A Model-Driven Approach to Establishment of Private Blockchain Business Networks

Aleksa Mirković, Branko Terzić, Dušan Gajić, Marina Nenić, Ivan Luković
Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
aleksa.mirkovic@uns.ac.rs, branko.terzic@uns.ac.rs, dusan.gajic@uns.ac.rs, marina.nenic@uns.ac.rs, ivan@uns.ac.rs

Abstract - This paper presents a tool easesthe process of configuring and developing HyperLedger Fabric Blockchain networks. The tool is intended tohelp the developers reduce the time needed for setup and focus more on the business logic implementation. The solution would providea domain specific language (DSL) which would lean on the code generator that produces configuration files for the Fabric network, as well as functions for interacting with the chaincode (smart contract). In the paper we present abstract syntax for the DSL as well as concrete textual syntax. All the provided concepts are specific for the Hyperledger Fabric network. Plans for future work include developing concrete graphical syntax and defining bigger set of concepts for modeling Fabric network.

I. INTRODUCTION

Enterprises are becoming more and more interested in the introduction of private blockchains (PBC) into their existing IT infrastructure since their application ensures transparency, traceability, and trust between parties involved into business processes without the need for a central authority. In order to achieve the three aforementioned core values, specialized infrastructure services, known as private blockchain business network (BN) services, need to be developed before actual smart contracts, which implement the business rules, can be developed and deployed. The need for a specific infrastructure and the distributed nature of BN itself are the main obstacles keeping PBC from the production readiness.

Contemporary cloud-based infrastructure has reached the maturity level which is sufficient to support the PBC BN specific requirements. Thus, some of the major Cloud Vendors (CV) offer a Blockchain as a Service (BCaaS) as a part of their standard service offering. However, BCaaS have introduced several challenges to the BN modeling and development. First of all, the usage of BCaaS implies that each organization involved into BN needs to setup its part of BN separately, using the CV-specific web interface. This implies that there is no well-defined BN modeling technique which can be applied to the process of BN establishment, independently from the chosen CV. Accordingly, there is no single place for keeping the whole BN specification in order to be easily accessible for in-place changes or reviews. Therefore, if there is a need to perform a change on the particular part of BN, it must be done by each party separately, usually resulting in building a completely new BN. Thus, it is hard for engineers to get a complete overview of the BN infrastructure, its participants, and their relationships, since there is an absence of the concise BN specification. Furthermore, blockchain cloud services are by a degree of magnitude more expensive than any other cloud service, so average users and startups usually use BCaaS just for production purposes. Consequently, predefined BNs are typically usedfor testing and development purposes, rather than custom BNs. A major problem with this approach is that predefined BN infrastructure and participant's setup does not match the given use-case and is used solely to significantly reduce development costs.

On the other hand, the microservice software architecture design principles and containerization techniques have introduced a new approach to the specification and development of distributed software systems. The use of various container orchestration tools, such as Kubernetes, has introduced a new approach to the custom BN infrastructure development, eliminating a need for using a BCaaS. The Kubernetes BN artifacts are always specified in the same format, such as YAML, and can be disposed to the cloud execution platform, using the standard command-line interface (CLI) commands. In this way, software engineers are able to develop the BN infrastructure which is more suitable for their use case, using a well-defined set of steps. However, the process of custom BN specification and development is not a trivial task, since it requires knowledge of various techniques and technologies which are not necessarily related to the PBC development domain. For example, in order to develop a basic custom BN, engineers should be familiar with the software containerization techniques and container managers, such as Docker or Kubernetes. Also, various configuration parameters, which differ depending on the BN participant role, need to be specified and configured correctly. Further, engineers should be familiar with the cryptographic techniques for issuing, signing, and revoking certificates which specify identify and roles for individual BN participants and users. Therefore, the process of BN network establishment is time consuming and requires repetitive and boilerplate program code constructs to be written in different layers of the BN specification. This can lead to another problem, since, while trying to reduce development time and effort by coping and pasting similar code constructs, engineers often unintentionally introduce mistakes through omitting

specific configuration parameter values. These mistakes usually do not cause runtime errors, but the BN inconsistent behavior which is hard to understand while debugging. Accordingly, separate teams of blockchain engineers are usually dedicated to the custom BN specification and development.

