University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of an Intent-based Blockchain Selection Framework

Patrick Widmer
Zürich, Switzerland
Student ID: 13-786-009

Supervisor: Eder J. Scheid, Bruno B. Rodrigues,
Prof. Dr. Burkhard Stiller
Date of Submission: January 8, 2020

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

In den letzten Jahren hat die Bekanntheit von Kryptowährungen stetig zugenommen. Die zugrundeliegende Technologie, ein verteiltes Kontenbuch, wird als Blockchain bezeichnet. Heute existieren zahlreiche Blockchainimplementierungen. Die Auswahl einer geeigneten Implementierung für eine bestimmte Anwendung ist komplex. Kürzlich wurden Ansätze vorgeschlagen, die diesen Prozess automatisieren. Diese Ansätze setzen auf policy-basierte Verwaltung. Die Formulierung dieser Policies setzt ein technisches Verständnis der zugrundeliegenden Implementierung voraus. Deshalb ist das Ziel dieser Arbeit die Entwicklung eines Prototyps, der die bestehende Lösung erweitert und die zugrundeliegenden Implementierungsdetails abstrahiert. Der Ansatz ermöglicht die Formulierung von Intents in natürlicher Sprache. Eine Zustandsmaschine wird verwendet um diese Intents in Auswahlpolicies umzuwandeln. Dieser Ansatz basiert auf früheren Lösungen aus dem Bereich der Netzwerkverwaltung. Die Resultate der Performanzauswertung des Prototyps zeigen einen vernachlässigbaren Mehraufwand. Die Anforderungen an die Blockchaintechnologie von Benutzern wird anhand einer Umfrage untersucht. Die Resultate dieser Umfrage zeigen, dass technische wie nicht-technische Benutzer gleichermassen von einem Intent-basierten Ansatz profitieren. Der entwickelte Protoyp unterstützt Parameter, die in den Auswahlpolicies nicht verfügbar sind. Die Implementierung des Policysystems muss erweitert werden um diese zusätzlichen Parameter zu unterstützen.

In the last years, cryptocurrencies have become increasingly popular. The underlying technology a distributed ledger is referred to as a blockchain. Nowadays, there is a wide variety of blockchain implementations. The selection of a suitable implementation for a particular application is complex. Recently, approaches have been proposed to automate this process. Specifically, these approaches are relying on policy-based management. However, the specification of these policies still requires a technical understanding of the underlying implementation aspects. Therefore, the goal of this thesis is the development of a prototype that extends the existing solution to abstract these underlying implementation details. The approach allows the specification of intents in natural language. Further, a state-machine-based refinement technique is proposed to transform these intents into low-level blockchain selection policies. It is inspired by previous approaches from network management. The results of the performance evaluation of the prototype implementation show a negligible overhead. Further, the blockchain usage requirements of users are assessed in a survey. The results of the survey suggest that technical and non-technical individuals benefit from an intent-based approach equally. The developed prototype supports parameters that are not available in the low-level policies. Therefore, the implementation of the policy system has to be extended to support the remaining options.

Acknowledgments

I am very grateful to a number of people who have helped me, directly or indirectly, with my work on this thesis. Without their continuous support, this work would not have been possible.

First and foremost, I would like to thank Eder John Scheid for his enduring support, technical feedback, and experienced guidance throughout this thesis. I would also like to thank Bruno Bastos Rodrigues for his valuable feedback. Moreover, I would like to thank Prof. Dr. Burkhard Stiller for the opportunity to work on this thesis.

I want to thank all the CSG members for their feedback during the midterm presentation and the participation in the survey. I would also like to thank all the students of the Blockchain CAS and the Blockchains and Overlay Networks course that participated in the survey. Finally, I would like to thank all the remaining participants of the survey.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 2

1.3 Thesis Outline 2

2 Background **3**

2.1 Blockchain 3

2.1.1 Consensus Mechanism 3

2.1.2 Deployment Type 4

2.1.3 Performance 5

2.1.4 Transaction Costs 5

2.2 Policy-based Network Management (PBNM) 5

2.2.1 PBNM Architecture 6

2.3 Intent-based Networking (IBN) 7

2.4 Policy Refinement 8

2.5 Policy-based Blockchain Agnostic Framework 8

3 Related Work **11**

3.1 Network Management 11

3.2 Cloud Management 12

3.3 Discussion 12

4 Intent Refinement Approach **15**

4.1 Design 15

4.1.1 Intent Refinement 15

4.1.2 Intent Grammar 16

4.1.3 Intent Parameters 16

4.1.4 Intent Refinement Toolkit (IRTK) 20

4.1.5 Intent Parsing 21

4.1.6 Intent Translation 24

4.1.7 Intent Validation 25

4.2 Implementation 25

4.2.1 Intent Refinement 26

4.2.2 Intent Parsing 27

4.2.3 Intent Translation 30

4.2.4 Intent Validation 31

4.2.5 Intent 32

4.2.6 Policy 33

5 Evaluation **35**

5.1 Performance Testing 35

5.1.1 Test System 35

5.1.2 Results 36

5.1.3 Discussion 41

5.2 Survey on Blockchain Usage Requirements 42

5.2.1 Discussion 48

6 Summary and Future Work **51**

6.1 Future Work 52

Abbreviations **57**

Glossary **59**

List of Figures **59**

List of Tables **62**

A Questionnaire **65**

B Installation Guidelines **71**

B.1 Dependencies 71

B.2 Database 72

B.3 Configuration 72

B.4 Tests 73

B.5 Usage 73

B.6 Troubleshooting 74

C Contents of the CD **77**

Chapter 1

Introduction

In 2009, the Bitcoin whitepaper was released [25]. It introduced an electronic cash system. Bitcoin does not rely on a central entity to coordinate the system. Instead, it records transactions in a decentralized manner using a peer-to-peer network. The underlying technology, a distributed ledger is known as a blockchain. The system is maintained by nodes in the network, which validate incoming transactions and compete against each other to collect a reward provided by the protocol.

