Interfacer: a Model-Driven Development Method for SDN Applications

João Eurípedes Pereira Júnior, Flávio de Oliveira Silva, João Henrique de Souza Pereira and Pedro Frosi Rosa

Abstract In order to deliver network services using Software-Defined Networking (SDN) elements of the three layers, namely infrastructure, control, and application are necessary. Infrastructure and Control layer elements are in a more mature state. They had received more extensive research and already count with some product options available in the market. SDN Applications, however, are not commodity software that can be deployed into different control and infrastructure layers. Applications should be built in a more customized fashion, and seamless integrated with existing SDN infrastructure and control. This paper explores the Interfacer, a Model-Driven Development (MDD) approach to SDN application development and integration in order to deliver high-quality network services. This method uses ontology-driven conceptual modeling to capture essential aspects of existing and to be developed components of a SDN network architecture as well how they should better interface and integrate into models. These models are transformed into source code that respects the requirements of existing components and enforce the requirements of SDN applications been developed leading to higher continuity and lower time to market and maintenance cost of SDN services. These models also promote communication and learning improvements in developer community accelerating the development process and minimizing risks. A case study is reported illustrating the application of the method. In this case study we perform the analysis and refactoring of the Entity Title Architecture (ETArch), an SDN based network architecture that is deployed over an OpenFlow capable infrastructure.

João Eurípedes Pereira Júnior
Federal University of Uberlândia, Brazil e-mail: joao@ufu.br

Flávio de Oliveira Silva
Federal University of Uberlândia, Brazil e-mail: flavio@ufu.br

João Henrique de Souza Pereira
Federal University of Uberlândia, Brazil e-mail: joaohs@ufu.br

Pedro Frosi Rosa
Federal University of Uberlândia, Brazil e-mail: pfrosi@ufu.br

1 Introduction

Software Defined Networking (SDN) facilitates the implementation of network services by combining components in a bottom-up approach. However, high quality network services rely on complex SDN applications which also demand a top-down approach to materialize a unified service view throughout the network while compatibility with underlying components is enforced by the bottom-up approach. Moreover, such SDN application implementation strategy requires additional processes and artifacts to manage and automate the development. Generally, this features are delivered by Model-Driven Development (MDD) methods. To our knowledge there is no model-driven development approach specific for developing complex SDN applications. In this paper we explore the Interfacer, a model-driven development approach for SDN applications.

SDN starts from three simple ideas: generalize network hardware establishing common interfaces in order to provide a standard collection of packet-processing functions, a logically centralized and decoupled control layer makes decisions upon an up-to-date global view of network state and summarizes the network state for applications and translates application requirements to low-level rules. Additionally, applications are programmable and network aware so that they can enforce their communication requirements. These principles make it possible to evolve the network infrastructure without having to change the underlying hardware, to evolve network applications with fewer changes in the underlying control software and enable expressing network algorithms in terms of appropriate abstractions for particular applications [1] [2]. However, obtaining these appropriate abstractions is not an easy task. Despite several initiatives in the related work the problem still is considered challenging.

This paper explores the fact that proposing good abstractions frequently require deep knowledge and long experience in a system or domain. It proposes a structure to support knowledge and experience, helping to materialize a top-down approach to manage SDN application development complexity. it uses Ontology-Driven Conceptual Modeling (ODCM) to build shared and formal conceptualizations of the problems, the method helps to isolate development responsibilities by introducing artifacts and promotes more core reuse by transformations. Additionally it provides a formal and shared understanding of the domain that developers can use learn, communicate, and write and test code. ODCM is the activity of formally capturing community knowledge about a domain or system. The aim of this method is to introduce representations of knowledge in the form of artifacts simplifying interface and abstraction definition and accelerating the development process.

The remainder of this work is organized as follows: Section 2 presents the state of the art by comparing our approach with the related work. Section 3 describes the Interfacer, a method that applies model-driven development to SDN applications. Section 4 shows a case study that applied the Interfacer into an SDN based network architecture, called Entity Title Architecture (ETArch). Finally, Section 5 presents some concluding remarks and future work.

2 Model-Driven Development for Networks

This Section presents the related literature available. We have first realized that research addressing the problem of SDN application development are frequently related to the concept of interface. Much research has been made in order to improve SDN application development (i.e. NetKAT [3], Frenetic [4], Ravel [5], Gavel [6]) and ONF initiatives such as Intents [7] Northbound Interface (NBI) and the Open Information Modeling (CIM) of Open Network Foundation (ONF). Besides the advancements in programming languages and interfaces, complex SDN application development continues to be a challenge.

