INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2020, VOL. 66, NO. 4, PP. 619–625
Manuscript received February 20, 2020; revised October, 2020. DOI: 10.24425/ijet.2020.134020

Modeling of Distributed Ledger Deployment View

Tomasz Górski, and Jakub Bednarski

Abstract—**The Distributed Ledger Technology (DLT) is a peer-to-peer model of sharing data among collaborating parties in a decentralized manner. An example of DLT is a blockchain where data form blocks in an append-only chain. Software architecture description usually comprises multiple views. The paper concentrates on the Deployment view of the DLT solution within the 1+5 architectural views model. The authors have proposed Unified Modeling Language (UML) extensibility mechanisms to describe the needed additional semantic notation to model deployment details. The paper covers both the network and node levels. The proposed stereotypes and tagged values have enriched *UML Deployment diagram*. We have gathered those modeling elements in dedicated *UML Profile for Distributed Ledger Deployment*. We have applied the profile to model *Deployment view* of a renewable energy management system that uses R3 Corda framework. The system records information about inbound and outbound energy to/from renewable energy grid.**

Keywords—**Software architecture, Deployment view, Architectural views model 1+5, Unified Modeling Language extensibility mechanisms, Distributed Ledger Technology, Blockchain**

I. Introduction

DISTRIBUTED Ledger Technology (DLT) has emerged as one of the most disruptive technologies in the last decade. It promises to change the way people do their business, track their products, and manage their personal data.

A distributed ledger, defined by Xu et al. [20], is an append-only set of transactions distributed across many machines. An append-only means that new transactions can be added but existing ones can not be changed or deleted. A blockchain is a distributed ledger that is structured into a linked list of blocks. Each block has an ordered set of transactions. A block is linked to its predecessor with using a cryptographic hash to secure the whole chain. Many scientists have paid attention to blockchain technology because it has potential benefits for many industries. In other words, blockchain technology finds its applications in many sectors. Al-Jaroodi et al. [1] explore the advantages and challenges of incorporating blockchain in different industrial applications. As far as smart cities are concerned, Shen et al. [17] have observed many uses. That technology is a natural choice when designing supply chain solutions. Gonczol et al. [4] present the current state of research and summarize the benefits and challenges of the distributed organization and the management of supply chains. The healthcare sector can benefit from blockchain technology because of privacy and decentralization. For example, Shahnaz et al. [16] present a framework that could be used for implementing blockchain technology for the Electronic Health Record system. The broader view on blockchain in healthcare present Metcalf et al. [12]. They show topics from over 50 authors, presenting the technical side of the technology and its practical applications around the globe. For example, Xia et al. [19] propose a blockchain-based system that addresses the problem of medical data sharing in a trust-less environment. Furthermore, Kaijun et al. [10] propose a public blockchain of agricultural supply chain system. Moreover, Wang et al. [18] describe an electronic large-scale voting scheme based on blockchain. Furthermore, we believe that blockchain has enormous potential in military sector. Górski et al. [9] present a solution which persists battleship position co-ordinates in cloud-based blockchain.

We have paid special attention to the energy sector. Next-generation grid demands technologies that enable the integration of distributed energy resources and consumers that both buy and sell electricity. Wang et al. [18] developed an optimization model and blockchain-based architecture to manage the operation of crowdsourced energy systems, with peer-to-peer energy trading transactions. They have built a solution on IBM Hyperledger Fabric. The authors of the paper have proposed the software architecture of the Electricity Consumption and Supply Management (ECSM) system, designed on R3 Corda platform, [8].

We can look at software architecture as the highest level breakdown of a software system into its parts. We think of architecture as a structure. Software architecture comprises software elements, relations among them, and properties of both elements and relations. But, we can look at software system at different angles. In other words, we have different software architectural views. Software architecture comprises models which represent different architectural views, e.g.: *Use case*, *Logical*, *Deployment*. Unified Modeling Language is the most commonly used graphical notation. UML helps in describing and designing software systems. It is especially useful for modeling systems designed under the object-oriented approach, [3], [25]. In order to model distributed ledger solutions, we propose extensions of UML modeling notation. That enrichment of UML elements applies to *Deployment* architectural view of blockchain solution.

We structure the paper as follows. The second section locates the main area of interest of the paper within 1+5 architectural views model. The next section describes R3 Corda framework. The fourth section contains description of UML extensibility mechanisms proposed in *UML Profile for Distributed Ledger Deployment* whereas the example of *Deployment view* design of Electricity Consumption and Supply Management system is presented in the sixth section. The

T. Górski is with Department of Computer Science, Polish Naval Academy, Gdynia, Poland (e-mail: t.gorski@amw.gdynia.pl).

J. Bednarski is with Department of Computer Science, Polish Naval Academy, Gdynia, Poland (e-mail: j.bednarski@amw.gdynia.pl).

following section presents an example of Gradle Domain Specific Language (DSL) script for the ECSM system, which can be used to deploy nodes of a distributed ledger network. The last one concludes the paper and outlines the direction of further work.

II. Deployment view

A variety of models exists, with differing sets of architectural views, such as, e.g.: 4+1, RM-ODP, Siemens, SEI, [15]. Yet, those models do not allow for a complete description of the blockchain solution architecture. The Architectural views model 1+5 for integration solutions was designed to model collaborating systems in the context of business processes, [5]. As far as DLT and blockchain solutions are concerned the 1+5 model fits perfectly, [6]. In that kind of solution, we have collaborating parties (e.g.: seller and buyer) based on rules defined in a smart contract. We can include aspects of collaborating parties through smart contracts in the 1+5 model in *Contracts View* and *Integrated services view*, [7]. The paper concentrates on *Deployment view* of the model (see Fig. 1).

