



Assignment of master's thesis

Title:	Business Rules in Blockchain Smart Contracts
Student:	Bc. Ondřej Šelder
Supervisor:	Ing. Marek Skotnica
Study program:	Informatics
Branch / specialization:	Managerial Informatics
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Business rules and their execution are adopted in almost every organization. The rules are usually expressed in a decision management notation (DMN) standardized by the OMG organization. This thesis aims to evaluate how the business rules can be executed in blockchain smart contracts and implement an open-source algorithm that generates the smart contract code from the DMN format. The results will be demonstrated in a simple case study.

Steps to take:

- Explore the DMN standard and current blockchain platforms
- Compare the suitability of different smart contract programming languages for the execution of business rules
- Design, implement and test an algorithm to convert DMN rules to selected blockchain programming language
- Perform a proof-of-concept case study to demonstrate the feasibility of the proposed algorithm



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Business Rules in Blockchain Smart Contracts

Bc. Ondřej Šelder

Department of Software Engineering
Supervisor: Ing. Marek Skotnica

May 2, 2022

Acknowledgements

I would like to express my gratitude to the people who supported me and helped me while I was completing my master's thesis. First and foremost, I would like to thank my supervisor Ing. Marek Skotnica for his help, time, and guidance. My appreciation goes also toward my family, and friends for all their support at the time I was working on this thesis and during my university studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 2, 2022

.....

Czech Technical University in Prague
Faculty of Information Technology
© 2022 Ondřej Šelder. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Šelder, Ondřej. *Business Rules in Blockchain Smart Contracts*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstract

Business rules now define activities throughout most organizations. Their use leads to increased consistency of outputs, better morale of employees, and more satisfied customers. Together with the blockchain network, they can significantly reduce transaction costs and automatize processes.

This thesis explores the possibilities of transforming business rules written in DMN notation to run in the blockchain. Suitable technology for the generation of smart contracts is analyzed. Then, the implementation and testing of the conversion itself are described. The thesis also contains a simple case study showing the benefits of the selected approach.

Keywords business rules, DMN, smart contract, Solidity, Plutus

Abstrakt

Podniková pravidla dnes definují aktivity ve většině organizacích. Jejich použití vede ke zvýšené konzistenci výstupů, lepší morálce zaměstnanců a spokojenějším zákazníkům. Společně s technologií blockchain s nimi lze také výrazně snížit transakční náklady a automatizovat procesy.

Tato práce zkoumá možnosti transformace podnikových pravidel napsaných v notaci DMN pro jejich spuštění v síti blockchain. Proces generace smart kontraktů, pro který bude vybrána vhodná technologie, je následně je implementován a otestován. Práce také obsahuje jednoduchou případovou studii, která představuje výhody zvoleného postupu.

Klíčová slova podniková pravidla, DMN, smart kontrakt, Solidity, Plutus

Contents

Introduction	1
1 Theoretical Foundations	3
1.1 Business Process Management	3
1.2 Business Rules and DMN	10
1.3 Blockchain Technology	15
1.4 Smart Contract	21
1.5 Ethereum Platform	25
1.6 Cardano Platform	29
1.7 Chapter Summary	35
2 Business Rule as Smart Contract	37
2.1 Representation of DMN Business Rule	37
2.2 Generation of Smart Contract	39
2.3 Chapter Summary	54
3 Proof of Concept: Implemented Generator	55
3.1 Used Technologies	55
3.2 Software Architecture	58
3.3 Testing	61
3.4 Chapter Summary	62
4 Proof of Concept: Case Study	63
4.1 Asylum Procedures in Europe	63
4.2 As-Is Analysis: Collection of Resources	64
4.3 As-Is Analysis: Modeled Process	70
4.4 To-Be Analysis: Modeled Process	75
4.5 To-Be Analysis: Simulation	84
4.6 Chapter Summary	96

Conclusion	97
Bibliography	99
A Acronyms	107
B Contents of enclosed CD	109

List of Figures

1.1 Business Process Management Lifecycle	4
1.2 Applicability of BPMN, BORM, and DEMO for Organizational Management Levels	5
1.3 Example of Communication between Pools in BPMN	6
1.4 BPMN Types of Tasks	8
1.5 BPMN Data Objects	9
1.6 BPMN Swimlines	9
1.7 BPMN and DMN Modeling Aspects	11
1.8 Decision Requirements Diagram with Value Expression	12
1.9 Example of the Decision Table	14
1.10 Technologies behind the Bitcoin	17
1.11 Graphic Representation of Blockchain	18
1.12 Definition of the Basic UTXO	18
1.13 Categorization of Consensus Algorithms	20
1.14 Adoption of TCP/IP and Blockchain	24
1.15 Structure of EVM Operations	26
1.16 Code Example of the Solidity	27
1.17 Architecture of the Plutus Platform	31
1.18 Code Example of the Plutus Tx	32
2.1 Solidity Template: General Contract	40
2.2 Solidity Template: Output Structure	40
2.3 Solidity Template: String Comparison	41
2.4 Solidity Template: Any Hit Policy	41
2.5 Solidity Template: Unique Hit Policy	42
2.6 Solidity Template: First Hit Policy	42
2.7 Solidity Template: Main Function for Priority Hit Policy	43
2.8 Solidity Template: Helper Function for Priority Hit Policy	43
2.9 Solidity Template: Output Order Hit Policy	44
2.10 Solidity Template: Rule Order Hit Policy	45

2.11 Solidity Template: Collect Hit Policy with Count Aggregation	45
2.12 Solidity Template: Collect Hit Policy with Sum Aggregation	45
2.13 Solidity Template: Collect Hit Policy with Min and Max Aggregations	46
2.14 Plutus Template: General Module	47
2.15 Plutus Template: Output Structure	48
2.16 Plutus Template: Function for Rule Evaluation	48
2.17 Plutus Template: Function for Rule Evaluation of Collect (Count)	48
2.18 Plutus Template: Function for Application of Rules	49
2.19 Plutus Template: String Comparison	49
2.20 Plutus Template: Unique Hit Policy	50
2.21 Plutus Template: Any Hit Policy	50
2.22 Plutus Template: Priority Hit Policy	51
2.23 Plutus Template: First Hit Policy	51
2.24 Plutus Template: Output Order Hit Policy	52
2.25 Plutus Template: Rule Order Hit Policy	52
2.26 Plutus Template: Collect Hit Policy with Min and Max Aggregations	52
2.27 Plutus Template: Collect Hit Policy with Sum Aggregation	53
2.28 Plutus Template: Collect Hit Policy with Count Aggregation	53
3.1 Interface of DasContract Editor	56
3.2 Abstraction: Class Diagram Selection	59
3.3 Solidity Converter: Class Diagram Selection	60
4.1 Flow Chart of the Irish Asylum Procedure	65
4.2 Descriptive As-Is Model of the Asylum Procedure: Part One	71
4.3 Descriptive As-Is Model of the Asylum Procedure: Part Two	72
4.4 Descriptive As-Is Model of the Asylum Procedure: Part Three	72
4.5 Descriptive As-Is Model of the Asylum Procedure: Part Four	73
4.6 Preview of the Analytic As-Is Model	73
4.7 Descriptive To-Be Model of the Asylum Procedure: Part One	76
4.8 Descriptive To-Be Model of the Asylum Procedure: Part Two	77
4.9 Descriptive To-Be Model of the Asylum Procedure: Part Three	78
4.10 Descriptive To-Be Model of the Asylum Procedure: Part Four	78
4.11 Executable BPMN Model for Asylum Procedure	79
4.12 Decision Table of the Review Initial Admissibility Business Rule	80
4.13 Decision Table of the Stream One Priority Business Rule	81
4.14 Decision Table of the Stream Two Priority Business Rule	82
4.15 Decision Table of the Final Decision Business Rule	83
4.16 Smart Contract Endpoint in Remix IDE	84
4.17 Checking Status of Tasks	85
4.18 Invalid Task Call	85
4.19 First Simulation Case: Preliminary Information	86
4.20 Second Simulation Case: Preliminary Information	87

4.21 Second Simulation Case: Admissibility Approval	88
4.22 Second Simulation Case: Take Back Request	89
4.23 Third Simulation Case: Preliminary Information	90
4.24 Third Simulation Case: Prioritization	91
4.25 Third Simulation Case: Final Decision	92
4.26 Third Simulation Case: Final Approval	93
4.27 Fourth Simulation Case: Preliminary Information	94
4.28 Fourth Simulation Case: Final Decision	95
4.29 Fourth Simulation Case: Final Approval	95

Introduction

Business Process Modeling is a great tool to design processes in organizations or evolve existing ones according to the management's strategy. The moving forces behind strategies are business rules, which are definitions of operations and constraints. Without them, the quality of products and services may decrease together with the morale in the organization. Business rules can be expressed using a variety of modeling standards.

Some of such standards are Decision and Model Notation (DMN) and Business Process Modeling Notation (BPMN). They both complement each other, but as the name suggests, the DMN is more focused on decision management. Business rules depicted in a certain level of DMN conformance can be made executable which streamlines automatization.

In business and government, where many transactions are occurring between multiple interested parties, blockchain technology can reduce the cost per transaction. Once there is this peer-to-peer network established, it is possible to enhance it with self-executing codes in the form of smart contracts.

The development of smart contracts requires a higher level of expertise. The code must be without errors since it cannot be modified once it is deployed. As the adoption of blockchain technology is spreading, many new software solutions are developed to overcome mentioned obstacles. DasContract is a visual language based on BPMN that simplifies the creation of smart contracts. It can also help to discover insecure parts in their design.

There are various possibilities for the selection of target smart contract language. Solidity and Plutus are programming languages with professional developer communities and research. Although, they are using different concepts and approaches.

This thesis aims to analyze the suitability of both programming languages for DMN business rules transformation and integrate sample generation to DasContract Editor. The application's offered solution is then showcased in a simple case study exploring the improvement of asylum procedures with generated smart contracts.

Structure of the Thesis

The thesis is structured as follows:

Chapter 1 – Theoretical Foundations – describes the DMN standard and how it can be used together with BPMN notation. Afterward, the blockchain and smart contract technology are summarized with a focus on Cardano and Ethereum platforms.

Chapter 2 – Business Rule as Smart Contract – explores ways of translating DMN business rules to Plutus and Solidity smart contracts. Then, it compares their suitability for such a process.

Chapter 3 – Proof of Concept: Implemented Generator – presents own implementation of smart contract generator from DMN business rules. It lays out the used technologies and software architecture of DasContract software that it extends.

Chapter 4 – Proof of Concept: Case Study – depicts the process of the asylum procedure in the BPMN model for further improvement using blockchain technology. The model together with business rules is then translated to a smart contract with the use of mentioned software.

Conclusion – ends the thesis with a summary of the used solution and evaluates the achievement of set goals. Lastly, possible directions for future research are offered.

Theoretical Foundations

In this chapter, the necessary background information and concepts are laid out by the thesis to support any following exploration of methods. There are two distinct topics in this chapter. In the first part, the DMN abstraction of business rules that is a part of Business Process Modeling is described. The second part of this chapter deals with the blockchain network. It starts with the basics of the technology continuing with extensions that are currently being worked on by Ethereum and Cardano platforms.

1.1 Business Process Management

A business process is a key tool for organizations to provide a service or product to their customers and users. There are more definitions, but all of them identify that it is a collection of activities that are achieving certain objectives. [1]

Their principle was known in the past, but the need to identify and work with such a concept was more prevalent past three decades. The field of Business Process Management arose from the effort of businesses to gain and maintain a competitive position in the market. The focus was on how to create, manage and alter processes to achieve bigger quantity and better quality of outputs, as well as reduce the cost and time to produce them. Another dimension in which the organizations can improve is to provide more services. [1]

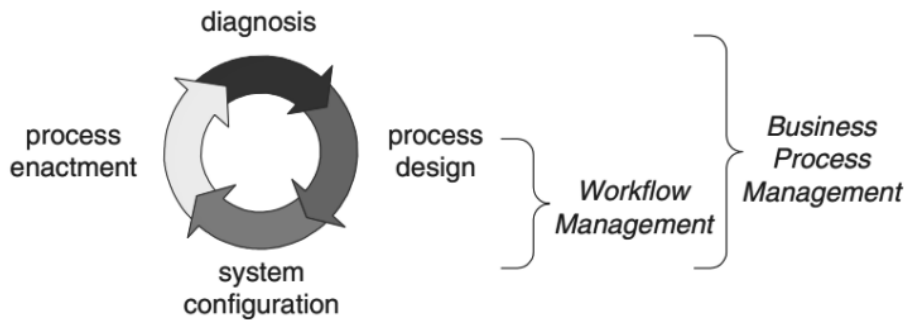
1.1.1 Application of Business Process Management

Aalst et al. [2] define Business Process Management as follows: *"Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information."*

Basically, Business Process Management is a field that utilizes information technology to effectively organize people and introduce changes to the business

processes bringing improvement. Aalst et al. [2] divide the lifecycle of Business Process Management into 4 stages.

Figure 1.1: Business Process Management Lifecycle [2]



The lifecycle that can be seen in the Figure 1.1 starts with Process design where as-is models are designed and potentially redesigned. The second phase called System configuration is the hardest from the view of standardization. The information systems must be configured so that they are ready for the execution of the models. The execution itself is done in the phase of Process enactment. The last phase is an examination of operational processes labeled Diagnosis. [1]

Business Process Management can be used on multiple levels and dimensions of organizations but it is not limited by this view. Bandara et al. [3] distinguish strategic, tactical, and operational levels in organizational management.

Business Process Management levels: [3]

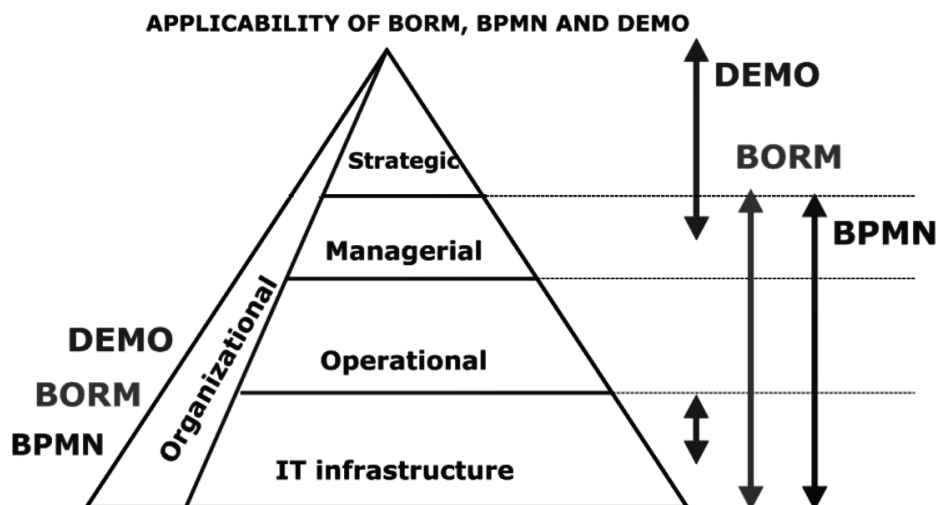
- **the strategic level** – top management support, business and IT alignment, process organisation and governance issues
- **the tactical level** – process modeling, process performance measurement and Business Process Management methodologies
- **the operational level** – technology capability, Service Oriented Architectures (SOA) maturity in the technology landscape, use of XML standards

1.1.2 Business Process Modeling

Business Process Modeling (BPM) is a key step in Business Process Management in depicting the current as-is or future to-be state of the processes

in organizations. The output of this procedure is a visual representation of the processes and it usually utilizes some modeling standard. The visualization requires modeling in multiple iterations and collection of the information through documentation exploration and discussions. [1]

Figure 1.2: Applicability of BPMN, BORM, and DEMO for Organizational Management Levels [1]



Conceptualization of a process model involves identifying and representing entities and relations between them. Aside from the BPM, the process modeling can be done for multiple other reasons and different modeling standards and methodologies were designed for them. [1]

Enterprise Modeling is a basis for Enterprise Engineering which is used for the analysis, design, engineering, and implementation of enterprises. The suitable methodology for it is the Design & Engineering Methodology for Organizations (DEMO). [1]

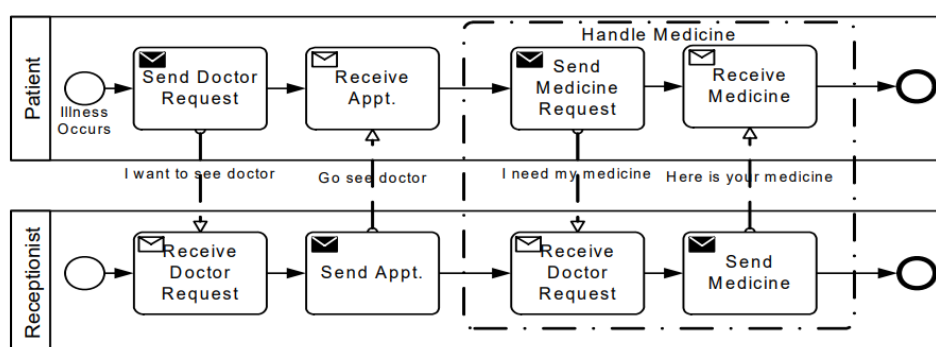
Software Engineering utilizes process modeling for a different reason. The Information Systems Development uses it for information systems implementation, based on understanding business processes. In this case, Business Objects Relation Modelling (BORM) is applicable. [1]

For the BPM that serves as input for Business Process Engineering, the reasons for process modeling are process validation, simulation, analysis, optimization; Business Process Re-engineering, and knowledge management. The useful standard for BPM is Business Process Model and Notation (BPMN). [1] The applicability of each mentioned methodology or standard is depicted in the Figure 1.2.

1.1.3 Business Process Model and Notation

In 2007, the Object Management Group created a Business Process Model and Notation (BPMN) to provide a notation that will be readable by business people and users or technical developers. It provides a bridge between design and implementation that is standardized. BPMN is also designed so that standardized formats for creating XML files are defined. Such an approach is allowing that BPMN processes can be exchanged between different software for visualisation. [4]

Figure 1.3: Example of Communication between Pools in BPMN [4]



Currently, the BPMN 2.0. is the actual version of the notation. A software can claim that it is compliant with this standard only when matching the compliance points in the specification. The software also claims compliance toward certain conformance types. The types of conformance defined in the specification are Process Modeling Conformance, Process Execution Conformance, BPEL Process Execution Conformance, and Choreography Modeling Conformance. BPEL is an abbreviation of Business Process Execution Language. To claim Process Modeling Conformance the implementation must support the BPMN core elements, process, collaboration, and conversation diagrams.

There is also a possibility to be compliant with its Descriptive, analytical and common executable conformance sub-classes. They are utilizing different subsets of modeling elements and also some additional constraints. [4] An example of a process modeled with the BPMN standard can be seen in the Figure 1.3.

1.1.3.1 BPMN 2.0. Elements

Since the main goal of the BPMN is to create a standard visual language that process modelers will recognize and understand, the key elements are available graphical shapes or icons, and their semantics. Essentially, the pro-

cess itself is modeled as a standard workflow connecting nodes with oriented edges. There are multiple types of nodes with their graphical visualization, rules, and constraints. The processes modeled as such are called orchestration processes. [4]

The main building blocks are flow objects, data, connecting objects, swimlanes, and artifacts. More info about those elements will be provided in the following sections. The process that is internal to a specific organization is called private and it is contained within a pool that sets its boundaries. When there are more participants introduced as their own processes inside the pool, the term public process is used. [4]

Once there are multiple participants, the communication between them is modeled in the collaboration processes with the use of message flows. The communicating pools can be modeled as white boxes or black boxes. The separations to pools and determination of their type depending on the modeler and the nature of the modeled process. There are more diagrams in the BPMN 2.0. like Choreography diagrams and Conversation diagrams. Those types of processes won't be covered or used in this thesis. [4]

1.1.3.2 BPMN Elements: Flow Objects

The behavior of business processes is mainly defined by three flow objects: Events, activities, and gateways. An Event is a circular symbol used in case a certain situation occurs during the course of the process. Usually, events have their triggers and results. Different kinds of them (message, timer, condition) are distinguished by the corresponding internal markers inside the open center. We are categorizing events either as Start, Intermediate, and End events or as Catching, Throwing, and Non-interrupting. [4]








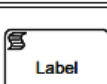
An Activity symbol is used for actual work that is performed in the organization in a process. When the Activity is atomic meaning that the work in the process is not broken down to a finer level of detail, it is called Task. The non-atomic Activities can be broken down into multiple types (collapsed, expanded, call...) of sub-processes. Both atomic and non-atomic are shaped as rounded rectangles. [4] The Figure 1.4 lists all types of Tasks.

For controlling the divergence and convergence of Sequence flows in a process, a gateway element is used. It is used either as a fork or a join of multiple paths. The type of behavior is indicated by internal markers inside the open center of a Gateway that has a diamond shape. The types of Gateway activity are Exclusive, Inclusive, Event-based, Parallel, Parallel Event-Based, and Complex. [4]

1.1.3.3 BPMN Elements: Data

Data can be defined in the diagrams with four available elements: Data Object, Data Input, Data Output, and Data Stores. The Data Object represents either

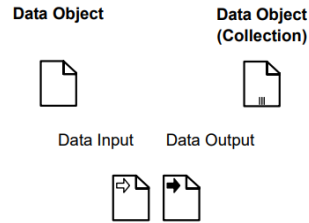
Figure 1.4: BPMN Types of Tasks [4]

Kind:	Depiction:	Specific Depiction Resolution:	
		bpmnElement:	BPMNShape Attributes:
Abstract Task		Task	None
Service Task		ServiceTask	None
Send Task		SendTask	None
Receive Task		ReceiveTask	None
User Task		UserTask	None
Manual Task		ManualTask	None
Business Rule Task		BusinessRuleTask	None
Script Task		ScriptTask	None

a singular object or a collection of objects and they provide information about what Activities are producing or requiring to be performed. The Data Input and Data Output are providing the same information for processes. All are represented by a page icon with a blank arrow inside (Data Input) or a black arrow inside (Data Output). [4] They are depicted in the Figure 1.5.

The Data Store element brings a mechanism for activities to get or set stored information that will be persistent beyond the scope of the process. The Data Store is visualized with a basic database icon and it can work as a reference appearing multiple times in the same process. [4]

Figure 1.5: BPMN Data Objects [4]



1.1.3.4 BPMN Elements: Connecting Objects

To connect multiple information in the diagram, four types of Flow Objects are used. They are Sequence Flows, Message Flows, and Associations. [4]

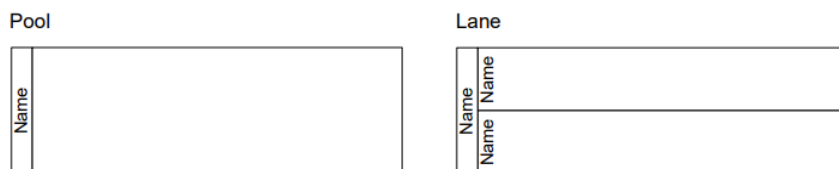
The order in which the Activities are performed is defined by a Sequence Flow that is represented by an edge with an arrowhead. The Message Flow is visualized as a dashed edge with an arrowhead and shows the flow of send and receive messages between two pools. [4]

An Association is depicted as a dashed edge or arrow. It links information, data, and artifacts with other graphical elements. The arrowhead on the edge is optional and indicates a direction of flow. [4]

1.1.3.5 BPMN Elements: Swimlines and Artifacts

Swimlines group previously described elements in two ways. A Pool represents a participant in the Collaboration diagram or a distinctive set of activities from other Pools. It is shaped like a rectangle with a separated section on the left that contains its name. As mentioned, if the Pool has no internal details it can be visualized as a black box. Lanes are sub-partitions within pools, splitting them with vertical or horizontal lines. They are used to organize and classify Activities. [4] The Figure 1.6 shows examples of how the swimlines can be modeled.

Figure 1.6: BPMN Swimlines [4]



Additional information is added to the process via Artifacts. The modeler can introduce their own set of Artifacts but the Group and Text Annotation are standardized by the specifications. A Group is used to group graphical

elements within the same category and Text Annotations provide additional information to the diagram's reader. [4]

1.2 Business Rules and DMN

Operations, constraints that are applied to organizations are called business rules. The reason for the necessity of business rules is to ensure consistency, which is important for an organization to function, and evolve, but also to deliver the products and services of the same quality. Ronald G. Ross in his book *Business Rule Concepts* [5] writes that the consistency of expressing business rules is ensured by well-defined business vocabulary.