Complexity of SDN application development makes modeling an ideal candidate for improvements as already verified by related works. Although the SDN architecture enables network programmability, SDN does not make it easy for developers and network operators [8]. The Model-Driven Networking also applies Model-Driven Development (MDD) to SDN development recognizing the benefits (e.g., reduced complexity, less error-prone, meaningful validation) to the development and management of applications but it aims at delivering simpler generated source code that can be executed. In other related work, MDD is applied to cloud computing. It defines MDD as a methodical software development approach based on three primary activities i.e. modeling, model transformation, and verification. It acknowledges that the application of such methods incorporates the features of fast development, reusability, and portability. Such sophisticated features are highly supportive and aligned with the implementation requirements of cloud computing. Consequently, MDD is considered an effective and attractive development approach for cloud computing [9].

This related work joins Infrastructure as a Code technique with MDD in order to deliver data-intensive architectures. It advocates that this trend of using software engineering techniques that reduce the space, time, and efforts between software development and operations, as well as the technical and organizational distance between these two types of software teams, is known as DevOps. As part of the DevOps menu, many practices entail re-using standard tools from software development (e.g., code-versioning, code-revision management, etc.) to manage what is known as the Infrastructure-as-Code (IaC) approach. In this context, it has experimented with MDD and seen that power of abstraction and its automation potential simplified IaC development[10]. Although model-driven development has been already proposed in other works, none of then focus specifically on SDN application that is currently a very important issue.

3 The Interfacer

The Interfacer is a Model-Driven Development approach that helps to manage SDN application development complexity by introducing artifacts in the development process. One type of artifact that the Interfacer introduces is the model. Models

capture high level (architectural) abstractions, giving a complete view of the solution and the relationship between its parts, establishing a formal and shared vocabulary that promotes developers team communication and learning. Another type of artifact the Interfacer introduces is the transformation code. Transformation code uses models as input and produces SDN application source code as output. The goal of transformation codes is to reduce abstraction from model level to programming language level (i.e. object-oriented, functional, procedural). These artifacts are introduced and improved in a cyclic way divided into 4 phases: Modeling, Transformation, Implementation, and Testing as shown in Figure 1.

Modeling captures essential knowledge of community about the system using ontology-driven conceptual modeling [11] and stores it into models expressed in OntoUML language. Transformation incorporates architectural decisions and transform model elements into generated code that instantiate good architectural patterns using Eclipse Acceleo plugin and OntoUML meta-model [12]. Implementation is the phase where SDN developers actually write some code. With knowledge and design issues isolated in precedent phases, this phase encompasses only programming language specific decisions that must obey earlier phases constrains simplifying and accelerating the implementation. In testing phase network services are tested and its compliance with the model and the architectural elements can be verified. The results are used as feedback to fix implementation or improve the model in subsequent cycles. Testing concerns not only code tests but also network service tests (such as performance, security) that can also be generated from models and transformations.

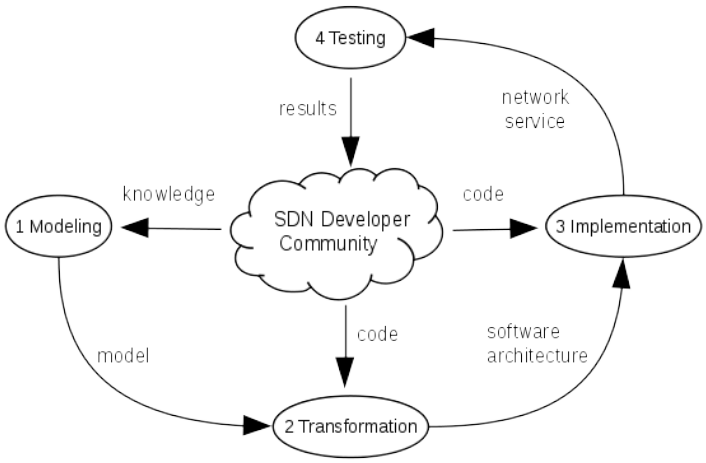

Fig. 1 Interfacer: Model-Driven Development for SDN Applications

The models created in the modeling phase must work in a complimentary way with SDN reference architecture. The concepts defined in these models should represent some SDN plane element and concepts from different planes must communicate through standard interfaces. Interfacer helps to seamless integrate SDN appli-

cations into existing infrastructures as it can model the interfaces of reused software (i.e. SDN Controllers) and abstraction of applications been developed. Additionally, the Interfacer helps to define interfaces that do not compromise the general purpose of SDN networks by leveraging a special application to manage application interfaces, so that network control can use standardized interfaces while (user) applications can define more specific interfaces when needed. Once development project is mature Interfacer provides much less knowledge to service time for huge alterations in the project meeting SDN demands for agility in network service deployments.