Since then, many blockchains have been developed [2]. However, the underlying protocols differ in specific design and implementation decisions. These characteristics have implications for different applications and use cases. Therefore, there is no blockchain implementation that suits every use case. Initially, the blockchain technology was used for cryptocurrencies in the financial industry. Nowadays, blockchains are applied in various industries, such as the pharmaceutical industry [4]. Moreover, governments are experimenting with the technology to develop electronic voting solutions [39].

1.1 Motivation

In the last years, blockchain technology has gained a lot of popularity. Initially, mainly technical individuals have worked with blockchain technology. However, the media coverage of blockchains has attracted a wide range of users. Still, to interact with these blockchain platforms users are required to understand underlying technical aspects, such as transaction models and private key mechanisms [12].

Therefore, the selection of a blockchain is becoming a limiting factor in the adoption of blockchain technology. There are approaches that try to automate the process and support users in selecting a well-suited blockchain implementation. For example, there is an approach that is inspired by policy-based management which has been already applied to network management [32]. The approach defines policies or blockchain selection rules that define a set of criteria to select a blockchain based on these conditions. However, the specification of policies still requires a technical understanding of the blockchain implementations.

1.2 Description of Work

This thesis explores an intent-based approach for blockchain selection. It introduces an additional layer of abstraction to the existing policy-based blockchain selection. Similarly, to intent-based networking for policy-based network management. The approach allows the specification of intents (*i.e.*, high-level abstract policies) in natural language. These intents are then refined into corresponding low-level policies. The advantage of this approach is that it abstracts low-level implementation details.

1.3 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides information on blockchain characteristics, policy- and intent-based network management and policy refinement. Moreover, it also discusses the concept of a policy-based blockchain agnostic framework. Chapter 3 discusses intent-based management approaches in various contexts. The design and implementation of a prototype is discussed in Chapter 4. Chapter 5 provides the results of the performance testing of the prototype. Furthermore, it presents and discusses the results of a survey on the blockchain usage requirements. Finally, Chapter 6 summarizes the most important findings and provides an outlook for potential extensions of this work.

Chapter 2

Background

This chapter describes the main concepts involved in this thesis. Section 2.1 introduces blockchains and lists the parameters and mechanisms that support them. Then, in Section 2.2 the Policy-based Network Management (PBNM) concept is presented with policy format examples. The core concept of this thesis, Intent-based Networking (IBN), is described in Section 2.3. Techniques to refine abstract high-level policies (*i.e.*, intents) into low-level policies are presented in Section 2.4. Finally, Section 2.5 describes a Policy-based Blockchain Agnostic Framework (PBBAF) and its components.

2.1 Blockchain

A blockchain is a distributed append-only immutable ledger [25]. It consists of an ordered, linear list of blocks. It is replicated across all the nodes of the network. In contrast to traditional databases, there is no central authority as a consensus mechanism ensures that the copies of the nodes are identical. The data of the blockchain is maintained in blocks, which contain a pointer to the previous block. A block maintains transaction data (payload), a pointer to the previous block, a cryptographically hashed value of a crypto puzzle, and a timestamp.

2.1.1 Consensus Mechanism

A consensus mechanism determines the participant that is able to append blocks to the blockchain. Consensus mechanisms are important because they help the blockchain network to maintain the same state across all peers and to prevent the double spending of coins. The creation of valid blocks is often referred to as *mining* in Proof-of-Work (PoW)-based blockchains. However, there are different consensus mechanisms with various characteristics [41].

PoW was introduced in the Bitcoin white paper [25]. In such a mechanism, nodes in the network compete against each other to solve a crypto puzzle. When a node has found a

solution to the crypto puzzle, the other nodes validate the result. A state of consensus is reached when the majority of the nodes accept the output of the mining. The node that successfully mined a block receives a reward. The reward of a successful inclusion are tokens of this blockchain. It consists of transaction fees and possibly a block reward provided by the protocol. The main disadvantage of this approach is the excessive power consumption [28].

Proof-of-Stake (PoS) is an alternative that addresses this issue of power consumption. It pseudo-randomly selects block creators and validators, based on a node's wealth and tokens at stake. Even though PoS reduces the power consumption compared to PoW, scalability is still an issue. In delegated Proof-of-Stake (dPoS) the nodes select a set of nodes as their delegates (witnesses). Each delegate is given a turn to create a block. After all the delegates have had a turn, they are shuffled again. Proof-of-Authority (PoA) relies on special validator nodes to create new blocks. Nodes earn the position of a validator. Each validator has a reputation attached, as an incentive to retain the position. However, these consensus mechanisms present a higher a degree of centralization because the number of delegates is limited. There are other consensus mechanisms [41], and each one of these implementations offers a trade-off between performance and centralization. Thus, one must consider those when selecting the most suitable blockchain.

2.1.2 Deployment Type

Blockchains can be classified according to the maintainers and writing and reading access rights of the distributed ledger. There are private and public blockchains [42]. Public blockchains are also referred to as permissionless, and private blockchains are also referred to as permissioned.

- A **public blockchain** can be accessed and used by anyone following the respective protocol.
- A **private blockchain** is controlled by a single organization that manages the permission.
- A **private group-based (consortium) blockchain** is controlled by a consortium with known members.