The business vocabulary contains each noun concept and verb concept of the operational business. The goal is that all structural components are unified and unique. Even the best design behavior cannot work properly without good structure. [5]

1.2.1 Decision Management and Decision Tables

Business rules directly support business operations by guiding day-to-day business activities and shaping operational business judgments. Those two roles coordinate business processes. There is also a role to make operational business decisions that leads to decision management and decision tables. [5]

It is important to separate business rules from processes. When the business rule is externalized as a separate resource, it is possible to develop them at their own pace. The rule independence also helps to deliver better process models that are more aligned with business bringing true agility. [5]

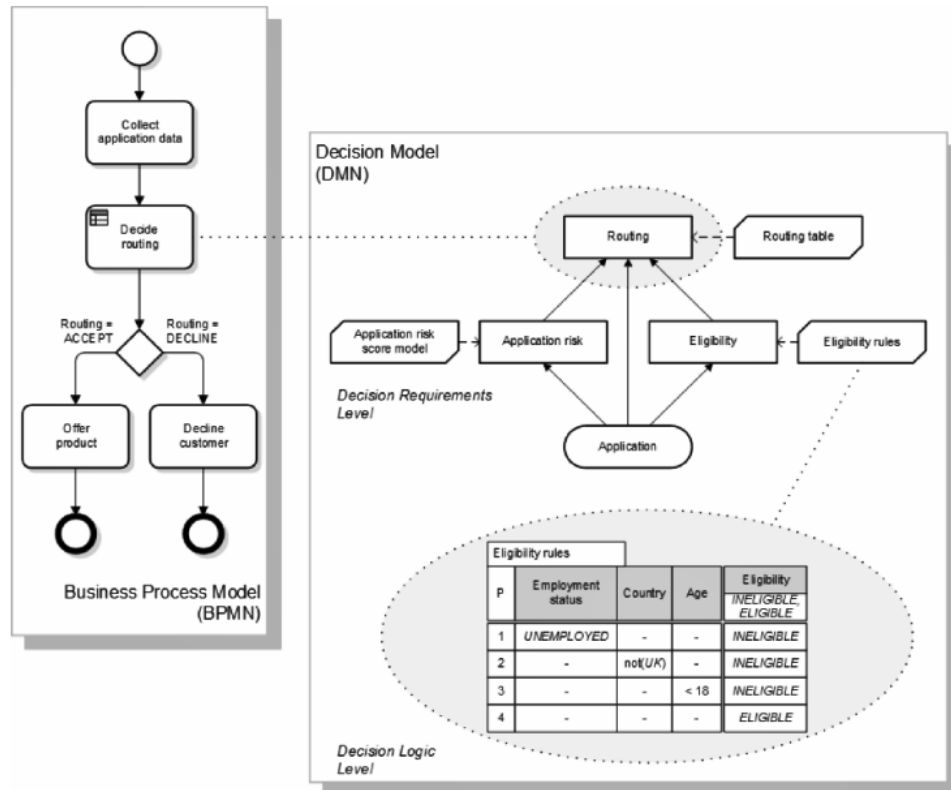
All business rules introduce exceptions to processes that are costly for the organization. The costs are not due to implementations and maintenance, but mainly because of the associated documentation, training, administration, and time for communicating them. The effectiveness lies in having less but well-defined and precise business rules. [5]

1.2.2 Decision Model and Notation

There are two different perspectives that are addressing decision-making. Business process models can describe the coordination of decision-making within business processes by defining specific tasks or activities. The decision-making takes place within these activities and tasks. The second perspective is a decision logic that can define the specific logic used to make individual decisions (business rules, decision tables, or executable analytic models). [6] The Decision Model and Notation (DMN) together with BPMN is a powerful tool for BPM as can be seen in the Figure [1.7]

There are 3 levels of conformance after which software implementation can claim that it is based on the DMN 1.3 Specification. [6]. The conformance

Figure 1.7: BPMN and DMN Modeling Aspects [6]



level 1 includes at least one Decision Requirements Diagram (DRD), decision tables, and decision logic, but those models are not executable. Any natural or unstructured language can be used for definitions of expressions. The conformance level 2 supports fully executable decision models and expressions written in Simplified Friendly Enough Expression Language (S-FEEL). The last conformance level 3 contains again fully executable decision models, expressions written in Friendly Enough Expression Language (FEEL), and the full set of boxed expressions. [7]

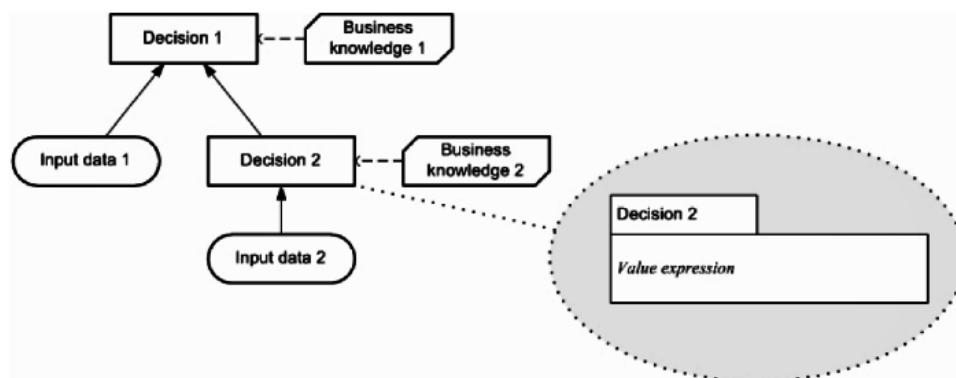
Business analysts can use decision modeling to understand operational decisions that are made on a periodic basis in an organization. Not only can DMN be used to model human decision-making or to model requirements for automated decision-making, but also for automated decision-making implementation itself. [6]

1.2.2.1 Decision Requirements Diagram

The Decision Requirements Graph (DRG) is an abstraction of decisions and their connection rules with one another. The graph can be visualized in one or more Decision Requirements Diagrams. The DRDs are modeling a domain of decision-making. It contains the most important element involved in the domain and the relations of those elements. The modeled elements are: [6]

- **Decision** element uses decision logic in boxed expressions to determine an output from several inputs.
- **Business Knowledge Model** depicts a function encapsulating business knowledge like a decision table, business rules, or an analytic model.
- **Input Data** denotes information used as an input for Decision elements.
- **Knowledge Source** represents authority for a Business Knowledge Model or Decision.
- **Decision Service** denotes a set of internally or externally invocable decisions that can be reused.

Figure 1.8: Decision Requirements Diagram with Value Expression [6]



The relations or more precisely dependencies represent requirements. An Information Requirement denotes outputs of Input Data and Decisions that are used as input to another Decision. A Knowledge Requirement is used where the decision logic of a Decision invokes a Business Knowledge Model or Decision Service. At last, an Authority requirement denotes a dependency of other elements acting as a source of knowledge or guidance. DRD also contains annotation artifacts like Text Annotation, Association, and Grouping. [6] An example of the DRD is depicted in the Figure [1.8]

1.2.2.2 DMN Boxed Expressions

Boxes expressions in DMN are graphical notations defining the logic of Decision elements and Business Knowledge Models. Together with DRDs, they are forming a complete and functional DMN decision model. [7]

The decision logic model of a business rule can be decomposed using this notation. The smaller pieces created this way can be then associated with DRG artifacts. Boxed expression can contain other boxed expressions. They are defined recursively. Each boxes expression shall contain its name that serves as a visual link. [6]

The DMN 1.3 Specification defines different kinds of boxed expressions: [6]

- boxed literal expression
- boxed invocation
- decision table
- boxed FEEL expression
- boxed context
- boxed list
- relation
- boxed function

Boxed literal expressions are boxed expressions represented by their text. There are two notational conventions provided to improve readability and avoid confusion. Typographical string literals can be represented as initialized or string literals but they should avoid commas. The specification also describes the typographical date and time literals. [6]

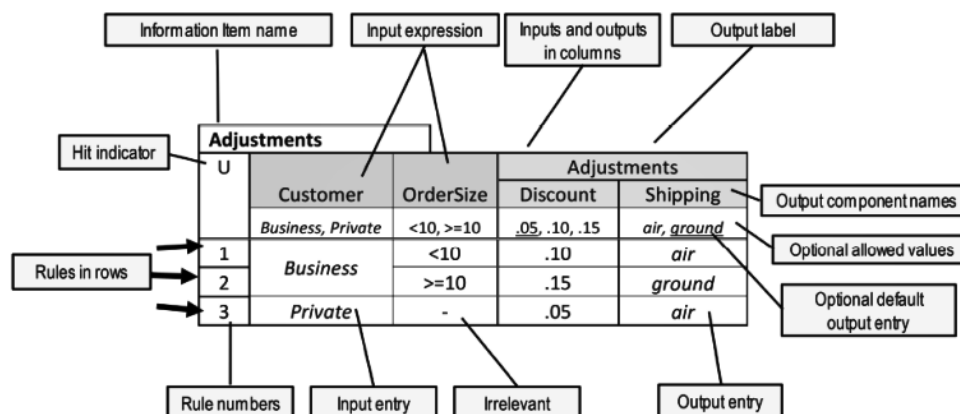
Boxed invocation is a container for parameter bindings. They provide context for business knowledge model evaluation. Such invoked model is represented by a box that contains a name and a list of bindings. Each binding is consisting of the name of a parameter and binding expression. [6]

1.2.2.3 Decision Table

A decision table is one way how to express decision logic in DMN. An example of such table can be seen in the Figure 1.9. The column in the table defines the input conditions or outputs. It should contain all inputs for the determination of the related output. [6]

The row is directly connected to one business rule that has the if-then form. There could be a case where multiple rows match the input values. For that, a specific hit policy has to be defined to specify which one will be applied. The hit policies are either single or multiple. [7]

Figure 1.9: Example of the Decision Table [6]



There are 4 types of single hit policy. Unique hit policy states that no overlaps are possible and all business rules should be disjoint. Any policy type allows that multiple rows can be applied but only if they are tied to the same output. If not, the result is undefined. [6]

Priority and First hit policies determine the output based on one matched rule. For Priority policy, the table has to be provided with a prioritized list of outputs in decreasing order of priority. The output is selected according to this list if there are multiple matches. The First hit policy determines the priority according to the order of rows. [6]

Multiple hit policies are returning a list of multiple outputs. There are 3 types with one having 5 subtypes. Output order has to be provided with a prioritized list of outputs and the output of the table is sorted according to this list. Rule order sorts its output according to the order of rows. [6]

If no order is applied the Collect hit policy is used and does not guarantee the ordering of the output list. It has 4 more variants that can return the sum, the count, the smallest, or the largest value of all the outputs. [6]

1.2.2.4 FEEL

If software states a certain conformance level, all used boxed expressions must apply to FEEL syntax specification. Conformance level 2 requires simplified S-FEEL syntax, which is just a subset of FEEL. This language is used for expressions inside the boxed expressions. [6]

It was designed to be side-effect free with syntax for a wide audience. FEEL also brings some features like a simple data model with numbers, dates, strings, lists, and contexts and three-valued logic (null, true, false). [6]

Not only the FEEL serves as a textual notation for boxed expressions but also as a robust language to represent the logic of DRGs. The main purpose

of this is to compose the semantics simply and uniformly. [6]

Aside from alphabetical characters, the language supports whitespaces and a few special characters. There exist various limitations that apply to variable and function names like reserved keywords. [7]

1.3 Blockchain Technology

A great amount of today's computing and information systems are distributed. Once the clock speed of processors got into its physical limits, hardware architecture became distributed. Both data storage and computational power perform their logic on multiple machines. Another reason for distributed systems is protection against human attacks and mistakes or geographical constraints and risks. Summarized, current computers are distributed for geography, parallelism, reliability, and availability reasons. [8]

The introduction of distributed systems brings certain coordination problems that need to be dealt with. With an increasing number of nodes in the network, the frequency of node failure is getting higher. The solution to many of those failures is their toleration with enough redundancy. [8]

Wattenhofer [8] distinguishes two types of distributed systems that do not have a central node: permissioned systems and permissionless systems. As the name suggests the permissioned systems require specific permission to access the consensus process. They are sometimes called private systems. On the other hand, permissionless systems are public for participation in the consensus process, which brings new coordination problems when achieving state replication. One of such permissionless systems is blockchain. [8]

In the financial tech industry circles, the term blockchain can be viewed as synonymous with the term state replication. Still, there are many other usages of that word depending on the context. From a technical perspective, the blockchain can be also described as a decentralized consensus mechanism, distributed shared ledger, or a data structure. [9]

1.3.1 State Replication

The main problem that blockchain solves is state replication. State replication is a key property for distributed systems and Wattenhofer [8] defines it as follows: *A set of nodes achieves state replication if all nodes execute a (potentially infinite) sequence of commands c_1, c_2, c_3, \dots , in the same order.*

The technology achieving state replication must be able to sustain both fault-tolerance and malicious behavior of its nodes. Paxos is an algorithm that can achieve a state replication even with a minority of crashing nodes in the distributed system. Practical Byzantine Fault Tolerance (PBFT) is a central protocol for asynchronous byzantine state replication. The finding of consensus is called byzantine when it can contain any node which can have arbitrary behavior. This includes actions like not sending any messages at

all or sending a wrong or false message to other nodes. Both Paxos and PBFT are permissioned systems that enable state replication under certain circumstances. The very first permissionless system was the Bitcoin network which uses decentralized consensus. [8]

1.3.2 Decentralization

One of the key characteristics of blockchain technology is decentralization. The basic idea behind decentralization is a distribution of control and authority out of a centralized node in a system. In organizations, this can lead to multiple benefits due to reduced workload from top management like increased efficiency, quicker decision making, and better motivation. [9]

The model of blockchain allows anyone to compete to become the decision-making authority. Thus decentralization arises from the absence of one central authority that governs the entire system. The decentralized consensus mechanism of Bitcoin known as the Proof of Work algorithm enables users to agree on something without the need for a central trusted third party, intermediary, or service provider. [9]

1.3.3 Bitcoin Network

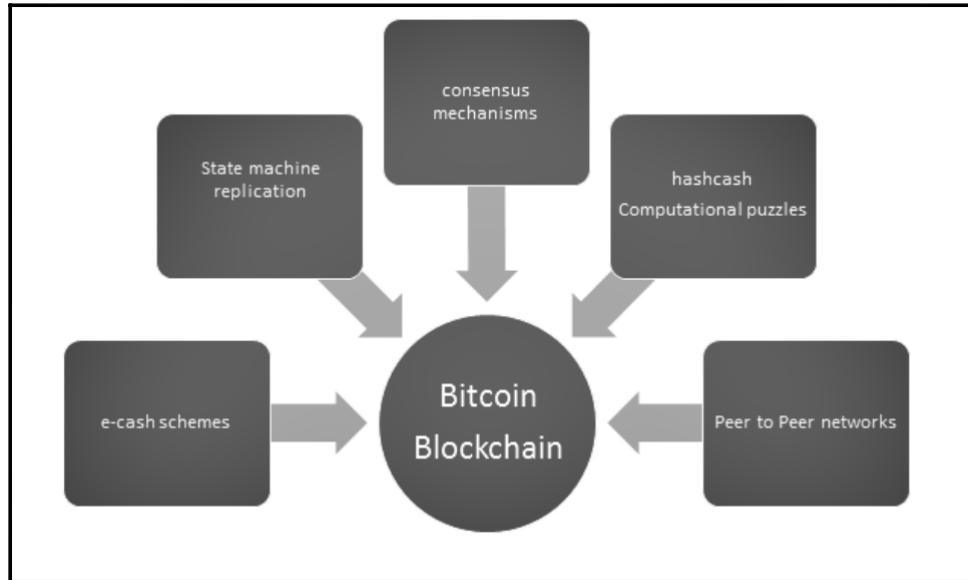
A significant reason for use of distributed systems and cryptography is to securely store transactions. In 1971 NASDAQ used distributed ledger in the first computerized stock market. Bitcoin was the first permissionless decentralized system that was able to reliably store transactions. The introduction of Bitcoin in 2008 [10] in the paper titled Bitcoin: Peer-to-Peer Electronic Cash System did not bring any new technologies. Even the term blockchain has origins in the cryptocurrency HashCash from 1997. The technologies like Proof of Work, Merkle trees, or public-key cryptography were discovered between the 1970s and early 1990s. [8] The graphic with the technologies behind the Bitcoin can be seen in the Figure 1.10.

The definition of the Bitcoin network provided by Wattenhofer [8] is this: *The Bitcoin network is a randomly connected overlay network of a few thousand nodes, controlled by a variety of owners. All nodes perform the same operations i.e., it is a homogenous network and without central control.*

The CAP Theorem states that it is impossible for a distributed system to provide the following characteristics at the same time:

- **Consistency** - *All nodes in the system agree on the current state of the system.* [8]
- **Availability** - *The system is operational and instantly processing incoming requests.* [8]
- **Partition Tolerance** - *It is the ability of a distributed system to continue operating correctly even in the presence of a network partition.* [8]

Figure 1.10: Technologies behind Bitcoin [9]



Bitcoin sacrifices consistency in favor of availability and partition tolerance. But it achieves eventual consistency which guarantees that the state replication is sooner or later gained even if there could be temporarily no consensus in the system. [8]

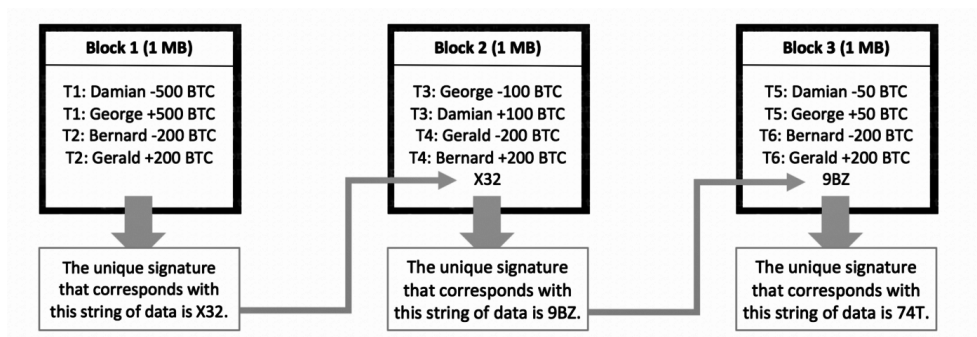
1.3.4 Anonymity and Irreversibility of Bitcoin

The aspects of anonymity and irreversibility of Bitcoin and blockchain, in general, arise from the uses of cryptography in different parts of the distributed filesystem. Anonymity in the system is achieved with the use of public-key cryptography and the irreversibility through hashing and verification of the blockchain's consistency. Still, various techniques have been developed and applied to trace the flow of transactions throughout the network and link them to actual nodes making the system pseudo-anonymous. [9] The Figure 1.11 depicts a representation of the blockchain data structure.

Bitcoin network is dependent on the use of asymmetric cryptography, hash functions, and digital signatures. The user of Bitcoin can generate multiple private keys that uniquely identify the owner of funds of an address. An address is a term interchangeable with a public key which is derived from the private key and identifies a recipient of a transaction. [8]

The cryptographic hash functions like SHA256 and RIPEMD160 are widely used. The other cryptographic tools that Bitcoin uses are Merkle trees and the Elliptic Curve Digital Signature Algorithm. [8]

Figure 1.11: Graphic Representation of Blockchain [11]



1.3.5 Unspent Transaction Output

Unspent Transaction Output (UTXO) is a concept of Bitcoin's distributed ledger. It is defined in the Figure 1.12. An output is a tuple of an amount of coins and most commonly a valid signature. The signature is associated with the private key of an address and it can be replaced or enhanced by any other spending condition. The output exists in an unspent state or a spent state and every output can be spent only once. The sum of unspent outputs associated with an address is a balance of the user's funds. The shared state of Bitcoin is consisting of a set of Unspent Transaction Outputs and additional global parameters. [8]

Figure 1.12: Definition of the Basic UTXO [12]

<i>Primitive types</i>		$txid \in \text{Txd}$	transaction id
		$ix \in \text{Ix}$	index
		$addr \in \text{Addr}$	address
		$c \in \text{Coin}$	currency value
<i>Derived types</i>			
$tx \in \text{Tx}$	=	$(inputs, outputs) \in \mathcal{P}(\text{TxlIn}) \times (\text{Ix} \mapsto \text{TxOut})$	transaction
$txin \in \text{TxlIn}$	=	$(txid, ix) \in \text{Txd} \times \text{Ix}$	transaction input
$txout \in \text{TxOut}$	=	$(addr, c) \in \text{Addr} \times \text{Coin}$	transaction output
$utxo \in \text{UTxO}$	=	$txin \mapsto txout \in \text{TxlIn} \mapsto \text{TxOut}$	unspent transaction outputs
$b \in \text{Block}$	=	$tx \in \mathcal{P}(\text{Tx})$	block
$pending \in \text{Pending}$	=	$tx \in \mathcal{P}(\text{Tx})$	pending transactions
<i>Functions</i>			
		$txid \in \text{Tx} \rightarrow \text{Txd}$	compute transaction id
		$ours \in \text{Addr} \rightarrow \mathcal{B}$	addresses that belong to the wallet
<i>Filtered sets</i>			
		$\text{Addr}_{ours} = \{a \mid a \in \text{Addr}, \text{ours } a\}$	
		$\text{TxOut}_{ours} = \text{Addr}_{ours} \times \text{Coin}$	

An input is a tuple that consists of a reference to the previous output

and most commonly signature that proves that the creator of the transaction is able to prove ownership of the referenced output. The transfer of coins from spenders to recipients is represented by a data structure. This structure consists of inputs that are removed from UTXO and new outputs that are added to UTXO. [8]

1.3.6 Proof of Work Algorithm

Due to byzantine nodes, there is a possibility that multiple transactions attempt to spend the same output. This situation is called double-spend. The shared state becomes inconsistent since only one transaction can be valid. Because of that, conflict resolution functionality is needed in the blockchain network. This mechanism decides which of those conflicting transactions should be accepted to achieve eventual consistency. [8]

Proof of Work (PoW) is a mechanism that allows a party to prove to another party that a certain amount of computational resources has been utilized for a period of time. A function $F_d(c, x) \rightarrow true, false$, where difficulty d is a positive number, while challenge c and nonce x are usually bit-strings, is called a Proof of Work function if it has following properties:

- $F_d(c, x)$ is fast to compute if d , c , and x are given.
- For fixed parameters d and c , finding x such that $F_d(c, x) = true$ is computationally difficult but feasible. The difficulty d is used to adjust the time to find such an x . [8]

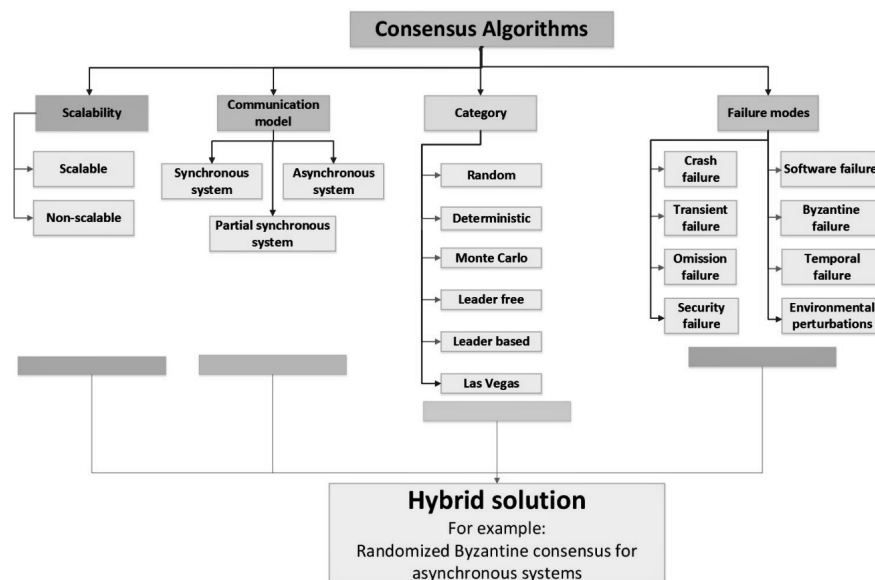
The so-called miner that finds a valid nonce for its PoW function can accept transactions to the memory pool and broadcast them inside a block. By doing that, he or she receives a reward transaction. This block consists of a list of transactions, nonce, and a reference to its previous blocks. This creates a Blockchain – the longest path from the root block. This structure functions as a consistent transaction history on which everyone eventually agrees. The value of the reward the miner receives is expected to be close to the energy and infrastructure used in the PoW mechanic. [8]

1.3.7 Other Consensus Algorithms

There is some inherent characteristic of the blockchain that is difficult to avoid. Bitcoin network is difficult to scale because it relies on confirmation in the blockchain. Transaction history is stored on every node and reconstructed from the root node. Another problem viewed by a part of the blockchain community is the large energy consumption of permissionless blockchains that are wasted because of the block validation happening in PoW. [8]

Other consensus algorithms are trying to solve the consumption problem. The most known of them is a Proof of Stake (PoS) algorithm and it will be

Figure 1.13: Categorization of Consensus Algorithms [13]



described in the next section. There are also enhanced variants of PoS like Delegated Proof of Stake or Leased Proof of Stake that are less common. Other ways of achieving consensus in public blockchains are Proof of Space and Proof of Retrievability which determines the power of the miner by his or her free disk storage. The already mentioned PBFT algorithm is suitable for permissioned distributed systems like Hyperledger Fabric. The same suitability also applies to Proof of Elapsed Time. [14] The Figure 1.13 contains the categorization of the consensus algorithms.

1.3.8 Proof of Stake Algorithm

Proof of Stake is trying to avoid this consumption by removing the principle of rewards received based on random miner solving a cryptographic puzzle. Instead of that, the rewards are distributed proportionally to the stake of owned funds in the system. [8]

Chain-based PoS introduced lottery tickets where accounts receive them by their economic proportion. Then a pseudorandom lottery winner is selected to add a block to the blockchain. This opens the problem that some actual winners do not produce the new block in time. A voting phase was added where the winner only proposes the new block and a committee then votes if the block will be accepted. Still, by its nature, the PoW is prone to certain attacks, like nothing at stake attacks or long-range attacks, that are possible in the PoS. They are theoretically possible in both algorithms but the enormous

amount of computing power to achieve successful attacks goes against the attacker in the PoW algorithm. [8]

1.4 Smart Contract

A smart contract can be viewed as a small decentralized program. The concept of smart contract was introduced independently from blockchain and does not need it to run. The security advantages of blockchain lead to a situation that it is becoming to be used as a decentralized execution platform for smart contracts. [9]

It is possible to implement some business logic and a limited amount of data in the form of a smart contract. But the nature of blockchain technology brings some constraints to those programs. The following section will be exploring smart contracts, their characteristics, variations, and limitations.