3.1 Knowledge to Model

In order to represent knowledge into models, we use ontology-driven conceptual modeling [13]. Ontology-driven conceptual modeling is the act of capturing and formalizing how a community perceives a domain or system of interest, using modeling primitives inherited from a foundational ontology. Moreover, we use the Unified Foundational Ontology (UFO) as foundational ontology to identify the fundamental distinctions between types and individuals, taking into consideration ontological properties, such as rigidity, identity, and dependence. In software-defined networking, the individuals can be the programmable switches, routers, or any hardware composing the network, and processes running software that controls the network, or provides some application to his users.

Unified Foundational Ontology [14] provides a set of universals that behaves as a set of meta-properties and are used as class or association stereotypes. A model is constructed by grouping some stereotyped classes and associations so that the classes and associations of the models will follow a formal meta-property behavior that represents an actual network behavior. Visually a model will appear like a UML class diagram, but it carries a much more expressive semantic due to the use of unified foundational ontology as stereotypes.

Models contain classes, which can be stereotyped as kind, subkind, phase, role or relator. The classes are connected by associations, which can be stereotyped as component of, member of, material, or mediation. The use of ODCM allows the simultaneous modeling of diverse aspects of the network like architecture, performance, security, energy-efficiency and user quality of service producing an agile, cheap, compliant and consistent architecture design that can be refined in successive cycles while automatically generates code that reflects model changes. It isolates the high-level aspects of design concerns formally promoting its completeness and soundness. A formal representation of knowledge about network leverages developer community communication and learning leading to more precise development process and results.

Figure 2 is a model diagram that depicts our vision of a reference SDN application consisting of several application components such as main, controller and client components. The main component resides in the core of SDN application layer. The controller component resides inside the SDN controller and is respon-

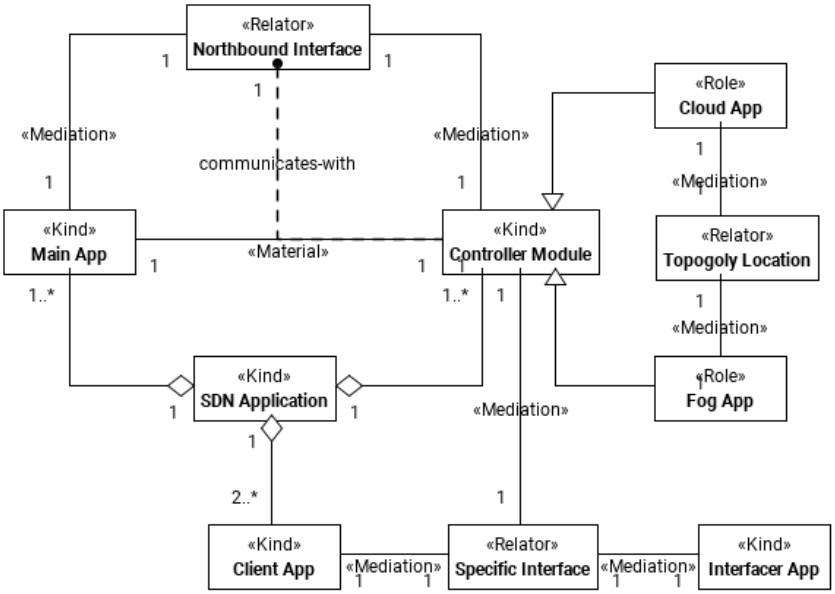

Fig. 2 Example of SDN Application Model

sible for the translation between application and controller abstractions. The client component runs inside the user device, it requests and uses network services provided by the main component. These components have different functions and may have different behaviors under different circumstances. We model these using roles (e.g. a controller component of the application may behave as a cloud controller component or as a fog controller component depending if it is located in the core or in the edge of the network). This models help to separate knowledge about the SDN application from source code allowing knowledge reusability and improving developer community learning and communication.

3.2 Model to Code

We use the Eclipse Acceleo [15], [16] plugin and the OntoUML meta-model in order to transform network models into architecture and test code. Writing a template in Acceleo model transformation language we can instruct Acceleo how to transform network models into programming language interfaces, classes, methods, and variables. Transformation code [17] isolates intermediary design concerns, separating architectural from implementation decisions.

