Model-Driven Networking: A Novel Approach for SDN Applications Development

Felipe A. Lopes, Marcelo Santos, Robson Fidalgo, Stenio Fernandes
Center of Informatics (CIn)
Federal University of Pernambuco (UFPE)
Recife, Brazil
{fal3, mabs, rdnf, sflf}@cin.ufpe.br

***Abstract*— Software-Defined Networking (SDN) has been receiving a great deal of attention from both academic and industry communities. One reason to this interest is that SDN enables the network programmability, through an external controller, which supports applications and policies built from SDN programming languages, thus breaking the traditional bind between control and data plane. Nevertheless, the application development in this context is still complex for such recent technology. Moreover, there is a strong need for methodologies and tools that explore the abstraction levels potentials supported by SDN. This paper presents a new approach based on the Model-Driven Engineering (MDE) paradigm, called Model-Driven Networking (MDN). MDN relies on a Domain-Specific Modelling Language (DSML) to create SDN applications. We argue that MDN raises the level of abstraction on development, thus reducing the complexity to implement SDN applications and avoiding inconsistent policies. In order to show the relevance and the technological viability of our proposal, we have specified a DSML and have built a tool for creating SDN applications using the MDN approach.**

Keywords— Domain-Specific Modeling Language; Model-Driven Engineering; Software-Defined Networking.

I. INTRODUCTION

The Internet infrastructure has become complex and hard to manage. The need to make the Internet more dynamic and to enable new management techniques has driven the development of a new paradigm called Software-Defined Networking (SDN). Concisely, SDN separates the control plane from the forwarding one, by moving such control plane to an external controller. This controller executes SDN policies and rules as algorithms or applications that determine the network behavior [1]. In such a scenario, the communication between the control and data plane can be enabled by the well-known OpenFlow protocol [2].

Although the SDN architecture enables network programmability, SDN does not make it easy for developers and network operators. Currently, many approaches exist to enable the development of SDN applications or even to specify policies for network behavior [3][4]. They are mostly based on *Domain-Specific Languages* (DSL) paradigm. However, there is still a number of issues regarding the development of correct and effective SDN applications.

In this context, considering that DSL hides low-level implementation details, which speeds up and makes easier the software development process [5], it is worth emphasizing the two types of DSLs: textual and visual [6]. A visual DSL, called *Domain-Specific Modeling Language* (DSML), tends to be easier than textual DSLs. Modeling languages provide higher level of abstractions than textual ones, especially because the visual notation of a DSML typically aims to abstract the textual representation of codes. Moreover, the *Computer-Aided Software Engineering* (CASE) tools supports DSMLs, which can early detect errors in the development process [7]. These concepts are part of a major paradigm called *Model-Driven Engineering* (MDE) [5].

Considering the concepts introduced above, this paper proposes a new concept called Model-Driven Networking (MDN), which associates the MDE and SDN paradigms. In order to demonstrate the feasibility, expressiveness, and usefulness of our proposal, we have specified a DSML and developed a CASE tool prototype that enables the development and management of SDN applications. Such composition makes possible for a network operator to interact with a MDN diagram by inserting, excluding, and editing network elements. MDN also automatically generates code and ensures that invalid constructions are avoided. The MDN tool is available at https://github.com/felipealencar/mdn.

The remainder of this paper is organized as follows: Sections II and III present the technical background on SDN and MDE. Section IV presents our proposal, whereas Section V covers the body of knowledge on similar approaches. Finally, Section VI draws some conclusions and provides directions for future work.

II. SOFTWARE-DEFINED NETWORKING (SDN)

The separation between control and data planes is the pillar of the SDN paradigm. SDN aims at reducing the complexity to deploy new protocols and new services, through simplification of network management [1]. SDN arises from the need that network operators and applications have to implement new strategies, supporting the increasing complexity of networks.