1.4.1 Codebase of Bitcoin

The functionality of Bitcoin can be extended beyond the transfer of funds if enough of the miner community decides. This is possible due to a custom scripting language called Script which supports many functions and evaluates to true or false values. To avoid DoS attacks, some of those functions were disabled. The Script is currently kept simple and does not have the complexity of other programming languages. The code is executed when a new transaction is validated. [8]

The Script is lacking loops to avoid any undesirable long-running codes in the network. Many words, commands, or functions known as Opcodes were also excluded throughout its existence due to found bugs. Therefore it is not a Turing complete language. The Opcodes categories (constants, flow control, stack, bitwise logic, splice, arithmetic, cryptography, and lock time) together form standard transaction scripts. The most used payment transaction types are Pay to Public Key Hash, Pay to Script Hash, and Pay to MultiSig. Null Data transaction type enables to store arbitrary data on the blockchain up to 40 bytes. [9]

1.4.2 Concept of Smart Contract

The term smart contract was first used by Nick Szabo in 1993 in an article discussing tools that can help small businesses to operate in multinational markets. [15] Szabo further defines the term in the glossary as follows: *A set of promises, including protocols within which the parties perform on the other promises. The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics, thus these contracts are "smarter" than their paper-based ancestors. No use of artificial intelligence is implied* [16]

Currently, it is strictly tied to blockchain which ensures the correct execution of an agreement between two or more parties. Special programming languages for writing smart contracts allow the implementation of business logic encoded in the blockchain network. Blockchain then acts as a mediator between sides of an agreement performing an agreed-upon action. It is a code that has associated storage in the blockchain and can execute complex logic. Smart contracts cannot be altered once they are deployed otherwise the concept is violated. There is one exception in a form of mutable storage with which updates are possible. [8]

1.4.3 Programming Languages for Smart Contracts

The smart contract is generally written in a higher programming language like Solidity and compiled down to Turing complete low-level programming language. [8] The programming languages were often based on existing languages for standard software development. The object-oriented paradigm of Python is used in Vyper and Serpent languages. [9]

On the other hand, Plutus Tx utilizes Haskell, which is a functional programming language, as the basis. It is basically a library for Haskell and compiled down to multiple lower layers. The compilation pipeline ends at Plutus Core which runs on the blockchain. [17] Nowadays there are also attempts to enable compilation to the blockchain of generally known and used languages like JavaScript. Still, Solidity is most popular for writing smart contracts and it became almost a standard for Ethereum blockchain. [9]

1.4.4 Smart Contract Templates

Most of the current blockchain use cases are in the financial industry. There is an idea to create domain-specific templates to make the conduction of smart contracts easier for less technical users. Domain-specific languages are nothing new to the financial industry and they are developed with limited expressiveness and specific areas of interest. [9] One of such domain-specific languages is Marlowe and it is built as a platform for decentralized finance (DeFi). [18]

The idea of a domain-specific programming language for writing smart contracts is further expanded in a form of graphical languages. On platforms like Caterpillar or DasContract, it is possible to define semantics with graphic elements that are then processed into smart contracts and possibly deployed to the blockchain. [9]

Caterpillar is a system built on top of the Ethereum blockchain and provides modeling tools to develop smart contracts. [19] DasContract is a similar system developed by CCMi Research at the Czech Technical University in Prague. It aims to provide a visual language and platform that can generate smart contracts written in multiple programming languages like Solidity or

Plutus Tx. The DasContract is based on DEMO, BPMN, and UML modeling standards and languages. [20]

1.4.5 Oracle

Since the technologies behind blockchain networks assure that the contracts and transactions in it are not corrupted, the way how to read external data is limited. The so-called Oracle is an interface that delivers external data to smart contracts in secured channels. [9]

Oracles would be able to deliver various types of data to blockchain: weather reports, real-world news, corporate actions, and data from the Internet of Things devices. Those data cannot be altered to assure trust. The centralization is also avoided by introducing distributed mechanisms to decentralized Oracles. [9]

The Oracles must be trusted entities. Even then, they are posing a security risk to the system that cannot be disregarded. There is also a concept of Smart Oracles which allows the execution of smart contract code by the Oracles. [9]

1.4.6 Smart Contract Platforms

This thesis will be mainly focused on the blockchain projects Cardano and Ethereum that also provides the platforms for smart contracts. There are other platforms supporting self-executable codes. Notable public blockchain systems are Nem, Stellar, Waves, Neo, EOS, and RSK. Except for the RIDE language of the Waves platform, every other utilizes existing language for smart contracts. [21]

In terms of permissioned systems, Hyperledger Fabric is a well-established blockchain platform that uses Java to write codes deployed to Docker containers. It uses the PBFT algorithm as a mechanism to achieve consensus. Other private blockchains to mention are Corda, Tendermint, and Quorum. [21]

1.4.7 Decentralized organizations

A smart contract or a set of smart contracts can be viewed as real human organizations with people and protocols. Such a model is called Decentralized organization and strictly relies on human input to execute business logic. [9]

The Distributed autonomous organization (DAO) also implements governance and business logic rules on top of a blockchain that are fully automated. Such a model can be applied in the real-world legal system in theory, but currently, they have no legal status and still have some technical problems. [9]

The first implementation of DAO was the Ethereum platform. *The DAO* was a venture capital fund project raising 168 million USD. A bug in the code of *The DAO* caused hacking attacks on the blockchain. This opened up a debate on the security and the quality of DAOs. Currently, formalization and standardizations of smart contracts are trying to avoid those incidents. [9]

sion is the degree of novelty to which software is new to the world. The second one is the amount of complexity and coordination effort that needs to be invested to produce value with the technology. The adoption of applications is then divided into 4 phases according to the levels of those dimensions. They are Single Use, Localization, Substitution, and Transformation. [22] You can see the examples for TCP/IP and blockchain in the Figure 1.14.

1.5 Ethereum Platform

Ethereum is a distributed state machine that allows running arbitrary computer programs on top of a blockchain. It was the first blockchain to introduce Turing complete language for writing smart contracts. The concept of this system was proposed by Vitalik Buterin in 2013. [9]

The actual Ethereum platform is still significantly evolving because the propositions made at the beginning are still in development. The first implementation was deployed in 2015 and it was called Frontier. Since then, 14 big releases were introduced. The initial DAO fork, Tangerine whistle, and Spurious Dragon code upgrades were a necessary response to DAO and Denial of Service (DoS) attacks on the network. At first, the Ethereum network was working with the PoW algorithm and the long-term migration to PoS is currently in process. [23]

1.5.1 Merge into Proof of Stake

The proposed PoS algorithm was a response to the energy consumption of the Bitcoin mining PoW consensus mechanism. Still, the Ethereum network was first secured by the PoW because it is less complicated and easier to implement. [24] The transition to PoS is currently in progress. To prevent forking of the established blockchain working with PoW, the so-called difficulty bomb was introduced that exponentially increases the computation required to write a block. [25]

The migration to PoS was frequently postponed and with that, the difficulty bomb needed to be delayed also. The Staking deposit contract upgrade was deployed in 2020 and it introduced staking where nodes could become validators of transactions after depositing some resources. This contract worked on the main Ethereum blockchain network called Mainnet but it was the first step toward the PoS network. [23]

There are three phases of which the transition to PoS is consisting. The deployment of Beacon Chain happened at the end of 2020 but only as a consensus layer in the network. The Mainnet was still handling the accounts and smart contracts. The Merge of those two blockchains is planned for 2022. Shard chains update is planned for the future to improve scalability and capacity. At first, the newly emerged blockchain was supposed to be renamed

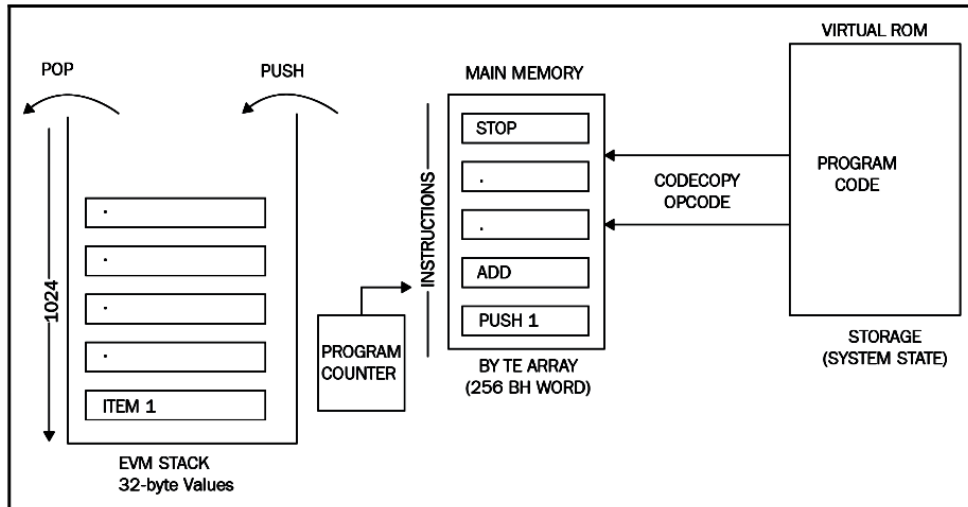
Ethereum 2.0. This idea was left due to possible misunderstandings and scams with other forked blockchains. [26]

1.5.2 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a Turing complete machine that transforms the system state from one state to another. It is a virtual state machine executing bytecode instructions similar to the Last in First Out queue. One of the goals is to have the EVM instruction set natively in CPUs which would make the smart contract execution faster and more efficient. [9]

To prevent DoS attacks through infinite loops, each instruction requires an amount of so-called gas. The size of this resource is controlled by the EVM based on the consumption of resources in the system. EVM measures the consumption of gas with its accounting mechanism. [9] The gas costs are covered with the Ethereum’s cryptocurrency called Ether. [25]

Figure 1.15: Structure of EVM Operations [9]



EVM does not have access to any external resources like networks or filesystem. There are two types of storage available to contracts. The memory type is a byte array cleared after code execution. Storage type is permanently stored on the Ethereum blockchain. The EVM supports exception handling. The execution halts and returns an error code in the case of not having enough resources or invalid instruction. [9] The Figure [1.15] shows the structure of the EVM operations.

1.5.3 Solidity

Solidity is a high-level programming language created by Dr. Gavin Wood for writing smart contracts that target the EVM. [25] This curly-bracket language is based on the object-oriented paradigm and it is similar to C++, JavaScript, or Python. It is a statically typed language that supports inheritance, libraries, and complex user-defined types. [27]

The smart contract written in the high-level Solidity language is then translated into Solidity inline assembly. Aside from that, the compiler generates Application Binary Interface (ABI). This is a JSON file describing the deployed contract and its smart contract functions. [27] The example of the Solidity code is depicted in the Figure 1.16.

Figure 1.16: Code Example of the Solidity

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.4;
3
4 error NotEnoughFunds(uint requested, uint available);
5
6 contract Token {
7     mapping(address => uint) balances;
8     function transfer(address to, uint amount) public {
9         uint balance = balances[msg.sender];
10        if (balance < amount)
11            revert NotEnoughFunds(amount, balance);
12        balances[msg.sender] -= amount;
13        balances[to] += amount;
14        // ...
15    }
16 }
```

1.5.4 Contract Structure and Data Types

The smart contract written in Solidity has a similar structure to classes of other object-oriented languages. The contract can contain declarations of state variables, functions, function modifiers, events, errors, structure types, and enumeration types constructs. [27]

As written, Solidity is statically typed language so each variable needs to be specified before compilation. Data of such declared variables are permanently stored on the blockchain given by the nature of the technology. [27]

Aside from the basic data types like boolean or integer, the language also defines address type for Solidity 20 bytes addresses. The reference type, in which its values can be modified through multiple different names, is consisting of structs, arrays, and mappings. The data area where the reference type has

its storage needs to be provided with the type definition. The data are is either memory, storage, or call data which contains the function arguments. [27]

The mapping types can be viewed as hash tables where all keys are initialized to a default value, but the keys are not stored in the mapping type. The value can be looked up with keccak256 hash. It is not possible to iterate over mappings because the keys cannot be enumerated. But it is possible to implement data structure on top of mapping type and iterate over that. [27]

1.5.5 Solidity Functions

Solidity functions are usually defined inside a contract and have public, internal or private visibility towards other contracts. Internal function calls, direct or recursive, are translated into simple jumps inside the EVM. Because of that that the current memory is not cleared and passing memory references to internally-called functions is more efficient. Still, excessive recursion should be avoided since there are 1024 available slots and each internal function uses up one. [27]

The functions can be also called externally using the `this.g()` and `c.g()` notation. This results in the message call being used without direct jumps inside the EVM. Because the actual contract has not been created yet, the function calls cannot be used in the constructor. The external calls have to be always used for other contacts and all function arguments are copied to memory. Those calls are part of the overall transaction but they are not creating one. An amount of gas can be also sent with a function call. [27]

1.5.6 Function Modifiers

The semantics of functions can be altered in a declarative way using function modifiers. They are applied to functions in a form of a list separated by whitespaces and the modifiers are executed in the presented order. The values of arguments are passed to modifiers explicitly at the point of invocation. Modifiers cannot be overloaded but it is possible to override them but only with the use of `virtual` and `override` keywords. [27]

The `constant` and `immutable` keywords can be declared for state variables which then cannot be modified after contract construction. The value of constant variables has to be fixed at compile-time, but the immutable variables can be assigned at construction time. The allowance of side effects on the memory allocator should lead to the construction of complex objects like lookup tables. Currently, only strings and value types can be immutable. [27]

When the function is marked as `view`, they promise not to modify the state. This includes writing to state variables, creating other contracts, sending Ether, using low-level calls, or calling a function that is neither declared `view` nor `pure`. The `pure` declaration promises that the state won't be read from and modified. The reading from immutable variables can be an oper-

ation that is not pure because it is not possible to evaluate such a function without knowledge of the current blockchain state. [27]

1.5.7 Advantages and Disadvantages of Solidity

There are some advantages of using Solidity over other smart contract programming languages. Since it was heavily influenced by C++, the constructs and programming style can be recognizable to new programmers. Also, the support for inheritance properties within its constructs is borrowed from object-oriented features. The generated ABI provides a possibility to determine if the returned data is right and valid. This also facilitates type-safe functions within a contract. [28]

In terms of disadvantages, Solidity is not as old as the general programming languages so there is less documentation and therefore a possibility of anti-patterns within the code structures. Still, this aspect is better when compared to other smart contract programming languages. The youth and limited expressiveness of the language can be an obstacle when learning the language. Also, the data can be brought into the blockchain only through transnational operations so managing some data actual can be costly. The nature of blockchain restrains updates of the deployed code but this will be a common problem for all similar languages. [28]

1.6 Cardano Platform

Cardano is a blockchain platform founded in 2015 by Charles Hoskinson and Jeremy Wood. It is one of the new generations of blockchain which allows running arbitrary computer programs on top of a blockchain. The core principle of Cardano is identifying and overcoming cascading disruptions, perturbations having a ripple effect on a system. [29]

The company behind Cardano is Input Output HK (IOHK) and it describes the network as the third generation of a blockchain. This distinction is a critique of the scalability aspect of Ethereum. The core concepts of Cardano are scalability of transactions, self-sustainability of the project, and interoperability supported by cross-chain transfers, multiple token types, and commonly used smart contracts languages. [30]

1.6.1 Cardano Roadmap

The first version of Cardano Mainnet was deployed in 2017 and the following phase of the project was called the Byron era. Its cryptocurrency called Ada was introduced with a fully working blockchain secured by the Ouroboros consensus protocol. Ouroboros is a PoS algorithm built on academic research that came up with a mathematically-proven secure mechanism. [31]

The following phase called the Shelley era focused on optimizing the decentralization of the network where the majority of nodes would be running less by the company's nodes. It also introduced delegation and a reward system to drive stake pools and community adoption. This system of incentives was designed using game theory. [31]

The Goguen era was focused on the integration of smart contracts into the blockchain. The peer-reviewed research and high-assurance development brought the ability to build decentralized applications. This resulted in the creation of the Plutus programming language. Plutus is a functional programming language based on Haskell that supports the development of both on-chain and off-chain code. The domain-specific programming language Marlowe for financial contracts which is built on Plutus was also introduced. [31]

Future eras called Basho and Voltaire will focus on delivering the promises of true decentralization and governance. The Basho era will introduce side-chains with the ability to support and switch between UTXO and account-based models. The Voltaire era will introduce voting by the stakeholders in the existing staking and delegation process, and a treasury system of transaction fees distribution. After that Cardano will be no longer under IOHK's management. [31]

1.6.2 Ouroboros Consensus Protocol

The consensus mechanism of Cardano is based on the PoS mechanism that was extended with rigorous security guarantees. The security aspects of the algorithm are comparable to the Bitcoin properties with much better qualitative efficiency advantages over numerous blockchains based on proof of physical resources. [32]

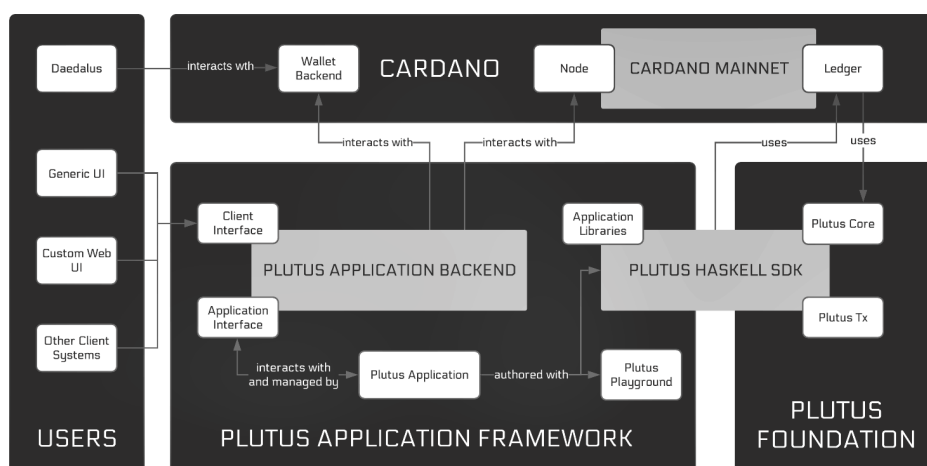
The challenge to designing a secure PoS mechanism is finding out the leader election process. Wrongly introduced entropy may result in grinding vulnerability. In this malbehavior, the sequence to bias the leader election is found and exploited. The concept focuses on the persistence and liveness of honest nodes so they are adopted and become immutable. It should be well resistant against double-spending attacks, transaction denial attacks, 51% attacks, nothing-at-stake, desynchronization attacks, and others. [32]

The main principle is that a snapshot of the current set of stakeholders is taken in epochs which are regular intervals. In each epoch, a set of randomly selected stakeholders form a committee. The randomness is created by the multiparty computation. This set participates in a coin-flipping protocol that determines the slot leader. The slot leader has then the right to mine a block. The mechanism was subsequently enhanced in the Byzantine Fault Tolerance, Praos, Genesis, and Hydra implementations. [32]

1.6.3 Plutus Platform

Plutus Platform is a platform for writing, testing, and maintaining Plutus smart contracts that can be then deployed to the Cardano network. The platform consists of the Plutus Foundation for writing the trusted kernel of code and its execution on the blockchain and the Plutus Application Framework for writing Plutus applications. [33] The architecture of Plutus Platform is depicted in the Figure 1.17.

Figure 1.17: Architecture of the Plutus Platform [33]



The actual smart contract running on the blockchain is Plutus Core. It is a programming language that is a variant of the lambda calculus. It is used for its simplicity, determinism, and cost control of code execution. The lambda calculus allows Cardano to have a formally verified evaluator that is easily targeted by compilers of functional programming languages. Plutus Tx language then serves as a higher programming language based on Haskell that is developed for writing smart contracts. [33]

Plutus Application Framework is a framework for developing distributed applications with the Cardano blockchain. It contains libraries, selection of use-cases written with the framework, and a web-based playground for learning and writing basic applications called Plutus Playground. [33]

1.6.4 Extended Unspent Transaction Output

One of the goals of Cardano is to provide a blockchain platform that could process multiple currencies. The standard UTXO model is strictly limited to Ada accounting. Because of that, Cardano blockchain uses an extension of UTXO that would also allow smart contract support. [12]

The Extended UTXO consists of two components. The first component is an extension of data in transactions, and an extension of the processing scheme performed by nodes. The second component is the off-chain extension of the wallet backend that coordinates the execution of on-chain code. On-chain code is the only part of the code compiled into Plutus Core. [12]

The possibility of smart contracts in UTXO is added through Plutus scripts. Plutus scripts are Plutus Core expressions stored on the blockchain ledger which have their addresses and can hold coins until certain conditions and processing. There are three types of Plutus scripts. [12]

The Validator script is carried by a transaction that spends funds from a script address. It serves as a validation of the spending and for that, it returns a boolean value. The Data scripts hold information about the state of the smart contract and it is carried by a transaction that is paying to a validator script. The last type is the Redeemer script is carried by a transaction that is spending funds from a script address. The Redeemer script represents an action that will be taken by the wallet like collecting funds or refunding. [12]

1.6.5 Plutus Tx and Functional Principles

Plutus Tx utilizes template metaprogramming support called Template Haskell which allows the development of source code generated from templates by Glasgow Haskell Compiler (GHC). Because of this Plutus Tx can execute its functions at compile time but to do that the arguments must be known at compile time and the function must be pure. A functional paradigm is beneficial when improving memory and time using this approach. [33]

Figure 1.18: Code Example of the Plutus Tx

```
1 {-# INLINABLE plusOne #-}
2 plusOne :: Integer -> Integer
3 plusOne x = x 'addInteger' 1
4
5 {-# INLINABLE myProgram #-}
6 myProgram :: Integer
7 myProgram = let
8     plusOneLocal :: Integer -> Integer
9     plusOneLocal x = x 'addInteger' 1
10
11     localTwo = plusOneLocal 1
12     externalTwo = plusOne 1
13     in localTwo 'addInteger' externalTwo
14
15 functions :: CompiledCode Integer
16 functions = $$ (compile [|| myProgram ||])
```


The Haskell data types and pattern matching can be used normally in Plutus Tx. Similar use can be applied for type classes but only for a subset of standard type classes. That is because their class methods must be inlinable so that the GHC compiler can inline the compiled function directly into the code. The module `PlutusTx.Prelude` is a replacement for the standard Haskell Prelude, but both of them can be used. [33] The example of the Plutus code is depicted in the Figure 1.18.

1.6.6 Plutus Tx Monads and Lifting

To preserve the functional principles in Haskell but also to handle side effects in pure functions the concept of monadic computation was introduced. For example, it allows communication with input/output devices or exception handling. [34] The Plutus Tx uses monads in the type class `MonadWallet`. This type class and any monad which implements it represents the off-chain code. [12]

PlutusTx also allows developers to generate code dynamically. This is useful when applying a function to an argument at runtime. To do it, the programmer needs to write the function as a static code and lift it using a `liftCode` function. The runtime in this case means the runtime of the main Haskell program and not the runtime of a Plutus Core program on-chain. [12]

1.6.7 Common Weaknesses and Optimization Techniques

There are some potential sources of vulnerabilities in applications in terms of Common Weakness Enumeration. The double satisfaction weakness can occur when payments are not correctly identified and overlap with other payments. To solve this problem it is recommended either to make the outputs unique or that transactions are allowing only one script. [30]

Another vulnerability can be found when exceeding certain hard limits defined by the protocol. Resources like transaction size, block size, UTXO size, and script execution units have their hard limits. This means that some data can grow in an unbounded way over time and reduce the amount of space available for other uses. There are many solutions to that like careful testing, bounding data usage, or reducing script size costs through reference inputs. [30]

Developers can use profiling to identify problem areas, but there are other techniques for optimization: using strict `let`-bindings to avoid recomputation, preferring higher-order functions, eliminating common sub-expressions, and using `error` keyword for faster failure. Following certain practices can make code secure and efficient, but all those recommendations make development much more difficult. [30]

1.6.8 Advantages and Disadvantages of Plutus

Similar advantages and disadvantages apply both to the Plutus and Cardano. They are backed by a strong team, financial resources, and methodology based on academic research, prototyping, and technical specifications. Because the Plutus Tx language is based on the Haskell language the constructs can be mathematically verified. [31] The performance can be also a significant advantage if good practices and principles are followed. [30]

But the academic research phase could be also proven as a weakness. Many of the promises are taking longer than expected and the functional paradigm can be less accessible for its developers. Because Cardano has a big competition in terms of the development of general-purpose smart contract programming language it can be less significant without a community that uses it. [35]

1.7 Chapter Summary

The chapter Theoretical Foundations established base knowledge for the generation of DMN business rules from smart contracts. All necessary elements were explored. First, the Business Process Management and Modeling were presented since they are the underlying principle of DMN notation. BPMN was also explored because together in combination with the DMN, it can significantly facilitate and improve the way that smart contracts are conducted.

The DMN standard was chosen for the formalization of business rules. The core element of the notation is a Decision Table which is one way of expressing the decision logic. It consists of input and output definitions, and conditions in a form of boxed FEEL expressions. The important output determination called Hit Policy is also described.

Currently, the blockchain networks serve mainly as a platform for cryptocurrencies. But the business, academia, and governments are exploring the potential of this technology for implementing DAO. this would be possible due to self-executing smart contracts stored on the blockchain system.

Two further platforms for the development and execution of smart contracts were laid out. Ethereum was the first project that provided the platform for smart contracts. Today its programming language Solidity is the leading solution for the creation of smart contracts. Cardano is a blockchain that chose a different approach that would improve the scalability of Ethereum. The Plutus Platform was created to support the goals of Cardano. In the following chapter, both those programming languages will be compared in terms of transformation from DMN business rules.

Business Rule as Smart Contract

The following chapter moves from collecting a theoretical basis to the actual exploration of ways to represent DMN business rules as smart contracts. It was described that for this translation, templates are generally used. This approach was selected also in this practical section of the thesis.

In the first part of this chapter, the components of the DRG are analyzed since they are the inputs in the process of generation. The suitability of those components for general execution and working in blockchain differs. Because of that, they might be excluded from the process.

The second part focuses on depicting previous components in multiple templates. Those templates will be represented in simplified pseudo-code excerpts. Examples of those codes in their target programming languages can be then found on the enclosed CD attached to this thesis. This part is also divided into the exploration of generation to Plutus Tx and generation to Solidity smart contracts. They are also compared at the end of the chapter.

2.1 Representation of DMN Business Rule

The components of the DRG or DRD models and ultimately the DMN business rules are stored and represented in an XML format. It then depends on the modelers how they display the data and alter their implementation but the elements should still follow the rules of the DMN specification. [6]

The modeling tools also manage the integration of business rules with BPMN notation since DMN itself cannot represent the full logic of processes in the organization. This applies more to the depiction of the process in a smart contract. The integration with BPMN won't be described in this chapter but will be outlined and used in the Proof of Concept part of the thesis.

2.1.1 DRD Elements

As described in the theoretical foundations the DRD model can consist of a decision, business knowledge model, input data, knowledge source, or decision service. Those elements are then connected with certain associations. The core component that describes the decision logic is the decision element with the decision table and the exploration will be solely focused on it.

Most of the modelers that I have worked with do not necessarily use the other elements for the description of execution logic. The decision can also use the literal FEEL expression to describe the logic. When using the DMN notation with BPMN models, the same can be handled by script tasks. The generation of decision tables is more complex and suitable for the goals of the use of visual language.

2.1.2 Decision Table

The decision table consists of a header defining the inputs and outputs and a body describing the rules determining the output value. There is also a column for annotations which serves for notes. Annotations are not generally reflected in the decision logic.

Both input and output definitions can be labeled with a name. The actual variables and expressions used in the logic are defined in different attributes and elements of the decision table in XML. The output is bound with a variable in the name attribute. The input is defined in the expression input element and can hold not only bound variables but also literal expressions for example in the FEEL language.

The DMN provides a number, string, boolean, days and time duration, years and months duration, time and date as FEEL data types. [6] The inputs and outputs can have predefined values for example for string but the usage of the valid formats is up to the person designing the model. Rules can hold values in those data types with certain conditions. These conditions specify the relations to the values in rules. Usually, this is possible for inequality comparison against number and date-time data types.

2.1.3 Hit Policy

The hit policy has to be also specified possibly with Collect aggregation if needed. By default, the outputs satisfying the rules should not overlap as the Unique hit policy states. For the determination of output if the rules are not disjunctive other hit policies have to be used.

The Any is one of the other hit policies returning single output. It allows overlapping rule outputs but they have to have the same values. If not, the outcome has to be determined either by a list of priorities as the Priority hit policy allows or by the First hit policy. This hit policy determines the output by the order of the rules from the top of the table to the bottom.

The hit policies returning multiple outputs are Output order, Rule order, and Collect hit policy. They either guarantee that the outputs will be returned in the order given by a list of priorities, in the order of matched rules or an unspecified order. For the Collect hit policy, four optional aggregation options can specify that sum, minimum, maximum, or a count of matched rules will be returned.

Here is the summarization of the hit policies:

- **Single** - Unique, Any, Priority, First
- **Multiple** - Output order, Rule order, Collect (Sum, Min, Max, Count)

2.2 Generation of Smart Contract

As was written, the generation will utilize the translation of the DMN business rules in XML format to smart contracts through template processors. For the target language, this section will explore the Plutus Tx based on a functional programming paradigm and the Solidity that builds on object-oriented programming languages. Because of that, the construction of different approaches has to be used for the templates.

The analysis of the structure of the smart contracts templates will be explored in the different sections starting with the Solidity language. The examples of the templates will be written in simplified pseudo-codes where the majority of the code will resemble the target smart contract programming language. The values inserted by the templating processor are represented by tags.

2.2.1 Generation of Solidity Smart Contract

In Solidity, a contract acts as a class. The decision logic of a business rule will be only a set of functions of this class. Those functions will be written in a procedural style. For the outputs, it was decided to provide it in a created structure and the assignment to the right values must be done in post-processing of the business rule task. That is because if a multiple hit policy is applied and more than one output is defined, the result must be wrapped in some way.

The input variables are taken from the input expressions of the decision table's inputs. It is assumed that the used variables were previously declared in the contract. The inputs have to be either variables or more complex expressions but then some additional logic has to be introduced. If the input variables are provided in the form of accessing state variables the function has to have a view modifier. When the inputs are provided through parameters the function can be marked pure.

2.2.1.1 General Templates

Generally, the function has to be contained inside a contract. The code must also start with a pragma directive stating the version of the contract. This simple template can be seen in the Figure 2.1 and it is the same for all following templates. The internal code structure is then generated based on a specific hit policy and will be explored in the next sections.

The output structure is also the same for all business rules except for the Collect hit policy with count aggregation that always returns an integer. The template for output structure is laid out in the Figure 2.2

Figure 2.1: Solidity Template: General Contract