The Menthor [18] tool is used to guide the model construction process and to verify OntoUML syntax. Menthor is an OntoUML language editor to build and val-

idate the network models. OntoUML is semantically supported by Unified Foundation Ontologies (UFO). The OntoUML metamodel contains all the UFO universals that are used to interpret the network models. The metamodel is imported into an Eclipse Acceleo Project where templates must be written with the instructions to transform model to code. OntoUML is a UML profile extension that integrates well founded ontological semantic with a graphical modeling language. Listing 1 shows an excerpt of a model transformation language of Eclipse Acceleo plugin.

```
[comment encoding = UTF-8 /]
[module ExampleModule('OntoUML')]

[template public generateControllerModule(aKind : Kind)]

[file (aKind.name + '.java', false, 'UTF-8')]
public [aKind.name/] implements IOFMessageListener,
    IFloodlightModule {

    @Override
    public String getName() {
    // TODO Auto-generated method stub
    return null;
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type, String
    name) {
    // TODO Auto-generated method stub
    return false;
    }
}
[/file]
[/template]
```

Listing 1 Acceleo model transformation language example

```
public ControllerModule implements IOFMessageListener,
    IFloodlightModule {

    @Override
    public String getName() {
    // TODO Auto-generated method stub
    return null;
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type, String
    name) {
    // TODO Auto-generated method stub
    return false;
    }
}
```

Listing 2 Excerpt of SDN application source code generated in file ControllerModule.java

A profile in the Unified Modeling Language (UML) provides a generic extension mechanism for customizing UML models for particular domains and platforms. Extension mechanisms allow refining standard semantics in a strictly additive manner, preventing them from contradicting standard semantics. Profiles are defined using stereotypes, tag definitions, and constraints which are applied to specific model elements, like Classes, Attributes, Operations, and Activities. A Profile is a collection of such extensions that collectively customize UML for a particular domain.

Using a modeling transformation language SDN architects can specify which type of programming structures should replace each occurrence of a determined stereotype as shown in listing 1 and steer developers to implement known best pattern as shown in listing 2. For example, each element of the model stereotyped with the universal kind should be replaced by a Java interface or class with the same name of the model element, and different role universals associated with this kind should be replaced by specific methods within his respective interfaces or classes.

3.3 Code to Service

This phase embraces a more traditional development style, however, it benefits from a formal and shared vocabulary delivered by the models of the first phase. This vocabulary must be broadcast inside the developers' team in order to facilitate communication and speed the learning. Some code convention may also be delivered from this vocabulary so that the whole code reflects the model concepts.

In order to transform the code into service developers must complete mandatory Java class methods and have a little understanding of how SDN infrastructure works. With this methodology interfaces and software architecture decisions can be separated from implementation decisions, and model improvements can be quickly transmitted to code in a top-down approach.

An Infrastructure-as-Code is used to enforce control of model and code version to allow reuse of implementation code compatible with different model versions. Additionally, modeling and transformation can also be used to provide the testing code to verify the service.

4 Applying Interfacer to the Entity Title Architecture (ETArch)

This section presents a case of study where we apply the Interfacer: Model-Driven Development for SDN Application method to the Entity Title Architecture, discussing his limitations and some results.

4.1 Entity Title Architecture (ETArch)

ETArch [19] is a clean-slate architecture with a natural match with SDN [20] concept. ETArch's goal is to support the new applications requirements such as multicast, mobility, QoS, and security. ETArch's design goals consider a new horizontal addressing scheme that solves the ambiguity of identification and location, then making easier mobility and multicast communication and mitigates the need of various addresses (MAC/IP/TCP), wherein ETArch uses only a unified address for communication between entities. A flexible Communication layer, semantic oriented, capable of supporting different applications requirements and an advanced network control plane to establish a communication context that provides guarantees to the communication requirements over time for satisfactory user experience. In more depth, ETArch has a communication context, namely *workspace*, which is a logical bus, naturally multicast. In practical terms, the workspace supports the dialog of a set of *entities*, which take part in a communication context. This communication context provides security, QoS, QoE, multicast communication, and mobility for these entities over time.