A. SDN Overview

Since the control plane is decoupled from the forwarding devices, all the network intelligence is moved to the SDN controller. Such controller becomes the network component responsible to manage the network devices [2].

To manage the network, SDN uses flow tables present in SDN switches. Such tables have flow entries, which contain policies or network rules defined by SDN controller. Flow entries instruct the switch about how to handle the incoming

This work is supported by the FACEPE under Grant no. IBPG-0738-1.03/12 for Felipe A. Lopes, and IBPG-1321-1.03/11 for Marcelo Santos; CNPq 304422/2013-4 for Stenio Fernandes.

packets. As SDN switches do not implement the control plane, they need to connect with an SDN controller to manage them through applications, policies, and algorithms. Currently, the OpenFlow protocol [2] is widely accepted as an open standard that allows controllers to modify the flow tables in switches. Figure 1 depicts the role of OpenFlow in SDN.

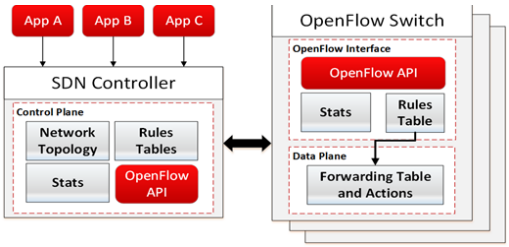

Fig. 1: The OpenFlow perspective in an SDN architecture.

B. *SDN Applications*

One of the main benefits provided by SDNs is the network programmability. Part of this programmability is made through SDN applications, which can deliver highly scalable, efficient, and manageable network services [1] (e.g. load balancing, access control). These applications may run on the SDN controller (cf. Fig. 1), resembling the common organization of traditional operating systems and software applications.

While most discussions involves the use of SDN in datacenters, research efforts started to explore how to use and to create SDN applications to build networks with flexible and more secure management. We emphasize that to date there is no a widely well-accepted standard process for developing SDN applications. Despite there are proposals involving DSL-based *SDN languages* to support the development of applications [3], they have drawbacks involving the dependency of controller, error-prone implementation, and lack of automated validation.

III. MODEL-DRIVEN ENGINEERING (MDE)

In order to add new capabilities on the network, both academic and industrial researchers have started to use methods from the Software Engineering discipline in achieving the fully potential of SDN through standardization of programming languages or interfaces. We now describe MDE to provide a view of its benefits for SDN development.

MDE is an approach used to reduce the complexity in software development [5]. It is composed of *process* and *analysis* with an underlying architecture of models as the key artifacts of development process. Such process is known as Model-Driven Development (MDD).

A. *Domain-Specific Modeling Language (DSML)*

According to [5], DSML formalizes the infrastructure and defines the behavior within specific domains (e.g., software-defined radio, online medical records, or database management). A DSML compose a MDE technology and consists of metamodels to define the domain concepts and relationships. Moreover, DSMLs are supported by Computer-Aided Software Engineering (CASE) tools, which enable the detection of errors in applications or even provide prevention of them. CASE tools also guide towards preferable design patterns, enable verification of completeness, support full code generation, and provide consistency for specifications [7].

Formally, in [8] a DSML is a 5-tuple composed of concrete syntax (CS), abstract syntax (AS), semantic domain (S), semantic mapping (MC), and syntactic mapping (MS), as defined in (1):

$$L = < CS, AS, S, M_C, M_S > \quad (1)$$

1) Semantic Domain: it presents a meaningful mathematical formalism, in terms that define the meaning of models;
2) Abstract Syntax: it defines the concepts, relationships, and integrity constraints present at the language;
3) Concrete Syntax: it defines the specific notation used to express models, which are graphical, textual, or both;
4) Semantic Mapping: it refers to syntactic of semantic domain;
5) Syntactic Mapping: it assigns syntactic constructions (graphical, textual, or both) for elements of abstract syntax.

Any DSML requires in its creation a precise specification of these five components.