```
1 pragma solidity <SolidityVersion>;
2
3 contract <ContractName> {
4     <OutputStructureDeclarations>
5     <BusinessRuleFunction1>
6     <OutputAssignment1>
7     <BusinessRuleFunction2>
8     <OutputAssignment2>
9     ...
10    <BusinessRuleFunctionM>
11    <OutputAssignmentM>
12 }
```

Figure 2.2: Solidity Template: Output Structure

```
1 struct <DecisionID>Output {
2     <Type1> <Variable1>;
3     <Type2> <Variable2>;
4     ...
5     <TypeM> <VariableM>;
6 }
```

2.2.1.2 Adjustment of Data Types

There also has to be some conversion of data format and adjustment of value comparison. There is no native function for the comparison of strings. The equality of strings can be done using a comparison of their hashes that will look like in the Figure 2.3. The date, time, and date-time data types have to be adjusted since Solidity currently does not implement those types. They can be represented as unsigned integers in a "yyyyMMddHHmmss" format.

Figure 2.3: Solidity Template: String Comparison

```

1  if (keccak256(abi.encodePacked(<StringValue1>)) ==
    keccak256(abi.encodePacked(<StringValue2>))) {
2      ...
3  }

```

In Solidity, there is no null or undefined value. Because of that, the violation of the hit policy leads to a thrown revert operation. This operation reverts all changes to the blockchain state. It is the outcome of the invalid design of the source decision table.

Figure 2.4: Solidity Template: Any Hit Policy

```

1  function <DecisionID>(<InputVariables>) public pure
    returns(<DecisionID>Output memory) {
2      <DecisionID>Output memory output;
3      bool matchedRule = false;
4      //RuleCheck1
5      if (<RuleCondition1> && ... && <RuleConditionK>) {
6          if (!matchedRule) {
7              output = <DecisionID>Output(<Value1>, ...,
                <ValueM>);
8              matchedRule = true;
9          } else if (output.<Variable1> != <Value1> ||
                ... || output.<VariableM> != <ValueM>) {
10             revert('Undefined output');
11         }
12     }
13     ...
14     <RuleCheckL>
15     if (!matchedRule) {
16         revert('Undefined output');
17     }
18     return output;
19 }

```

2.2.1.3 Templates for Single Hit Policies

The implementation of Any and Unique hit policies are similar as can be seen in the Figure [2.4](#) and the Figure [2.5](#). The rules are checked in the order of rows of the source decision table. If the condition is satisfied, another check is made that there was not another matched rule before. The difference between Any hit policy is that it checks if the previously matched rule returns the same output. If so, the function evaluation continues.

Figure 2.5: Solidity Template: Unique Hit Policy

```
1 ...
2 //RuleCheck1
3 if (<RuleCondition1> && ... && <RuleConditionK>) {
4     if (!matchedRule) {
5         output = <DecisionID>Output(<Value1>, ..., <
6             ValueM>);
7         matchedRule = true;
8     } else {
9         revert('Undefined output');
10    }
11 }
12 <RuleCheckL>
13 ...
```

Figure 2.6: Solidity Template: First Hit Policy

```
1 ...
2 //RuleCheck1
3 if (<RuleCondition1> && ... && <RuleConditionK>) {
4     output = <DecisionID>Output(<Value1>, ..., <ValueM
5     >);
6     return output;
7 }
8 ...
9 <RuleCheckL>
10 revert('Undefined output');
11 ...
```

The template in the Figure [2.7](#) depicts the First hit policy which returns the first output that meets the conditions. The test if the rule was already matched is left out.

Again, the Priority hit policy resembles the Unique hit policy. The template for it can be seen in the Figure [2.7](#). A priority list must be provided to this function for example in the first row of the decision table. The revert operation is exchanged with a call of a helper function resolving the priority of two matched outputs. The template of the helper function is laid out in the Figure [2.8](#).

Figure 2.7: Solidity Template: Main Function for Priority Hit Policy

```

1 //RuleCheck1
2 if (<RuleCondition1> && ... && <RuleConditionK>) {
3     if (!matchedRule) {
4         output = <DecisionID>Output(<Value1>, ..., <
5             ValueM>);
6         matchedRule = true;
7     } else {
8         output = <DecisionID>decideByPriority(
9             priorities, output, <DecisionID>Output(<
10                Value1>, ..., <ValueM>));
11     }
12 }
13 ...
14 <RuleCheckL>
15 ...

```

Figure 2.8: Solidity Template: Helper Function for Priority Hit Policy

```

1 for (uint i = 0; i < 3; i++) {
2     if (priorities[i].<Variable1> == currentOutput.<
3         Variable1>
4         && ...
5         priorities[i].<VariableM> == currentOutput.<
6             VariableM>) {
7         return currentOutput;
8     } else if (priorities[i].<Variable1> == newOutput.<
9         Variable1>
10        && ...
11        && priorities[i].<VariableM> == newOutput.<
12            VariableM>) {
13         return newOutput;
14     }
15 }
16 revert('Undefined output');

```

2.2.1.4 Templates for Multiple Hit Policies

The Output order hit policy is similar to the Priority hit policy in the terms of the priority list that has to be provided. The helper function is not necessary since there are multiple outputs returned. There is a list of boolean values declared with the priority list containing flags for matched outputs. The template for this hit policy is present in the Figure [2.9](#). It was decided that the generation will use mainly native functions of Solidity so that the version updates of the language do not require that many adjustments in the process.

Figure 2.9: Solidity Template: Output Order Hit Policy

```
1 //RuleCheck1
2 if (<RuleCondition1> && ... && <RuleConditionK>) {
3     for (uint i = 0; i < <NoUniqueOutputs>; i++) {
4         if (!existsInOutput[i]
5             && priorities[i].<Variable1> == currentOutput.<
6             Variable1>
7             && ...
8             && priorities[i].<VariableM> == currentOutput.<
9             VariableM>) {
10            existsInOutput[i] = true;
11            outputSize++;
12            matchedRule = true;
13            break;
14        }
15    }
16    <RuleCheckL>
17
18    //Put matched outputs to an array in the order of
19    priority list
20    <ForLoopOutputAssignment>
21    ...
```

The Rule order hit policy can be implemented in the same way as the Output order but the generator will fill the priority list with the outputs in the order of rules in the source table. Another possible way of depicting the Rule order hit policy can be seen in the Figure [2.10](#). It adds matched outputs after the last element in an array. Collect hit policy template is the same since the output order is not guaranteed but it has to be implemented in some way.

2.2.1.5 Templates for Collect Aggregations

The template in the Figure [2.11](#) depicts the Count aggregation of the Collect hit policy. It has the simplest logic, where a numeric variable is incremented when the rule is matched. The Sum aggregation is valid only for integer outputs and all the outputs variables are summated. The template for this aggregation is in the Figure [2.12](#).

The Min and Max aggregations are almost identical differing in the inequality symbol used. The DMN documentation does not specify the selection of extreme values when multiple outputs are defined in the table. The minimum and maximum will be considered for each output variable. The Figure [2.13](#) shows both of the aggregations valid only for number and date-time values.

Figure 2.10: Solidity Template: Rule Order Hit Policy

```

1 //RuleCheck1
2 if (<RuleCondition1> && ... && <RuleConditionK>) {
3     matches[outputSize] = <DecisionID>Output(<Value1>,
4         ..., <ValueM>);
5     outputSize++;
6     matchedRule = true;
7 }
8 ...
9 <RuleCheckL>
10
11 <DecisionID>Output[] memory output = new <DecisionID>
12     Output[](outputSize);
13 for (uint i = 0; i < outputSize; i++) {
14     output[i] = matches[i];
15 }
16 ...

```

Figure 2.11: Solidity Template: Collect Hit Policy with Count Aggregation

```

1 ...
2 //RuleCheck1
3 if (<RuleCondition1> && ... && <RuleConditionK>) {
4     count++;
5 }
6 ...
7 <RuleCheckL>
8 ...

```

Figure 2.12: Solidity Template: Collect Hit Policy with Sum Aggregation

```

1 //RuleCheck1
2 if (<RuleCondition1> && ... && <RuleConditionK>) {
3     output.<Variable1> += <Value1>;
4     ...
5     output.<VariableM> += <ValueM>;
6     matchedRule = true;
7 }
8 ...
9 <RuleCheckL>
10 ...

```

Figure 2.13: Solidity Template: Collect Hit Policy with Min and Max Aggregations

```
1 //RuleCheck1
2 if (<RuleCondition1> && ... && <RuleConditionK>) {
3     if (!matchedRule) {
4         output.<Variable1> = <Value1>;
5         ...
6         output.<VariableM> = <ValueM>;
7     } else {
8         if (output.<Variable1> <InequalitySymbol> <
9             Value1>) {
10            output.<Variable1> = <Value1>;
11        }
12        ...
13        if (output.<VariableM> <InequalitySymbol> <
14            ValueM>) {
15            output.<VariableM> = <ValueM>;
16        }
17    }
18    matchedRule = true;
19 }
20 ...
21 <RuleCheckL>
22 ...
```

2.2.2 Generation of Plutus Smart Contract

For the generation of smart contracts written in Plutus Tx, a different implementation approach was used. Again, the decision logic of a business rule will be only a set of functions of this class. In the case of Plutus, there are much more native functions provided. There were also more helper functions created for the templates than in the Solidity generation exploration. Those functions are often reused for multiple different hit policies.

The approach of using a custom output structure was also used. The same applies to the expectation of how the input variables are defined. Because the modification of state variables is as limited as possible in the Haskell, all generated functions are pure.

2.2.2.1 General Templates

As for the Solidity, certain parts of the code are general for all of the templates. Module definition is similar to contract definition in Solidity. The templates were designed so the same import clauses are required. Both of the initial parts of the code can be seen in the template in the Figure [2.14](#)

Figure 2.14: Plutus Template: General Module

```

1  module <ModuleName> where
2
3  import Playground.Contract
4  import Plutus.Contract
5  import PlutusCore.Default qualified as PLC
6  import PlutusTx
7  import PlutusTx.Lift
8  import PlutusTx.Builtins
9  import PlutusTx.Prelude
10 import Data.Maybe
11 import Prelude qualified as Haskell (Show, show, String
    )
12 import Data.ByteString.Char8 qualified as C

```

The Output structure definition is again the same for the templates except for the Count aggregation of Collect hit policy. This custom data type declaration together with standard instance declaration and the lifting of the value is depicted in the Figure [2.15](#). The off-chain part of the code is left out from the templates presented here but it is present in the enclosed CD to support their testing.

The functions representing the rules and their application to a list of inputs are a bigger exception to the similarities with Solidity. This approach would be also possible to do in Solidity but the functional basis of Plutus Tx encourages such design. The Figure [2.16](#) shows the general template for rule evaluation

2. BUSINESS RULE AS SMART CONTRACT

and the Figure 2.17 shows the same for the specific case of Count aggregation. The Figure 2.18 depicts the function applying previously defined rules.

Figure 2.15: Plutus Template: Output Structure

```
1 data <DecisionID>Output = <DecisionID>Output
2   { <Variable1> :: <Type1>
3     , ...
4     , <VariableM> :: <TypeM>
5   }
6   deriving stock (Haskell.Show, Generic)
7   deriving anyclass (FromJSON, ToJSON, ToSchema,
8                     ToArgument)
9 makeLift ''<DecisionID>Output
```

Figure 2.16: Plutus Template: Function for Rule Evaluation

```
1 //RuleCheckFunction1
2 <DecisionID>Rule1 :: <InputDataTypes> -> Maybe <
3   DecisionID>Output
4 <DecisionID>Rule1 <InputVariables>
5   | <RuleCondition1> && ... && <RuleConditionK> =
6     Just <DecisionID>Output
7     { <Variable1> = <Value1>;
8       , ...
9       , <VariableM> = <ValueM>;
10    }
11 | otherwise = Nothing
```

Figure 2.17: Plutus Template: Function for Rule Evaluation of Collect (Count)

```
1 //RuleCheckFunction1
2 <DecisionID>Rule1 :: <InputDataTypes> -> Maybe Bool
3 <DecisionID>Rule1 <InputVariables>
4   | <RuleCondition1> && ... && <RuleConditionK> =
5     Just True
6   | otherwise = Nothing
```


Figure 2.18: Plutus Template: Function for Application of Rules

```

1 <DecisionID>ApplyRules :: <InputDataTypes> -> [Maybe <
  DecisionID>Output]
2 <DecisionID>ApplyRules <InputVariables> = map (\f -> f
  <InputVariables>) [<RuleCheckFunction1>, ..., <
  RuleCheckFunctionM>]

```

2.2.2.2 Adjustment of Data Types

Plutus Tx has problems because it uses both built-in types that are easily compilable to Plutus Core and also similar types to the ones of Haskell's Prelude. Because of this, off-chain strings work better with Prelude's code and on-chain with built-in strings. When comparing Prelude's string with string values defined in functions, values have to be converted as it is in the Figure 2.19. Aside from Solidity, Plutus supports DateTime data type.

The null value in Plutus Tx is represented by the Maybe data type. This type can be wrapped around other types returning either "Nothing" or "Just a" where a represents the value of the wrapped data type. Still, the returning "Nothing" means an invalid design of the decision table.

Figure 2.19: Plutus Template: String Comparison

```

1 (toBuiltin $ C.pack <StringValue1>) == <StringValue2>

```

2.2.2.3 Templates for Single Hit Policies

As can be seen in the Figure 2.20 the Unique hit policy is the only one from single hit policies that represent its logic only with one original function. These functions check if the mapping function returned only one matched rule. The Any hit policy code from the Figure 2.21 needs to have the check if all returned outputs have the same value. For that, it needs equality instance declaration and function for comparison of output structure.

The implementations of the First hit policy in the Figure 2.23 and the Priority hit policy in the Figure 2.22 take multiple matched values from the mapping function. Then the main function returns only the first element. Except for the Priority hit policy that applies a function on this list that sorts it according the predefined priorities.

2. BUSINESS RULE AS SMART CONTRACT

Figure 2.20: Plutus Template: Unique Hit Policy

```
1 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
2 <DecisionID>Get <InputVariables>
3   | (length $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>) == 1
4     = Data.Maybe.listToMaybe $ catMaybes $ <
    DecisionID>ApplyRules <InputVariables>
5   | otherwise = Nothing
```

Figure 2.21: Plutus Template: Any Hit Policy

```
1 instance Eq <DecisionID>Output where
2   a == b = ((<Variable1> a == <Variable1> b) && ...
3           && (<VariableM> a == <VariableM> b))
4
5 anyCheck :: Eq a => [a] -> Bool
6 anyCheck outputs = and (map (head outputs ==) (tail
7   outputs))
8
9 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
10 <DecisionID>Get <InputVariables>
11   | (anyCheck $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>) == True
12     = Data.Maybe.listToMaybe $ catMaybes $ <
    DecisionID>ApplyRules <InputVariables>
13   | otherwise = Nothing
```

Figure 2.22: Plutus Template: Priority Hit Policy

```

1 <DecisionID>PriorityCheck :: ...
2 instance Eq ...
3
4 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
5 <DecisionID>Get <InputVariables>
6     | (length $ <DecisionID>PriorityCheck $ catMaybes $
    <DecisionID>ApplyRules <InputVariables>) /= 0
7     = Data.Maybe.listToMaybe $ <DecisionID>
    PriorityCheck $ catMaybes $ <DecisionID>
    ApplyRules <InputVariables>
8     | otherwise = Nothing

```

Figure 2.23: Plutus Template: First Hit Policy

```

1 instance Eq ...
2
3 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
4 <DecisionID>Get <InputVariables>
5     | (length $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>) /= 0
6     = Data.Maybe.listToMaybe $ catMaybes $ <
    DecisionID>ApplyRules <InputVariables>
7     | otherwise = Nothing

```

2.2.2.4 Templates for Multiple Hit Policies

The Output order hit policy utilizes the same functions and instance declarations as the Priority hit policy except it returns all mapped values. Its template is in the Figure [2.24](#). The Rule order hit policy has its counterpart in the First hit policy as can be seen in the Figure [2.25](#). Collect hit policy template is again the same since the output order is not guaranteed but it has to have some implementation.

The mapping functions enable the application of a simple function to the returned list and get the desired logic. The laziness of the Plutus Tx guarantees effective evaluation with this approach. For the Min, Max aggregation needs for their templates in the Figure [2.26](#) additional function to find extreme. The Sum aggregation from the Figure [2.27](#) alters this with its summation function. The Count aggregation of the Collect hit policy returns only the length of the mapped list. It is the simplest template as can be seen in the Figure [2.28](#).

Figure 2.24: Plutus Template: Output Order Hit Policy

```

1 <DecisionID>PriorityCheck :: ...
2 instance Eq ...
3
4 <DecisionID>Get :: <InputDataTypes> -> Maybe [<
    DecisionID>Output]
5 <DecisionID>Get <InputVariables>
6     | (length $ <DecisionID>PriorityCheck $ catMaybes $
    <DecisionID>ApplyRules <InputVariables>) /= 0
7     = Just $ <DecisionID>PriorityCheck $ catMaybes
    $ <DecisionID>ApplyRules <InputVariables>
8     | otherwise = Nothing

```

Figure 2.25: Plutus Template: Rule Order Hit Policy

```

1 <DecisionID>Get :: <InputDataTypes> -> Maybe [<
    DecisionID>Output]
2 <DecisionID>Get <InputVariables>
3     | (length $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>) /= 0
4     = Just $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>
5     | otherwise = Nothing

```

Figure 2.26: Plutus Template: Collect Hit Policy with Min and Max Aggregations

```

1 <DecisionID>Minimum :: ...
2
3 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
4 <DecisionID>Get <InputVariables>
5     | (length $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>) /= 0
6     = foldl <DecisionID>Minimum Nothing $ map Just
    $ catMaybes $ <DecisionID>ApplyRules <
    InputVariables>
7     | otherwise = Nothing

```

2.2.3 Comparison between Solidity and Plutus Generation

Concerning the suitability for generation using templates, both explored languages are appropriate for this process. From the point of the design, I think that the Solidity language leaves the developer bigger freedom for his im-

Figure 2.27: Plutus Template: Collect Hit Policy with Sum Aggregation

```

1 <DecisionID>Addition :: ...
2
3 <DecisionID>Get :: <InputDataTypes> -> Maybe <
    DecisionID>Output
4 <DecisionID>Get <InputVariables>
5     | (length $ catMaybes $ <DecisionID>ApplyRules <
        InputVariables>) /= 0
6     = foldl <DecisionID>Addition Nothing $ map Just
        $ catMaybes $ <DecisionID>ApplyRules <
        InputVariables>
7     | otherwise = Nothing

```

Figure 2.28: Plutus Template: Collect Hit Policy with Count Aggregation