Public blockchains are completely distributed and decentralized. Access to the network is unrestricted. All nodes are equal. All nodes can create blocks and perform validation. Public blockchains are transparent. However, the operation of a public blockchain results in excessive power consumption. Also, the data is publicly available and can be read by anyone. Private blockchains restrict the access to the network. Thus, they are not completely decentralized. Not every node might be able to read the data. Usually, only selected nodes can create blocks and perform validation. Private blockchains decrease the power consumption. However, it is not clear whether they can be considered real blockchains or not, because they often can be replaced by generic databases.

2.1.3 Performance

There are different factors that affect the performance of a blockchain [10]. For example, the deployment type and consensus mechanism affect the performance. In general, public blockchains tend to be slower than private ones. Mainly, because public blockchains require computationally expensive consensus mechanisms to secure the blockchain state due to the presence of untrusted stakeholders.

There are different approaches to measure the performance of a blockchain [10]. The throughput or transaction rate of a blockchain is defined as the maximum number of transactions that can be confirmed over a specific period of time [13]. It depends on different factors. For example, the block size determines the maximum number of transactions that can be stored within a single block. Another factor is the latency or block time of a blockchain. It is defined as the interval in which a new block is appended to the blockchain. For example, Bitcoin creates a new block every 10 minutes containing 2200 transactions on average and Ethereum [5] creates a new block every 14 seconds with 120 transactions on average [3]. Thus, Bitcoin's transaction rate can be calculated using Equation (2.1) resulting in 3.6 transactions per second, and Ethereum's transaction rate results in 8.6 transactions per seconds as shown in Equation (2.2).

$$3.6 \text{ tps} = \frac{2200 \text{ transactions}}{600 \text{ seconds}} \tag{2.1}$$

$$8.6 \text{ tps} = \frac{120 \text{ transactions}}{14 \text{ seconds}} \tag{2.2}$$

2.1.4 Transaction Costs

Different blockchains use different formulas to calculate transaction costs or transaction fees. PoW-based blockchains provide a block reward to miners. Theoretically, transaction fees are optional. However, miners are free to choose which transactions they want to include in a block. Therefore, in practice transaction fees are used as an additional incentive to miners to include particular transactions. Some blockchains, *e.g.*, Stellar and EOS define a fixed transaction fee. For example, Stellar calculates the transaction fee based on the computational power used by a transaction. The formula used by Stellar is shown in Equation (2.3) [17].

$$transaction\ fee = number\ of\ operations \times base\ fee \tag{2.3}$$

2.2 Policy-based Network Management (PBNM)

In the past, network management became increasingly complex, due to an increasing number of devices involved. Moreover, the types of involved devices have become more

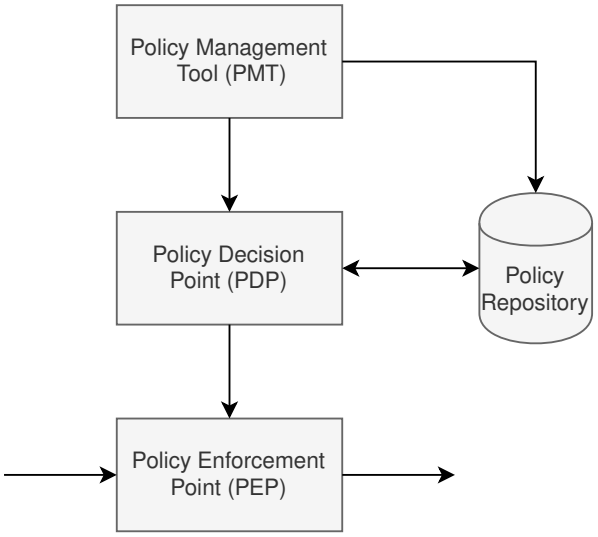

Figure 2.1: PBNM architecture

diverse. This lead to the creation of approaches tied to specific situations and configurations, in which the underlying process relies heavily on manual interactions by network operators. Therefore, new approaches were researched and developed to automate this process, reducing the manual work of network operators and configuration errors. One such approach is Policy-based Network Management (PBNM) [40]. The management of configurations with policies was originally proposed for the management of distributed systems [36].

2.2.1 PBNM Architecture

A policy is a sequence of Event-Condition-Action (ECA) rules [6]. ECA rules are part of event-driven computing. In event-driven computing, actions are triggered by events given a specified condition is satisfied. Originally, ECA rules were used in active databases [26].

There are different approaches to specify policy. The Internet Engineering Task Force (IETF) policy specification is based on a sequence of rules. A rule is a condition-action pair. This approach is simple compared to the Ponder policy specification [14] which is more complex. Ponder distinguishes between different types of policies. There are access control and obligation policies. Access control policies are further divided into authorization, delegation, and refrain policies.

The PBNM architecture is shown in Figure 2.1. It is composed of four components:

- The **Policy Management Tool (PMT)** is used by an operator to define and update policies. Also, the PMT validates and refines policies. It stores policies in the policy repository and notifies the PDP about changes.
- The **Policy Repository** stores the policies in an interoperable, *i.e.*, vendor- and device-agnostic format.