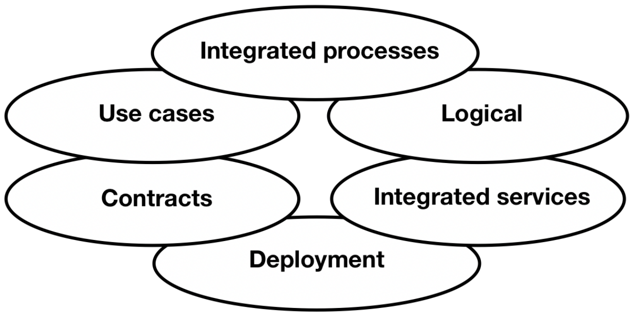


Fig. 1. Architectural views model 1+5.

The deployment view defines the physical environment in which the solution is intended to run, including: hardware environment (e.g., processing nodes, network interconnections), technical environment requirements for each node (or node type), the mapping of software elements to the runtime environment that will execute them. As part of deployment view we propose to model two aspects: overall network view and a single node configuration.

III. R3 Corda Framework

Many Distributed Ledger Technology platforms are available on the market. A significant number of comparative analyses concern that kind of framework. In one of the recent analyzes, Chowdhury et al. [2] evaluate the feasibility of the most well-established, both private and public DLT platforms, i.e.: Bitcoin, Ethereum, Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Burrow, EOS, Multichain, R3 Corda, Cardano, IOTA and Walton-Chain. They have selected a broad range of quantitative evaluation criteria, e.g.: block size, block creation time, cost, energy consumption, consensus. Because of major limitations of public blockchain platforms, we have concentrated on private ones. We have found that R3 Corda, a private DLT framework, consumes almost negligible energy. The second advantage of that framework is scalability. Corda is among there frameworks with the shortest *block creation time* (0.5 second). In addition, the block size is configurable.

The financial industry imposed requirements on Corda's design, but field experience has shown that Corda has broad applicability, well beyond banking. That is why we have used software engineering techniques to support the design of solutions using R3 Corda framework. A Corda network is an authenticated peer-to-peer network of nodes where each node is a Java Virtual Machine run-time environment hosting Corda services and executing applications known as CorDapps. It is a permissioned network that makes up the condition that all participants must have verifiable identities using public-key infrastructure. In Corda network, we can distinguish the following node types: Network Map, Notary, Oracle and DLTNode. A Corda network is a fully connected graph. The graph edges represent the potential to communicate, not persistent connections (see Fig. 2).

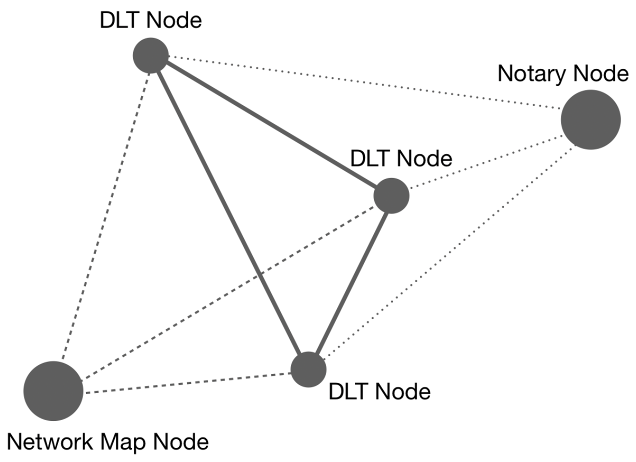


Fig. 2. Example of Corda network graph of connections.

Network Map Node publishes a list of peers. The framework uses Advanced Message Queuing Protocol (AMQP) over Transport Layer Security (TLS) between nodes which is implemented using Apache Artemis, an embeddable message queue broker. Corda transactions operate using consumable states analogous to the latest entry of a blockchain ledger that can validate a new transaction. The consensus in Corda is reached at transaction level by involving relevant parties only. The framework uses services to facilitate communication among nodes. The following services are the integral parts of the framework: permissioning, network map, notary, oracle, identity and support. Using nodes and services, we can make up Corda network.

IV. Stereotypes

The authors have proposed modeling constructions to represent parameters of nodes configuration which can be modeled in UML Deployment diagram. All proposed stereotypes and tagged values, that describe the needed additional semantic structures, have been included in *UML Profile for Distributed Ledger Deployment* and modeled in Visual Paradigm Enterprise. We have placed the designed profile in the GitHub repository, [22].

In the Unified Modeling Language there are three types of mechanisms that allow on extending the language. A stereotype is one of them. Stereotypes allow designers to create new model elements and in that way extend the vocabulary of UML. Stereotypes provide additional, specific semantic meaning to existing elements.

A Corda network comprises nodes that communicate using protocols to create and validate transactions. Nodes host and run services. So, we have identified the following UML stereotypes corresponding to services:

- ≪permissioningService≫ — applied to the service used to provision TLS certificates,
- ≪networkMapService≫ — applied to the service which enforces rules regarding the information that nodes must provide and Know Your Customer (KYC) processes that they must complete before being admitted to the network,
- ≪notaryService≫ — applied to the service which is used to provide transaction ordering and time stamping,
- ≪oracleService≫ — applied to the service which signs transactions if they state a fact and that fact is considered to be true.
- ≪identityService≫ — controls admissions of participants into Corda Network. The service receives certificate signing requests from prospective network participant and reviews the information submitted. That participant can register itself with a signed participation certificate in the network map service.
- ≪supportService≫ — manages and resolves inquiries and incidents relating to the identity, doorman and notary services.