```

1 <DecisionID>Get :: <InputDataTypes> -> Integer
2 <DecisionID>Get <InputVariables>
3     = length $ catMaybes $ <DecisionID>ApplyRules <
        InputVariables>

```

plementation. It is also readable for more people due to its procedural and object-oriented approach.

Once I've overcome the limitations and constraints of Plutus' functional paradigm, the implementation is more elegant than in the case of Solidity. The characteristics of Plutus Tx make it more effective for the evaluation in blockchain, but the work on it is still evolving. Because of that, the generation would have to be adjusted more dramatically once the updates are made.

2.3 Chapter Summary

In this chapter, the DMN business rules undergo an analysis that explored a way to capture them in a smart contract. The logic of decision tables is mostly defined by their rules and hit policy. Multiple pseudo-code examples for different cases were presented. They will serve as templates in the implementation of the process combining them with data models.

The exploration was made for two distinct programming languages. Solidity has its basis in the procedural and object-oriented paradigm. Unlike that, Plutus is a functional language. In both cases, the logic is captured by the main function and an output structure. Depending on the hit policy, supporting helper functions are also generated.

Both Plutus and Solidity are suitable for template processors. From the point of modeling, Solidity might be understandable to more designers when implementing some custom scripts. On the other hand, Plutus Tx has the potential to be a more effective language to run on a blockchain.

In the next step, the explored generation will be integrated into an already existing tool as a Proof of Concept. The outcome will be presented in the following chapter. As a target smart contract language, Solidity was chosen.

Proof of Concept: Implemented Generator

In the previous chapter, the generation of Solidity and Plutus smart contracts was explored. This chapter will describe the implementation of the presented approaches for Solidity. The source code together with instructions to build and run it is present on the enclosed CD attached to the thesis.

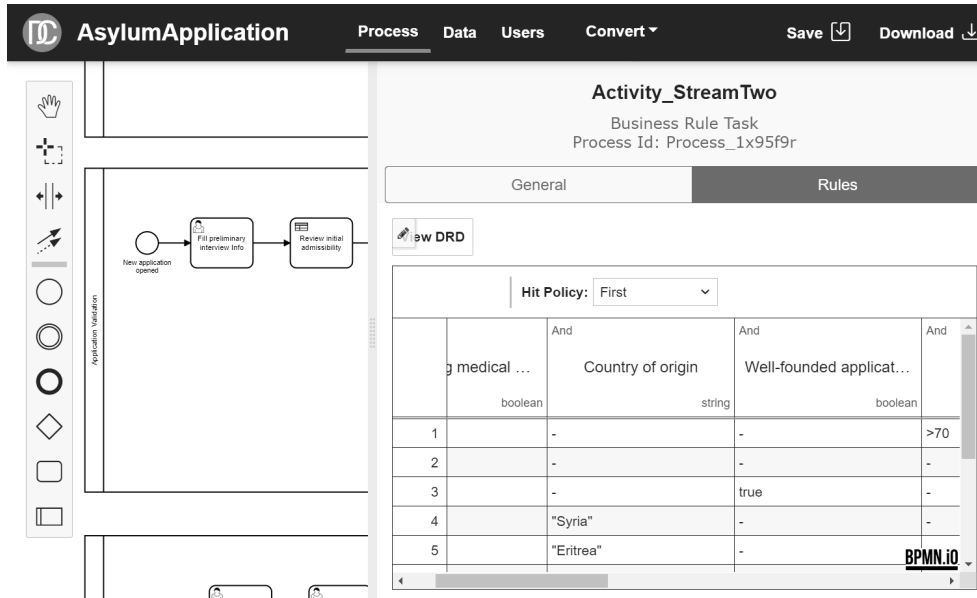
The chapter is separated into 3 sections. The first section describes the used technologies for the implementation. In the following section, how the classes and packages are structured in the code and the components are described. This section is focused mainly on the parts extended by this thesis. The chapter is ended with a smaller section reporting the testing approach of the implementation.

3.1 Used Technologies

For the transformation from DMN business rules to Solidity smart contract, the use of templating processor was chosen. Liquid language together with some of its template engines like Fluid or Scriban are solutions to achieve this way of generation. Both engines are built for .NET with similar characteristics differing only in popularity.

It was decided that this thesis will extend DasContract Editor since it provides multiple tools that are useful for the generation. Development of the modeling application and the complementary BPMN generation would be out of the scope of his thesis and it will help when presenting the results of the Proof of Concept. The DasContract Editor is an ongoing project to which multiple students are contributing in their theses so the future development is also assured.

Figure 3.1: Interface of DasContract Editor



3.1.1 DasContract Editor

DasContract editor is a tool allowing its users to design a smart contract using DasContract visual language. The input graphical language is based on the combination of DEMO modeling language, BPMN, and UML. It is a project maintained by CCMi Research Group at the Faculty of Information Technologies of CTU. Currently, there is a second version of the language called DasContract 2.0. [20] The first concepts were introduced in 2018. [36] The descriptions of the language that utilized DEMO methodology [37] and BPMN standard [38] followed. The first editor and Solidity converter was previously worked on in bachelor's and master's theses. Those works were focused on the editor: its user interface [39], redesign, additional functionalities [40], and Solidity conversion of BPMN with other models [41].

My implementation will be developed with the new version of the editor. There is also a Plutus convertor being implemented at the time of writing this thesis, but it will not be a part of the source code on the enclosed CD.

3.1.2 .NET Core and Standard

The back-end of the editor is written in .Net Core 5, which is an implementation of the .NET Standard. It is an open-source development platform for building applications. It allows writing .NET solutions in C#, F#, or Visual Basic. [42] The logic behind DasContract Editor is written in C#. The classes

and conversion mechanisms are written with the use of the .NET Standard, which is a base set of APIs common to all .NET implementations.

3.1.3 Blazor Framework

The user interface in the front-end of the editor is written using the Blazor framework. This package allows rendering HTML and CSS for hybrid desktop and mobile applications. The developer can write both client-side and server-side of the application using just C# language. The whole logic is then written in .NET. [43]

To save the contract, the user can either download it to its machine or save it in his browser. This is possible with Blazored LocalStorage which provides access to the browser's local storage. Not only does this saves storage on the server-side but improves performance since the serialization and deserialization are handled more efficiently. [44]

3.1.4 DMN and BPMN Modelers

The bpmn.io maintained by Camunda provides multiple tools to view and edit diagrams inside other projects. The dmn-js is a library that allows embedding the DMN modeler into applications. [45] The same applies to the bpmn-js library. [46] These components can be dynamically attached to or detached from any element on the page. Both modelers support current versions of the standards, meaning BPMN 2.0 and DMN 1.3.

3.1.5 Liquid and Scriban

Liquid is a templating language that combines its own syntax with HTML. The language is written without a concept of state so the user does not need to know the data content. It allows data manipulation in the template using custom logic and filters. Liquid is an open-source language created by Shopify. [47] DasContract uses a C# port of the Liquid templating language called Liquid.NET. [48]

Scriban is a scripting language and template engine for .NET. It was mainly developed for parsing and working with Liquid templates. Aside from Fluid, it is not an engine specifically for Liquid, but it is compatible with it. It efficiently works with CPU and Garbage Collector. The Lexer/Parser that it uses is also faster than regex-based parsers. [49]

3.2 Software Architecture

Since this thesis is mainly focused on the transformation process from the business rule, the front-end of the DasContract won't be described. The conversion from BPMN models to Plutus smart contracts is also available but not discussed in this chapter.

The used technologies for the front-end were described in the previous section. Still, to simplify it, the Blazor app consists of BPMN and DMN modelers, an editor of data entities, custom codes, and form fields. Modules displaying the converted codes and managing its projects are also included.

The main components that were extended are the Abstraction component and Solidity Converter. Abstraction contains classes representing multiple data models for form fields, DMN business rules, BPMN elements, and the contract itself. The Solidity Converter components work with the Liquid templates that are configured and composed based on the specific data model.

3.2.1 Abstraction Component

The root abstraction in the DasContract Editor is a contract. This class contains all DMN and BPMN elements together with the definition of a data model, users, and user roles. The core element for the data model is an entity. For the Solidity generation, the root entity must exist and it serves as solidity contracts' state variables.

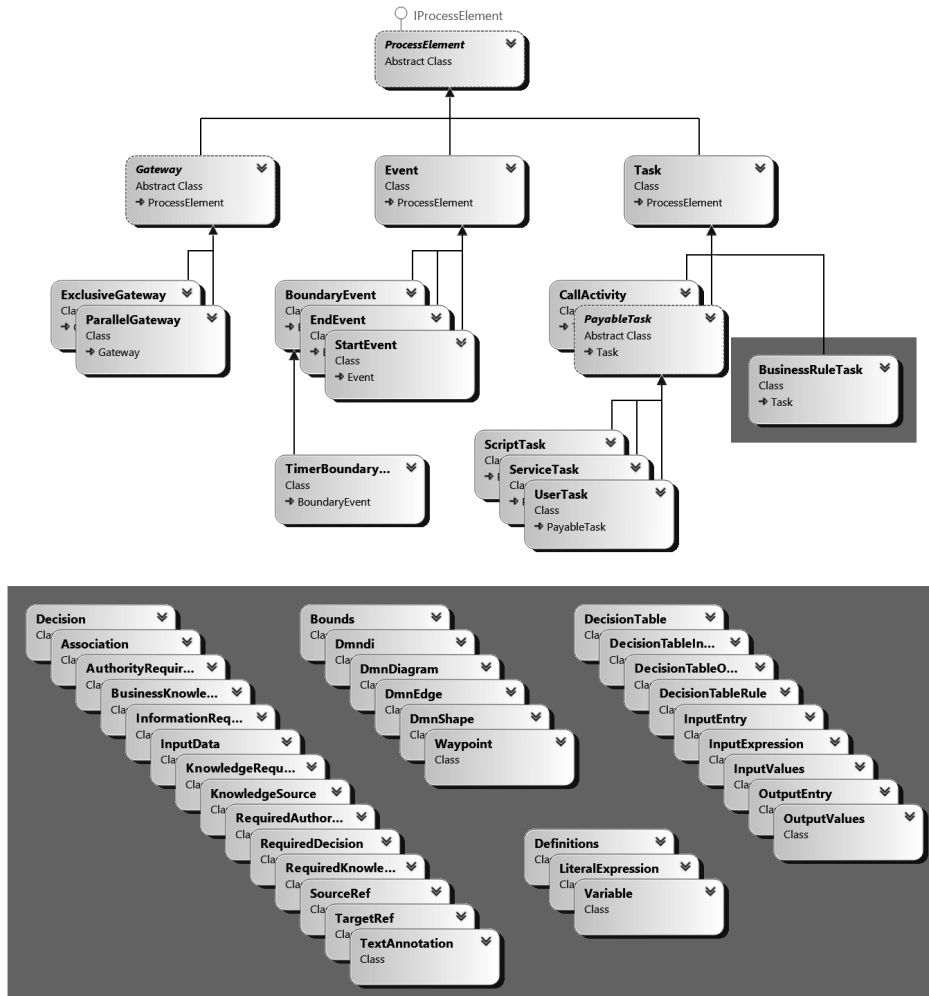
The abstraction component also contains classes for custom exceptions and UI for user forms. The DasContracts allows definitions of form fields of BPMN's user task that can be then subsequently displayed as user form. The main set of classes is the processes. They represent every BPMN element from events, gateways, and tasks to custom elements like roles and tokens.

3.2.1.1 XML Serialization of Business Rules

Previously, there were no abstractions for the elements of DMN notation. The DMN modeler was injected and handled all serialization and deserialization of the decision's XML representation. The whole business rule was stored in the business task abstraction as a string. The whole XML definition was in one line.

For this thesis, 32 classes representing XML elements were created for automatic deserialization by XMLSerializer. The business rule is still stored in the previously defined way, but the deserialization is done each time before the conversion. The deserialized business rule is then used as a data model for the generation. The DMN elements can be divided into elements of the DRD diagram, elements of the decision table, and information for the visual representation of previous elements.

Figure 3.2: Abstraction: Class Diagram Selection

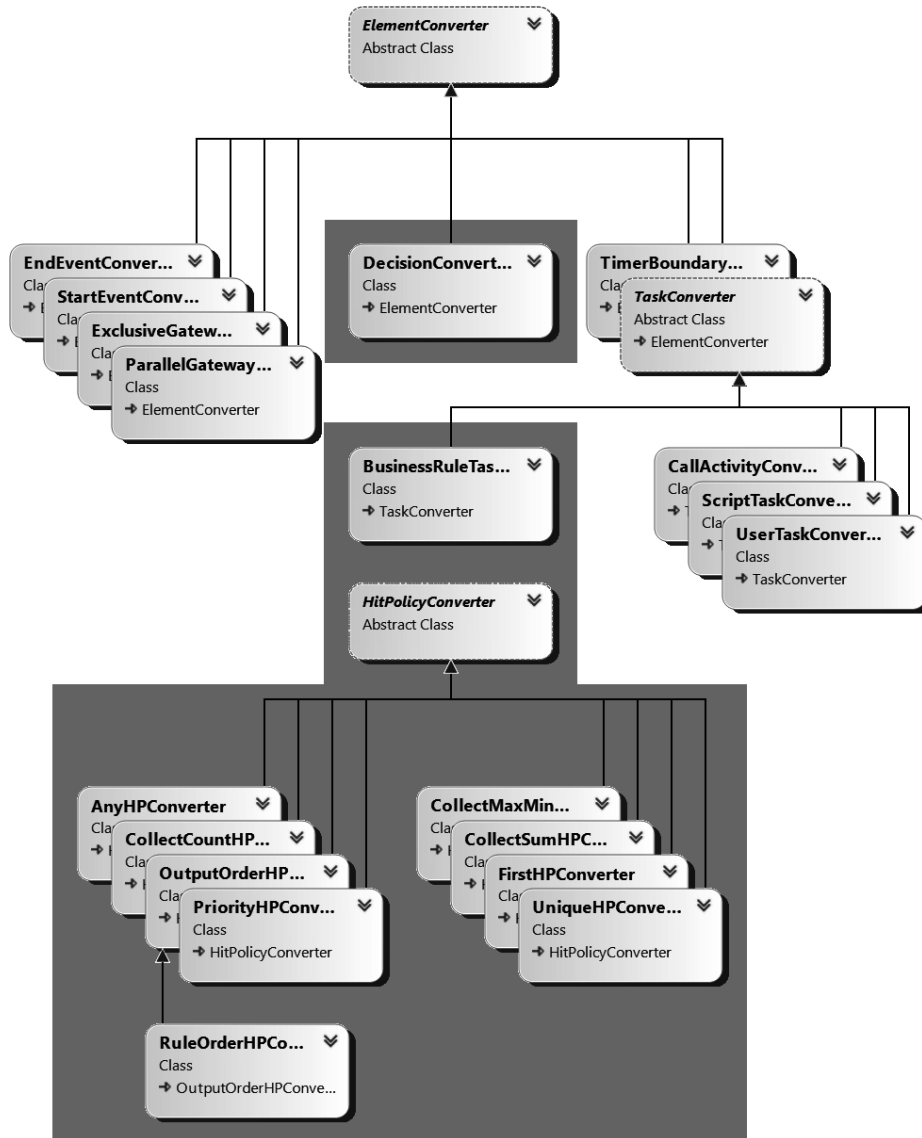


The added and modified classes of the DasContract Editor can be seen in the Figure [3.2](#). They are marked with a grey color. It is only a part of the Abstraction component. The full class diagram is stored on the enclosed CD in the attachment.

3.2.2 Solidity Converter Component

The second component is the Solidity Converter which handles the transformation from modeled abstractions into actual Solidity smart contracts. The first part of the converter is Solidity Components. Those classes handle the generation of smaller Liquid templates representing Solidity constructs like function, if-then statement, for loop, struct, and more.

Figure 3.3: Solidity Converter: Class Diagram Selection



The second part is individual converters of DasContract elements. The contract and process converters initiate the transformation using converters of BPMN elements like tasks, events, and gateways. Those converters then subsequently utilize the previously mentioned Solidity Components. At the end of the process, the final Liquid template is composed and ready for the template engine to fill the data.

3.2.2.1 Conversion of DRD and Business Rule Task

To implement the generation logic from the previous chapter, the DMN converters had to be implemented. Aside from that, the BPMN Business Task element was excluded from the transformation and had to be also implemented. The `BusinessRuleTaskConverter` contains the abstraction of the business rule. Just with this data, the decisions are identified and their conversion is called.

Because the DRD can obtain multiple decisions, additional logic had to be created that generates the declaration of the decision functions in the same order. If the DRD contains cycles, an exception is thrown and conversion is unsuccessful. Cycles in the DRD are an invalid design of the business rule.

3.2.2.2 Conversion of Hit Policies

The main decision logic is determined by the decision table's hit policy. For the Collect hit policy, it is also determined by the type of aggregation. The transformation of this logic is handled by the converter of a specific hit policy. There is one parent class holding the general logic that is identical to other hit policies. Another converter inherits logic from this class and extends it with the generation of specific decision logic and helper functions.

The selection and allocation of the converter are managed by the `DecisionConverter` which decides based on the abstraction of the decision. The Collect hit policy without aggregation uses the same converter as the Rule order hit policy. The same applies to the Min and Max aggregation of the Collect hit policy which uses one common class.

The added and modified classes of the `DasContract Editor` can be seen in the Figure [3.3](#). They are marked with a grey color. It is only a part of the `Solidity Converter` component. The full class diagram is stored on the enclosed CD in the attachment.

3.3 Testing

To test the proposed algorithm, two types of testing were chosen. Unit tests were created mainly to check that the individual classes generating specific templates are working correctly. They were also used for the testing of entities' serialization and generation of BPMN business task elements. The unit test is written using `xUnit.NET` and `.NET Test SDK`.

The generated smart contracts that were correctly generated then undergo manual testing that checked that the codes can be successfully deployed to the Ethereum blockchain. The codes that passed such testing are present on the enclosed CD in the attachment of this thesis. The contracts can be deployed using `Ganache` or `Remix Environments`. One of the manual testing and its simulation will be also resented in the next chapter.

3.4 Chapter Summary

The first part of the Proof of Concept for the presented way was focused on the implementation of an algorithm to generate smart contracts from DMN business rules. Used technologies, the structure of the code, and the description of the testing were described.

The solution developed in this thesis extended the already existing tool. DasContract Editor is a project providing modeling tools for the creation of smart contracts. The main technologies are Liquid Templates, Blazor Framework, .NET Core, and .NET Standard. For the testing, manual and unit tests were used.

Proof of Concept: Case Study

The preceding chapters laid the theoretical basis and exploration of technologies that would allow the generation of business rules to smart contract form. The last chapter of this thesis follows with a performance of the Proof of Concept case study that should highlight the possible applications of the generated business rule for the blockchain network.

As a process that could be improved with the use of business rules implemented with blockchain technology, the asylum procedure was chosen. Its characteristics like its international scope and the centralized identity of applicants make it ideal for blockchain applications.

4.1 Asylum Procedures in Europe

Article 14 of the Universal Declaration of Human Rights document lead to a convention of European countries. Most world countries agreed on the definition of the term refugee and some general approaches to the problematics in 1951 in Geneva on the Convention Relating to the Status of Refugees. [50]

From this commitment, European countries devised various complex asylum systems utilizing many procedural tools. European Council on Refugees and Exiles set its goal to protect and advance human rights that were agreed upon in 1951. [51]

It manages the Asylum Information Database mapping and collecting information about different asylum procedures in European countries. Those countries have tools to process the application for refugee status and assess the applicant's claim but also tools to transfer responsibility for the application to another European state based on the Dublin procedure. [52]

This procedure is defined in the Dublin III Regulation issued by the European Parliament. It determines on which condition the application can be transferred to a different state and in which processes those states can submit the reasons and basis for this responsibility. The identity of applicants can be

verified in European Asylum Dactyloscopy Database that was established for this regulation. [53]

4.1.1 Asylum Process in Republic of Ireland