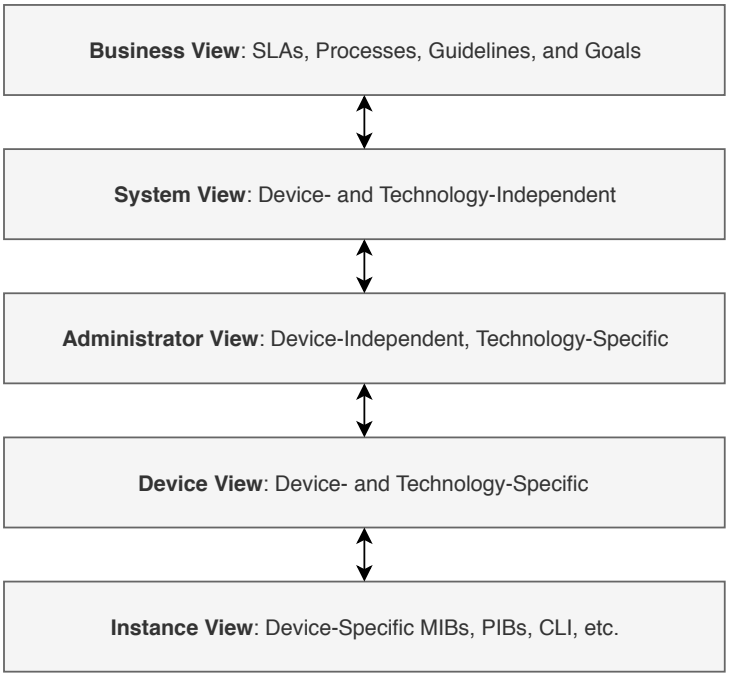

Figure 2.2: Policy continuum

- The **Policy Decision Point (PDP)** evaluates the policies from the repository. It installs new devices or rules and communicates policies to the PEP.

- The **Policy Enforcement Point (PEP)** applies and executes the policies from the PDP. For example, by routing incoming packets.

2.3 Intent-based Networking (IBN)

Formally, an intent in the context of autonomic networking is defined in Request for Comments (RFC) 7575 [1] by the IETF as "an abstract, high-level policy used to operate the network". In this sense, IBN can be viewed as an evolution of PBNM. It addresses the business view, *i.e.*, the layer with the highest level of abstraction, from the policy continuum [15].

The policy continuum is a layered concept which is shown in Figure 2.2. Each layer corresponds to a view with a particular abstraction level. These layers allow policy authors to express policies based on their technical knowledge and requirements. The main goal of IBN is the specification of high-level objectives in a declarative manner without knowing all the underlying implementation details. It enables individuals without a technical background and understanding to configure networks. Intents are translated into low-level policies in a dedicated refinement process.

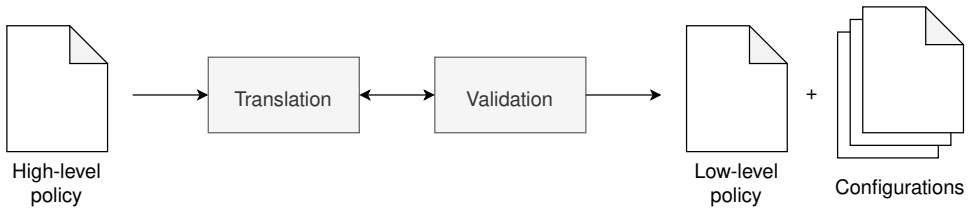

Figure 2.3: Generic refinement process

2.4 Policy Refinement

Policy refinement is the process of translating high-level policies into low-level policies and configurations [29, 34, 33]. A generic refinement process is depicted in Figure 2.3. High-level policies correspond to the business or system view of the policy continuum [15]. Low-level policies address the administrator, device, or instance view of the policy continuum. High-level policies declaratively describe the desired state of a system, *i.e.*, they do not specify how this state shall be achieved. Low-level policies target options for particular technologies or even devices. Therefore, in the refinement process, options from high-level policies are mapped to low-level options. Usually, refinement includes validation mechanisms, *e.g.*, to identify invalid or conflicting policies [24, 8, 9].

There are three types of approaches to refinement [29], namely rule-based, classification-based refinement and Case-Based Reasoning (CBR), and logical based approaches of refinement. Rule-based approaches are the most automated but also more domain-specific, *i.e.*, less generic. Logical-based approaches are more generic but also harder to model. Classification-based refinement and CBR is less complete than the other approaches, *e.g.*, it does not support verification or conflict detection.

2.5 Policy-based Blockchain Agnostic Framework

Recently, a Policy-based Blockchain Agnostic Framework (PBBAF) has been proposed [30]. The goal of the PBBAF is to combine policy-based management with a modular blockchain interoperability API to automate the process of selecting a blockchain to store incoming transactions. Figure 2.4 shows the architecture of the proposed framework. It is composed of the following components:

- The **Connector** is the entry point for users to interact with the framework.
- The **Blockchain Costs Monitor** periodically retrieves variable blockchain parameters, *e.g.*, transaction fees.
- The **Analytical Solver** periodically calculates usage scenarios, *e.g.*, cost thresholds.
- The **Database** stores various data, *e.g.*, transaction templates for different blockchains.
- The **OpenAPI** provides adapters to store transactions on different blockchains.

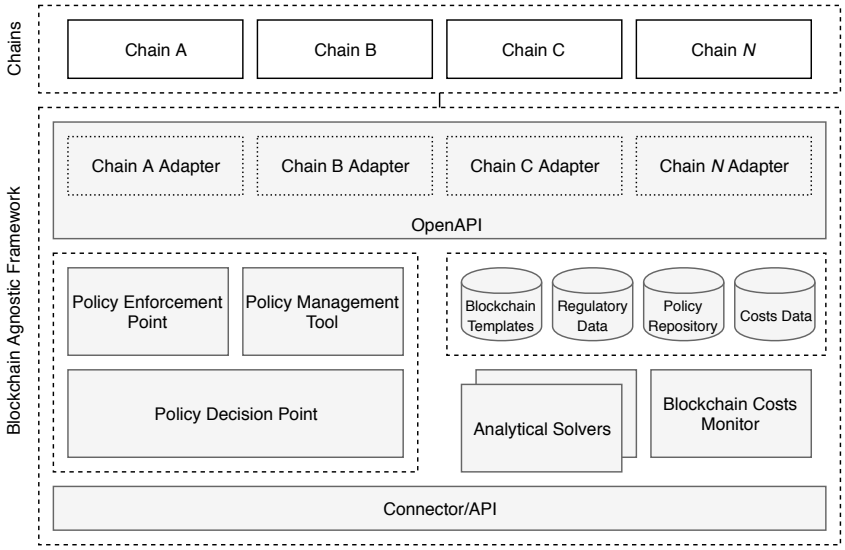

Figure 2.4: Blockchain Agnostic Framework architecture

- The **Policy-based Management** consists of PDP and PEP which are responsible for determining an active policy, and selecting a blockchain, respectively.