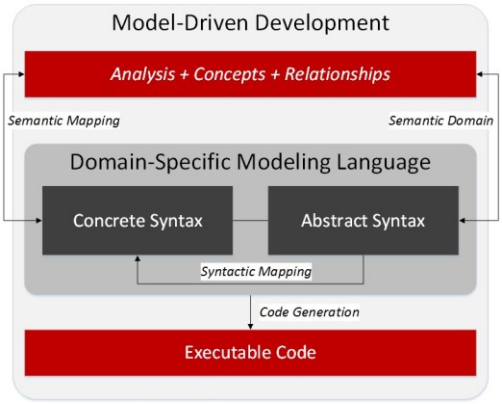

Fig. 2 Scheme for Model-Driven Development and the components of DSMLs.

B. *Frameworks for MDE*

Currently, several tools exist to support the MDE approach and the transformation of models to executable code through the creation of graphical editors (e.g., Graphical Modeling Framework (GMF), MetaEdit+, and AToM³) [9]. Due to the particularities of GMF (i.e., open-source, public license, and support to integrity constraints), it is the tool used in our proposal.

The GMF is a framework that supports the mapping between models and artifacts of MDE, including executable code. It enables the generation of editors (e.g., CASE tools) that supports the MDE paradigm.

The actions and artifacts mentioned in this section, which are used to apply the MDE approach in several fields of knowledge, bring several benefits to the development and management of complex applications (e.g., reduced complexity, less error-prone, meaningful validation) [5].

IV. MODEL-DRIVEN NETWORKING

The concepts defined so far provide the necessary background to understand a prospective relationship between MDE and SDN. This section presents the workflow and DSML created to offer a new approach in SDN applications development, called Model-Driven Networking (MDN).

Due to lack of space, we have omitted the description of mappings and present: (1) the specification of metamodel based on Eclipse implementation of MOF metalanguage; (2) the graphical elements; and (3) the code generation. As result, these elements (besides the mappings of DSML) compose a MDE-based graphical editor for modeling SDN applications, hereinafter called MDN editor.

In order to offer and build a DSML based on the MDE paradigm, it is first necessary to define a domain scope [10]. Although there is no well-known complete set of definitions for SDN applications, this paper uses the domain scope defined in [11]. Such domain scope focuses in management entities:

i. *Time*: network operators need to implement policies where the network behavior depends on date or time on some day.
ii. *Data usage*: network operators specify constraints where the network behavior depends on the amount of data utilized by its users.
iii. *Status*: the network should enable operators to specify privileges for different groups of users or devices, according to their status.
iv. *Flow*: the controllers manages the network behavior information of flows, which the packet header provides.

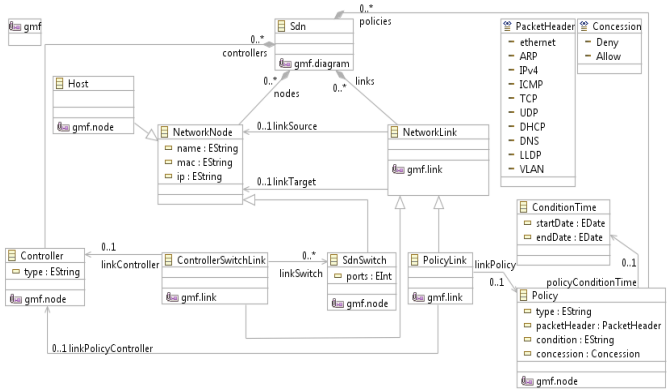

Fig. 3 Abstract syntax of MDN.

After the definition of domain scope, the semantic domain (i.e., first requirement of the DSML specification) provides entities to describe through metamodeling, defining their attributes and relationships. For instance, a link must connect at most one source and one destination (e.g., host, switch). The Fig. 3 depicts our metamodel implementation using ECore (the MOF specification of Eclipse Project [12]). The metamodel definition was created in parallel with the domain model, which respectively refer to abstract syntax and concrete syntax of our DSML. The creation of such artifacts meets the second and third requirements of DSML specification. Thus, we defined our concrete syntax by assigning a visual element for each abstract entity from metamodel. The complete list is available online at https://github.com/felipealencar/mdn.

