the IDE on a productive networking environment by administrators with little programming knowledge.

We will continue implementing the aforementioned features of the method in future versions of NetIDE. This includes a specification language for SDN App composition, a topology pattern specification language, and editors for other kinds of metadata. Finally, the whole method will be supported by the IDE. We will assert the feasibility of the method in various case studies and use their results to improve the method.

REFERENCES