There are already prototype implementations available for some components of the PB-BAF. Bifrost [31, 19] is an implementation of the OpenAPI component. It implements different adapters to connect to nodes of different blockchains. It relies on transaction templates to create transactions and store data on blockchains. Already stored data can be retrieved or migrated to a different blockchain. The Bifrost prototype is implemented in Python.

Also, there is a Policy-based Blockchain Selection Framework (PBS) [22, 32] which implements the Policy-based Management component. It automates the process of selecting a blockchain based on a set of predefined requirements. It provides a GUI to manage the blockchain-selection policies. Policies are validated and stored in a repository. For every incoming transaction, it selects an active policy from all the policies in the repository. Then, this active policy determines on which blockchain the incoming transaction is stored. The PBS Framework consists of a REST API server and a client. It is implemented in JavaScript using the Express framework.

Chapter 3

Related Work

Intent-based Management is still an infant research area. However, this concept is being applied and researched in different contexts, such as network management [34, 18, 38, 20] and cloud management [7, 21]. Moreover, intent-based management is discussed by the Internet Engineering Task Force (IETF) in several drafts [11, 23, 37]. It should be noted that these drafts are valid for a limited period of time and may be replaced or become obsolete. Nevertheless, they are an indication of the growing interest and research in the intent topic.

3.1 Network Management

Researchers developed an IBN solution, called INSpIRE [34]. INSpIRE includes a refinement technique that translates intents into a set of configurations for managing service chains in various environments. The solution supports both homogeneous environments that consist only of Virtual Network Functions (VNFs), and heterogeneous environments that consist of VNFs and physical middleboxes. INSpIRE determines the VNFs that are required to fulfill an intent, chains these VNFs based on their dependencies, and presents low-level details to network devices for posterior traffic steering.

In [18], the authors propose an intent-based approach to manage Virtual Networks (VNs) based on software-defined Network Virtualization (NV). The goal of the approach is to automate the management and configuration of VNs based on intents, *i.e.*, high-level requirement specifications. The implementation is based on an open-source Software-Defined Networking (SDN) controller. Moreover, the approach is able to provide multiple VNs over the same physical infrastructure.

An intent-based approach has been proposed to create secure services based on SDN network orchestration [38]. The approach relies on SDN network orchestration and intents to automate the configuration and deployment of secure services. In an experimental evaluation, the authors found that the overhead of the approach is negligible.

An extension of IBN has been proposed [20]. The approach enhances the refinement process by employing machine-learning techniques and feedback from the network operators.

They introduce an intermediate representation extracted from intents specified in natural language. The prototype translates intents from natural language into the intermediate representation and finally, into low-level network configuration rules.

In [16], the authors describe an intent-based approach to discover and deploy networks. The intents use a verb-object-subject sentence structure and are specified as tuples. The refinement of the intents is based on an ontology which is composed of hierarchical categories of operations. They use *Maat* agents to mediate between user intents and policies of network operators. There are still open challenges for the proposed approach such as scalability of the deployment and security-related aspects.

3.2 Cloud Management

In [7], the authors propose an intent-based approach to cloud service management. The objective of the approach is to automate or support the decision-making process for cloud resources of cloud operators. The approach accepts the requirements of cloud users as intents in a declarative manner. The authors provide a prototype implementation and evaluate its performance.

In [21], the authors propose an intent-based approach to manage cloud infrastructure. The approach distinguishes between high-level intents from the underlying infrastructure implementations. In contrast to existing systems that resolve conflicts during run-time, the approach detects and resolves conflicts already during the specification. The approach is called label management service and improves scalability.

3.3 Discussion

In network management, there are intent-based approaches that rely on natural language for intent specification. However, the refinement of these approaches is more complex than the refinement described in this thesis. The refinement of INSpIRE [34] relies on softgoals and operationalizations. The refinement process described in [20] relies on machine learning techniques. The platform described in [18] includes a refinement process but also enforces the policies. There are also intent-based approaches in network management that do not rely on natural language to specify intents. Specifically, intents in [38] are specified as a set of key-value pairs in the JSON format. Furthermore, there is an approach [16] that is based on verb-object-subject tuples.

In cloud management, there are intent-based approaches that rely on natural language to specify intents. However, the underlying refinement processes are also more complex than the refinement described as part of this thesis. The refinement process described in [7] employs linear regression techniques to calculate parameters. Similarly, the refinement process used in [21] relies on complex label trees. Table 3.1 provides an overview of the similarities and differences in refinement.

Table 3.1: Overview of related work

	Intent specification	**Refinement**	**Area**
[34]	Natural language	Softgoals and operationalizations	Network management
[20]	Natural language	Machine learning	Network management
[18]	Natural language	Multi-layer translation	Network management
[38]	JSON	N/A	Network management
[16]	Tuples	Ontology	Network management
[7]	Natural language	Linear regression	Cloud management
[21]	Natural language	Label trees	Cloud management
This Work	Natural language	State machine and lookup tables	Blockchain management

In summary, there are various intent-based approaches in network management and in cloud management. In most of these approaches, intents are specified in natural language. However, the refinement processes of these approaches differ and are tied to specific use cases. Therefore, these approaches cannot be directly applied to a novel context such as blockchain selection.