For the purposes of this thesis's case study, Ireland's asylum procedure was chosen but the ties to the Dublin procedure do not limit the feasibility for other EU member states.

The process of assessment of the application for refugee status changed significantly after 6th January 2017, when the International Protection Act (IPA) 2015 came into effect. The new process grew in complexity beyond its originally defined boundaries because four years after the commencement of this act it still has to deal with many transitional cases from the old procedure and also it was altered due to the ongoing Covid-19 crisis. [54]

Outside the regular procedure for the applications, there is a possibility for accelerated assessment for cases that meet the defined circumstances for prioritization. As of December 2020, the regular processing of the application could take up to 18 months, which is about 3 months longer than it took before the Covid-19 outbreak. The accelerated procedure could speed the process up to 14 months. Both processes could take even longer if the decisions are taken to appeal courts and institutions. [54]

The number of given decisions in 2020 was 2 276 and 68% of the applications were rejected. The number of pending applications at the end of the year is 5 279. [55]

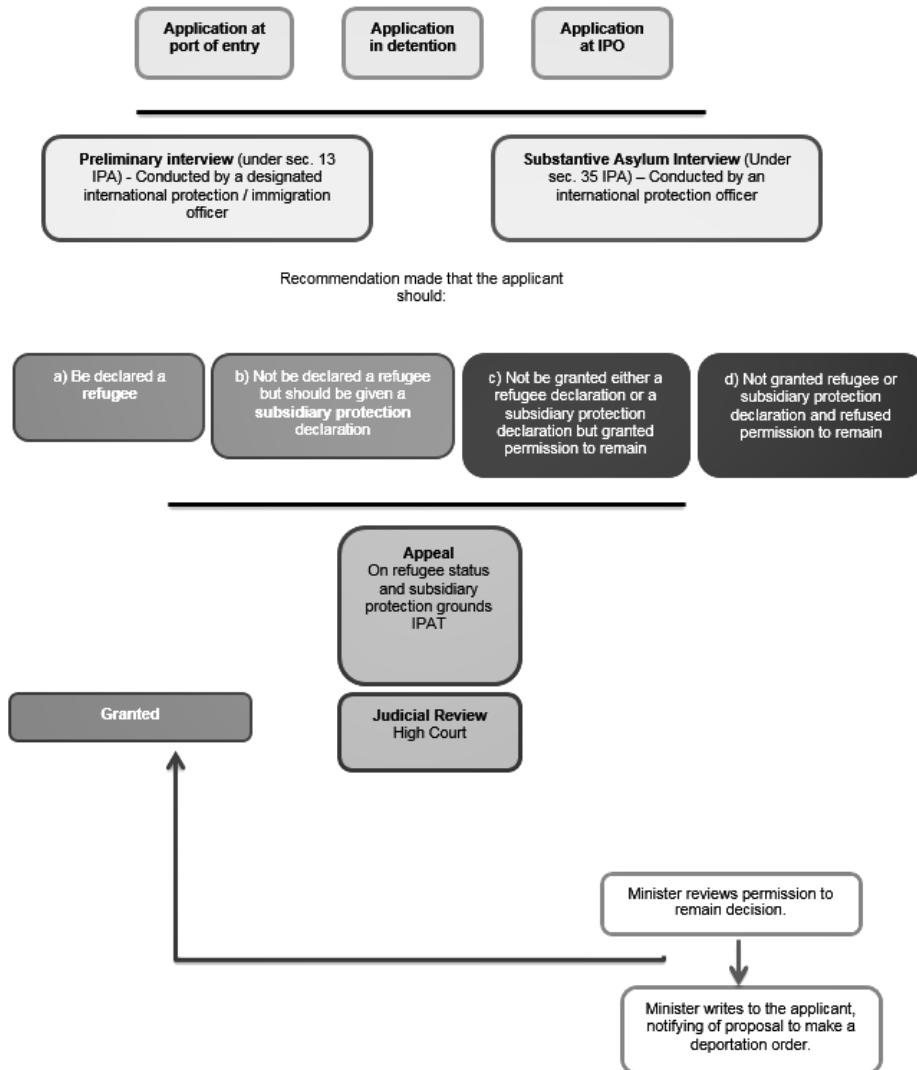
4.2 As-Is Analysis: Collection of Resources

For the research part of this PoC case study, the resources obtained from personal interviews, direct observations, and group meetings were skipped. The information was mainly gained from process descriptions, surveys, laws, and official websites of the Republic of Ireland.

The following sections will give information about how the processes begin and what are their possible outcomes. Other than that, responsible authorities for parts of the asylum procedure are identified. Subsequently, the main procedures are described before the next documentation part of the As-Is analysis.

The procedure had to undergo many temporary changes in its processes because of the COVID-19 pandemic (ongoing while writing this work). Those changes may evolve throughout time and some can even remain, but this thesis will take the description of the processes mainly from the time before the disease has spread.

Figure 4.1: Flow Chart of the Irish Asylum Procedure [56]



4.2.1 Stages and its authorities

The whole procedure around the application and subsequent decision on it can be divided into eight stages:

- Application at the border
- National security clearance
- Refugee status determination
- Dublin procedure
- Accelerated procedure
- Appeal
- Judicial review
- Subsequent application (admissibility)

Some of those stages may not have taken place in specific instances, but this depends on the nature of the actual process. [\[57\]](#)

4.2.1.1 Stages under Garda National Immigration Bureau

Garda National Immigration Bureau (GNIB) is a unit of the Garda Síochána, which is the national police and security service of Ireland. This unit is responsible for the stage of the application at the border and national security clearance.

The website of the GNIB currently describes one change in the process, which is that registrations for people living in the Dublin area are newly covered by the Naturalisation and Immigration Service. [\[58\]](#)

4.2.1.2 Stages under International Protection Office

International Protection Office (IPO) is a part of Immigration Service Delivery. The main responsibility of this office is an examination of the applications and their processing. IPO is the competent authority in the stages of refugee status determination, Dublin procedure, and accelerated procedure. [\[59\]](#)

4.2.1.3 Stages under International Protection Appeals Tribunal

International Protection Appeals Tribunal (IPAT) is a subject that was created under the International Protection Act 2015. Its function is to determine appeals to the decisions of the IPO, so its only responsibility in the description is the appeals stage. [\[60\]](#)

4.2.1.4 Stages under High Court

The High Court of Ireland, the first instance when dealing with the most serious and important civil and criminal cases, is the determining authority for the judicial review stage. This means that it serves as an appeal court for the IPAT when its decision is challenged. [61] [62]

4.2.1.5 Stages under the Minister for Justice and Equality

In general, the Minister for Justice and Equality in the Department of Justice and Equality works to advance community and national security, promote justice and equality, and safeguard human rights. In the asylum procedure, it serves as the competent authority for the subsequent application (admissibility). [63]

4.2.2 Beginning of the process and its End States

There are three application points: Application at a port of entry, application in detention, and application at IPO. [56] In general, the procedures after the application are the same for all of the mentioned applications. For application at the frontiers, a person must first indicate that he or she needs asylum. After this, they are given permission to enter and remain until the procedure is over, and a temporary residence certificate. Subsequently, they are viewed as an applicant for international protection under Section 16 IPA. [64]

The asylum application can end in 4 states: Declaration of refugee status, declaration of subsidiary protection, rejection of the application with permission to remain, and rejection of the application with a deportation order. [56].

4.2.3 Description of the Main Procedure

The application begins with filling out the application form. The first short interview is then taken by an immigration official or an international protection officer. This depends on the place of the application. [64]

4.2.3.1 Preliminary Interview

In the following steps, the applicant's age must be determined. If the age is higher than 13 years, fingerprints and photography is taken for identification purposes. This is also done for the dependent children if there are any. The applicant can state that he or she is under 14 years old, but this must be believed by the IPO officer. If such an applicant is also unaccompanied, his or her acquisition of fingerprints is skipped.

Then the actual preliminary interview is conducted, where its outcome with the information in the application is examined for initial admissibility of the application. The request may be decided to be inadmissible if that

person already received refugee status or subsidiary protection in another EU Member State. The inadmissibility decision is also taken if another Member State is considered the first country of asylum. Also, if the applicant does not cooperate in this interview it may lead to his or her detention. After the interview is finished, the applicant is provided with detailed information on the asylum process in a preferable language and he or she is also provided with the Application for International Protection Questionnaire. [62]

4.2.3.2 Substantive Interview

After the applicant receives the Application for International Protection Questionnaire, he or she has 20 working days to complete this questionnaire. When this is done, the questionnaire is reviewed and the application is categorized and prioritized. Then also before the interview can be conducted, the language interpreter has to be determined which could be difficult if the applicant uses some narrowly used language. If the waiting time would exceed 6 months, the applicant can ask for provisioning of the interview date.

When the substantive interview can be realized, the applicant is informed and the interview takes place. It is possible that during the interview there will be problems with the interpreter. In this case, a new interview must be done. The interview is done face to face in Dublin. After it is finished the applicant has to sign the interview's transcript and is issued a Temporary Residence Certificate.

At the end of the main procedure, the application is given with the final decision by the IPO. This decision is reviewed by the Minister for Justice and Equality who then declares the resulting status of the applicant. [62]

4.2.4 Implementation of Dublin III Regulation

After the IPO acquires the applicant's fingerprints and his or her photography is taken, the Dublin Unit of the IPO can start with the Dublin Procedure. The main goal of this procedure is to find out if the applicant is not subject to international protection in another country. The biometric information is given to the EURODAC which analyses the identification material against previous records in the database. If there are any, the regular process can continue as usual.

If the applicant is subject to the Dublin III Regulation or previously lodged in another Member State, he or she is provided with Dublin Procedure Information Leaflet and interviewed to explain newly found information. The only aspect that can go against the Dublin Regulation is that the applicant has any dependent children whose interest would be against the regulation. The interest must be consulted with Tusla Child and Family Agency.

The responsible Member State has then time to raise any objection to the decision which has to be provided with supporting documents. This could pro-

long the process by months. In the other scenario, the applicant is transferred to the selected Member State. [65]

4.2.5 Accelerated Procedure

There are three ways that can truncate the procedure by giving certain applications higher priority. First, the Minister for Justice and Equality can take regard and speed up the process most often in problematic cases where the applicant cannot provide evidence to his or her claims by very specific explanations. The Minister has also designated some countries as Safe Countries of Origin which can lead to the accelerated procedure. Those countries are Albania, Bosnia and Herzegovina, North Macedonia, Kosovo, Montenegro, Serbia, Georgia, and South Africa.

The main prioritization is done according to the IPO and the UN High Commissioner for Refugees statement. The first division comes in the respect of validity of International Protection Act 2015 procedures. Cases opened after and during the commencement of the Act are put into Stream One category and case before the commencement of the Act into Stream Two. Both streams are processed in specific priorities and then on order of older cases first. In the streams the priority is done in following order: [62]

- **Stream One:**
 - subsidiary protection recommendations
 - appeals at the Refugee Appeals Tribunal
 - refugee status recommendations
- **Stream Two:**
 - reasons of high/low age
 - well-founded applications
 - countries of origin: Syria, Eritrea, Iraq, Afghanistan, Iran, Libya and Somalia
 - applicants with severe/life threatening medical condition

4.2.6 Appeal and Judicial Review

For the decisions given by the Dublin procedure and the admissibility procedure, there is a possibility to raise an appeal to International Protection Appeals Tribunal. This has to be done within 10 days of the decisions. The applicant can choose if he or she wants to get an oral hearing or the decision

over the appeal can be done behind closed doors. The IPAT either affirms or declines the initial decision.

Again, the decisions laid out by the IPAT can be rejected by the applicant who can challenge it a push it to the High Court. This process on a point of law only under Irish administrative law needs special permission since it is a costly and long process. [62]

4.2.7 Blockchain Network for the Asylum Procedure

Since the generator would only provide smart contracts (in the case of this thesis the scope is even narrower) for the blockchain network, the actual establishment of the system and deployment of the smart contracts on it is a different topic. For the asylum procedure, there is one blockchain solution already being developed by institutions of the EU.

4.2.7.1 European Blockchain Services Infrastructure

European Blockchain Services Infrastructure (EBSI) is a platform enabling the trustful sharing of evidence and credentials, which could be official documents or others. The EBSI has been deploying a network of distributed nodes across Europe since 2020. The main goal should be accelerating the creation of cross-border services for public administrations. The provided solution would be already in compliance with EU regulations. [66]

The first use-case that EBSI was focused on were self-sovereign identity, diploma management, document traceability trust data sharing. In 2021, three more use cases were being worked on: SME Financing, European social security pass, and asylum process management. [66]

On the other hand, EBSI does not provide a way for its users to add their smart contracts. They are managing their own library of smart contracts that are being enhanced with new use cases and scripts. Those smart contracts are using Solidity language. [67]

4.3 As-Is Analysis: Modeled Process

Previously collected and summarized description of the process was modeled using BPMN. The focus was to model the asylum procedure as faithful as possible. Still, the gaps in the resources were filled by this thesis and they could differ from the real process. This is not an issue as the case study is Proof of Concept.

4.3.1 BPMN Descriptive Model

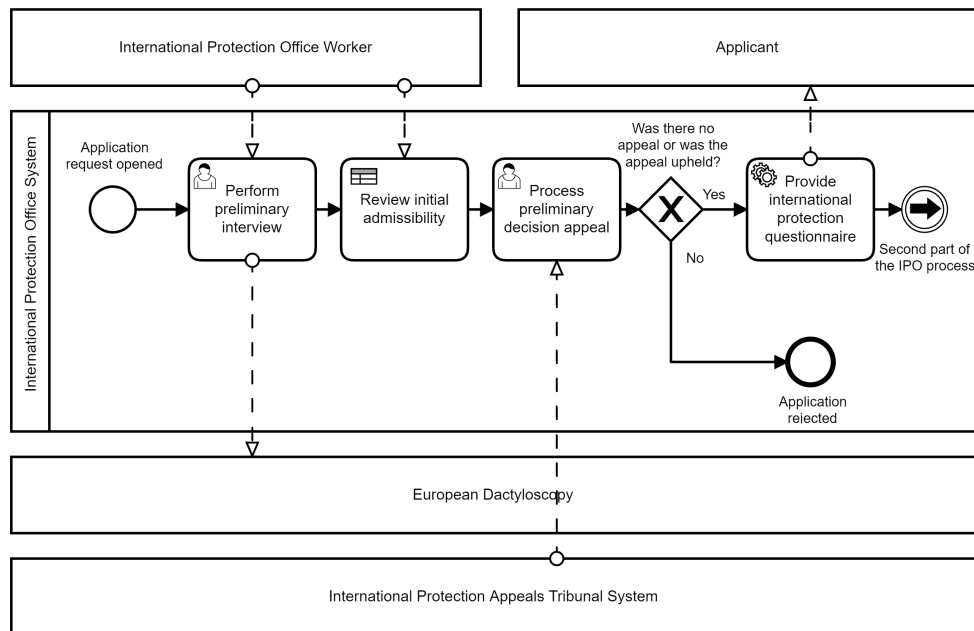
Firstly, the described asylum procedure was captured in the Descriptive BPMN model which can be seen in the Figures [4.2], [4.3], [4.4], and [4.5]. It skips the ini-

4.3. As-Is Analysis: Modeled Process

tial phase of border procedure and the phase dealing with appeals and judicial review. The main focus of the model is the information system of the IPO and the information system of the Ministerial Decision Unit. The model considers other sides but they are modeled as black boxes and their logic is not the focus of this thesis.

The system of the IPO is the place where the application is created and goes through all stages that collect the information from the applicant in the interviews. It also communicates with the other Dublin countries through the Dublin information system. The last task for this system is to give a first instance decision in a form of a report. The Ministerial Decision Unit then reviews the report and delivers the final decision. This decision is then confirmed by the Minister for Justice.

Figure 4.2: Descriptive As-Is Model of the Asylum Procedure: Part One



4. PROOF OF CONCEPT: CASE STUDY

Figure 4.3: Descriptive As-Is Model of the Asylum Procedure: Part Two

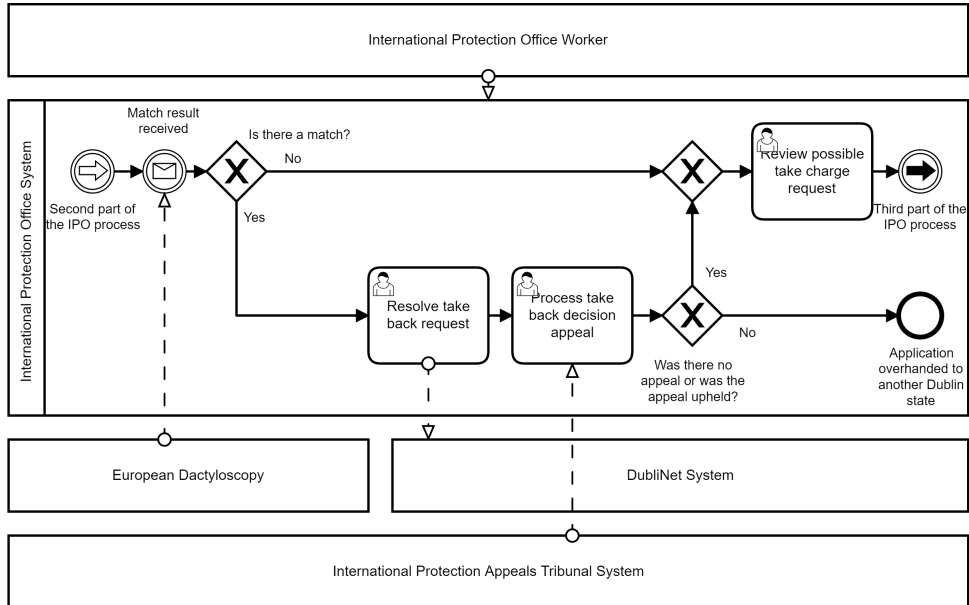


Figure 4.4: Descriptive As-Is Model of the Asylum Procedure: Part Three

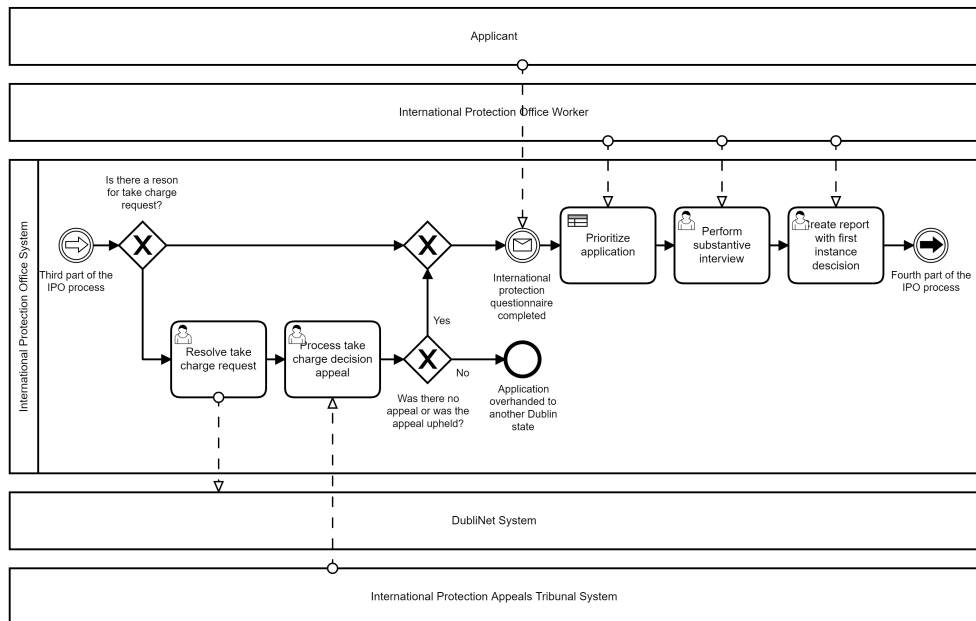
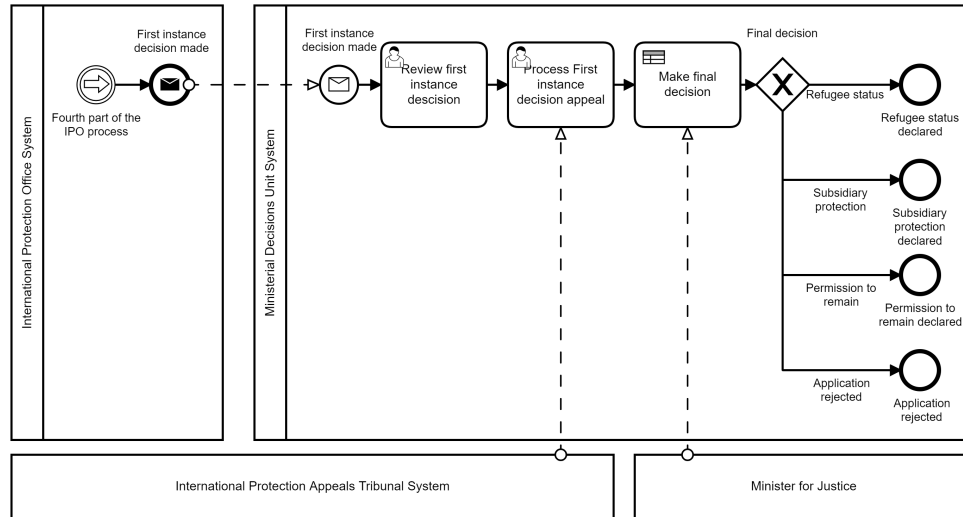


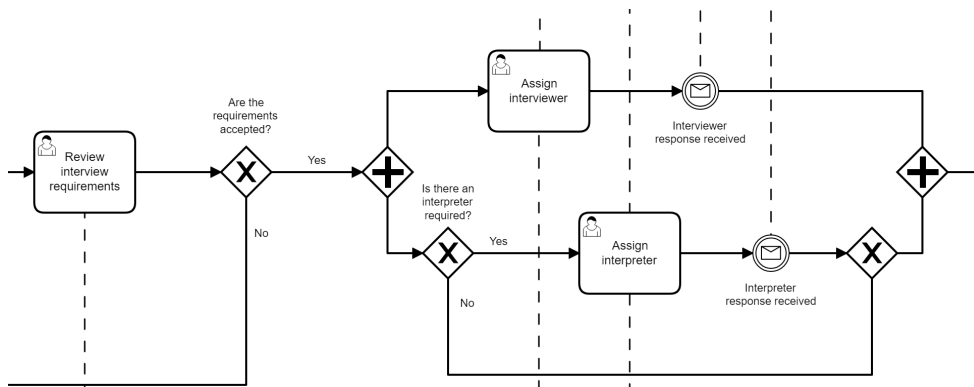
Figure 4.5: Descriptive As-Is Model of the Asylum Procedure: Part Four



4.3.2 BPMN Analytic Model

The analytic BPMN model is depicting the process on a deeper level than the descriptive. The scope of the model is too big to be added to the text of this thesis, but the full version can be found on the enclosed CD in the attachment. It models all the situations that are described in the section Collection of Resources.

Figure 4.6: Preview of the Analytic As-Is Model



Still, the preview of the model can be seen in the Figure [4.6](#). This preview implies that the complete model confirms, that the communication between other systems and sides, takes an equal part in the process. Aside from the

processes depicted as black boxes, there are others in this model. It considers 11 sides.

The main ones are used information systems and the applicant that were mentioned in the previous systems. There are two new roles present in the substantive interview as can be seen also in the preview: Interpreter and Interviewer. The model also depicts other institutions that the IPO needs to communicate with.

4.3.3 Business Rules in such Process