The figure (see Fig. 3) shows stereotypes declared for services in UML Profile diagram.

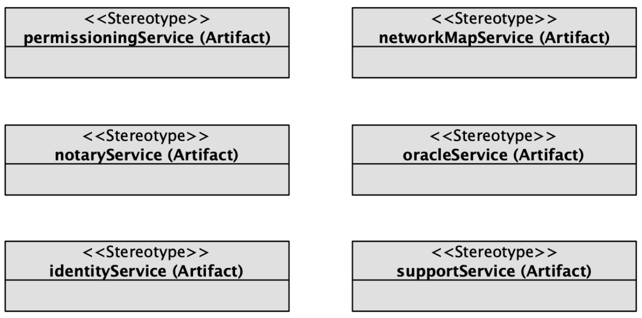


Fig. 3. UML Profile diagram with stereotypes for DLT services.

Services run on specific nodes. So, we have identified the following stereotypes for nodes:

- ≪NetworkMapNode≫ — applied to the node which runs the network map,
- ≪NotaryNode≫ — applied to nodes running a notary service,
- ≪OracleNode≫ — applied to nodes that link the ledger to the outside world by providing facts that affect the validity of transactions,
- ≪DLTNode≫ — applied to nodes have a vault and may communicate with other nodes, notaries and oracles and evolve their private ledger.
- ≪CordaNode≫ — abstract node which gathers common properties for all types of nodes in Corda network.

Nodes communicate with each other using communication protocols.

We have added the following stereotypes for connection links, to mark communication protocols:

- ≪HTTPS≫ — Hypertext Transfer Protocol Secure (HTTPS) extends the Hypertext Transfer Protocol encrypted using Transport Layer Security.
- ≪AMQP/TLS≫ — Advanced Message Queuing Protocol is used by nodes to send encrypted messages in peer-to-peer communication with Transport Layer Security.

The figure (see Fig. 4) shows stereotypes declared for communication protocols in UML Profile diagram.

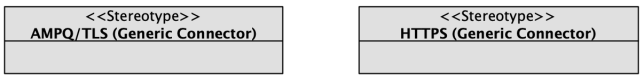


Fig. 4. The profile stereotypes for communication protocols.

The table (see Table I) contains the summary of proposed stereotypes, with extended elements, in the *UML Profile for Distributed Ledger Deployment*.

TABLE I
THE SUMMARY OF STEREOTYPES IN THE PROFILE.

Stereotype	Extended UML element
≪CordaNode≫	Node
≪DLTNode≫	Node
≪OracleNode≫	Node
≪NotaryNode≫	Node
≪NetworkMapNode≫	Node
≪permissioningService≫	Artifact
≪networkMapService≫	Artifact
≪notaryService≫	Artifact
≪oracleService≫	Artifact
≪identityService≫	Artifact
≪supportService≫	Artifact
≪HTTPS≫	Generic Connection
≪AMQP/TLS≫	Generic Connection

V. TAGGED VALUES

We can set deployment parameters up for a single node configuration. Each parameter has its name and a value. We have used tagged values to model deployment parameters. A tagged value is a tag value pair that can add properties to model elements in UML. Defined tagged values describe the parameters of distributed ledger node's deployment configuration. Starting from UML 2.0, we consider tagged values as attributes of stereotype. The full list contains 52 tagged values which represent configuration parameters of Corda nodes. Subset of tagged values available for all types of nodes we have placed in ≪CordaNode≫ stereotype. Complete set of configuration fields for nodes of R3 Corda v.4.3 can be found at the enclosed link, [21].

Deployment configuration of each node comprises required and additional sets of parameters. We have declared tagged values for all required parameters of deployment configuration and placed them in ≪CordaNode≫ stereotype:

- myLegalName — the legal identity of the node,
- devMode — the flag sets the node to run in development mode,
- p2pAddress — host and port on which the node is available for protocol operations over ArtemisMQ,
- rpcSettings.address — host and port for the RPC server binding,
- rpcSettings.adminAddress — host and port for the RPC admin binding,
- rpcSettings.standAloneBroker — indicates whether the node will connect to a standalone broker for RPC.
- rpcSettings.useSsl — indicates whether or not the node should require clients to use Secure Sockets Layer (SSL) for Remote Procedure Call (RPC) connections,
- rpcSettings.ssl.keyStorePath — absolute path to the key store containing the RPC SSL certificate,
- rpcSettings.ssl.keyStorePassword — password for the key store,
- rpcUsers — a list of users who are authorised to access the RPC system, e.g., [[user: ”thomas”, ”password”: ”password”, ”permissions”: [”ALL”]]],
- configFile — a file with complete list of configuration parameters for the node, e.g., ./build/nodes/thomas/Thomas windmill.conf.

Moreover, we have proposed tagged values for additional deployment parameters but common for all types of nodes. We have attached them also to the ≪CordaNode≫ stereotype, but we present only the subset of that tagged values:

- attachmentCacheBound — optionally specifies how many attachments should be cached locally,
- detectPublicIp — flag toggles the auto IP detection behaviour,
- flowMonitorPeriodMillis — duration of the period suspended flows waiting for IO are logged,
- flowTimeout.timeout — the initial flow timeout period,
- flowTimeout.maxRestartCount — the number of retries the back-off time keeps growing for. For subsequent retries, the timeout value will remain constant,
- h2Settings — sets the H2 JDBC server host and port,
- jmxReporterType — provides an option for registering an alternative JMX reporter,
- keyStorePassword — the password to unlock the KeyStore file containing the node certificate and private key,
- messagingServerAddress — address of the ArtemisMQ broker instance,
- networkServices.doormanURL — root address of the network registration service,
- trustStorePassword — the password to unlock the Trust store file containing the Corda network root certificate.