Chapter 4

Intent Refinement Approach

This chapter describes an approach to refine high-level policies, *i.e.*, intents to low-level policies, in the context of blockchain selection. Section 4.1 describes the design of the refinement process. Section 4.2 discusses an implementation of the refinement.

4.1 Design

The refinement process is described in Section 4.1.1. Section 4.1.2 provides a formal definition of an intent grammar for an intent specification language. The available options are explained in Section 4.1.3. The Intent Refinement Toolkit (IRTK) is a prototype of the refinement process and is discussed in Section 4.1.4. The parsing process and the underlying state machine are described in Section 4.1.5. Section 4.1.6 describes the translation step. Finally, the validation logic is discussed in Section 4.1.7.

4.1.1 Intent Refinement

The refinement process is depicted in Figure 4.1. It takes an abstract high-level policy, *i.e.*, an intent as input. In a first phase, this intent is parsed and validated. In case the given intent is complete and valid, the output is a parsed intent. The parsed intent is an instance of an intermediate data structure storing all the parsed options. In a second phase, the parsed intent is translated into a set of low-level policies. Again, the translation

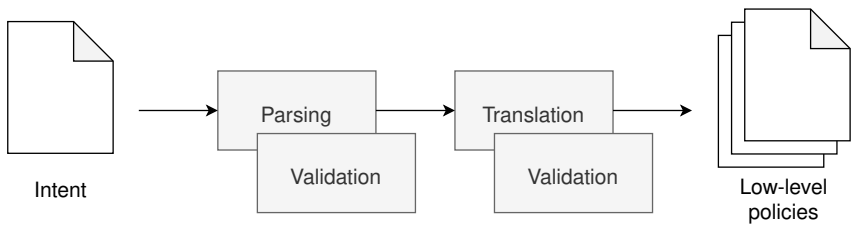

Figure 4.1: Refinement process

also validates the parsed intent. Finally, if there is no validation error, the translation returns a set of low-level policies.

4.1.2 Intent Grammar

Listing 4.1 specifies the grammar of the intent language using the Extended Backus Naur Form (EBNF) [35]. EBNF is an extension of the Backus Naur Form (BNF), which additionally supports the specification of options and repetitions. BNF and EBNF are notation techniques that allow the formal specification of context-free grammars. They are often used to specify the syntax of programming languages. The notation is explained in Table 4.1. Moreover, commas indicating sequences are omitted, for clarity. Further, trailing semicolons or dots indicating the end of the right-hand-side expression are not used. Similarly, whitespace is omitted from the specification. Instead, every symbol in a sequence is implicitly separated by whitespace.

```
1  <intent> ::= "for" <users> [ "in" "the" <timeframe> ] "select" ( "the"
2      <profile> [ <filters> ] "blockchain" [ ( "from" | "except" )
3      <blockchains> ] | <blockchain> ) [ "with" <modifiers> ] ( "until"
4      "the"  <interval> "costs" "reach" [ <currency> ] <threshold> | "as"
5      "default" )
6  <users> ::= <user> { ( "," | "and" ) <user> }
7  <user> ::= character { character }
8  <timeframe> ::= "day" | "night" | "morning" | "afternoon"
9  <profile> ::= "cheapest" | "fastest"
10 <filters> ::= <filter> { ( "," | "and" ) <filter> }
11 <filter> ::= "private" | "public" | "fast" | "cheap" | "stable"
12     | "popular"
13 <blockchains> ::= <blockchain> { ( "," | "and" ) <blockchain> }
14 <blockchain> ::= "Bitcoin" | "EOS" | "Ethereum" | "Hyperledger"
15     | "IOTA" | "Multichain" | "Stellar"
16 <modifiers> ::= <modifier> { ( "," | "and" ) <modifier> }
17 <modifier> ::= "encryption" | "redundancy" | "splitting"
18 <interval> ::= "daily" | "weekly" | "monthly" | "yearly"
19 <currency> ::= "CHF" | "EUR" | "USD"
20 <threshold> ::= integer | real
```

Listing 4.1: Intent grammar

4.1.3 Intent Parameters

An intent can be composed by different parameters, Table 4.2 shows the classification of the available options based on their usage. *Conditions* are used by the PDP to select an active policy from a set of policies. *Selection* strategies are used by the PDP to select a blockchain from the active policy to store incoming transactions. *Filters* are used by the

Table 4.1: EBNF notation

Notation	Description
`[...]`	Optional expression
`{...}`	Repeated expression
`(...)`	Grouped expression
`<...>`	Non-terminal symbol
`"..."`	Terminal symbol
`...`	Literal class, *e.g.*, `integer` or `character`
`...|...`	Choice between expressions

Table 4.2: Classification of configuration parameters

Usage	Parameters
Condition	Users, Timeframe, Cost Interval, Cost Currency, Cost Threshold
Selection	Profile, Blockchain
Filter	Deployment Type, Transaction Rate, Transaction Costs, Maturity, Whitelist, Blacklist
Modifier	Redundancy, Encryption, Splitting

PDP to restrict the pool of blockchains. Filters can be further divided into static and dynamic options. Static filter options, such as the deployment type of a blockchain, remain constant over time. Dynamic filter options, such as transaction cost, change over time. *Modifiers* are used to alter the filtering or selection process. Most of the configuration options are directly mapped onto options available in the low-level policies. Some options, *e.g.*, the timeframe provide an abstraction of the options from the low-level policies. There are options, *e.g.*, `encryption` which are not available in the low-level policies. Finally, there are options, *e.g.*, `turing complete` which are only available in the low-level policies.

