Article

The 1+5 Architectural Views Model in Designing Blockchain and IT System Integration Solutions

Tomasz Górski

Department of Computer Science, Polish Naval Academy of the Heroes of Westerplatte (PNA), 81-127 Gdynia, Poland; t.gorski@amw.gdynia.pl

Abstract: Service fulfillment for clients increasingly involves cooperation between information technology (IT) systems. Designing such solutions requires an architectural approach that ensures symmetry between the communicating parties. For the design of such systems, the author introduces the 1+5 architectural views model. The model contains three new architectural views. For business process modeling, it ensures the *integrated processes* view. Integration aspects cover two additional views: *integrated services*, and *contracts*. Moreover, new stereotypes and tagged values have been added to the unified modeling language (UML). The author has introduced two profiles: *UML profile for integration flows*, and *UML profile for distributed ledger deployment*. Communication between systems requires flows that arrange mediation mechanisms. The paper describes an *integration flow diagram* that extends a UML activity diagram. In the case of blockchain, the author has proposed the *smart contract design pattern*. The paper describes three case studies that have employed the model to design various solutions. The 1+5 model has proven to be well suited for designing both centralized integration environments with enterprise service bus (ESB) and distributed blockchain solutions with peer-to-peer (P2P) connections.

Keywords: 1+5 architectural views model; unified modeling language; design pattern; interoperability; blockchain

check for updates

Citation: Górski, T. The 1+5 Architectural Views Model in Designing Blockchain and IT System Integration Solutions. *Symmetry* **2021**, *13*, 2000. https://doi.org/10.3390/sym13112000

Academic Editor: Chin-Ling Chen

Received: 13 September 2021
Accepted: 20 October 2021
Published: 22 October 2021

1. Introduction

When designing an IT system, we most often must consider the aspect of information exchange with other systems. Moreover, the requirements for the system stem from the business model. Business processes often cross organizational boundaries involving collaboration between companies.

The idea for the architectural model results from participation of the author in the project of building the check-in system for the Polish Border Guard. The key issue was placing the requirements in the right business context. The conducted business analysis identified key business processes and significant use cases (the number of use cases was reduced by more than six times). Communication between systems is another important area. To work properly, the check-in system had to communicate with several other systems, including the central register of issued and invalidated passport documents and the visa information system. The last key element is the non-functional requirements imposed on the implementation of use cases. This applies especially to aspects of performance, but also to safety, reliability, and usability. For example, when you cross the border of the European Union, you are obliged to go through passport control. Usually, it lasts only several dozen seconds. During this time, the check-in system communicates with several external systems, checking whether there are any contraindications for the entry or exit of the person to/from the European Union.

With the above in mind, the author has proposed the 1+5 model of architectural views for designing an IT system that requires cooperation with other systems [1,2]. The model has six views but three of them are new. It is the *integrated processes* view that enables the description of business processes that can cross the boundaries of the organization. The second is the *integrated services* view that allows you to show communication between systems and design integration flows. The last one is the *contracts* view that enables you to define cooperation rules and non-functional requirements.

The paper is arranged as follows. Section 2 discusses the related work. The third section describes the 1+5 architectural views model. Section 4 introduces UML profiles for modeling integration flows and blockchain network deployment. The next section encompasses a description of the *smart contract design pattern*. Section 6 shows three case studies where the 1+5 model has been used. The section also presents the application of the *integration flow diagram* to the design of the flow of prescription. The last section summarizes the article and outlines directions for further work.

2. Related Work

The article concentrates on the architectural description in UML and distributed systems in terms of blockchain and distributed ledger technologies. Therefore, the search for articles concentrated on these two topics. The results of the literature review have been divided into corresponding paragraphs. The first one contains papers that show the newest applications of UML. It also shows the research results of employing the model-driven engineering approach. The second one encompasses papers that describe the latest advances in blockchain technology. Special attention has been paid to the matter of blockchain solutions for the renewable energy sector.

Professionals usually reach for UML for software architecture modeling. A large group of practitioners (109) has been surveyed by Ozkaya et al. [3] to discover usage of UML diagrams and architectural viewpoints. It appears that the most popular are the information (99%) and the functional (96%) viewpoints. In the survey, they have also surveyed professionals on the usage of UML diagrams in modeling different aspects of software architecture. For example, the UML class diagram is commonly used for data structure modeling (85%), the UML deployment diagram is applied to physical structure modeling (71%) and functional to physical components mapping (53%). Moreover, the UML activity diagram is also used for software building and release processes modeling (20–22%). Table 1 contains the results of UML diagram usage, where the numbers mean the percentage of professionals who selected the considered diagram to model a chosen aspect of software architecture.

Table 1. Usage of UML diagrams by professionals.

UML Diagram	Software Architecture Aspect of Modeling
class	data structure (85%), functional structure (71%)
deployment	physical structure (71%), functional to physical components mapping (53%)
activity	flow of data (65%), software delivery (20–22%)
sequence	data lifecycle models (47%)
component	software module composition (47%), system configuration (21%)
package	software module structure (47%)

Additionally, the uses of UML are still the focus of research. For example, Chavez et al. [4] strive to ensure coherence amid a class in the UML model and its implementation in Java. Software models that are expressed in UML may need to employ precise semantics. To achieve that, professionals can use object constraint language (OCL). As far as OCL is concerned, Clarisó et al. [5] show a method that may boost the usability of the UML class diagram. Moreover, Lu et al. [6] present the use of OCL for guaranteeing the quality of cancer data in medical registries. New research shows more and more uses of UML combined with MDE. Assunção et al. [7] have used UML class diagrams to generate product line architecture variants. Arora et al. [8] have obtained test scenarios from a UML activity diagram by analyzing alternative flows of events with the application of a bio-inspired algorithm. Test cases are the subject of research by Yousaf et al. [9]. They have chosen interaction flow modeling language models of the user interface as a starting point. The bottom-up approach is shown by Arcaini et al. [10]. They merge test cases for subsystems to achieve ones for the whole system.

Blockchain is one of the most disruptive technologies. Papers by Monrat et al. [11], and Al-Jaroodi et al. [12] examine the benefits and difficulties of employing blockchain in various business uses. In my opinion, blockchain has tremendous potential in the energy segment. Smart microgrids demand technologies that facilitate prosumers on the electricity exchange. Wang et al. [13] designed a model and blockchain framework on hyperledger fabric that allows point-to-point trading in crowdsourced energy systems. Jamil et al. [14] go on even further. They introduced a predictive energy exchange framework with a day-ahead controlling feature, whereas Lu et al. [15] concentrated on improving the system's availability while introducing a renewable energy selling method. Górski et al. [16] introduce the idea of an electricity consumption and supply management (ECSM) system. It is a renewable energy platform that has been developed with the use of the R3 Corda framework [17]. Saxena et al. [18] developed a private blockchain-based residential energy trading system. The system allows homeowners to select bidding strategies. It also reduces the peak demand for the energy of the community as a whole. They have used the hyperledger fabric framework [19]. Furthermore, Cioara et al. [20] proposed the construction of virtual power plants by applying a permissionless blockchain. Their model captures the prosumer constraints such as energy production and usage. The development of renewable energy sources grids may lead to their decentralized management with the active role of prosumers.

Researchers and practitioners use various blockchain frameworks. Chowdhury et al. [21] have published an analysis that assesses the suitability of both permissioned and permissionless blockchain frameworks. Moreover, one of the main elements in blockchain technology is a consensus algorithm. Especially in public ones, the proof-of-work consensus mechanism is the most proliferated. In such a network, reaching the consensus requires an intensive mining process [22]. Nguyen et al. [23] conduct performance analysis of the proof-of-stake consensus mechanisms. They encompass a broad range of mechanisms from basic to highly complicated.