The table (see Table II) contains declarations, with data type and default value, of selected tagged values common for all types of distributed ledger nodes.

TABLE II
SELECTED TAGGED VALUES FOR ≪CORDANODE≫ STEREOTYPE.

Tagged value name	Type	Default value
attachmentCacheBound	Integer	1024
detectPublicIp	Boolean	false
flowMonitorPeriodMillis	Integer	60
flowTimeout.timeout	Integer	30
flowTimeout.maxRestartCount	Integer	6
h2Settings	Text	NULL
jmxReporterType	Text	JOLOKIA
keyStorePassword	Text	cordacadevpass
messagingServerAddress	Text	Not defined
myLegalName	Text	Not defined
networkServices.doormanURL	Text	Not defined
p2pAddress	Text	Not defined
rpcSettings.useSsl	Boolean	false
trustStorePassword	Text	trustpass

Furthermore, we have identified the following deployment parameters for distributed ledger notary node type (≪NotaryNode≫), [21]:

- notary.validating — determines if notary is validating or non-validating one,
- notary.serviceLegalName — if the node is part of a distributed cluster, the parameter specifies the legal name of the cluster.
- notary.raft.nodeAddress — host and port to which to bind the embedded Raft server,
- notary.raft.clusterAddresses — must list the addresses of all the members in the cluster,
- notary.bftSMaRt.replicaId — index of the current replica. All replicas must specify a unique replica id,
- notary.dftSMaRt.clusterAddresses — must list the addresses of all the members in the cluster.

The table (see Table III) contains selected tagged values that we have identified for ≪NotaryNode≫ node.

TABLE III
TAGGED VALUES FOR ≪NOTARYNODE≫ STEREOTYPE.

Tagged value name	Type	Default value
notary.validating	Boolean	false
notary.serviceLegalName	Text	Not defined
notary.raft.nodeAddress	Text	Not defined
notary.raft.clusterAddresses	Text	Not defined
notary.bftSMaRt.replicaId	Text	Not defined
notary.dftSMaRt.clusterAddresses	Text	Not defined

During declaring tagged values, we have intentionally omitted the deployment parameters of R3 Coda 4.3 version that are marked as deprecated, internal options, or unsupported configuration.

Finally, we have applied tagged values to previously declared stereotypes for distributed ledger nodes. Thanks to generalization relationship, from the object-oriented approach, all defined stereotypes inherit common tagged values from ≪CordaNode≫ stereotype.

The following UML Profile diagram shows inheritance hierarchy of nodes' stereotypes with tagged values (see Fig. 5).