Both in the new process and the old process the found business rules are the same. It is possible that they would be refined differently once they would have to be modeled but it is something that is not the focus of this thesis. There were 4 business rules identified in the process:

- **Review initial admissibility** - Determines that the application is valid for the process based on the information given in the preliminary interview.
- **Prioritize stream one application** - Prioritization of the applications made after the commencement of the IPA.
- **Prioritize stream two application** - Prioritization of the applications made before the commencement of the IPA.
- **Make final decision** - Determines the final decision in considering both decision of the first instance report and the decision by the IPAT.

As was mentioned, there are some gaps in the descriptions of the prioritization. The applications are prioritized under certain circumstances but they are not disjunctive. Therefore for the DMN notation, the hit policy was set to first meaning that the output will consider only the highest priority condition.

4.3.4 Gaps, Bottlenecks, and Weaknesses

The resources that were the basis for the previous section were mainly collected from the Asylum Information Database which does detailed work in describing procedures of different European countries in similar forms. Still, it contains many gaps for example where certain processes start and wait for other and non-deterministic descriptions of prioritization. Also, the implementation of the Dublin III Regulation can bring to the process more delays due to uncertain evidence for the responsible Member State.

4.4 To-Be Analysis: Modeled Process

In the third section of this PoC case study, the previous process will be enhanced by the blockchain network and tasks that are performed by generated smart contracts. The goal of this To-Be analysis is to show the feasibility of generated DMN rules in the blockchain. Similar solutions are being developed during the writing of this work, which will be described at the end of the section.

4.4.1 Advantages of the New Procedure

As one of the disadvantages of the previous process, the delays caused by the Dublin procedure were mentioned. This could be improved if the evidence about who is responsible would be already available to both sides. If we introduce a blockchain network to the system, we could not only get a trustful system that could track all data objects throughout all asylum procedures in Europe but also, the possibility to improve and keep the original processes with smart contract programming.

The advantage of the approach that utilizes a blockchain network is also implied by the use of the EURODAC database in all Member States that verifies if the identification information was already recorded in the system. That is what the blockchain's original purpose is to prevent double-spending on the same transaction or token.

The solution that uses smart contract codes generated from the DMN notation or the BPMN model can also solve the second problem with the old procedure. Since the processes have to be modeled first, the designer would be pushed to model in a way that is more deterministic and understandable for future use.

4.4.2 BPMN Descriptive Model

For a better preview of how this model would be implemented, a new BPMN model on the descriptive level of abstraction was created. It can be seen in the Figures [4.7](#), [4.8](#), [4.9](#), and [4.10](#). The first noticeable change is the addition of the blockchain system. Some of the logic was moved from the IPO system and will be handled by smart contracts.

The Ministerial Decision Unit System is removed and its logic will be handled solely by the blockchain. Only some small application or API will be needed for the Ministerial Decision Unit Officers and Minister for Justice to access the data in it. In total, 10 tasks from the descriptive BPMN model of the as-is state were migrated to blockchain which is roughly two-thirds of the tasks. This of course doesn't reflect the actual scope of the workload.

4. PROOF OF CONCEPT: CASE STUDY

Figure 4.7: Descriptive To-Be Model of the Asylum Procedure: Part One

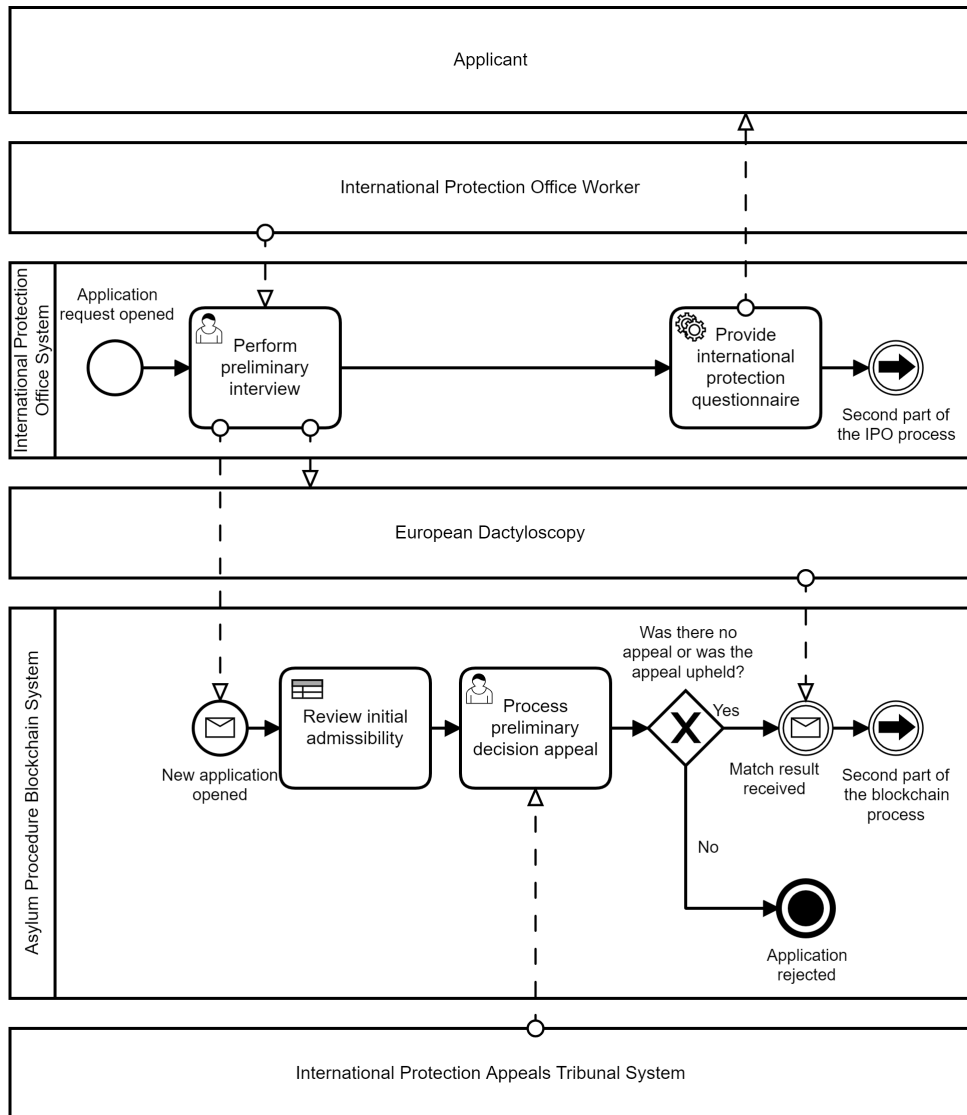
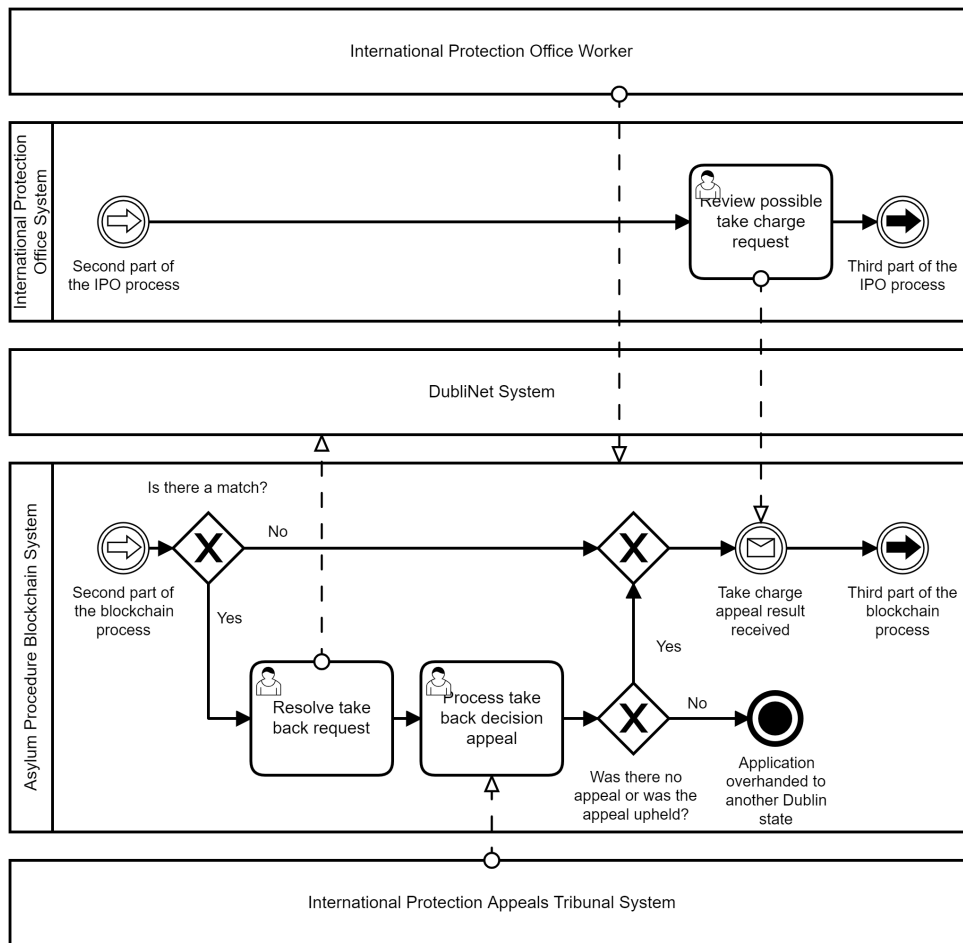


Figure 4.8: Descriptive To-Be Model of the Asylum Procedure: Part Two



4. PROOF OF CONCEPT: CASE STUDY

Figure 4.9: Descriptive To-Be Model of the Asylum Procedure: Part Three

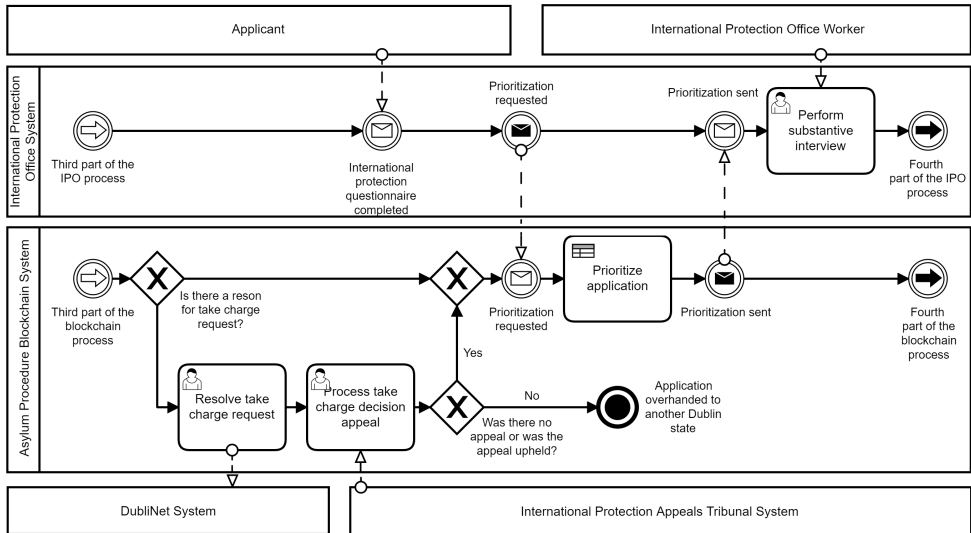
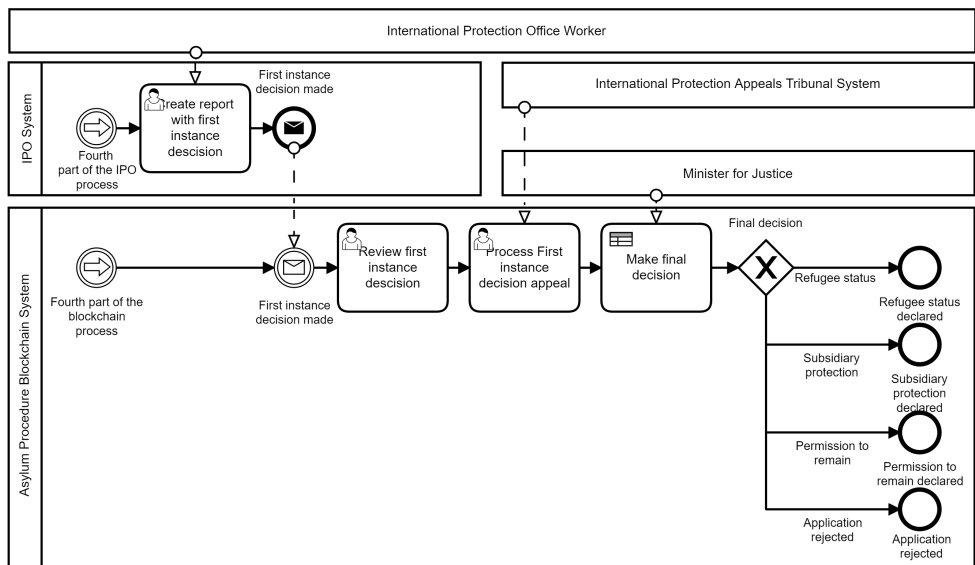


Figure 4.10: Descriptive To-Be Model of the Asylum Procedure: Part Four



4.4.3 BPMN Analytic Model

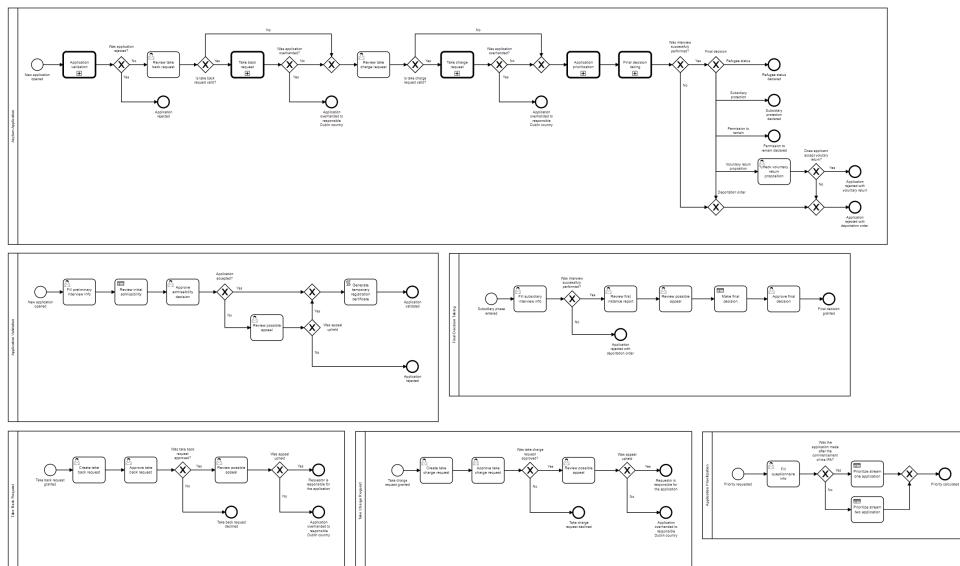
Same as for the as-is model, the size of the analytic to-be BPMN model doesn't allow it to be contained in the written part of the thesis. It can be found in the attachment on the enclosed CD. This logic was similar in the as-is model but now it is completely moved into the smart contract.

The main logic that would be handled by the blockchain system is to reference and change the statuses of the application. This reference together with the nature of the network provides a trusted basis for the application's status. Other Dublin states can avoid communication to check the status with the other countries thus speeding up the process. The same applies to propagating the statuses of upheld decisions by the IPAT. The logic of business rules was also moved onto the blockchain network.

4.4.4 BPMN Common Executable Model

The common executable BPMN model should be in such status that it could be executable by the related software for BPMN execution. The last level of process detail was created in the DasContract Editor since the DasContracts uses a modified version of the BPMN. The scope of the process can be seen in the Figure [4.11](#). Some parts laid out in the analytic model were altered during the creation of the executable model in the DasContract. It also contains only the part of the process working in the blockchain.

Figure 4.11: Executable BPMN Model for Asylum Procedure



4. PROOF OF CONCEPT: CASE STUDY

The main process contains 5 call activities. Those processes won't be shown in the text of the thesis, but they can be found on the enclosed CD. The subprocesses are the following: Application validation, Take back request, Take charge request, Application prioritization, and Final decision taking.

4.4.5 DMN Decision Tables

There were no DRD identified in the process. Each business rule task includes only one decision table. The identified business rules were described in the previous sections and they were kept for the to-be model. For the tables to be included in the text, they had to be narrowed and the names of the inputs are not visible. They are described below.

Figure 4.12: Decision Table of the Review Initial Admissibility Business Rule

Hit Policy: <input type="text" value="First"/>			
When	And	And	Then
Existing internat... boolean	Non-refouleme... boolean	Country of origin string	Initial admissibi... string
-	-	"Bosnia and Herzegovina"	"Inadmissible"
-	-	"North Macedonia"	"Inadmissible"
-	-	"Georgie"	"Inadmissible"
-	-	"Kosovo"	"Inadmissible"
-	-	"Montenegro"	"Inadmissible"
-	-	"Republic of Albania"	"Inadmissible"
-	-	"Republic of Serbia"	"Inadmissible"
-	-	"Republic of South Africa"	"Inadmissible"
true	true	-	"Inadmissible"
-	-	-	"Admissible"

- **Decision Table: Review initial admissibility** - depicted in the Figure [4.12](#)
 - Existing international protection - Input Column
 - Non-refoulement principle - Input Column
 - Non-refoulement principle - Input Column
 - Country of origin - Input Column
 - Initial admissibility - Output Column

Figure 4.13: Decision Table of the Stream One Priority Business Rule

Hit Policy: <input type="text" value="First"/> ▾			
When	And	And	Then
Pending refgee... boolean	Pending appeal... boolean	Pending subsidi... boolean	Stream one pri... number
-	-	true	1
-	true	-	2
true	-	-	3

- **Decision Table: Prioritize stream one application** - depicted in the Figure [4.13](#)
 - Pending refgee status recommendation - Input Column
 - Pending appeal at the former RAT - Input Column
 - Pending subsidiary protection recommendation - Input Column
 - Stream one priority - Output Column

4. PROOF OF CONCEPT: CASE STUDY

Figure 4.14: Decision Table of the Stream Two Priority Business Rule

Hit Policy: First <input type="button" value="v"/>						
When	And	And	And	And	And	Then
Threatening me...	Country...	Well-fo...	Age	Part of a fa...	Unac...	Stream...
boolean	string	boolean	number	boolean	boolean	number
-	-	-	>70	false	-	1
-	-	-	-	-	true	1
-	-	true	-	-	-	2
-	"Syria"	-	-	-	-	3
-	"Eritrea"	-	-	-	-	3
-	"Iraq"	-	-	-	-	3
-	"Afghanistan"	-	-	-	-	3
-	"Iran"	-	-	-	-	3
-	"Libya"	-	-	-	-	3
-	"Somalia"	-	-	-	-	3
true	-	-	-	-	-	4

- **Decision Table: Prioritize stream two application** - depicted in the Figure [4.14](#)
 - Threatening medical condition - Input Column
 - Country of origin - Input Column
 - Well-founded application - Input Column
 - Age - Input Column
 - Part of a family group - Input Column
 - Unaccompanied minor in time of application - Input Column
 - Stream two priority - Output Column

Figure 4.15: Decision Table of the Final Decision Business Rule

Hit Policy: <input type="text" value="First"/>		
When	And	Then
Ministerial unit ... string	Decision behin... string	Final decision string
-	"Refugee status"	"Refugee status"
-	"Subsidiary protection"	"Subsidiary protection"
-	"Permission to remain"	"Permission to remain"
"Refugee status"	-	"Refugee status"
"Subsidiary protection"	-	"Subsidiary protection"
"Permission to remain"	-	"Permission to remain"
"Voluntary return proposition"	-	"Voluntary return proposition"
"Deportation order"	-	"Deportation order"

- **Decision Table: Make final decision** - depicted in the Figure [4.15](#)
 - Ministerial unit decision - Input Column
 - Decision behind upheld appeal - Input Column
 - Final decision - Output Column

4.4.6 Generated Executable Model

The BPMN and DMN models were shown in the last sections. There is also a data model and set of roles defined for the contract. Those elements won't be presented. The data model contains only basic entities and their attribute to support the simulation.

The roles were defined and assigned to the corresponding task but the logic was not converted to the contract. One of the reasons was that the scope of the smart contract was too big. The generated contract has approximately 800 lines of code and its size is 34 413 bytes. This is mentioned since the Spurious Dragon, the allowed size for a smart contract is 24 576. Still, for the testnet, the generated code was successfully deployed.

4.5 To-Be Analysis: Simulation

The generated source code was deployed to Remix IDE. This online tool [68] provides a Solidity compiler and smart contract deployment directly in the browser. The process can be then simulated using a simple interface that also allows debugging options. The following sections will describe multiple simulation cases accompanied by adjusted excerpts from the mentioned environment.

Figure 4.16: Smart Contract Endpoint in Remix IDE

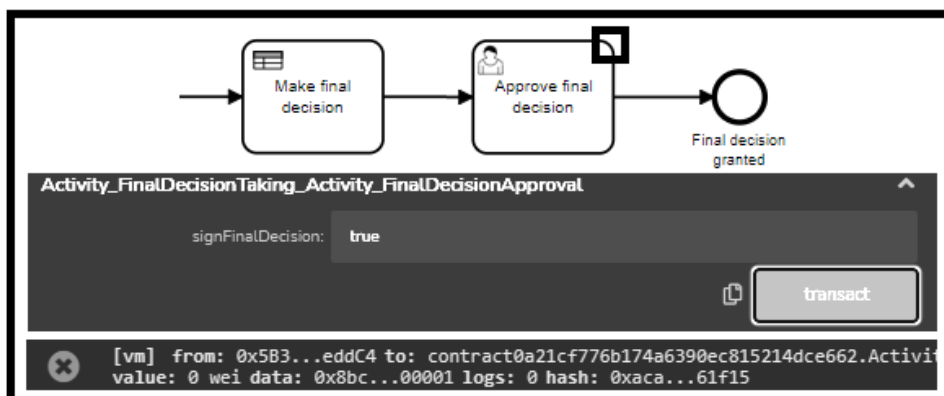


The Figure 4.16 contains a preview of the smart contract's endpoint that is possible to call. There are all endpoints for corresponding calls but only certain are available. The check if a certain call is possible is through the methods `isStateActive` and the name of the task. These checks can be seen in the Figure 4.17. If the user would call a task that is not yet active he would receive an error message as is depicted in the Figure 4.18.

Figure 4.17: Functions for Checking Status of Tasks



Figure 4.18: Invalid Task Call

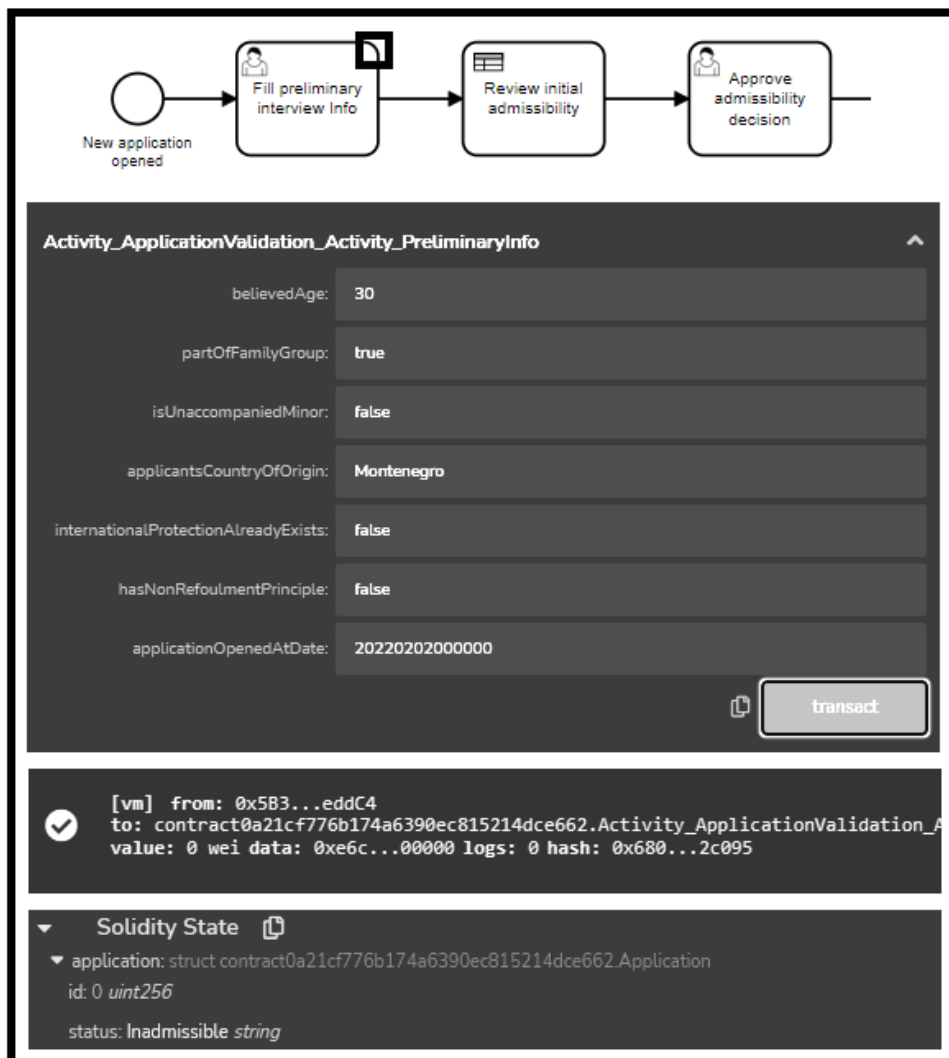


4.5.1 First Simulation Case: Inadmissible Application

The first simulation case is the simplest one depicting a situation where the application is rejected right after the first phase. The preliminary interview is the first point where the information about the applicant is collected. Right after that, the business rule determining admissibility is applied.

The Figure 4.19 shows that the application from the first case was determined as inadmissible. This is because the applicant stated the country of origin that is defined in a list of safe countries. The case would be followed by rejection.

Figure 4.19: First Simulation Case: Preliminary Information



4.5.2 Second Simulation Case: Overhanded Application

The second case simulates a situation where the application is deemed admissible (see the Figure 4.20 and the Figure 4.21), but it is overhanded to another Dublin country. This can occur when the applicant's fingerprints are already present in the EURODAC system because he already applied in another country.

First, the IPO worker reviews the results of the EURODAC system. If there is a matched record of fingerprints he has to create a so-called take back request. This request is then either accepted or declined by the requested Dublin country. This phase is depicted in the Figure 4.22. No new business rules are triggered in this case.

Figure 4.20: Second Simulation Case: Preliminary Information



4. PROOF OF CONCEPT: CASE STUDY

Figure 4.21: Second Simulation Case: Admissibility Approval

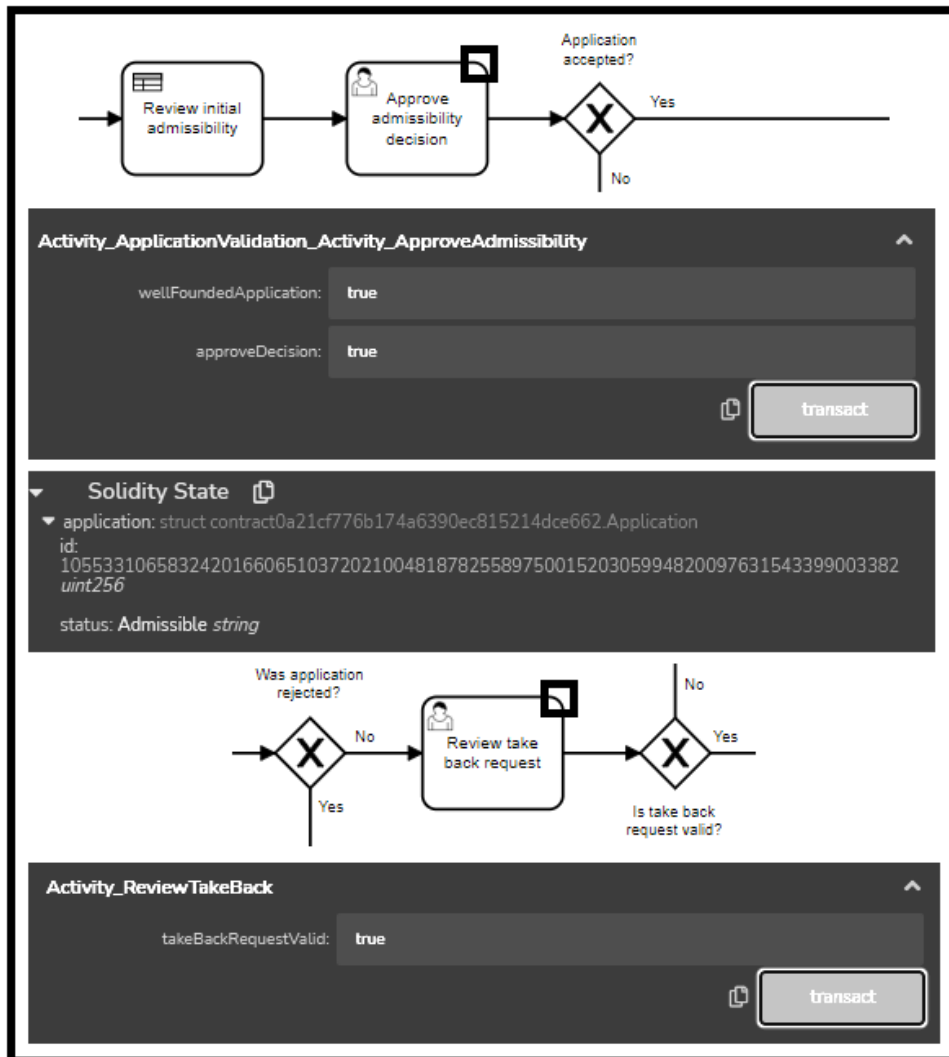
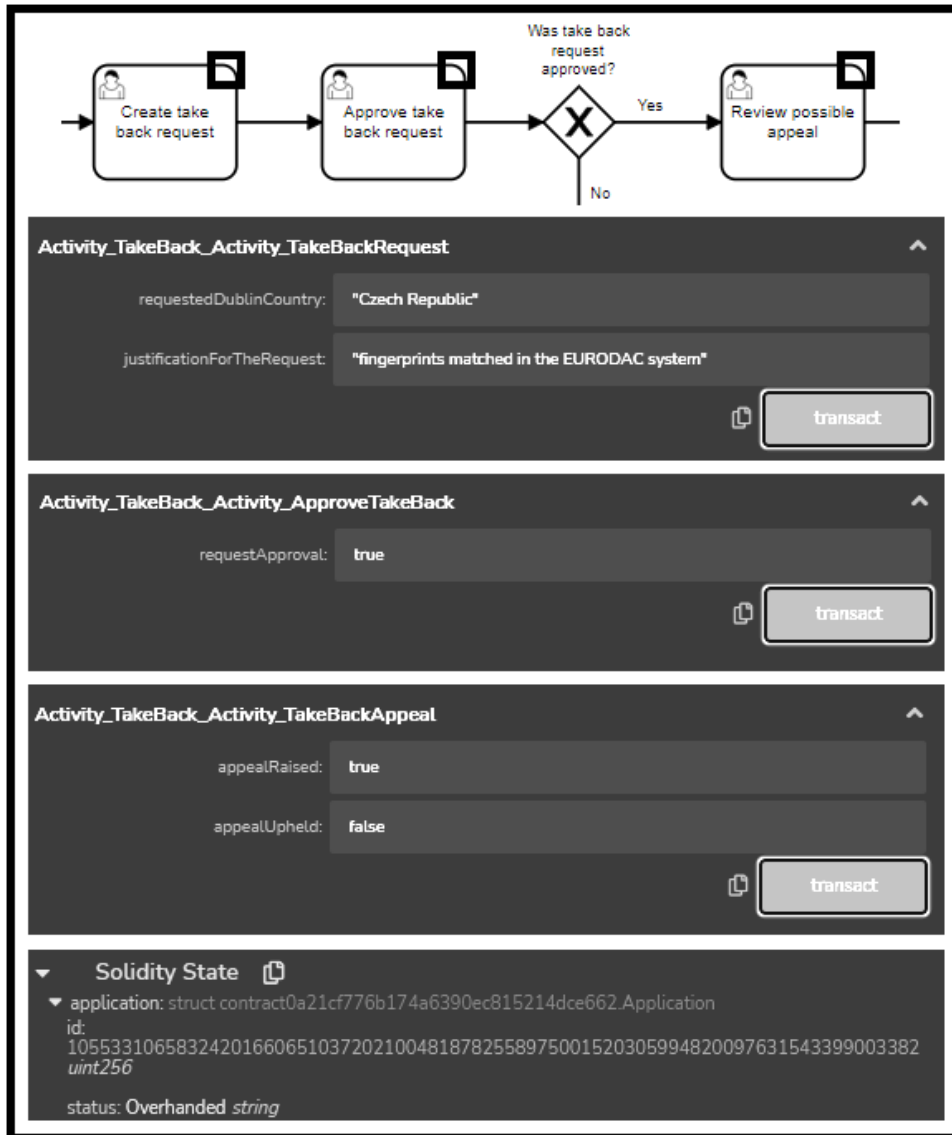


Figure 4.22: Second Simulation Case: Take Back Request



4.5.3 Third Simulation Case: Refugee Status

The third case is the first that is successfully completed. In this case, three business rule tasks are occurring. Since the application was created recently as can be seen in the Figure 4.23, the application will be processed in the stream one backlog. The result of the prioritization is depicted in the Figure 4.24.

The final decision is grated after the application passes through all tasks that can be seen in the Figure 4.25. This case simulates a situation where the application is rejected in the first instance report. Then the appeal is upheld and the decision is changed to Refugee status. The last confirmation of the decision is depicted in the Figure 4.26.

Figure 4.23: Third Simulation Case: Preliminary Information

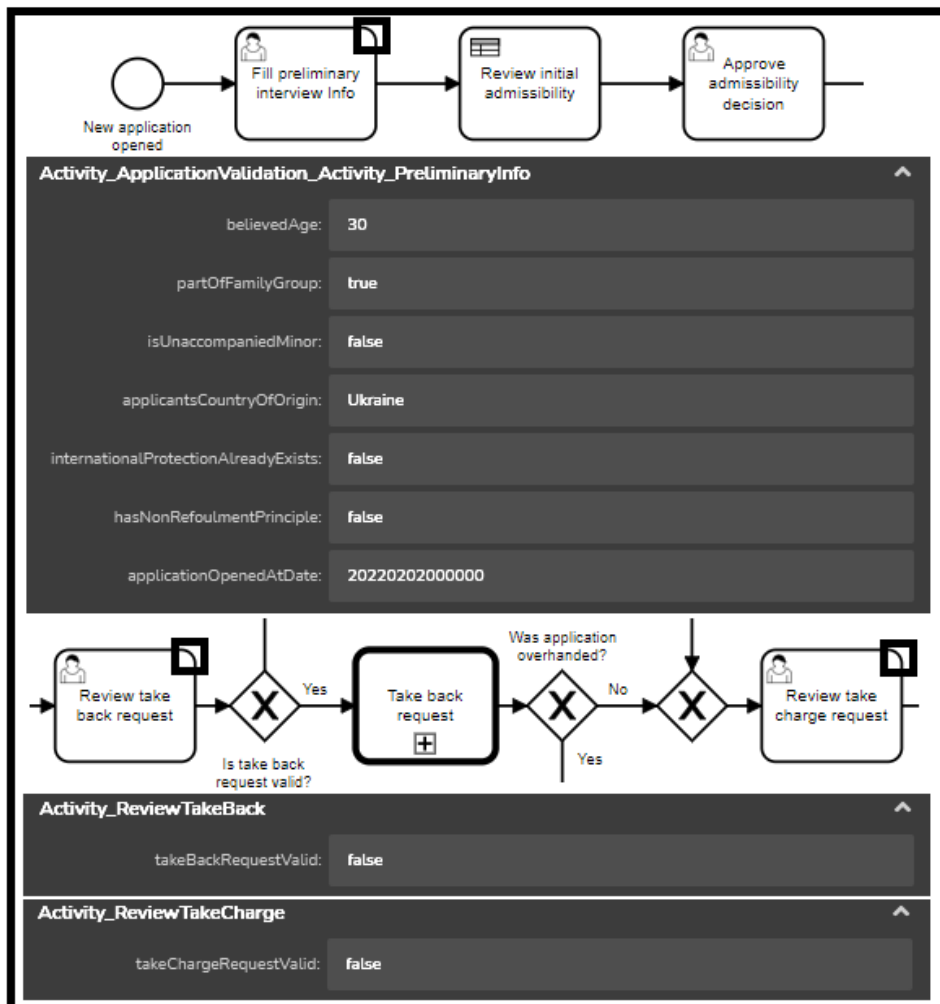
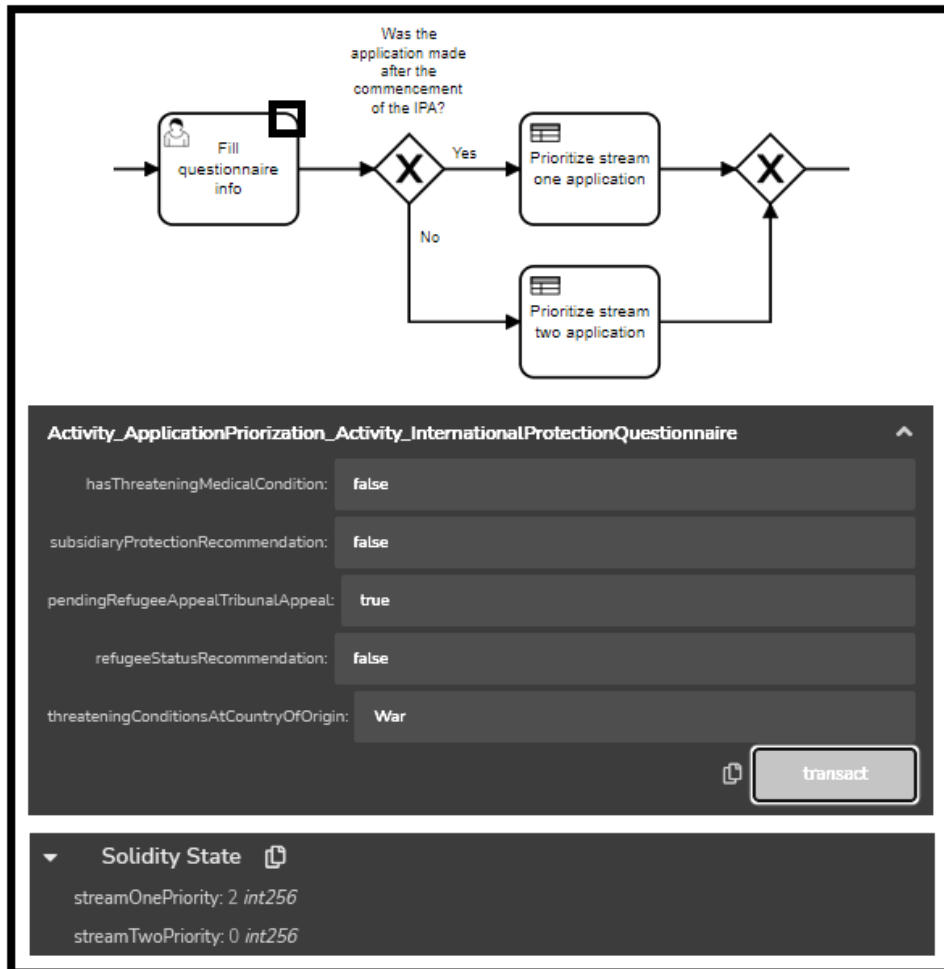


Figure 4.24: Third Simulation Case: Prioritization



4. PROOF OF CONCEPT: CASE STUDY

Figure 4.25: Third Simulation Case: Final Decision

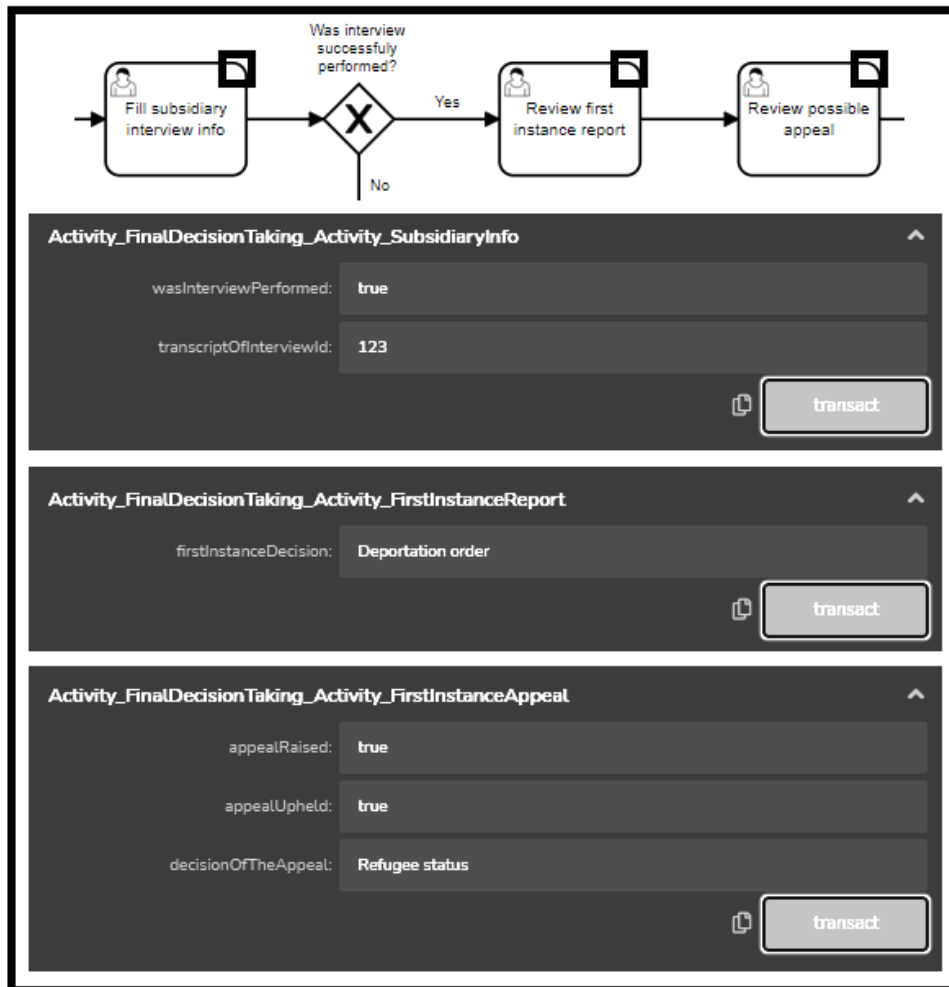


Figure 4.26: Third Simulation Case: Final Approval



4.5.4 Fourth Simulation Case: Permission to Remain

The last simulation case is similar to the previous one. It was designed to trigger the last unused business rule assigning the priority for the stream two backlogs. The input stating the date when the application was opened must be set to a much older date as it is in the Figure 4.27. This situation is not so probable but it was defined in the process for applications that are more sporadic and hanging in the system longer time. They also include a business case with a lot of different conditions.

The last phase of the process goes without problems and objections. The priority is assigned for stream two as can be seen in the Figure 4.28. The final decision (depicted in the Figure 4.29) is determined as Permission to remain and approved by the Minister for Justice.

Figure 4.27: Fourth Simulation Case: Preliminary Information

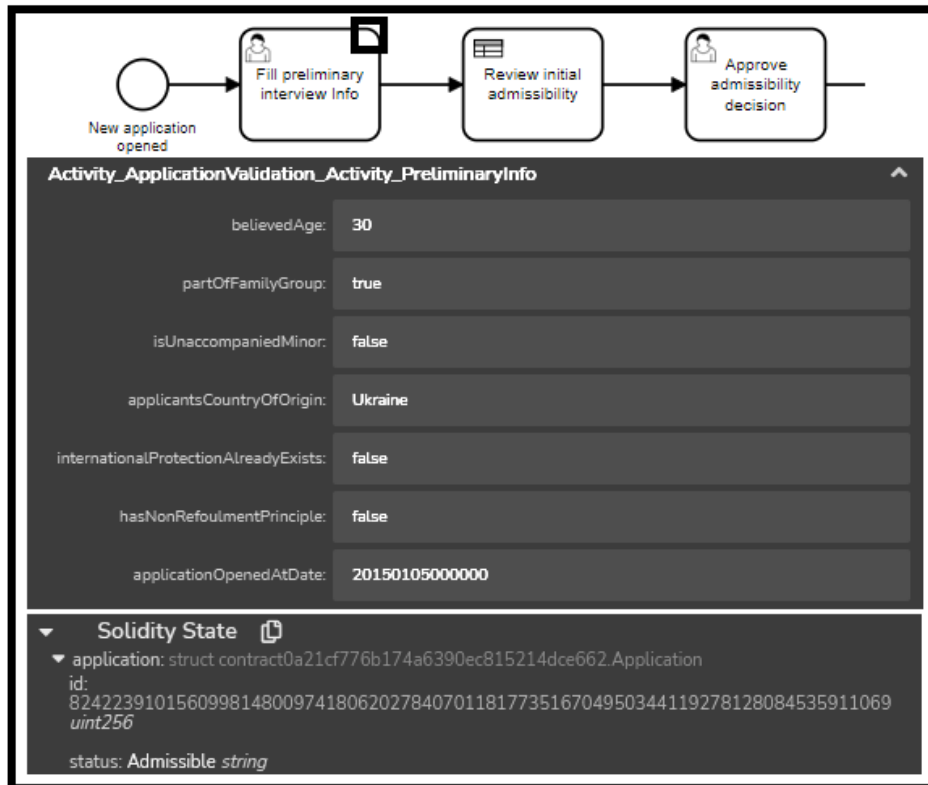
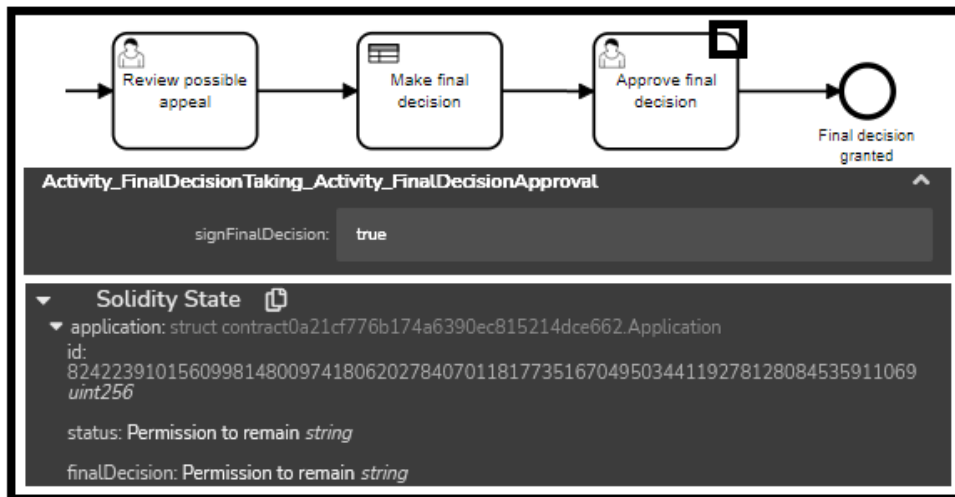


Figure 4.28: Fourth Simulation Case: Final Decision



Figure 4.29: Fourth Simulation Case: Final Approval



4.6 Chapter Summary

The final chapter provided a Proof-of-Concept case study in which blockchain technology was introduced to the existing asylum management process of the Republic of Ireland. Aside from the process' introduction, the first part of the chapter was functioning as a collection of resources describing the current procedures in the asylum process. The BPMN models on multiple levels were then created from the description.