Current research results on blockchain focus mainly on smart contracts. Distributed applications consist of smart contracts with imposed verification rules. In addition, blockchain nodes constitute the deployment environment. A properly configured deployment environment hosts distributed applications. Ozkaya et al. [3] found that the functional and information views are the most popular views in software architecture modeling. In the author's opinion, a complete architectural view model is needed. The professionals select and employ the views required for the design of a particular solution. As far as blockchain is concerned, the paper concentrates more on the *deployment* view (*UML profile for DLT deployment*). However, it also includes at least limited support for the design of smart contracts in the form of the pattern in the *contracts* view. As far as the integration of IT systems is concerned, the paper puts more emphasis on the *integrated services* view (*UML profile for integration flows*).

3. The 1+5 Model of Architectural Views

Software teams use architecture descriptions to improve communication and cooperation among stakeholders, enabling them to work in a comprehended and coherent manner. The ISO/IEC/IEEE 42010:2011 standard introduces terms defining software engineering architecture description [24]. *Architecture description* is a work product used to express an architecture. *Architecture* encompasses key ideas or characteristics of a system included in its parts, relationships, and principles of its design and development. Finally, the *architec-*

ture view shows the architecture of a system from a specific perspective. An architecture description comprises architecture views and models. Kruchten [25] presents the software architecture model 4+1 that encompasses various architectural views: *scenarios*, *logical*, *development*, *process*, and *physical*. The name *use cases* is used interchangeably with the name *scenarios*. There are other models of architectural views [26], e.g., SEI, Siemens, RM-ODP. There is definitely a lack of such description well suited to cooperating information technology systems and especially blockchain solutions. The 1+5 architectural views model has been proposed to design software systems that realize common business processes and must exchange information [1] (Figure 1).

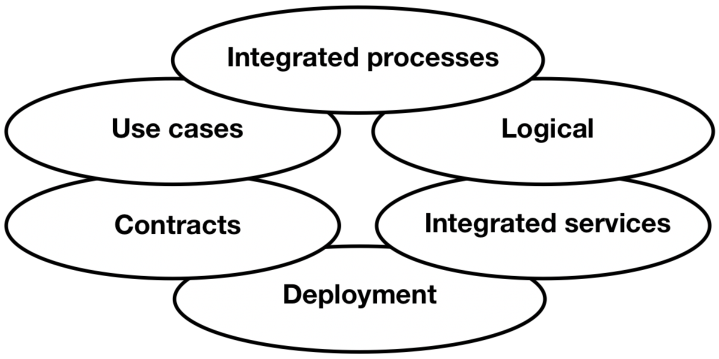


Figure 1. The graphical abstract of the 1+5 architectural views model.

Within the *integrated processes* view we can model business processes. Secondly, we can identify the required software systems support. Furthermore, we can look at the required collaboration among those software systems. Generally, we have human and automated tasks within a business process. The latter ones become services. They automate the execution of a business process. We can model processes in a UML activity diagram or business process model and notation business process diagram. In the *use cases* view we can describe the functions of the software system. The functionality scope is depicted in the UML use case diagram. A new stereotype ≪IntegratedSystem≫ has been proposed to represent an external system. The stereotype can be applied to an actor. Within the *logical* view we can design realizations of the identified use cases. Usually, we use three UML diagrams: class, sequence, and communication. The UML class diagram can be also used to show the structure of business entities used in business processes. The *contracts view* shows agreements inflicted on collaborating parties. The view uses stereotypes from service-oriented architecture modeling language (SoaML) to denote providers and consumers of services. The ≪provider≫ stereotype marks a component that realizes the service. The ≪consumer≫ stereotype marks a component that uses the service. We can use the UML component diagram to show contracts. Moreover, a contract can be used to specify non-functional requirements. The *integrated services* view shows the interaction between service providers and consumers. We can use a UML component diagram. A new stereotype ≪ESB≫ has been introduced to clearly identify a central point of communication, the enterprise service bus. Using a service usually involves integration flow. The flow consists of mediation mechanisms. The set of mediation mechanisms has been put in the *UML profile for integration flows*. Moreover, the *integration flow diagram* has been proposed to model an integration flow. The flow is invoked and executed by the enterprise service bus. The physical runtime installation for the software system can be shown in the *deployment* view. The installation consists of the hardware and the execution environment required to execute the developed software system. We can use the UML deployment diagram. The view also depicts the placement of software components onto physical nodes. The applicability of the 1+5 model for IT systems integration was presented by the author at the EUROCAST 2019 conference [27].

4. UML Profiles

UML offers the following extensibility mechanisms: constraints, stereotypes and tagged values [28]. Constraints define rules for protecting the integrity of an element in the model. Stereotypes are new types of modeling elements that extend the semantics of the UML metamodel existing elements. The specification treats the tagged value as a property that is a name-value pair. The tagged value can be an attribute of a model element. Profiles group extension mechanisms.

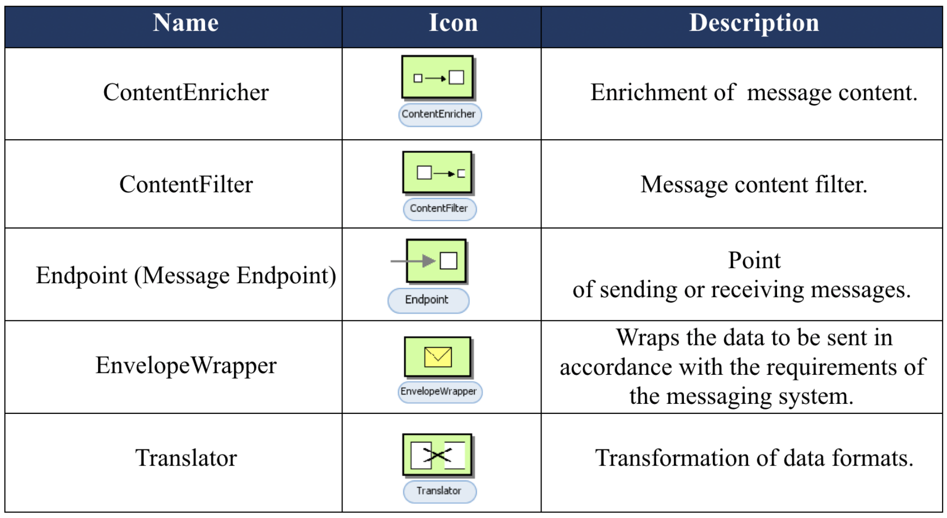
Stereotypes and tagged values have been applied for defining profiles with the required semantic UML enrichment for facilitating the modeling of selected aspects of integration platform. The first UML profile groups stereotypes for mediation mechanisms of integration flows usually used by enterprise service buses. The profile can be used in the *integrated services* view of the 1+5 model. The second UML profile groups stereotypes that represent blockchain nodes. Furthermore, each node may be described by deployment parameters. They are modeled as tagged values. In turn, this profile can be applied in the *deployment* view of the 1+5 model.

4.1. UML Profile for Integration Flows

Software systems may use different data formats, operate at various communication protocols, and store data using incompatible structures. Integration flows ensure all needed transformations to send the message from one system to another. Integration flows involve using mediation mechanisms. The *UML profile for integration flows* profile contains stereotypes that represent enterprise integration patterns. Stereotypes in the profile enable the modeling of mediation flows. Each stereotype represents a mediation mechanism. All stereotypes refer to messages. They allow modeling of the following aspects of messages: transformation, systems, routing, and endpoints. The profile encompasses 40 mediation mechanisms from enterprise integration patterns. A few of them have been shown here:

- Aggregator—combines the results of related messages so that they could be processed as a whole;
- DynamicRouter—messages are sent only to those customers who meet certain conditions;
- MessageFilter—message filtering to prevent these unwanted;
- Resequencer—laying related messages in order;
- Splitter—divides a complex message into several smaller messages which are then separated.

Each stereotype in the profile has its own icon (Figure 2).