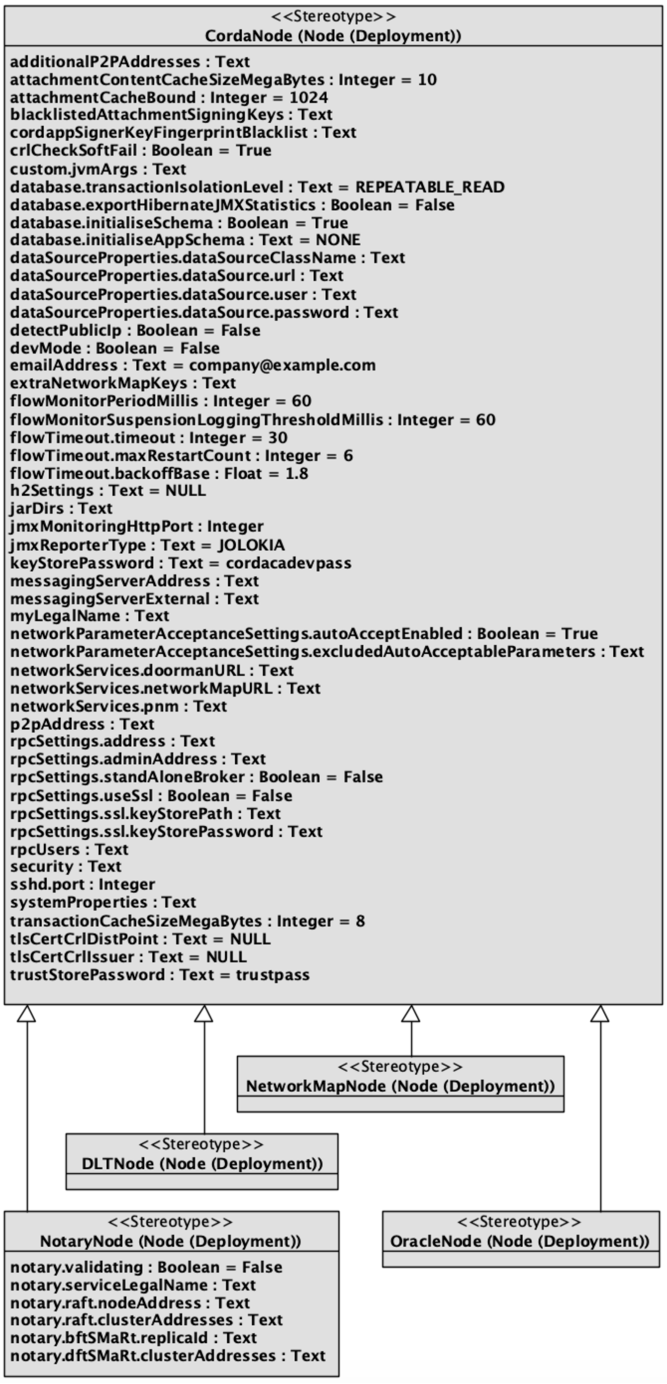

<<Stereotype>>
CordaNode (Node (Deployment))
additionalP2PAddresses : Text
attachmentContentCacheSizeMegaBytes : Integer = 10
attachmentCacheBound : Integer = 1024
blacklistedAttachmentSigningKeys : Text
cordappSignerKeyFingerprintBlacklist : Text
crlCheckSoftFail : Boolean = True
custom.jvmArgs : Text
database.transactionIsolationLevel : Text = REPEATABLE_READ
database.exportHibernateJMXStatistics : Boolean = False
database.initialiseSchema : Boolean = True
database.initialiseAppSchema : Text = NONE
dataSourceProperties.dataSourceClassName : Text
dataSourceProperties.dataSource.url : Text
dataSourceProperties.dataSource.user : Text
dataSourceProperties.dataSource.password : Text
detectPublicIp : Boolean = False
devMode : Boolean = False
emailAddress : Text = company@example.com
extraNetworkMapKeys : Text
flowMonitorPeriodMillis : Integer = 60
flowMonitorSuspensionLoggingThresholdMillis : Integer = 60
flowTimeout.timeout : Integer = 30
flowTimeout.maxRestartCount : Integer = 6
flowTimeout.backoffBase : Float = 1.8
h2Settings : Text = NULL
jarDirs : Text
jmxMonitoringHttpPort : Integer
jmxReporterType : Text = JOLOKIA
keyStorePassword : Text = cordacadevpass
messagingServerAddress : Text
messagingServerExternal : Text
myLegalName : Text
networkParameterAcceptanceSettings.autoAcceptEnabled : Boolean = True
networkParameterAcceptanceSettings.excludedAutoAcceptableParameters : Text
networkServices.doormanURL : Text
networkServices.networkMapURL : Text
networkServices.pnm : Text
p2pAddress : Text
rpcSettings.address : Text
rpcSettings.adminAddress : Text
rpcSettings.standAloneBroker : Boolean = False
rpcSettings.useSsl : Boolean = False
rpcSettings.ssl.keyStorePath : Text
rpcSettings.ssl.keyStorePassword : Text
rpcUsers : Text
security : Text
sshd.port : Integer
systemProperties : Text
transactionCacheSizeMegaBytes : Integer = 8
tlsCertCrlDistPoint : Text = NULL
tlsCertCrlIssuer : Text = NULL
trustStorePassword : Text = trustpass
<<Stereotype>>
NetworkMapNode (Node (Deployment))
<<Stereotype>>
DLTNode (Node (Deployment))
<<Stereotype>>
NotaryNode (Node (Deployment))
notary.validating : Boolean = False
notary.serviceLegalName : Text
notary.raft.nodeAddress : Text
notary.raft.clusterAddresses : Text
notary.bftSMaRt.replicaId : Text
notary.dftSMaRt.clusterAddresses : Text
<<Stereotype>>
OracleNode (Node (Deployment))

Fig. 5. UML Profile diagram presents stereotypes for nodes with tagged values.

All proposed stereotypes and tagged values we have placed in *UML Profile for Distributed Ledger Deployment*.

VI. Deployment view of ECSM

We have presented the concept of the ECSM and its implementation during ICSEng 2018, Sydney, Australia, [8]. Electricity Consumption and Supply Management system provides the functionality to monitor and record continuously information about inbound and outbound energy to/from a node in renewable energy grid. Information about inbound/outbound energy is a part of a smart contract that is confirmed and stored in every node. We have three types of nodes in such a renewable energy grid: prosumer, energy stock exchange and power grid. Prosumer's node can sell energy to other nodes or to the grid. The energy stock exchange node has two main roles: confirms the energy price for each transaction, provides real time energy price. Except monitoring inbound and outbound energy, the solution will provide the possibility to manage in automatic and manual way when energy should be sent to the energy grid. This approach gives benefits for both sides. It allows to maximize profits by energy farms owners and send electricity to the grid, when price is above the profitability ratio which can be calculated automatically based on stored data. Moreover, it gives the possibility to request additional energy by electricity distribution companies during higher energy demands. This approach should make settlements between producers and electricity sellers easier, and should give real time view for both sides. Moreover, it should give the possibility to connect new energy production points to the existing grid and build distributed network of energy suppliers. Corda DLT has been applied to manage actions among nodes and store transactions of selling energy. Each element in the system is actually DLT node. Proof-of-Concept (PoC) of ECSM has been implemented in a Java language with use of IntelliJ IDEA framework, [27]. The source code of PoC is available at GitHub repository, [23].

Deployment view of ECSM was designed in Visual Paradigm Enterprise. We have used *UML Deployment diagram* to show *Deployment view* of ECSM system. The figure (see Fig. 6) shows UML Deployment diagram for configuration of Corda distributed ledger network for ECSM PoC.