Another important concept of ETArch is the Domain Title Service (DTS), which is a distributed environment that manages all of the control mechanisms at a global scale and solves issues, such as management of entity lifecycle, calculation of the best route for the extension of workspaces, and guarantee of communication quality. These functions are materialized by a DTS Agent called Domain Title Service Agent (DTSA), which is an ETArch controller based on the SDN concept. To sum up, the key element design applied to ETArch is flexibility, which allows this architecture to handle a broad array of requirements, while it can still be generic enough to support the technological evolution. Through these considerations, ETArch can uphold different services from distinct paradigms such as Service-Centric, User-Centric, Information-Centric, and the current Host-Centric.

4.2 Limitations of Entity Title Architecture

Despite ETArch's design goals consider a horizontal addressing scheme that makes mobility and multicast easier, and its capacity to uphold distinct network service paradigms. It has only delivered a content-centric transport service known as Workspace. Although, initial implementation was straightforward, further development proved the architecture has serious service limitations. The first, regards to service extensibility, the use of automata to define a communication between what instantiates client and main components of SDN application leads to a simple language, however every feature addition implies in code changes in both components. The second, regards to service scalability, architecture application and control are tightly coupled meaning that every DTSA is an application and also a controller. Additionally, this condition makes the architecture interface abusive (i.e. prohibiting new versions of the SDN controller), as it do not use the standard northbound

interface, without mention the effort maintain the distributed applications of DTS synchronized.

4.3 Refactoring ETArch

By refactoring the architecture we eliminated these limitations, whose causes were only identified as we created models to understand how the code base reflected upon SDN architectural principles. We enforced the use of northbound interface between the main component of SDN application and the controller modules by introducing transformation code. This measure have made application and control loosely coupled and respected standard northbound interface. Further, we have centralized the application that can now communicate with several controllers through this interface, thus making the specific protocol for DTSA inter communication no more necessary.

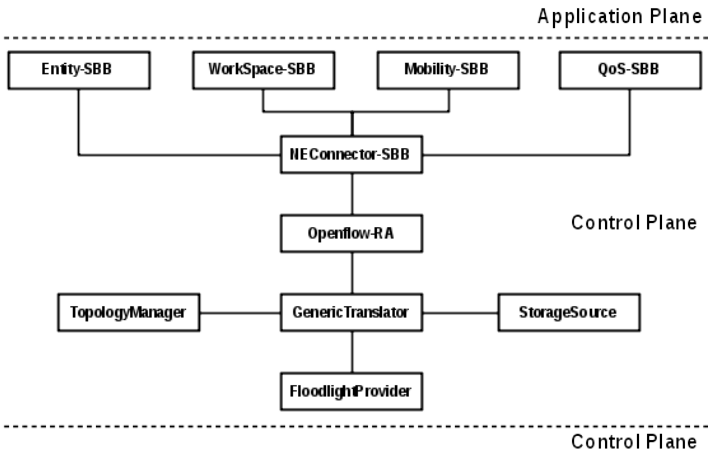

Fig. 3 ETArch components before Interfacer

Figure 3 shows the main components of ETArch before Interfacer. Entity-SBB, Workspace-SBB, Mobility-SBB, NEConnector-SBB, and Openflow-RA are JAIN SLEE components and represent the main component of SDN application. TopologyManager, FloodlightProvider, StorageSource, and GenericTranslator are Floodlight SDN controller modules. The dashed lines delimit SDN planes showing how ETArch components fit into SDN architecture. Figure 4 shows the main components of ETArch after Interfacer was briefly applyed to architecture, resulting in an analysis and refactoring. Note that there are only controller modules in control plane and only application modules in application plane. They must always communicate through northbound interface.

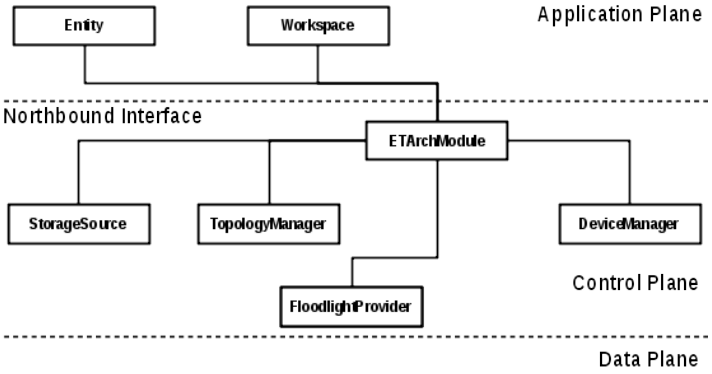

Fig. 4 ETArch components after few Interfacer iterations

5 Concluding Remarks and Future Work

In this paper we have explored the Interfacer, a method for developing complex SDN applications in order to deliver high-quality network services. The method was applied to ETArch, an analysis, that identified ETArch's limitations, and a refactoring, that qualitatively improved ETArch's scalability, were performed as a case study. Alhought, service extensibility issues were not improved by the refactoring, it opened the possibility for future investigation on how the replacement of automata by grammars can help to produce more extensible and expressive communication amid SDN application components while minimizing client application changes.