Name	Icon	Description
ContentEnricher	ContentEnricher	Enrichment of message content.
ContentFilter	ContentFilter	Message content filter.
Endpoint (Message Endpoint)	Endpoint	Point of sending or receiving messages.
EnvelopeWrapper	EnvelopeWrapper	Wraps the data to be sent in accordance with the requirements of the messaging system.
Translator	Translator	Transformation of data formats.

Figure 2. Selected mediation mechanisms.

The profile has been designed using the IBM Rational Software Architect. The profile file EIP.epx can be used to model integration flows and is stored under Git version control, [29]. A broader description of the profile can be found in the paper [30].

4.2. UML Profile for DLT Deployment

The profile for modeling the deployment of a blockchain network is of a completely different nature and is at a much more detailed level of accuracy. In the case of the deployment view, we operate with a specific deployment environment. To talk about the further possibility of automating the deployment of solutions, we must be able to define in the model their individual, specific parameters. That is why the profile is so close to the physical files.

The profile concentrates on the platform-specific model (PSM) to express the precise deployment configuration of the R3 Corda framework. The profile has been configured in such an elastic way that it encompasses Corda's versions from 4.3 to 4.6. Stereotypes have been applied for nodes, services, and communication protocols. The network map node hosts the network map service. Notary's node responsibility is signing transactions. Oracle node provides facts from the outside world that are needed to commit transactions. The DLT node can transact with other nodes. The Corda node is an abstract one. That abstract node is needed to hold properties that are shared among all nodes. Nodes in the Corda network communicate using protocols through transactions. Nodes host and run services. The permissioning service provides TLS certificates. The network map service enforces rules allowing nodes admission to the network. The notary service signs transactions providing proper timing. Two additional stereotypes have been added for communication protocols: HTTPS, and AMPQ/TLS. Table 2 summarizes stereotypes in the profile for nodes, services, and protocols.

Table 2. Stereotypes for nodes and services.

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

Next, deployment parameters have been identified for each type of node. The range of configuration parameters has been significantly expanded in the current paper. In the profile, the tagged value represents the deployment parameter. As a result, the number of tagged values has doubled. For a node in the Corda network, we can specify values for 107 different deployment configuration parameters. For example, the profile encompasses the deployment configuration of the enterprise nodes. The tuning section has been also included in the profile. Up-to-date documentation of the R3 Corda framework in version 4.6

is accessible at the link [17]. According to UML 2.0 specification, tagged values are attributes of a stereotype. Therefore, proper tagged values have been attached to stereotypes of nodes. In particular, the set of tagged values common for all types of nodes has been attached to the abstract ≪CordaNode≫ stereotype.

Table 3 shows a few tagged values for deployment parameters associated with the ≪CordaNode≫ stereotype.

Table 3. Tagged values for the ≪CordaNode≫ stereotype.

Name	Type	Default
myLegalName	Text	Not defined
cryptoServiceTimeout	Integer	10,000
database.initialiseSchema	Boolean	true
database.initialiseAppSchema	Text	NONE
detectPublicIp	Boolean	false
enterpriseConfiguration.mutualExclusion.on	Boolean	false
enterpriseConfiguration.healthCheck	Boolean	true
enterpriseConfiguration.externalBridge	Boolean	false
enterpriseConfiguration.maintenanceMode	Text	Not defined
p2pAddress	Text	Not defined
messagingServerAddress	Text	Not defined
networkServices.doormanURL	Text	Not defined
networkServices.pnm	Text	Not defined
flowTimeout.maxRestartCount	Integer	6
flowTimeout.timeout	Integer	30
flowMonitorPeriodMillis	Integer	60
tuning.backchainFetchBatchSize	Integer	50
tuning.flowThreadPoolSize	Integer	30
tuning.rpcThreadPoolSize	Integer	4
rpcSettings.useSsl	Boolean	false
transactionCacheSizeMegaBytes	Integer	8

In the paper, the scope of configuration parameters for the notary node has been also significantly extended. As a result, the number of tagged values has quadrupled. A total of 27 tagged values have been defined for deployment parameters of the ≪NotaryNode≫ stereotype. It results from considering the deployment parameters of the notary node operating in the high availability mode.

Table 4 contains a few tagged values for deployment parameters associated with the ≪NotaryNode≫ stereotype.

Table 4. Tagged values for the ≪NotaryNode≫ stereotype.

Name	Type	Default
notary.serviceLegalName	Text	Not defined
notary.validating	Boolean	false
notary.extraConfig.maxInputStates	Integer	2000
notary.extraConfig.backOffBaseMs	Integer	20
notary.extraConfig.batchSize	Integer	32
notary.raft.nodeAddress	Text	Not defined
notary.raft.clusterAddresses	Text	Not defined
notary.dftSMaRt.clusterAddresses	Text	Not defined
notary.bftSMaRt.replicaId	Text	Not defined
notary.jpa.connectionRetries	Integer	2
notary.jpa.backOffIncrement	Integer	500
notary.jpa.backOffBase	Float	1.5
notary.jpa.maxQueueSize	Integer	100,000

All stereotypes for Corda network nodes have a common set of tagged values. Because they inherit from the ≪CordaNode≫ stereotype.

Figure 3 presents an inheritance tree of Corda node stereotypes with tagged values.