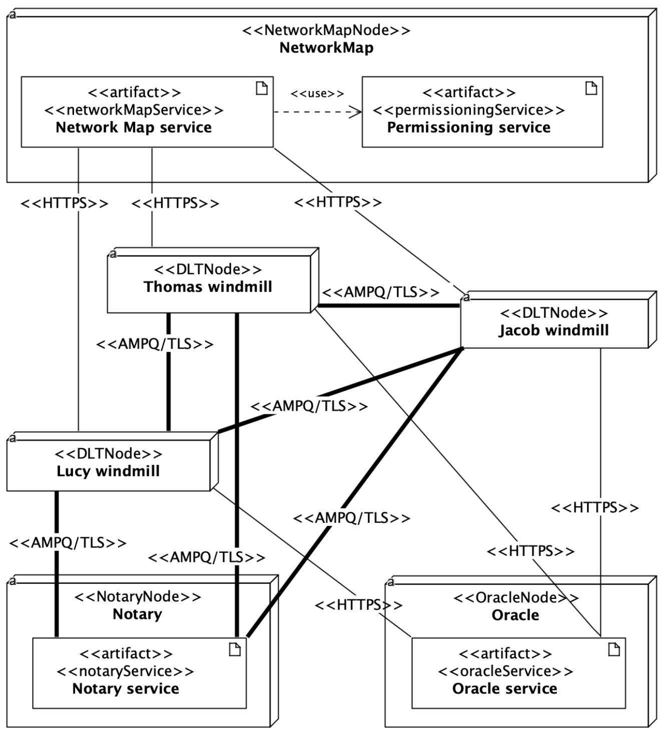
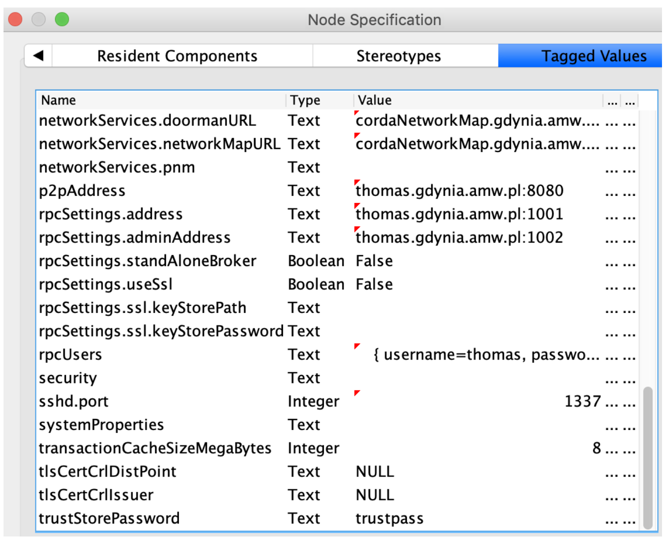


Fig. 6. UML Deployment diagram for part of ECSM.

We have identified all required nodes with matching stereotypes, e.g., ≪NetworkMapNode≫ node that will run network map service. We have modeled artifacts that represent services required by Corda platform. So, we have mark those artifacts with corresponding stereotypes, e.g., ≪oracleService≫. We have placed artifacts on suitable nodes, e.g., component with ≪notaryService≫ stereotype on node with ≪NotaryNode≫ stereotype. Moreover, we have identified DLT nodes that can exchange energy. We have marked those nodes with ≪DLTNode≫ stereotype. All DLT nodes have been connected in peer-to-peer manner. Connections among DLT nodes have been marked with ≪AMPQ/TLS≫ stereotype. Each of distributed ledger nodes (≪DLTNode≫) communicates over HTTPS with ≪NetworkMapNode≫ node and for that reason those connections have been marked with corresponding stereotype. Distributed ledger nodes communicates with oracle service over HTTPS and with notary service over AMPQ/TLS. So, ≪HTTPS≫ stereotype has been applied to connectors with oracle service and connectors with notary service have been enriched with ≪AMPQ/TLS≫ stereotype.

Furthermore, we can specify deployment configuration of each node. In order to do that, we have used tagged values available for each node type. Fist of all, we have specified deployment parameters for each of ≪DLTNode≫ nodes. Those nodes take part in transactions.

The figure (see Fig. 7) shows *Node Specification window* of Visual Paradigm with tagged values for ≪DLTNode≫ that are available in *UML Profile for Distributed Ledger Deployment*.



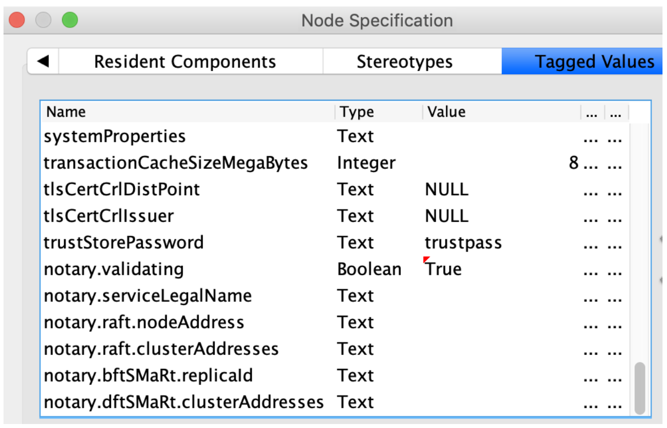
Name	Type	Value	...	...
networkServices.doormanURL	Text	cordaNetworkMap.gdynia.amw....	...	...
networkServices.networkMapURL	Text	cordaNetworkMap.gdynia.amw....	...	...
networkServices.pnm	Text		...	...
p2pAddress	Text	thomas.gdynia.amw.pl:8080	...	...
rpcSettings.address	Text	thomas.gdynia.amw.pl:1001	...	...
rpcSettings.adminAddress	Text	thomas.gdynia.amw.pl:1002	...	...
rpcSettings.standAloneBroker	Boolean	False	...	...
rpcSettings.useSsl	Boolean	False	...	...
rpcSettings.ssl.keyStorePath	Text		...	...
rpcSettings.ssl.keyStorePassword	Text		...	...
rpcUsers	Text	{ username=thomas, passwo...	...	...
security	Text		...	...
sshd.port	Integer	1337	...	...
systemProperties	Text		...	...
transactionCacheSizeMegaBytes	Integer	8	...	...
tlsCertCrlDistPoint	Text	NULL	...	...
tlsCertCrlIssuer	Text	NULL	...	...
trustStorePassword	Text	trustpass	...	...