Besides the aforementioned challenges, the BN engineering teams are also facing the BN complexity in early development phases. The BN complexity is mainly caused by the large number of different PBC participants and their numerous relationships. Therefore, engineers are typically trying to reduce the BN complexity by specifying various BN models. These models depict individual BN participants and their relationships, and, in the end, they are used just for documentation purposes. On the other hand, Model-Driven Software Engineering (MDSE) practitioners argue in favor of models as a formal way to describe the entities from the specific domain and use of such specifications as primary artifacts in the development process. Therefore, it could be beneficial for blockchain engineers to have a DSL which can be used to specify the BN infrastructure and its participants and to use such a BN specification for generating executable BN program code. Generated program code should implement the BN infrastructure services which can be then disposed the cloud execution platform and used by parties involved in the BN.

The paper is organized as follows. Research questions are stated in section III. In section IV we describe the tools that were used to implement the solution presented in this paper. Implementation of the software solution is described in V section. The final section of the paper offers main conclusions and offers some directions for further work.

II. RESEARCH QUESTIONS

In this research, we aim to resolve challenges related to custom BN modeling, development, and disposition to cloud execution platforms. Accordingly, the first goal of this research is to introduce a model-driven approach in order to address: (i) the BN modeling challenges, by providing a DSL as a formal modeling technique; and (ii) the development and disposition challenges, by using the BN DSL specification for generation of all the required BN code constructs, so it can be deployed to the target cloud execution environment. By doing this, we first want to increase the abstraction level of custom BN specification process, in order to enable various blockchain domain experts to develop their BNs even they are not familiar with concrete PBC development frameworks. In this way, we also want to eliminate potential errors caused by the need for specification of repetitive configuration parameters and boilerplate code constructs. Second, our goal is to ease the process of custom BN specification and development in practice, make it less dependent from the specific CVs or development teams and therefore less time-consuming.

Third, we want to decrease the development and testing costs in order to enable various individuals and organizations to dispose their custom BNs to cloud execution platforms without the need for the expensive BCaaS services. We also consider this as a big step towards expanding the PBC development community, while at the same time enabling great ideas and projects to reach the production readiness more quickly.

The second goal of this research is to develop a software tool, so that the proposed model-driven approach can be utilized in real-world projects.

While surveying the state-of-the-art literature in this area, we have found several papers utilizing MDSE in specification and development of PBC-related artifacts. In [1], authors have introduced a MSDE-based approach for specification and generation of smart contracts for asset management. They have also developed a software tool called Lorikeet. Lorikeet can automatically create well-tested smart contract code from specifications that are encoded in the business process and data registry models based on the implemented model transformations. Authors of [2] present an initial approach for generating smart contracts for coordinating the usage of cyber-physical system elements from UML state charts. While the current target platform is Ethereum, the proposed approach can easily be extended to other blockchain platforms. In comparison to our approach, the aforementioned research papers are focused on the MDSE utilization in specification and generation of smart contracts, rather than the BN infrastructure services.

Finally, in [3] authors propose a model-driven approach, combining an ontology and a layer model, that is capable of capturing the properties of existing blockchain-driven business networks. Therefore, this approach is aimed to be applied on the existing blockchain BNs in order to describe and understand existing BN infrastructure, rather than specify and develop a new one.