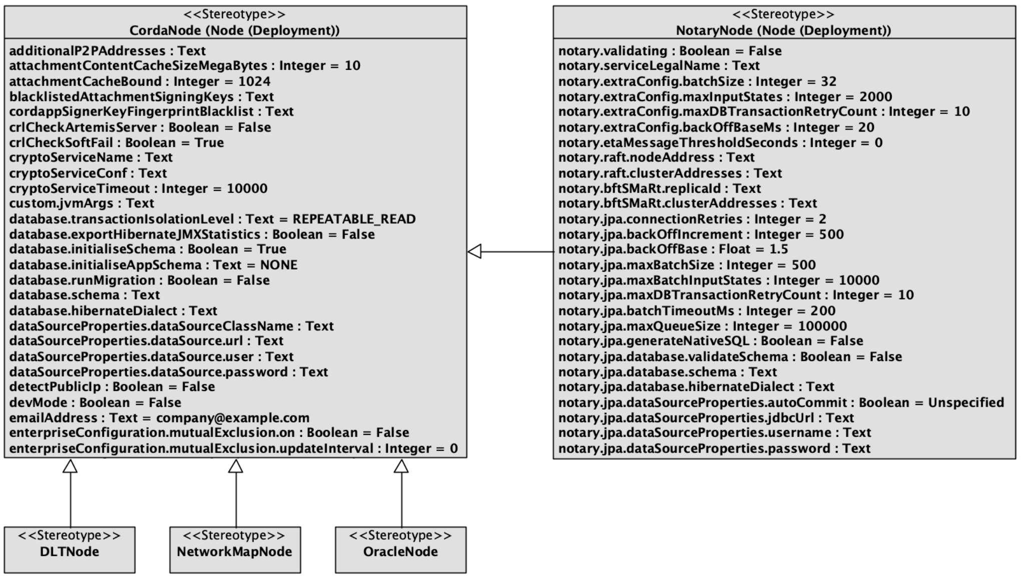

<<Stereotype>>
CordaNode (Node (Deployment))
additionalP2PAddresses : Text
attachmentContentCacheSizeMegaBytes : Integer = 10
attachmentCacheBound : Integer = 1024
blacklistedAttachmentSigningKeys : Text
cordappSignerKeyFingerprintBlacklist : Text
crlCheckArtemisServer : Boolean = False
crlCheckSoftFail : Boolean = True
cryptoServiceName : Text
cryptoServiceConf : Text
cryptoServiceTimeout : Integer = 10000
custom.jvmArgs : Text
database.transactionIsolationLevel : Text = REPEATABLE_READ
database.exportHibernateJMXStatistics : Boolean = False
database.initialiseSchema : Boolean = True
database.initialiseAppSchema : Text = NONE
database.runMigration : Boolean = False
database.schema : Text
database.hibernateDialect : Text
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
enterpriseConfiguration.mutualExclusion.on : Boolean = False
enterpriseConfiguration.mutualExclusion.updateInterval : Integer = 0
<<Stereotype>>
NotaryNode (Node (Deployment))
notary.validating : Boolean = False
notary.serviceLegalName : Text
notary.extraConfig.batchSize : Integer = 32
notary.extraConfig.maxInputStates : Integer = 2000
notary.extraConfig.maxDBTransactionRetryCount : Integer = 10
notary.extraConfig.backOffBaseMs : Integer = 20
notary.etaMessageThresholdSeconds : Integer = 0
notary.raft.nodeAddress : Text
notary.raft.clusterAddresses : Text
notary.bftSMaRt.replicaId : Text
notary.bftSMaRt.clusterAddresses : Text
notary.jpa.connectionRetries : Integer = 2
notary.jpa.backOffIncrement : Integer = 500
notary.jpa.backOffBase : Float = 1.5
notary.jpa.maxBatchSize : Integer = 500
notary.jpa.maxBatchInputStates : Integer = 10000
notary.jpa.maxDBTransactionRetryCount : Integer = 10
notary.jpa.batchTimeoutMs : Integer = 200
notary.jpa.maxQueueSize : Integer = 100000
notary.jpa.generateNativeSQL : Boolean = False
notary.jpa.database.validateSchema : Boolean = False
notary.jpa.database.schema : Text
notary.jpa.database.hibernateDialect : Text
notary.jpa.dataSourceProperties.autoCommit : Boolean = Unspecified
notary.jpa.dataSourceProperties.jdbcUrl : Text
notary.jpa.dataSourceProperties.username : Text
notary.jpa.dataSourceProperties.password : Text
<<Stereotype>>
DLTNode
<<Stereotype>>
NetworkMapNode
<<Stereotype>>
OracleNode

Figure 3. The UML profile diagram depicts stereotypes with tagged values.

For the ≪CordaNode≫ stereotype the figure shows only selected tagged values, whereas for the ≪NotaryNode≫ one it presents the complete set of them. The *UML profile for distributed ledger deployment* contains all 11 stereotypes and 134 tagged values. The profile has been designed using the visual paradigm tool. The profile is stored under Git version control [31].

5. Smart Contract Design Pattern

The 1+5 model fits the architecture description of distributed ledger and blockchain solutions. In distributed ledger solutions, transactions occur between two collaborating parties. Both collaborators must follow rules enclosed in a smart contract. To properly model smart contracts the following UML stereotypes have been identified:

- ≪Contract≫—the agreement inflicted on transactions among blockchain nodes;
- ≪VerificationRule≫—a condition that needs to be met;
- ≪State≫—denotes a fact, which is stored in a blockchain node;
- ≪Flow≫—a course of actions among nodes in reaching a consensus on the transaction.

All of them have been used to propose the *smart contract design pattern* for the R3 Corda distributed ledger framework. That pattern allows the design of smart contracts in a flexible manner with the application of a dynamic list with verification rules. The pattern uses the *Contract* interface from the Corda framework. By default, the concrete class implements the interface. The *verify()* method is declared in the interface. In the method, the concrete class implements the rules that verify the contract. However, it is embedded in the body of the method and any change to the rules requires recompilation. In the proposed approach, we have a list of verification rules in the method. The *verify()* method will work correctly regardless of how many and what rules we introduce in the list of rules. In the case of extending the rules, we add the appropriate class for a new verification rule.

Figure 4 presents the *smart contract design pattern* with the complete set of interfaces and classes. For readability, the figure shows only two concrete verification rules.

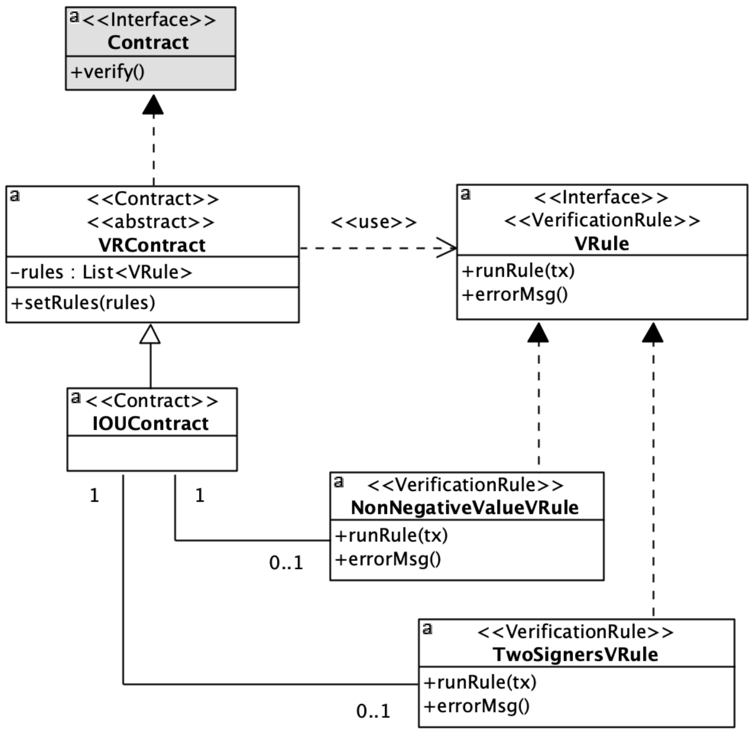
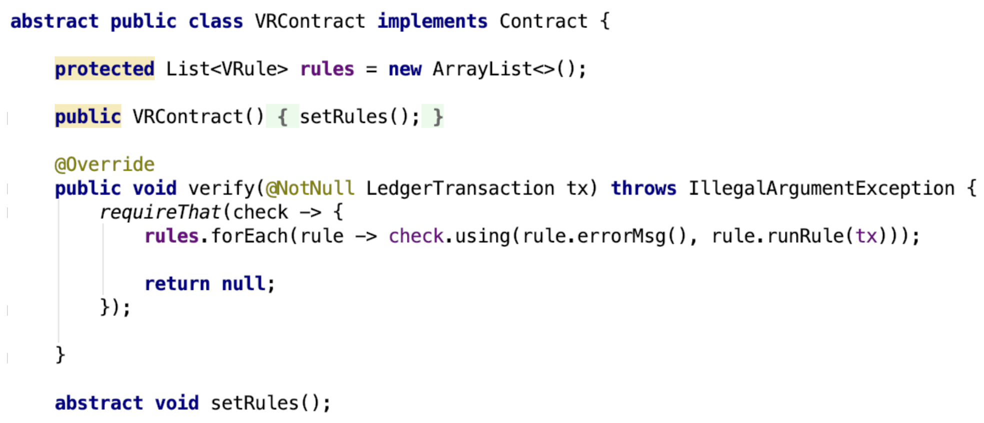


Figure 4. The UML class diagram shows the smart contract design pattern.

The pattern has two layers: abstract and actual. In the abstract layer, there is the abstract class *VRContract* that declares a dynamic list of verification rules. That layer also includes the *VRule* interface that provides the definition for verification rule. The abstract class uses the definition of that interface. In the actual layer, there are classes that represent contract and verification rules. The concrete implementation of the *runRule()* method is provided by each of verification rule classes. Using the *verify()* method, the *IOUContract* class iterates across the dynamic list of verification rules. The abstract class *VRContract* declares attribute *rules* that stores list of verification rules and implements *verify()* method.