Fig. 7. Selected tagged values available for DLT node deployment configuration.

We can set values for all tagged values which have been declared for ≪CordaNode≫. We can do that, thanks to the generalization relationship applied between ≪CordaNode≫ and ≪DLTNode≫ stereotypes in design of the profile. Default values are available for some deployment parameters in R3 Corda framework. The corresponding tagged values in the profile received the same default values, e.g.: attachmentCacheBound has default value 1024.

For ≪NotaryNode≫ node we have the same set of tagged values as for ≪DLTNode≫ node. But, in ≪NotaryNode≫ node there is an additional section of tagged values devoted for configuration of notary service. In our example we have one node with ≪NotaryNode≫ stereotype called *Notary*.

The figure shows *Node Specification window* of Visual Paradigm with tagged values for ≪NotaryNode≫ (see Fig. 8).



Name	Type	Value	...	...
systemProperties	Text		...	...
transactionCacheSizeMegaBytes	Integer	8	...	...
tlsCertCrlDistPoint	Text	NULL	...	...
tlsCertCrlIssuer	Text	NULL	...	...
trustStorePassword	Text	trustpass	...	...
notary.validating	Boolean	True	...	...
notary.serviceLegalName	Text		...	...
notary.raft.nodeAddress	Text		...	...
notary.raft.clusterAddresses	Text		...	...
notary.bftSMaRt.replicaId	Text		...	...
notary.dftSMaRt.clusterAddresses	Text		...	...

Fig. 8. A section of tagged values for notary node deployment configuration.

The design of ECSM is available at GitHub repository, [24].

Thanks to the proposed stereotypes and tagged values, we can model the Deployment view of Distributed Ledger solution. Furthermore, we can model it with the application of standard modeling notation, Unified Modeling Language, but extended with a dedicated profile. In software engineering, we usually have several environments that organize the development process, e.g.: development, tests, integration, preproduction, and production. Using UML packages we can model and manage different deployment environments, i.e.: dev, test, and prod.

VII. Deployment script for node configuration

Gradle is a general purpose build management system, which supports the automatic download and configuration of dependencies or other libraries. Gradle builds are described via one or multiple *build.gradle* files. At least one *build.gradle* file is typically located in the root folder of the project. Each of these files defines a project and its tasks. These build files are based on a Groovy Domain Specific Language.

Gradle uses the following files to configure and deploy distributed ledger network: *build.gradle*, *node.conf*, *reference.conf*.

The *build.gradle* file consists of three types of elements:

- task — a *Task* represents a single atomic piece of work for a build, such as compiling classes.
- block — a build script is made up of zero or more statements and script blocks. Statements can include method calls, property assignments, and local variable definitions.
- property — in the *node* block, we set values of properties based on corresponding values of tagged values.

We have used the *deployNodes* task to configure required deployment parameters for distributed ledger nodes from *UML*

Deployment model for ECSM PoC. That task is used by *build.gradle* to configure and deploy the DLT network. We have used the *node* block and *property* for node configuration.

Example of *deployNodes* task of the script for configuration of nodes in ECSM PoC we present in Listing 1.

Listing 1. Gradle script for configuration of distributed ledger nodes.

```
task deployNodes(type: net.corda.plugins.Cordform, dependsOn: ['jar']) {
  directory "./build/nodes"
  node {
      name "O=Notary,L=London,C=GB"
      notary = [validating: false]
      devMode false
      p2pAddress "notary.corda.amw.gdynia.pl:10002"
      rpcSettings {
        useSsl false
        standAloneBroker false
        address "notary.corda.amw.gdynia.pl:10003"
        adminAddress "notary.corda.amw.gdynia.pl::10103"
      }
      rpcUsers = [[ user: "notary", "password": "password", "permissions": ["ALL"]]]
      configFile = "./build/nodes/notary/notary.conf"
  }
  node {
      name "O=Thomas windmill,L=Gdynia,C=PL"
      devMode false
      p2pAddress "thomas.corda.amw.gdynia.pl:10002"
      rpcSettings {
        useSsl false
        standAloneBroker false
        address "thomas.corda.amw.gdynia.pl:10003"
        adminAddress "thomas.corda.amw.gdynia.pl::10103"
      }
      rpcUsers = [[ user: "thomas", "password": "password", "permissions": ["ALL"]]]
      configFile = "./build/nodes/thomas/thomas.conf"
  }
  node {
      name "O=Jacob windmill,L=Gdansk,C=PL"
      devMode false
      p2pAddress "jacob.corda.amw.gdynia.pl:10002"
      rpcSettings {
        useSsl false
        standAloneBroker false
        address "jacob.corda.amw.gdynia.pl:10003"
        adminAddress "jacob.corda.amw.gdynia.pl::10103"
      }
      rpcUsers = [[ user: "jacob", "password": "password", "permissions": ["ALL"]]]
      configFile = "./build/nodes/jacob/jacob.conf"
  }
  node {
      name "O=Lucy windmill,L=Plock,C=PL"
      devMode false
      p2pAddress "lucy.corda.amw.gdynia.pl:10002"
      rpcSettings {
        useSsl false
        standAloneBroker false
        address "lucy.corda.amw.gdynia.pl:10003"
        adminAddress "lucy.corda.amw.gdynia.pl::10103"
      }
      rpcUsers = [[ user: "lucy", "password": "password", "permissions": ["ALL"]]]
      configFile = "./build/nodes/lucy/lucy.conf"
  }
}
```

We can also use *node.conf* files to set values for all deployment parameters for each node.