Another future concern is apply this method to other well known open source SDN architectures, evaluate how its applications perform in terms of SDN architectural principles, and propose some guide lines to fixing eventual issues using models and transformation code.

Acknowledgements This work was has been partially funded by Coordination for the Improvement of Higher Education Personnel (CAPES), FAPEMIG and PROPP/UFU.

References

1. M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2661061.2661063
2. O. N. Foundation. (2018, jun) Northbound interfaces. [Online]. Available: https://www.opennetworking.org/sdn-definition/
3. C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "Netkat: Semantic foundations for networks," *SIGPLAN Not.*, vol. 49, no. 1, pp. 113–126, Jan. 2014. [Online]. Available: http://doi.acm.org/10.1145/2578855.2535862

4. N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2034574.2034812
5. A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, "Ravel: A database-defined network," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 5:1–5:7. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890970
6. O. L. Barakat, D. Koll, and X. Fu, "Gavel: Software-defined network control with graph databases," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, March 2017, pp. 279–286.
7. O. N. Foundation. (2016, oct) Intent nbi - definition and principles. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-523_Intent_Definition_Principles.pdf
8. F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "Model-driven networking: A novel approach for sdn applications development," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 770–773.
9. A. W. Muzaffar, S. R. Mir, M. W. Anwar, and A. Ashraf, "Application of model driven engineering in cloud computing: A systematic literature review," in *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ser. ICC '17. New York, NY, USA: ACM, 2017, pp. 137:1–137:6. [Online]. Available: http://doi.acm.org/10.1145/3018896.3036380
10. M. Artac, T. Borovšak, E. D. Nitto, M. Guerriero, D. Perez-Palacin, and D. A. Tamburri, "Infrastructure-as-code for data-intensive architectures: A model-driven development approach," in *2018 IEEE International Conference on Software Architecture (ICSA)*, April 2018, pp. 156–15 609.
11. M. Verdonck, F. Gailly, R. Pergl, G. Guizzardi, B. Martins, and O. Pastor, "Comparing traditional conceptual modeling with ontology-driven conceptual modeling: An empirical study," *Information Systems*, vol. 81, pp. 92 – 103, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437918303727
12. R. Pergl, T. P. Sales, and Z. Rybola, "Towards ontouml for software engineering: From domain ontology to implementation model," in *Model and Data Engineering*, A. Cuzzocrea and S. Maabout, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 249–263.
13. N. Guarino and C. A. Welty, *An Overview of OntoClean*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 201–220. [Online]. Available: https://doi.org/10.1007/978-3-540-92673-3_9
14. G. Guizzardi, *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology, 2005.
15. J. Mtsweni, "Exploiting uml and acceleo for developing semantic web services," in *2012 International Conference for Internet Technology and Secured Transactions*, Dec 2012, pp. 753–758.
16. H. Benouda, M. Azizi, R. Esbai, and M. Moussaoui, "Mda approach to automate code generation for mobile applications," in *Mobile and Wireless Technologies 2016*, K. J. Kim, N. Wattanapongsakorn, and N. Joukov, Eds. Singapore: Springer Singapore, 2016, pp. 241–250.
17. L. M. Rose, N. Matragkas, D. S. Kolovos, and R. F. Paige, "A feature model for model-to-text transformation languages," in *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, ser. MiSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 57–63. [Online]. Available: http://dl.acm.org/citation.cfm?id=2664431.2664440
18. J. Guerson, T. P. Sales, G. Guizzardi, and J. P. A. Almeida, "Ontouml lightweight editor: A model-based environment to build, evaluate and implement reference ontologies," in *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, Sept 2015, pp. 144–147.
19. F. de Oliveira Silva, M. Goncalves, J. de Souza Pereira, R. Pasquini, P. Rosa, and S. Kofuji, "On the analysis of multicast traffic over the Entity Title Architecture," in *2012 18th IEEE International Conference on Networks (ICON)*, 2012, pp. 30–35.
20. J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing Software-Defined Networks: A Survey," *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.