The *verify()* method uses lambda expression to check whether all verification rules are met (see Figure 5).



```
abstract public class VRContract implements Contract {

    protected List<VRule> rules = new ArrayList<>();

    public VRContract() { setRules(); }

    @Override
    public void verify(@NotNull LedgerTransaction tx) throws IllegalArgumentException {
        requireThat(check -> {
            rules.forEach(rule -> check.using(rule.errorMsg(), rule.runRule(tx)));

            return null;
        });

    }

    abstract void setRules();
```

Figure 5. The source code of the *VRContract* abstract class.

The *IOUContract* class instantiates objects of concrete verification rules classes and adds them to an immutable list of those verification rules objects stored in the *rules* attribute. The source code of a smart contract implementation using *smart contract design pattern* is available at the GitHub repository, [32].

6. Case Studies

6.1. e-Prescription

The first case study shows the use of the model to design a prescription circulation. The initial situation includes separate IT systems of clinics and pharmacies with a paper circulation of such documents. The aim was to build an integration solution enabling the electronic circulation of prescriptions and their realizations.

When you visit a doctor usually it ends with a written prescription. To realize the prescription, you step into a pharmacy. The prescription is realized by a pharmacist in full. The aim was to ensure the electronic flow of the prescription and the realization. In that case, the prescription is in electronic form, and it does not require you to carry it with yourself. You can realize those prescription positions that you want or are available in the pharmacy. It required designing applications for medical clinics and pharmacies. Both have been implemented in the IntelliJ IDEA tool using Java Server Faces. The source code of both applications is accessible on demand at GitHub repositories [33,34]. The e-Prescription application implements two main uses cases—the *write prescription* for issuing the prescription, and the *get prescriptions* for viewing those issued for the patient. The *get prescriptions* use case must also be accessible for the e-Pharmacy. The pharmacist should be able to locate the prescription for realization.

Figure 6 depicts use cases of the e-Prescription application. Thanks to the use of the ≪IntegratedSystem≫ stereotype, the external system is clearly visible in the diagram. We can see that the *get prescriptions* use case can be invoked by the e-Pharmacy.

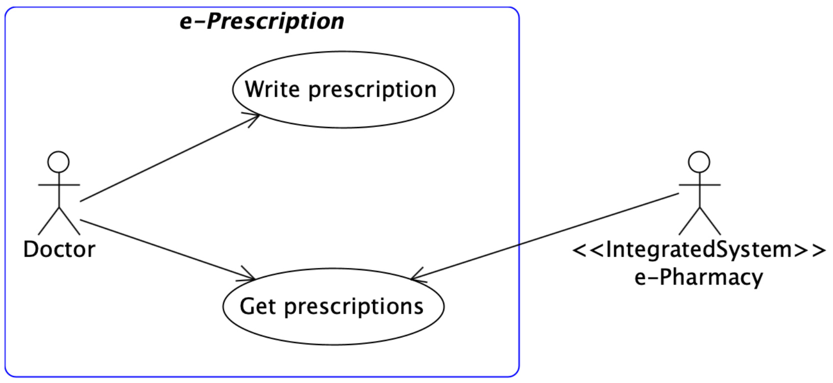


Figure 6. Use case diagram for e-Prescription.

The e-Pharmacy application also implements two main uses cases. The *realize prescription* for realizing the prescription and the *get prescription's realization* for viewing realized prescription for the patient. The *get prescription's realization* use case must be also accessible for the e-Prescription. The doctor should be able to view the prescription's realization. Figure 7 shows use cases of e-Pharmacy.

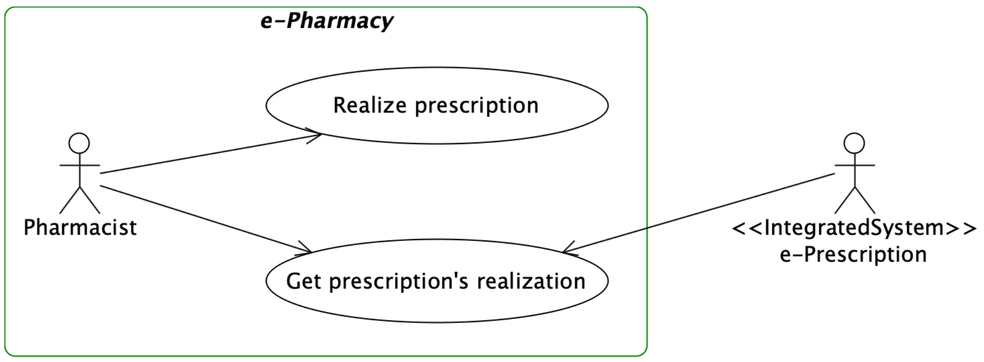


Figure 7. Use case diagram for e-Pharmacy.

Both use cases that involve integration between applications of a medical clinic and pharmacy have been exposed as services onto ESB. Figure 8 presents providers and consumers of services into the UML component diagram. The service exposed from a component is represented as a realized interface. Services needed by a component are depicted as a required interface. The e-Prescription application provides the *Prescription* service. The e-Pharmacy provides the *PrescriptionRealization* service. The component with the ≪ESB≫ stereotype manages communication between those two applications.

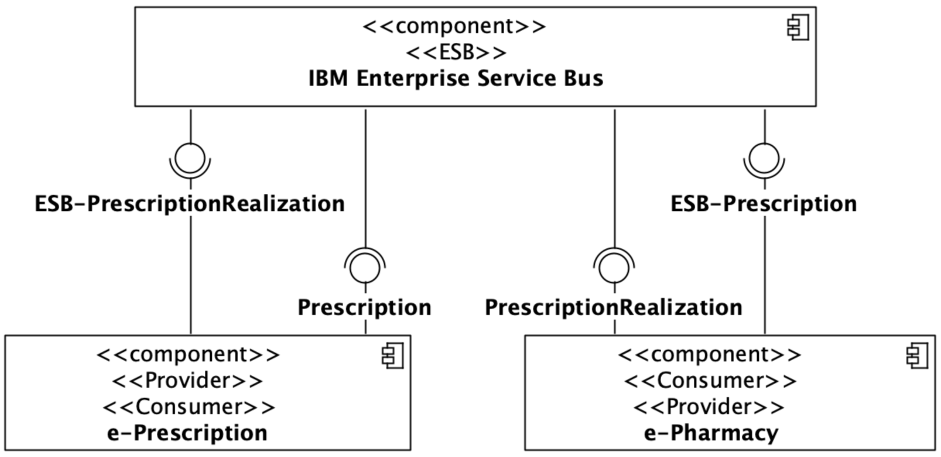


Figure 8. The UML component diagram with service providers and consumers.

Each application has a different data format for a prescription and its realization. To read a prescription in the e-Pharmacy application, it should be transformed to the right format. Likewise, reading a prescription's realization in the e-Prescription also requires transformation to the applicable format. Therefore, service execution onto ESB invokes specific integration flow.

Figure 9 depicts the *integration flow diagram* for sending prescriptions to the e-Pharmacy. In the diagram, we can see mediation mechanisms (*UML profile for integration flows*) arranged in a flow.