VIII. Conclusion

Distributed Ledger Technology involves the application of peer-to-peer architecture to collaborating nodes. The paper concentrates on the Deployment view of the Architectural views model 1+5. We have proposed an UML profile for modeling the Deployment view of distributed ledger solution — *UML Profile for Distributed Ledger Deployment*. We have identified stereotypes for nodes, services and connectors. Those modeling elements allow for presenting the structure of a distributed ledger network. We took a step deeper in modeling and proposed the manner of modeling deployment configuration for a single node. We have used tagged values to achieve that aim. We have designed the profile in Visual Paradigm. We have applied the profile in designing ECSM for a renewable energy grid.

Further studies move into the direction of applying Model-Driven Development to design model-to-code transformation for generating deployment scripts for distributed ledger network configuration.

References

[1] J. Al-Jaroodi and N. Mohamed: "Blockchain in Industries: A Survey", *IEEE Access*, 7, 36500–36515 (2019)

[2] M.J.M. Chowdhury, M.S. Ferdous, K. Biswas, N. Chowdhury, A.S.M. Kayes, M. Alazab and P. Watters: "A Comparative Analysis of Distributed Ledger Technology Platforms", *IEEE Access*, 7, 167930–167943 (2019)

[3] M. Fowler: "UML Distilled. A brief guide to the standards Object oriented Language", Boston, USA: Addison-Wesley, (2005)

[4] P. Gonczol, P. Katsikouli, L. Herskind, N. Dragoni: "Blockchain Implementations and Use Cases for Supply Chains–A Survey", *IEEE Access*, 8, 11856–11871 (2020)

[5] T. Górski: "Architectural view model for an integration platform", *Journal of Theoretical and Applied Computer Science*, 6(1) 25–34 (2012)

[6] T. Górski: "Verification of Architectural Views Model 1+5 Applicability", in Extended abstracts book of the 17th International Conference on Computer Aided Systems Theory Las Palmas de Gran Canaria, Spain, 138–139 (2019)

[7] T. Górski and J. Bednarski: "Modeling of Smart Contracts in Blockchain Solution for Renewable Energy Grid", in Extended abstracts book of the 17th International Conference on Computer Aided Systems Theory Las Palmas de Gran Canaria, Spain, 140–141 (2019)

[8] T. Górski, J. Bednarski and Z. Chaczko: "Blockchain-based renewable energy exchange management system", in Proceedings of 26th International Conference on Systems Engineering, ICSEng 2018, Sydney, Australia (2018)

[9] T. Górski, K. Marzantowicz and M. Szulc: "Cloud-Enabled Warship's Position Monitoring with Blockchain", in Smart Innovations in Engineering and Technology, 1nd ed. vol. 1, Klempous, R. and Nikodem, J., Ed. Cham, Switzerland: Springer, 53–74 (2020)

[10] L. Kaijun, B. Ya, J. Linbo, F. Han-Chi and I. van Nieuwenhuyse: "Research on agricultural supply chain system with double chain architecture based on blockchain technology", *Future Generation Computer Systems*, 86 641–649 (2018)

[11] R.C. Martin: "Clean Architecture: A Craftsman's Guide to Software Structure and Design", Prentice Hall, (2017)

[12] D. Metcalf, J. Bass, M. Hooper, A. Cahana and V. Dhillon: "Blockchain in Healthcare: Innovations that Empower Patients, Connect Professionals and Improve Care", Boca Raton, USA: CRC Press, Taylor & Francis Group, (2019)

[13] D. Mohanty: "R3 Corda for Architects and Developers: With Case Studies in Finance, Insurance, Healthcare, Travel, Telecom, and Agriculture", Noida, Uttar Pradesh, India: Apress, (2019)

[14] T. Pender: "UML Bible", Wiley; 1st ed., (2003)

[15] N. Rozanski and E. Woods: "Software Systems Architecture. Working with Stakeholders using Viewpoints and Perspectives", Pearson India; 2nd ed., (2015)

[16] A. Shahnaz, U. Qamar and A. Khalid: "Using Blockchain for Electronic Health Records", *IEEE Access*, 7 147782–147795 (2019)

[17] C. Shen and F. Pena-Mora: "Blockchain for Cities — A Systematic Literature Review", *IEEE Access*, 6 76787–76819 (2018)

[18] S. Wang, A.F. Taha, J. Wang, K. Kvaternik and A. Hahn: "Energy Crowdsourcing and Peer-to-Peer Energy Trading in Blockchain-Enabled Smart Grids", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(8) 1612–1623 (2019)

[19] Q. Xia, E.B. Sifah, K.O. Asamoah, J. Gao, X. Du and M. Guizani: "MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain", *IEEE Access*, 5 14757–14767 (2017)

[20] X. Xu, I. Weber, M. Staples: "Architecture for Blockchain Applications:, Springer, (2019)

[21] docs.corda.net/corda-configuration-file.html#configuration-file-fields. Accessed, February, 5, 2020

[22] github.com/drGorski/UMLProfileForDLT. Updated, February, 19, 2020

[23] github.com/drGorski/renewableEnergyBlockchain. Updated, February, 8, 2019

[24] github.com/drGorski/designECSM. Updated, February, 5, 2020

[25] www.omg.org/spec/UML/2.5.1/. Accessed, February, 5, 2020

[26] www.corda.net/. Accessed, February, 5, 2020

[27] www.jetbrains.com/idea/. Accessed, February, 5, 2020

[28] www.visual-paradigm.com. Accessed, February, 5, 2020