TABLE I. GRAPHICAL ELEMENT CONCEPTUAL

Visual Notation	Element
Controller ip 10.0.0.10	*Abstract Syntax*: Controller. *Attributes*: entity name (EString); IP (EString).

B. Rules and Validation of SDN Applications

One of the problems present at the development of SDN applications is the inconsistency between policies. Currently, with DSLs or some API for SDN, network operators and developers specifies these policies in an error-prone textual way. The use of MDN solves such problem by using a strategy based on the Object Constraint Language (OCL). OCL applies rules of building and perform validation in system models [13]. In MDN, we defined a set of OCL constraints to identify inconsistencies (if any) in such models (e.g., ambiguous policies, duplicated MACs). Table 3 shows two examples of constraints based on OCL for SDN.

TABLE II. CONSTRAINTS FOR SDN MODELS

Objective	OCL Rule
Avoid null IP	`constraint notNullIP {` `  check : self.ip.isDefined()`

Unique MAC	`constraint uniqueMAC {`
	`  check {`
	`  Network.all.select(n|n.mac = self.mac);`

C. Modeling an SDN Application for Access Control

In order to validate the modeling of SDN applications, this approach utilized the MDN editor itself, generated through the previous steps, to create an SDN application by modeling.

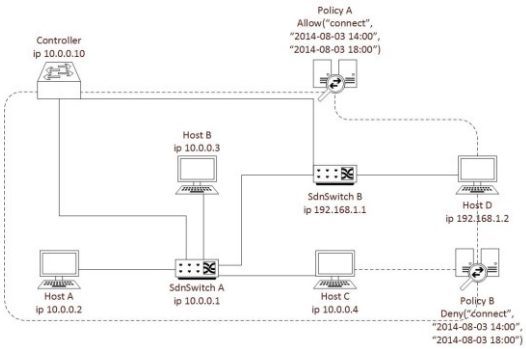

Fig. 4 Example of access control application modeled in MDN.

The modeled application aims to control network access of users or devices based on time, but it is noteworthy the extensibility of scope. Fig. 4 depicts this context with a model based on our concrete and abstract syntaxes.

D. Code Generation

The MDN enables the code generation from models. To implement our approach, we generate executable code to POX controller, but such generation is extensible for any controller. MDN uses Eclipse Generation Language (EGL) [12] to perform the code generation through EGL-based templates.

Due to lack of space, we do not show the complete way to generate code in the MDN editor. Concisely, the Fig. 5 presents this generation that replaces template tags ("[% %]") with information provided at model above (cf. Fig. 4). Then, POX controller executes the generated code.

```
[% var header = getCodeHeader(); %]
def mdn_handler (event):
 packet = event.parsed
ip = packet.find('[%=host.packetHeader%]')
  # This packet isn't IP!
  if ip is None:
   return
```

Fig. 5 Snippet from one of the templates used to generate code.

Although the MDN's current code generation focuses on POX controller, its extensible structure enables the addition of more controllers or programming languages. This might be necessary to increase the models compatibility, enabling them for managing networks with different underlying infrastructure (e.g., different versions of OpenFlow specification).

V. Related Work

Increasing the abstraction in development or management of SDN applications is the objective of several research projects. To the best of our knowledge, the closest related work found in literature to modeling SDN and its applications are Miniedit [14], Virtual Network Descriptor (VND) [15], and Common Information Model for SDN (CIM-SDN) [16].