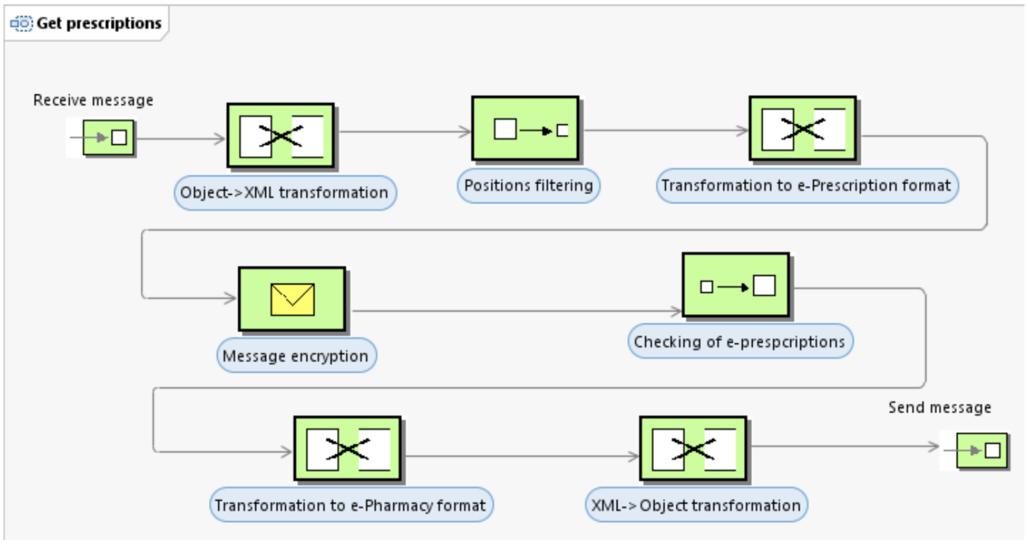

Get prescriptions
Receive message
Object->XML transformation
Positions filtering
Transformation to e-Prescription format
Message encryption
Checking of e-prespcriptions
Send message
Transformation to e-Pharmacy format
XML-> Object transformation

Figure 9. The integration flow diagram for sending prescriptions to the e-Pharmacy.

The 1+5 model has shown to be helpful in the architectural description of that integration solution [35]. The solution ensures the electronic circulation of prescriptions and their realizations.

6.2. Communication between ARMA and AFQI

The second case study shows the use of the model to design one-way communication between the Agency for Restructuring and Modernization of Agriculture (ARMA) and the Agricultural and Food Quality Inspection (AFQI). The initial situation encompasses the separate IT systems of ARMA and AFQI. The aim was to build an integration solution that would enable the transfer of a large set of multiple documents while ensuring an appropriate level of performance.

The academic project was devoted to checking the possibility of building an electronic collaboration platform between ARMA and AFQI. In Poland, competency for supervising the quality of food products is trusted to ARMA. Certification bodies are responsible for issuing and revoking certificates for organic farming. A certification body sends the *list of organic producers* both to ARMA and AFQI.

Figure 10 shows the flow of information among the Polish authorities participating in the process of supervising the quality of agricultural and food commodities.

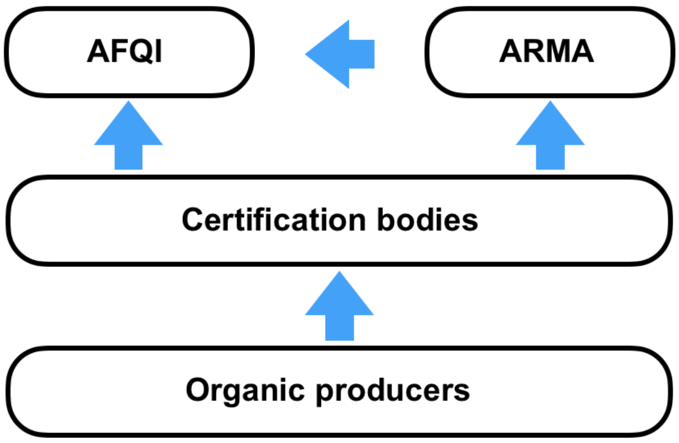


Figure 10. The authorities in the process of supervising the quality of agri-food products in Poland.

The ARMA supervises the whole process and checks the lists from certification bodies. In the event of non-compliance, ARMA punishes the certification body. This is done indirectly by AFQI. ARMA sends to AFQI a call to punish the certification body. The problem lays in the size of such a request. One call to punish may encompass 700 MB of data. Originally, the request data were burned to a DVD and sent by post. As a result, it usually took 2–3 business days for the request to reach AFQI. Therefore, the integration solution with ESB has been proposed to facilitate communication between those two authorities. In that case, the *1+5* model has been applied to design a single use case to send a request to punish a certification body.

Figure 11 shows the UML case diagram for the solution.

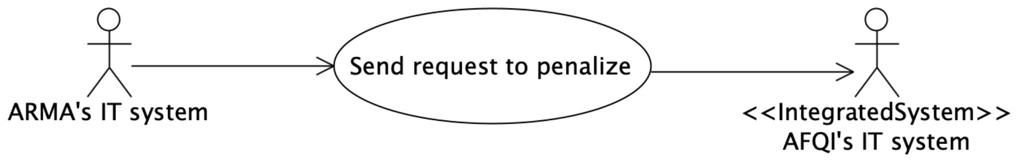


Figure 11. The Use case diagram with the *Send request to penalize*.

Specific requirements were imposed on the integration solution, e.g.: automate the transfer of data, ensure reliable message delivery, process each message once only, and compliance of XML documents with XSD schema. The message encompassed files in PDF and JPG formats encoded by the base64 algorithm. Simple object access protocol (SOAP) message transmission optimization mechanism (MTOM) is used to transfer binary data using web services. The XML optimized packaging (XOP) mechanism is used to send data as multipurpose internet mail extension (MIME) attachments of a SOAP message. The data in a SOAP message is referenced in the <xop: Include> tag. As a result, text data are separated from binary ones.

Figure 12 shows the UML component diagram with ESB and cooperating systems.

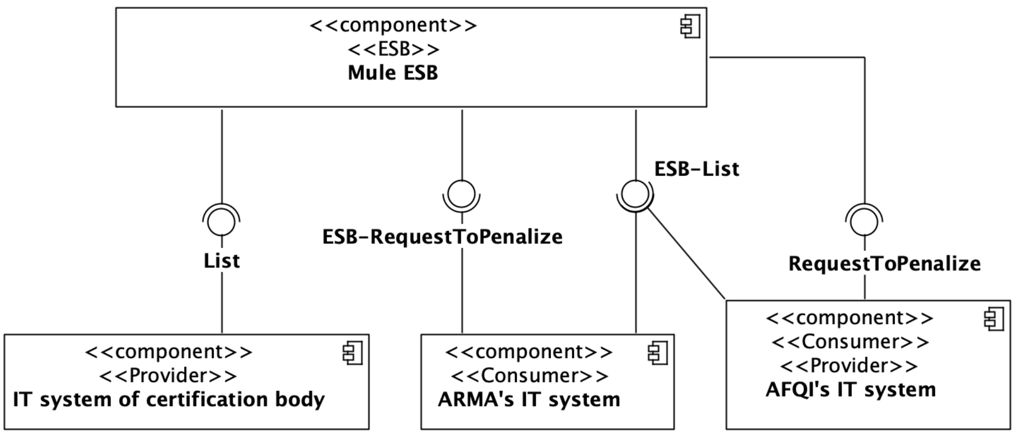


Figure 12. The UML component diagram with cooperating IT systems.

There were two main tasks: send files electronically, and deliver them in a reasonable time. The second was crucial because of the large size of documentation. One request to penalize, available for performance tests, contained 675.71 MB of data. It encompassed graphical files: photos and sketches of plots. The communication between systems has been treated as a queuing system. The required length of the queue and number of serving processes have been determined by applying Little's law, [36]. At the design level, the RabbitMQ message broker has been applied as a queue and Mule ESB as a service node. Additionally, to effectively deal with a short-term heavy load, the *decoupled invocation* pattern has been also incorporated [37]. Furthermore, each of 169 files from the documentation set has been transferred as a single message. As a result, ARMA can send to AFQI a request to penalize within 2 min.

A broader description of the integration with the performance analysis of the solution can be found in the paper [38].

6.3. ECSM

The third case study shows the use of the model to design a renewable energy management solution. The essence is to record the use of energy in the prosumer node and the energy produced by the node but used in other network nodes. Blockchain technology has been used in this solution, in particular the R3 Corda environment.