Although intents are specified in natural language, the negation of options is not supported. Negations could be implemented only for a subset of the available options, *e.g.*, `private`. The negation of `private`, *i.e.*, `not private` is equivalent to `public`. Similarly, all the options that could be negated can already be specified without negation. Therefore, it was decided to not support negation of options to avoid unnecessary complexity.

Each of these options can have different parameters. *Conditions* can be composed of users, timeframes, cost interval, cost currency and a cost threshold. The specification of at least one user is required. Timeframe and cost currency are optional. Cost-related parameters, *i.e.*, cost interval, cost currency, and cost threshold can only be specified for non-default policies. Moreover, for non-default policies the specification of a cost interval and a cost threshold is required.

Users A set of users, which can be composed of on one or more user. Policies in the PDP are organized on a per-user basis. Every user defines his/her own policies which are independent from policies defined by other users.

Timeframe The timeframe in which a policy can be activated by the PDP. The allowed values are `day`, `night`, `morning` and `afternoon`. If specified, the policy can be activated only within the specified timeframe. If omitted, the policy can be activated anytime. The starting and ending times for the timeframes can be customized by the user in a separate configuration file.

Cost Interval The interval defines at which rate the accumulated transaction costs are reset. The allowed values are `daily`, `weekly`, `monthly` and `yearly`. An interval is required, except for the default policy. The default policy does not specify an interval, because it does not have a cost threshold.

Cost Currency The currency of the specified cost threshold. The allowed values are `CHF`, `EUR` and `USD`. The default value is `USD`. In case a non-default currency is specified, the cost threshold is converted from the specified currency into US Dollars. This is necessary, because of the design of the PDP. The default policy specifies the currency which is used by all other policies and can not be changed. Therefore, US Dollars (USD) is always specified as currency in the default policy and the threshold values are converted if necessary.

Cost Threshold The threshold for the specified interval. The allowed value is either an integer or a float. A threshold is required, except for the default policy. The default policy does not specify a threshold. The PDP keeps track of the accumulated transaction costs within the specified interval. For each incoming transaction, it compares the accumulated transaction costs with the specified threshold. When the accumulated transaction costs exceed the threshold, the policy is deactivated.

Selections can be composed of profile or blockchain. Generally, the specification of a profile is required and the specification of a blockchain is optional. Moreover, they are mutually exclusive, *i.e.*, it is not possible to specify a profile and a blockchain for the same policy. However, a blockchain can be specified instead of a profile.

Profile The profile determines how the PDP selects a blockchain from the blockchain pool. The `fastest` profile selects the fastest blockchain, *i.e.*, the blockchain with the highest transaction rate. In case of a tie, the cheapest blockchain is chosen to resolve the tie. The `cheapest` profile selects the cheapest blockchain, *i.e.*, the blockchain with the lowest transaction cost. In case of a tie, the fastest blockchain is chosen to resolve the tie. Depending on the profile different filter options might be available. For example, `cheap` can only be specified with the `fastest` profile. It is ignored with the `cheapest` profile. Analogously, `fast` can only be specified with the `cheapest` profile and is ignored with the `fastest` profile.

Blockchain A particular blockchain to store incoming transactions. Currently the allowed values are `Bitcoin`, `EOS`, `Ethereum`, `Hyperledger`, `IOTA`, `Multichain`, and `Stellar`. However, the values depend on the blockchains that are supported by the PEP. Either a profile or a blockchain has to be specified. Instead of specifying a profile, a particular blockchain can be specified explicitly. If specified, transactions are always stored on the specified blockchain. Although, the `blockchain` option is listed as a selection, it is implemented as a filter option. The PDP always expects a

selection profile. It does not support the specification of a single blockchain directly. As a workaround, the blockchain pool can be restricted to only contain a particular blockchain. Therefore, the specified selection profile does not matter and the PDP will always select the only blockchain in the pool.

Filters can be composed of deployment type, transaction rate, transaction cost, maturity, and whitelist or blacklist. All the available options are optional. Moreover, `private` and `public`, and whitelist and blacklist are mutually exclusive.

Deployment Type A filter option specifying the deployment type of blockchains. The allowed values are `private` and `public`. They are mutually exclusive. It determines whether a private, public, or any type of blockchain can be used. If omitted, all types of blockchains are considered.

Transaction Rate A filter option, restricting the blockchain pool. The allowed value is `fast`. If specified, only blockchains with a transaction rate equal or higher than a threshold defined in an external configuration are considered. This option is only available in combination with the `cheapest` profile. If this option is specified in combination with the `fastest` profile, it is ignored.

Transaction Costs A filter option restricting the blockchain pool. The allowed value is `cheap`. If specified, only blockchains with a transaction cost equal or less than a threshold defined in an external configuration are considered. This option is only available in combination with the `fastest` profile. If this option is specified in combination with the `cheapest` profile, it is ignored.

Maturity Filter options restricting the blockchain pool. The allowed values are `stable` and `popular`. The values are not mutually exclusive, *i.e.*, they can be combined. They determine, whether a stable and/or popular blockchain can be used. If omitted, all blockchains are considered.

Whitelist and Blacklist Filter options restricting the blockchain pool. The allowed values are the same as the **Blockchain** parameter. A whitelist or blacklist can be specified only with a profile. In case a blockchain is specified instead of a profile, whitelist and blacklist are not available, because the blockchain pool contains only the specified blockchain anyway. If specified, the PDP only considers blockchains from the whitelist or a blockchain not on the blacklist, respectively. If omitted, all available blockchains are considered.

Modifiers can be composed of redundancy, encryption, and splitting. All the available options are optional.