The second part of this chapter used the BPMN models as a basis to which the new logic utilizing blockchain network was added to improve its attributes and characteristics. One of the old business rules was then taken described and translated into a smart contract as a Proof of Concept. This smart contract is then deployed and presented in a simulated blockchain environment.

Conclusion

The main goal of this thesis was to evaluate how smart contracts can be generated from business rules. The source decision tables and diagrams were captured in DMN notation. Two distinct programming languages for writing smart contracts were compared. The implementation of the explored process followed with its demonstration in a simple case study.

DMN notation is a standardized way to represent business rules inside an organization. The applicability can be enhanced when the notation complements BPMN process models. Cardano and Ethereum projects provide well-established platforms for writing smart contracts. The approaches to the development of Solidity and Plutus languages, together with big communities, make them suitable as target languages.

The theoretical foundations were followed with the design of the conversion process. During this section, the distinguishing characteristics of Solidity and Plutus were noticeable but similar when considering the usability. The proposed generation was then implemented and integrated into a DasContract Editor that effectively utilizes a Liquid template processor. Solidity was selected as a target language since the BPMN conversion was already present in the editor. This was beneficial for the case study.

The developed algorithm was then tested both with manual and unit tests. The Proof of Concept case study proposed a way to improve the existing process of asylum procedure with blockchain technology and generated smart contracts. The demonstration of the developed algorithm was successful. Using the presented approach, operating costs of asylum procedures can be significantly reduced.

After all goals of this thesis were completed, there are many possible directions that future research could take. The smart contract platform is still in development so the conversion mechanism will have to evolve also. Multiple variations of modeling methods and target smart contract languages are available for exploration.

Bibliography

- [1] Vejrážková, Z. *Business Process Modeling and Simulation: DEMO, BORM and BPMN*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Prague, 2013.
- [2] Aalst, W. M. P.; Hofstede, A. H. M.; et al. Business Process Management: A Survey. [online], 2003, [2022-03-05]. Available from: https://www.researchgate.net/publication/221586027_Business_Process_Management_A_Survey
- [3] Bandara, W.; Indulska, M.; et al. Major Issues in Business Process Management: An Expert Perspective. [online], 2007, [2022-03-05]. Available from: https://www.researchgate.net/publication/221409577_Major_Issues_in_BPM_Major_issues_in_Business_Process_Management_an_Expert_Perspective
- [4] Object Management Group. Business Process Model and Notation Specification. [online], [2022-03-05]. Available from: <https://www.omg.org/spec/BPMN/2.0/PDF>
- [5] Ross, R. G. *Business Rule Concepts: Getting to the Point of Knowledge*. Business Rule Solutions Inc, fourth edition, 2013, ISBN 978-0941049146, 162 pp.
- [6] Object Management Group. Decision Model and Notation Specification. [online], [2022-01-09]. Available from: <https://www.omg.org/spec/DMN/1.3/PDF>
- [7] Red Hat. Red Hat Decision Manager Documentation: Chapter 1. Decision Model and Notation (DMN). [online], [2022-03-05]. Available from: https://access.redhat.com/documentation/en-us/red_hat_decision_manager/7.12

- [8] Wattenhofer, R. *Blockchain Science: Distributed Ledger Technology*. Inverted Forest Publishing, third edition, 2019, ISBN 978-1793471734, 289 pp.
- [9] Bashir, I. *Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks*. Birmingham: Packt Publishing, 2017, ISBN 978-1787125440.
- [10] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. [online], 2008, [2022-04-28]. Available from: <https://bitcoin.org/bitcoin.pdf>
- [11] Jimi, S. How does blockchain work in 7 steps — A clear and simple explanation. [online], 2018, [2022-04-28]. Available from: <https://blog.goodaudience.com/blockchain-for-beginners-what-is-blockchain-519db8c6677a>
- [12] Brünjes, L.; Vinogradova, P. *Plutus: Writing reliable smart contracts*. Input Output HK, 2019. Available from: <https://leanpub.com/plutus-smart-contracts>
- [13] Chaudhry, N.; Yousaf, M. M. Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities. [online], 2018, [2022-04-28]. Available from: https://www.researchgate.net/publication/330880555_Consensus_Algorithms_in_Blockchain_Comparative_Analysis_Challenges_and_Opportunities
- [14] Wahab, A.; Memood, W. Survey of Consensus Protocols. [online], 2018, [2022-04-05]. Available from: <https://arxiv.org/ftp/arxiv/papers/1810/1810.03357.pdf>
- [15] Szabo, N. Multinational Small Business. [online], 1993, [2022-04-05]. Available from: <https://nakamotoinstitute.org/multinational-small-business/>
- [16] Szabo, N. Smart Contracts Glossary. [online], 1995, [2022-04-05]. Available from: <https://nakamotoinstitute.org/smart-contracts-glossary/>
- [17] Jones, M. P. Plutus Tx: compiling Haskell into Plutus Core. [online], 2021, [2022-04-05]. Available from: <https://iohk.io/en/blog/posts/2021/02/02/plutus-tx-compiling-haskell-into-plutus-core/>
- [18] Input Output Hong Kong. Learn about Marlowe. [online], [2022-04-06]. Available from: <https://docs.cardano.org/marlowe/learn-about-marlowe>

-
- [19] López-Pintado, O. Caterpillar: A BPMN-based BPMS for Ethereum. [online], [2022-04-06]. Available from: <https://github.com/orlenysl/Caterpillar>
- [20] CCMi Research at the Czech Technical University in Prague. DasContract: A visual language to define contracts between people, companies, and governments. [online], [2022-04-06]. Available from: <https://github.com/CCMiResearch/DasContract>
- [21] Sharma, P.; Jindal, R.; et al. A review of smart contract-based platforms, applications, and challenges. [online], 2021, [2022-04-21]. Available from: https://www.researchgate.net/publication/357850224_A_review_of_smart_contract-based_platforms_applications_and_challenges
- [22] Iansiti, M.; Lakhani, K. R. Harvard Business Review: The Truth About Blockchain. [online], 2017, [2022-04-06]. Available from: <https://hbr.org/2017/01/the-truth-about-blockchain>
- [23] ethereum.org. The history of Ethereum. [online], 2022, [2022-04-08]. Available from: <https://ethereum.org/en/history/>
- [24] ethereum.org. Consensus Mechanisms. [online], 2022, [2022-04-08]. Available from: <https://ethereum.org/en/developers/docs/consensus-mechanisms/>
- [25] ethereum.org. Ethereum Glossary. [online], 2022, [2022-04-09]. Available from: <https://ethereum.org/en/glossary/>
- [26] ethereum.org. Upgrading Ethereum to radical new heights. [online], 2022, [2022-04-08]. Available from: <https://ethereum.org/en/upgrades/>
- [27] ethereum.org. Solidity 0.8.14 Documentation. [online], [2022-04-09]. Available from: <https://docs.soliditylang.org/en/latest/>
- [28] Quantum Systems Limited. Solidity – Ethereum’s Programming Language for Smart Contraction. [online], 2018, [2022-04-10]. Available from: <https://www.qualium-systems.com/blog/blockchain/solidity-ethereums-programming-language-for-smart-contraction/>
- [29] IOHK Limited. About Input Output - IOHK. [online], [2022-04-10]. Available from: <https://iohk.io/en/about/>
- [30] IOHK Limited. Documentation for the Cardano ecosystem. [online], [2022-04-10]. Available from: <https://docs.cardano.org/>
- [31] IOHK Limited. Cardano Roadmap. [online], [2022-04-10]. Available from: <https://roadmap.cardano.org/en/>

- [32] Kiayias, A.; Russell, A.; et al. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. [online], 2019, [2022-04-10]. Available from: <https://eprint.iacr.org/2016/889.pdf>
- [33] IOHK Limited. Plutus 1.0.0 documentation. [online], [2022-04-28]. Available from: <https://plutus.readthedocs.io/en/latest/>
- [34] Joeris, B. Haskell Fundamentals Part 2. [online], 2014, [2022-04-11]. Available from: <https://app.pluralsight.com/library/courses/haskell-fundamentals-part2/>
- [35] Mike Thoma. CORE Report: Cardano (Abridged). [online], 2021, [2022-04-11]. Available from: <https://www.cryptoeq.io/corereports/cardano-abridged>
- [36] Hornáčková, B.; Skotnica, M.; et al. Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering. [online], 2019, [2022-04-30]. Available from: https://www.researchgate.net/publication/330008379_Exploring_a_Role_of_Blockchain_Smart_Contracts_in_Enterprise_Engineering_8th_Enterprise_Engineering_Working_Conference_EEWC_2018_Luxembourg_Luxembourg_May_28_-_June_1_2018_Proceedings
- [37] Skotnica, M.; Pergl, R. Towards Model-Driven Smart Contract Systems – Code Generation and Improving Expressivity of Smart Contract Modeling. [online], 2020, [2022-04-30]. Available from: https://www.researchgate.net/publication/338360280_Das_Contract_-_A_Visual_Domain_Specific_Language_for_Modeling_Blockchain_Smart_Contracts
- [38] Skotnica, M.; Klicpera, J.; et al. Towards Model-Driven Smart Contract Systems – Code Generation and Improving Expressivity of Smart Contract Modeling. [online], 2020, [2022-04-30]. Available from: <http://ceur-ws.org/Vol-2825/paper1.pdf>
- [39] Drozdík, M. *Open-Source Legal Process Designer in .NET Blazor*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, Prague, 2020.
- [40] Ančinec, P. *Domain-Specific Languages for Off-chain UI in Decentralized Applications*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, Prague, 2021.
- [41] Frait, J. *Generating Ethereum Smart Contracts from DasContract Language*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology, Prague, 2020.

-
- [42] Microsoft. .NET documentation. [online], [2022-04-24]. Available from: <https://docs.microsoft.com/en-us/dotnet/fundamentals/>
- [43] Microsoft. ASP.NET Core Blazor. [online], [2022-04-24]. Available from: <https://docs.microsoft.com/cs-cz/aspnet/core/blazor/>
- [44] Chris Sainty. GitHub Repository - Blazored LocalStorage. [online], [2022-04-24]. Available from: <https://github.com/Blazored/LocalStorage>
- [45] bpmn-io. dmn-js - DMN for the web. [online], [2022-04-24]. Available from: <https://github.com/bpmn-io/dmn-js>
- [46] bpmn-io. bpmn-js - BPMN 2.0 for the web. [online], [2022-04-24]. Available from: <https://github.com/bpmn-io/bpmn-js>
- [47] Shopify. Liquid. [online], [2022-04-24]. Available from: <https://shopify.github.io/liquid/>
- [48] Mike Bridge. GitHub Repository - Liquid.NET. [online], [2022-04-24]. Available from: <https://github.com/mikebridge/Liquid.NET>
- [49] Alexandre Mutel. GitHub Repository - scriban. [online], [2022-04-24]. Available from: <https://github.com/scriban/scriban>
- [50] The UN Refugee Agency. Convention and Protocol Relating to the Status of Refugees. [online], [2022-01-08]. Available from: <https://www.unhcr.org/3b66c2aa10>
- [51] European Council on Refugees and Exiles. Our Work. [online], [2022-01-08]. Available from: <https://ecre.org/our-work/>
- [52] Asylum Information Database. Admissibility, responsibility and safety in European asylum procedures. [online], [2022-01-08]. Available from: https://asylumineurope.org/wp-content/uploads/2020/11/admissibility_responsibility_and_safety_in_european_asylum_procedures.pdf
- [53] The European Parliament and the Council of the European Union. Regulation (EU) No 604/2013 of the European Parliament and of the Council. [online], [2022-01-08]. Available from: <https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2013:180:0031:0059:en:PDF>
- [54] European Council on Refugees and Exiles. Short overview of the asylum procedure. [online], [2022-01-08]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/general/short-overview-asylum-procedure/>

- [55] European Council on Refugees and Exiles. Country Report: Statistics. [online], [2022-01-08]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/statistics/>
- [56] European Council on Refugees and Exiles. Flow Chart: Republic of Ireland. [online], [2021-11-26]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/general/flow-chart/>
- [57] European Council on Refugees and Exiles. List of Authorities Intervening in each Stage of the Procedure: Republic of Ireland. [online], [2021-11-26]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/general/list-authorities-intervening-each-stage-procedure/>
- [58] An Garda Síochana. Immigration (GNIB). [online], [2021-11-26]. Available from: [https://www.garda.ie/en/about-us/organised-serious-crime/immigration-gnib-/](https://www.garda.ie/en/about-us/organised-serious-crime/immigration-gnib/)
- [59] International Protection Office. Our Responsibilities. [online], [2021-11-26]. Available from: <http://www.ipo.gov.ie/en/ipo/pages/internationalprotection>
- [60] The International Protection Appeals Tribunal. About Us. [online], [2021-11-26]. Available from: http://www.protectionappeals.ie/website/rat/ratweb.nsf/page/about_the_tribunal-en
- [61] Courts Service. High Court. [online], [2021-11-26]. Available from: <https://www.courts.ie/high-court>
- [62] European Council on Refugees and Exiles. Regular Procedure: Republic of Ireland. [online], [2021-11-26]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/procedures/regular-procedure/>
- [63] Department of Justice. About Us. [online], [2021-11-26]. Available from: https://www.justice.ie/en/JELR/Pages/About_Us
- [64] European Council on Refugees and Exiles. Border Procedure: Republic of Ireland. [online], [2021-11-26]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/procedures/border-procedure-border-and-transit-zones/>
- [65] European Council on Refugees and Exiles. Dublin Procedure: Republic of Ireland. [online], [2021-11-26]. Available from: <https://asylumineurope.org/reports/country/republic-ireland/asylum-procedure/procedures/dublin/>

- [66] CEF Digital. EBSI Documentation: What is EBSI? [online], [2022-01-09]. Available from: <https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=381517902>
- [67] CEF Digital. EBSI Documentation: Smart Contracts. [online], [2022-01-09]. Available from: <https://ec.europa.eu/cefdigital/wiki/display/EBSIDOC/Smart+Contracts>
- [68] ethereum.org. Remix - Ethereum IDE. [online], [2022-04-30]. Available from: <https://remix.ethereum.org/>

Acronyms

ABI	Application Binary Interface
BORM	Business Objects Relation Modelling
BPEL	Business Process Execution Language
BPM	Business Process Modeling
BPMN	Business Process Modeling Notation
DAO	Distributed Autonomous Organization
DeFi	Decentralized Finance
DEMO	Design & Engineering Methodology for Organizations
DMN	Decision Model and Notation
DoS	Denial of Service
DRD	Decision Requirements Diagram
DRG	Decision Requirements Graph
EVM	Ethereum Virtual Machine
FEEL	Friendly Enough Expression Language
GHC	Glasgow Haskell Compiler
GNIB	Garda National Immigration Bureau
IOHK	Input Output Hong Kong Limited
IPA	International Protection Act
IPAT	International Protection Appeals Tribunal

A. ACRONYMS

IPO International Protection Office

JSON JavaScript Object Notation

PBFT Practical Byzantine Fault Tolerance

PoS Proof of Stake

PoW Proof of Work

UML Unified Modeling Language

UTXO Unspent Transaction Output

SOA Service Oriented Architectures

S-FEEL Simplified Friendly Enough Expression Language

XML Extensible Markup Language

Contents of enclosed CD

	readme.md.....	the file with CD contents description
	models.....	the thesis model directory
	class-diagrams.....	class diagrams of the DasContract Editor
	dmn-and-bpmn-models.....	models for the asylum procedure
	src.....	the directory of source codes
	code-examples.....	code examples in Solidity and Plutus
	dascontrac-editor.....	implementation sources
	thesis.....	the directory of \LaTeX source codes of the thesis
	text.....	the thesis text directory
	thesis.pdf.....	the thesis text in PDF format