The ECSM system allows for continuous monitoring and recording of information about energy generated and used by prosumers. The system is blockchain-based and, using smart contracts, in each node stores information about inbound and outbound energy. The solution concentrates on the *deployment* view. In designing the *UML deployment model* of the ECSM system the *UML profile for distributed ledger deployment* has been applied, [16]. Nodes have been labeled with proper stereotypes. For example, nodes that represent prosumers with renewable sources of energy have been labeled with the ≪DLTNode≫ stereotype. Thanks to that, all tagged values associated with the stereotype have been available for configuration in the UML deployment model. The *UML deployment model* has been divided into deployment environments, e.g., *dev* for development, *test* for testing, and *prod* for production. The UML package groups nodes and represents the deployment environment. We can use separate UML deployment diagrams to depict deployment environments. The *UML deployment model* for ECSM that uses stereotypes and tagged values from the *UML profile for distributed ledger deployment* is available at the GitHub repository, [39].

Figure 13 depicts the UML deployment diagram for a part of the ECSM.

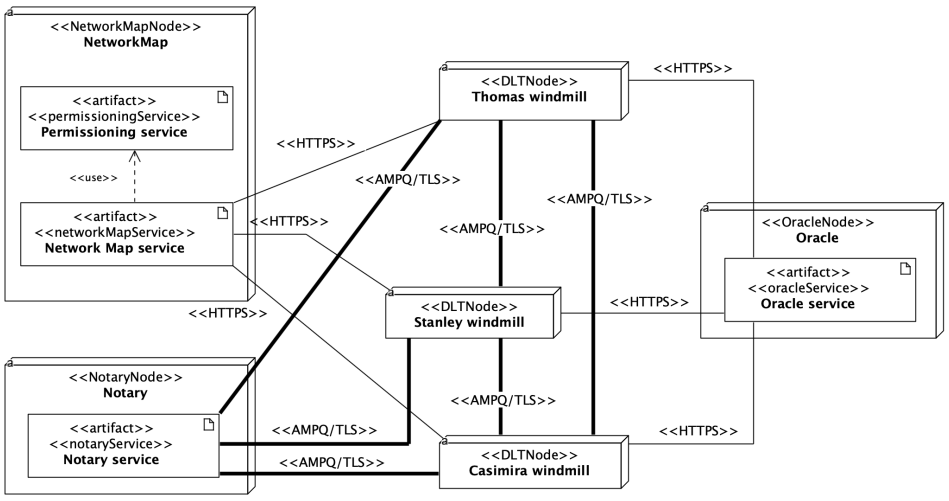


Figure 13. The UML deployment diagram for part of the ECSM system.

The *logical* view presents design of functions elicited in the *use cases* view. The UML class diagram presents classes and interfaces needed for the design of a smart contract (see Figure 14). Elements from the Corda environment have a gray background, and classes and interfaces that should be implemented are presented in white.

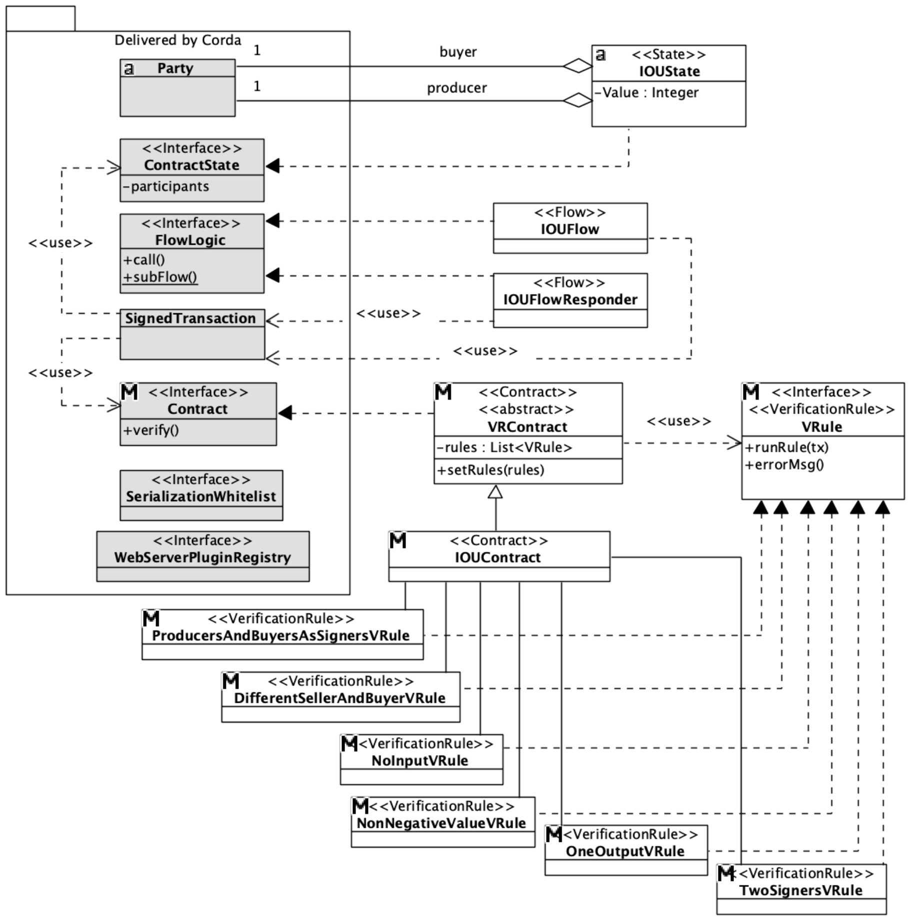


Figure 14. The UML class diagram for the smart contract design.

The source code of the application with the smart contract implementation is available at the GitHub repository, [32].

7. Conclusions

The paper describes the 1+5 model dedicated to the design of cooperating IT systems in the context of the realization of common business processes. The application of the model required the creation of additional semantic structures of the UML language. The article describes two UML profiles: *UML profile for integration flows*, and *UML profile for distributed ledger deployment*. The *integration flow diagram*, a specialized version of the UML activity diagram, has also been proposed. The paper contains examples of using the model to design the integration of IT systems in service architecture and the blockchain-based renewable energy management system. The model has been verified in both approaches: centralized architecture with a central element that is ESB and distributed blockchain technology architecture with peer-to-peer connections.

The *UML profile for distributed ledger deployment* has been proposed for R3 Corda distributed ledger up to version 4.6. The profile is at the platform-specific model (PSM) level. The deployment level is strongly associated with the specific blockchain platform. As further work, the author sees the need to propose profiles at the PSM level for other blockchain platforms, e.g.: private Hyperledger Fabric, and public Ethereum. It is also planned to generalize common features and define the profile for blockchain at the platform-independent model level. That may be the right step in the direction of the portability of smart contract applications between blockchain platforms. The work has stirred in the direction of the continuous delivery approach for generating complete deployment packages for blockchain nodes with up-to-date smart contract applications. It means using Git repositories for source-code storage and the Jenkins server for task automation. It is also planned to use the Kubernetes platform for automating the deployment of blockchain distributed applications in containers. Finally, within the *contracts* view, it is taken into account to propose mechanisms for modeling the security of blockchain applications. At the opposite level of abstraction are the following views: *integrated processes*, *use cases*, and *logical*. It is planned to provide the complete 1+5 model for distributed applications and the solution for their continuous deployment.