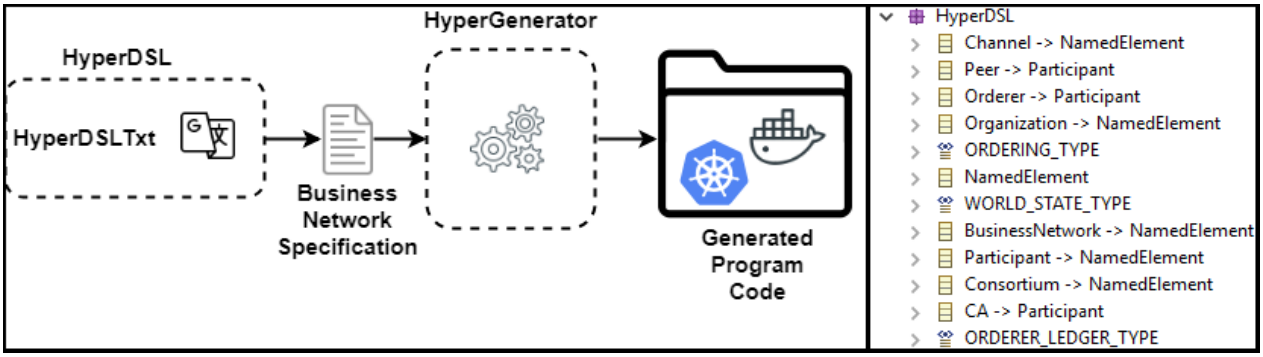

Figure 1 – Solution modules

III. METHODOLOGY

Our research is composed of the following methodological steps:

(i) develop the model-driven approach and (ii) develop HyperBuilder: a software tool which enables utilization of the developed model-driven approach. The development of the proposed model-driven approach includes the following steps: (i) develop the abstract syntax of the HyperDSL language in the form of a meta-model that conforms to the Ecore meta-meta- model; (ii) develop the concrete textual syntax of HyperDSL as a visual representation of the modeling concepts specified in the HyperDSL meta-model and (iii) develop a set of M2T transformations in order to generate executable programming code based on the HyperDSL BN specification.

For the development of HyperBuilder, we have used the Eclipse Modeling Framework (EMF). HyperDSL concrete syntax has been developed using the Xtext language, while individual M2T transformations were developed using the Xtend language.

IV. IMPLEMENTATION

HyperBuilder has two main modules and several submodules. Those two modules are called HyperDSL and HyperGenerator. They are presented in the left side of Figure 1.

The HyperDSL module enables use of the concrete textual syntax of HyperDSL

The HyperGenerator module contains several code generators, which use the HyperDSL BN specification as their input and generate executable Kubernetes program code as output. The generated program code implements the BN infrastructure services and can be deployed to a cloud execution platform. On the right side of Figure 1, the HyperBuilder meta-model is presented together with the description of several HyperDSL concepts (with the corresponding meta-model classes and attributes written in italics inside the parentheses). The main concept is the private blockchain BN (*BusinessNetwork*) which is composed of a set of consortiums (*Consortium*). Each consortium is described by name (*Name*) and domain (*Domain*). Each participant is additionally described by hostname (*Hostname*), its membership service provider name (*MSPName*) and port number (*Port*). Consortiums contain a set of organizations (*Organization*) which are exchanging transactions through channels (*Channel*), crating ledger data as a result of transactions. Transaction data are stored on the organization's peer (*Peer*). Each organization can additionally establish its certificate authority (*CA*) to generate certificates and key material in order to configure and manage identity in the BN. Each BN must have at least one ordering service defined. The BN ordering service (*Orderer*) provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. Currently, two types of ordering services are supported: (i) Solo, which is more suitable for development purposes since it is not fault-tolerant and (ii) Kafka, the production-ready crash fault-tolerant ordering service which uses the Apache Kafka message provider.

In the Figure 2, concrete network definition is given. The concrete textual syntax is JSON-like, since most of developers are familiar with JSON.

Based on this definition, an instance of a meta-model is being created. Templates for code generations are created using Xtend language. Based on the instance

of meta-model, templates are being populated. The final output of this process are generated files shown in Figure 3.

Generated files include those that are necessary for generating crypto materials for network members, as well as the configuration for channel creations. Besides this, tool is also generating files that allow creating containers for the Network elements (Peers, Orderers, etc.).