TABLE III. Features Comparison for SDN Modeling

Main Features for SDN Modeling	Miniedit	VND	CIM-SDN	MDN
Topology Model	YES	YES	YES	YES
Model behavior and rules	NO	NO	NO	YES
Model applications	NO	NO	NO	YES
Support for multiple controllers	YES	YES	YES	YES
Support for DSLs	NO	NO	NO	YES
Generate executable code	YES	YES	YES	YES
Validation	NO	NO	YES	YES
Descriptive graphical elements	YES	YES	NO	YES
Percentage	50%	50%	37.5%	100%

The Miniedit and VND refers to the creation of virtual topologies in SDN through a graphical interface. These topologies are simulated at Mininet, the underlying SDN simulator [14].

CIM-SDN offers an abstraction for SDN management, modeling SDN elements (e.g., hosts, controllers, switches). Similar to our proposal, CIM-SDN also validates an SDN structure through OCL, finding inconsistences and avoiding error-prone buildings in network.

In order to identify our contributions, Table III shows the comparison of features for SDN modeling present in Miniedit, VND, CIM-SDN, and our proposal MDN.

To realize the comparison in Table III, we identified the main features (first column) of each proposal related to modeling SDN, and then we verified the possibility to perform these functions through the MDN.

We state that MDN clearly fills a gap in the literature, especially regarding the SDN applications development.

VI. Concluding Remarks

In this paper, we propose a novel approach to address current issues involving the development of SDN applications and its complexity. Our main contribution, namely Model-Driven Networking (MDN), provides a DSML deployed in an editor that enables the modeling of SDN applications.

The benefits of the MDN proposal are manifold: i) it enables formal modeling of SDN applications; ii) it is capable of generating executable code; iii) it is vendor agnostic; and iv) it is capable of graphically describing the elements of an SDN architecture (including its applications).

As future work, we envisage a formal mathematical definition of domain scopes to develop SDN applications. We also plan to define code-to-model transformations, and evaluate qualitatively our MDN editor and its concrete syntax.

Acknowledgments

Authors would like to thank the Pernambuco State Government (Brazil), FACEPE (Grants no. IBPG-0738-1.03/12 and IBPG-1321-1.03/11), CAPES, and CNPq (Grant no. 304422/2013-4).

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[2] N. McKeown and T. Anderson, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM …*, pp. 69–74, 2008.

[3] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128–134, Feb-2013.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," *Proc. 10th USENIX Conf. Networked Syst. Des. Implement.*, pp. 1–14, 2013.

[5] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer (Long. Beach. Calif).*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[6] A. Van Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography.," *Sigplan Not.*, vol. 35, no. June, pp. 26–36, 2000.

[7] A. F. Case, "Computer-Aided Software Engineering (CASE): Technology for Improving Software Development," *ACM SIGMIS Database*, vol. 17, no. 1, pp. 35–43, 1985.

[8] D. Harel and B. Rumpe, "Meaningful Modeling: What's the Semantics Much confusion surrounds the proper definition of complex modeling," *IEEE Comput.*, vol. 37, no. 10, pp. 64–72, 2004.

[9] N. Baetens, "Comparing graphical DSL editors: AToM3, GMF, MetaEdit," 2011.

[10] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *IEEE Softw.*, vol. 20, no. 5, pp. 36–41, Sep. 2003.

[11] H. Kim and N. Feamster, "Improving network management with software defined networking," *Commun. Mag. IEEE*, no. February, pp. 114–119, 2013.

[12] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[13] "OCL 2.0." [Online]. Available: http://www.omg.org/spec/OCL/2.0/. [Accessed: 18-Sep-2014].

[14] B. Lantz, B. Heller, and N. Mckeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[15] R. Fontes, "Visual Network Description: A Customizable GUI for the Creation of Software Defined Network Simulations," in *27th European Simulation and Modelling Conference - ESM'2013*, 2013, no. Lantz.

[16] B. Pinheiro, R. Chaves, E. Cerqueira, and A. Abelem, "CIM-SDN: A Common Information Model extension for Software-Defined Networking," *2013 IEEE Globecom Work. (GC Wkshps)*, pp. 836–841, Dec. 2013.