Funding: The research has been conducted within the *architectural views model of cooperating IT systems* project financed by the statutory resources of the Department of Computer Science, PNA.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Górski, T. Architectural view model for an integration platform. *J. Theor. Appl. Comput. Sci.* **2012**, *10*, 25–34.
2. Górski, T. *Integration Platforms. Selected Issues*; Wydawnictwo Naukowe PWN (State Scientific Publishing House): Warsaw, Poland, 2012; ISBN 978-83-01-17071-4. (In Polish)
3. Ozkaya, M.; Erata, F. A survey on the practical use of UML for different software architecture viewpoints. *Inf. Softw. Technol.* **2020**, *121*, 106275. [CrossRef]
4. Chavez, H.M.; Shen, W.; France, R.B.; Mechling, B.A.; Li, G. An Approach to Checking Consistency between UML Class Model and Its Java Implementation. *IEEE Trans. Softw. Eng.* **2016**, *42*, 322–344. [CrossRef]
5. Clarisó, R.; González, C.A.; Cabot, J. Smart Bound Selection for the Verification of UML/OCL Class Diagrams. *IEEE Trans. Softw. Eng.* **2019**, *45*, 412–426. [CrossRef]
6. Lu, H.; Wang, S.; Yue, T.; Ali, S.; Nygård, J.F. Automated Refactoring of OCL Constraints with Search. *IEEE Trans. Softw. Eng.* **2019**, *45*, 148–170. [CrossRef]
7. Assunção, W.K.G.; Vergilio, S.R.; Lopez-Herrejon, R.E. Automatic extraction of product line architecture and feature models from UML class diagram variants. *Inf. Softw. Technol.* **2020**, *117*, 106198. [CrossRef]
8. Arora, V.; Singh, M.; Bhatia, R. Orientation-based Ant colony algorithm for synthesizing the test scenarios in UML activity diagram. *Inf. Softw. Technol.* **2020**, *123*, 106292. [CrossRef]
9. Yousaf, N.; Azam, F.; Butt, W.H.; Anwar, M.W.; Rashid, M. Automated Model-Based Test Case Generation for Web User Interfaces (WUI) From Interaction Flow Modeling Language (IFML) Models. *IEEE Access* **2019**, *7*, 67331–67354. [CrossRef]

10. Arcaini, P.; Gargantini, A.; Riccobene, E. Decomposition-Based Approach for Model-Based Test Generation. *IEEE Trans. Softw. Eng.* **2019**, *45*, 507–520. [CrossRef]
11. Monrat, A.A.; Schelén, O.; Andersson, K. A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities. *IEEE Access* **2019**, *7*, 117134–117151. [CrossRef]
12. Al-Jaroodi, J.; Mohamed, N. Blockchain in Industries: A Survey. *IEEE Access* **2019**, *7*, 36500–36515. [CrossRef]
13. Wang, S.; Taha, A.F.; Wang, J.; Kvaternik, K.; Hahn, A. Energy Crowdsourcing and Peer-to-Peer Energy Trading in Blockchain-Enabled Smart Grids. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *49*, 1612–1623. [CrossRef]
14. Jamil, F.; Iqbal, N.; Imran; Ahmad, S.; Kim, D. Peer-to-Peer Energy Trading Mechanism Based on Blockchain and Machine Learning for Sustainable Electrical Power Supply in Smart Grid. *IEEE Access* **2021**, *9*, 39193–39217. [CrossRef]
15. Lu, X.; Guan, Z.; Zhou, X.; Du, X.; Wu, L.; Guizani, M. A Secure and Efficient Renewable Energy Trading Scheme Based on Blockchain in Smart Grid. In Proceedings of the IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1839–1844. [CrossRef]
16. Górski, T.; Bednarski, J.; Chaczko, Z. Blockchain-based renewable energy exchange management system. In Proceedings of the 26th International Conference on Systems Engineering, ICSEng 2018, Sydney, Australia, 18–20 December 2018; pp. 1–6. [CrossRef]
17. Documentation for Corda Enterprise 4.6 Version. Available online: https://docs.corda.net/docs/corda-enterprise/4.6.html (accessed on 20 October 2021).
18. Saxena, S.; Farag, H.E.Z.; Brookson, A.; Turesson, H.; Kim, H. A Permissioned Blockchain System to Reduce Peak Demand in Residential Communities via Energy Trading: A Real-World Case Study. *IEEE Access* **2021**, *9*, 5517–5530. [CrossRef]
19. Hyperlegder Fabric. Available online: www.hyperledger.org/use/fabric (accessed on 20 October 2021).
20. Cioara, T.; Antal, M.; Mihailescu, V.T.; Antal, C.D.; Anghel, I.M.; Mitrea, D. Blockchain-Based Decentralized Virtual Power Plants of Small Prosumers. *IEEE Access* **2021**, *9*, 29490–29504. [CrossRef]
21. Chowdhury, M.J.M.; Ferdous, M.S.; Biswas, K.; Chowdhury, N.; Kayes, A.S.M.; Alazab, M.; Watters, P. A Comparative Analysis of Distributed Ledger Technology Platforms. *IEEE Access* **2019**, *7*, 167930–167943. [CrossRef]
22. Gramoli, V. From blockchain consensus back to Byzantine consensus. *Future Gener. Comput. Syst.* **2020**, *107*, 760–769. [CrossRef]
23. Nguyen, C.T.; Hoang, D.T.; Nguyen, D.N.; Niyato, D.; Nguyen, H.T.; Dutkiewicz, E. Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access* **2019**, *7*, 85727–85745. [CrossRef]
24. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000). ISO/IEC/IEEE Systems and Software Engineering—Architecture Description. 2011; pp. 1–46. Available online: ieeexplore.ieee.org/servlet/opac?punumber=6129465 (accessed on 20 October 2021). [CrossRef]
25. Kruchten, P. Architectural Blueprints—The 4+1 View Model of Software Architecture. *IEEE Softw.* **1995**, *12*, 42–50. [CrossRef]
26. Rozanski, N.; Woods, E. *Software Systems Architecture. Working with Stakeholders Using Viewpoints and Perspectives*, 1st ed.; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2008.
27. Górski, T. Verification of Architectural Views Model 1+5 Applicability. In *Computer Aided Systems Theory—EUROCAST 2019*; Lecture Notes in Computer Science; Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A., Eds.; Springer: Cham, Switzerland, 2020; Volume 12013, pp. 499–506. [CrossRef]
28. Pender, T. Customizing UML Using Profiles. In *UML Bible*; Wiley Publishing, Inc.: Indianapolis, IN, USA, 2003; pp. 687–723.
29. GitHub Repository with the UML Profile for Integration Flows. Available online: github.com/drGorski/UMLProfileForIntegrationFlows/blob/master/EIP.epx (accessed on 20 October 2021).
30. Górski, T. UML profiles for architecture description of an integration platform. *Bull. Mil. Univ. Technol.* **2013**, *LXII*, 43–56. Available online: https://www.researchgate.net/publication/246548013_UML_profiles_for_architecture_description_of_an_integration_platform (accessed on 20 October 2021).
31. GitHub Repository with the UML Profile for Distributed Ledger Deployment. Available online: github.com/drGorski/UMLProfileForDLT (accessed on 20 October 2021).
32. GitHub Repository with the *Smart Contract Design Pattern* Implementation. Available online: github.com/drGorski/renewableEnergyBlockchain (accessed on 20 October 2021).
33. GitHub Repository with the e-Prescription Application. Available online: github.com/drGorski/ePrescription (accessed on 20 October 2021).
34. GitHub Repository with the e-Pharmacy Application. Available online: github.com/drGorski/ePharmacy (accessed on 20 October 2021).
35. Górski, T. Architecture of Integration Platform for Electronic Flow of Prescriptions. *Ann. Coll. Econ. Anal. Warsaw Sch. Econ.* **2012**, *25*, 67–83. Available online: http://rocznikikae.sgh.waw.pl/p/roczniki_kae_z25_05.pdf (accessed on 20 October 2021). (In Polish)
36. Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*; Wiley-Interscience: New York, NY, USA, 1991; ISBN 0471503361.
37. Rotem-Gal-Oz, A. *SOA Patterns*; 1st ed.; Manning Publications: Shelter Island, NY, USA, 2012; ISBN 978-1933988269.
38. Górski, T. The use of Enterprise Service Bus to transfer large volumes of data. *J. Theor. Appl. Comput. Sci.* **2014**, *8*, 72–81.
39. GitHub Repository with the ECSM Design. Available online: github.com/drGorski/designECSM (accessed on 20 October 2021).