Also, there are .sh files that are used purely for network creation and its deletion.

```
BN<bn1>{
    domain:' example.com'
    worldStateType:LEVELDB
    network:[
        Peer<org1peer1>{
            domain:'org1.com'
            isAnchor:true
            hostName:'peer1'
            port: 30110
        },
        Peer<org2peer1>{
            domain:'org2.com'
            isAnchor:true
            hostName:'peer2'
            port: 30120
        },
        CA<org1ca1>{
            domain:'org1.com'
            hostName:'ca'
            port: 30054
        },
        CA<org2ca1>{
            domain:'org2.com'
            hostName:'ca'
            port: 30055
        },
        Orderer<orderer>{
            domain:'orderer.com'
            hostName:'bn'
            port: 31010
            type:SOLO
            ledgerType:FILE
        }
    ]
```

```
participants:[
    Organization<org1>{
        domain:'org1.com'
        MSPName:'org1MSP'
        networkServices:[
            org1peer1,org1ca1
        ]
    },
    Organization<org2>{
        domain:'org2.com'
        MSPName: 'org2MSP'
        networkServices:[
            org2peer1,org2ca1
        ]
    },
    Channel<ch1>{
        cOrganizations:[
            org1,org2
        ]
        cPeers:[
            org1peer1,org2peer1
        ]
    },
    Consortium<con1>{
        domain: 'con1.example.com'
        organizations:[
            org1, org2
        ]
        channels:[
            ch1
        ]
    }
]
```

Figure 2 – Network definition example

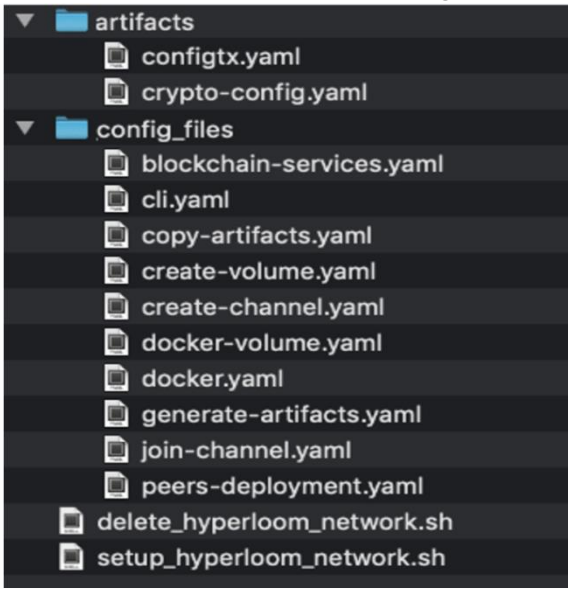

Figure 3 – Generated files

V. CONCLUSION

In this paper we present a solution that would help developers in creating Fabric Blockchain network, without the need to have a deep understanding of Fabric network configuration files. This solution would significantly reduce the time needed for the setup, and let developers focus more on writing the actual smart contracts. Also, it would save a lot of money, since it eliminates the need for expensive blockchain cloud services in the development and testing phases. The tool is able to generate all the configuration files needed for the Fabric blockchain network to be hosted and run on the Kubernetes cluster.

It uses Ecore meta-modeling framework, Xtext for specifying concrete textual syntax and Xtend for code generation.

This solution is generating artifacts for Hyperledger Fabric version 1, but with small efforts it can be modified so that it generates artifacts for the version 2.

Plans for future work include developing more meta-modeling concepts, as well as making existing more detailed. Also, the plan is to develop graphic syntax for the DSL, since it would be easier to use graphical syntax,

because most of Fabric networks have many relations between its elements.

ACKNOWLEDGMENT

Research presented in this paper was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models

REFERENCES

[1] Tran, A., Lu, Q. and Weber, I., Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management. In *Proc. BPM 2018*, Sydney, Australia, September 9 – 14, 2018, (pp. 56-60), 2018.

[2] Garamvölgyi, P., Kocsis, I., Gehl, B. and Klenik, A., 2018, June. Towards Model-Driven Engineering of Smart Contracts for Cyber-Physical Systems. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)* (pp. 134-139), IEEE Press, 2018.

[3] Seebacher, S. and Maleshkova, M., 2018, January. A Model-driven Approach for the Description of Blockchain Business Networks. In *Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS-11*), January 3 – 6,