Redundancy A modifier option. The allowed value is `redundancy`. If specified, the transaction data is stored on a blockchain and in a conventional database.

Encryption A modifier option. The allowed value is `encryption`. If specified, the transaction data is encrypted before storing it. However, encrypting the transaction data increases the size of the transaction data.

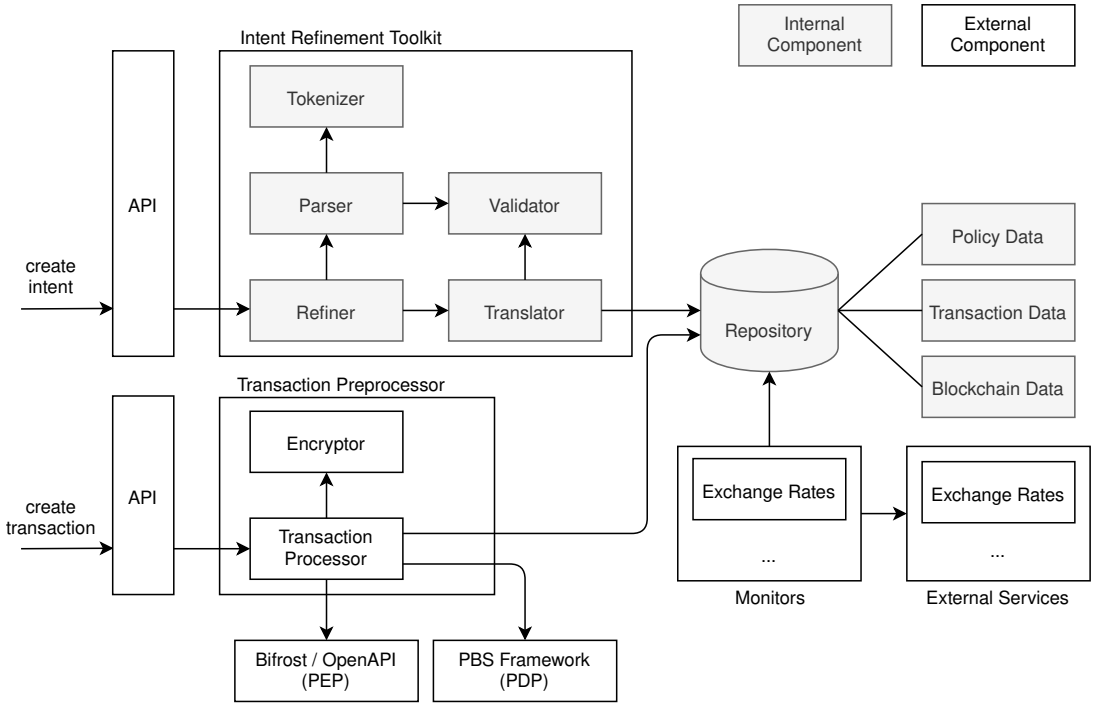

Figure 4.2: IRTK architecture

Splitting A modifier option. The allowed value is `splitting`. If specified and the blockchain pool consists of more than one blockchain after filtering, the PDP selects a different blockchain for every incoming transaction.

4.1.4 Intent Refinement Toolkit (IRTK)

Figure 4.2 depicts the architectural design of the Intent Refinement Toolkit (IRTK) and related components. Internal components are illustrated in grey and external components in white. It exposes an API that can be used by users or other systems to create intents.

The entrypoint of the IRTK is the refiner component. It acts as a wrapper for the parser and translator components. The parser component relies on the tokenizer component to tokenize an intent and on the validator component to validate the intent while parsing it. The translator component relies on the validator component that validates an intent while translating it to low-level policies. Also, the translator relies on the repository that provides exchange rates for currencies. Monitor services periodically call external services to update these exchange rates and store them in the repository.

A transaction preprocessor intercepts all calls to submit new transactions. It interacts with the repository and the PBS Framework to check whether the active policy has `redundancy` or `encryption` enabled. In case encryption is enabled, the transaction processor component calls an encryption component that encrypts the transaction data before submitting it to the PEP. In case redundancy is enabled, the transaction processor simply submits the transaction twice, once for the blockchain determined by the PDP and once

for a conventional database. This transaction preprocessor component is needed, because the PBS Framework does not support `encryption` and `redundancy` natively.

4.1.5 Intent Parsing

The parser component of the IRTK is implemented as a state machine. More precisely, it is a Deterministic Finite Automaton (DFA). It is responsible for parsing and validating intents. Figure 4.3 depicts the notation used in the state chart, and Figure 4.4 illustrates a high-level overview of the state chart. It consists of composite states which are illustrated in dedicated state charts. It is split into smaller parts, both for practical reasons and understandability. There are optional states, *e.g.*, the timeframe state, that can be skipped. Also, there are states which can be visited repeatedly, *e.g.*, the users and the modifiers states. Moreover, there are two accepting states, *i.e.*, the policy and the default policy states. These correspond to valid intent specifications that can be translated to low-level policies. Finally, there is an error state. It is not illustrated in the state chart. However, each state can lead to the error state, if it encounters an invalid event. The error state does not define any outgoing transitions. Thus, the error state acts as a sink state that ignores all further tokens. The states of the parser state machine are named according to the tokens they expect.

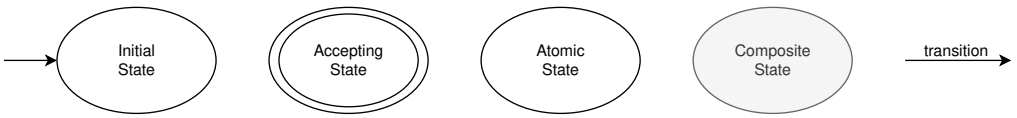

Figure 4.3: State chart legend

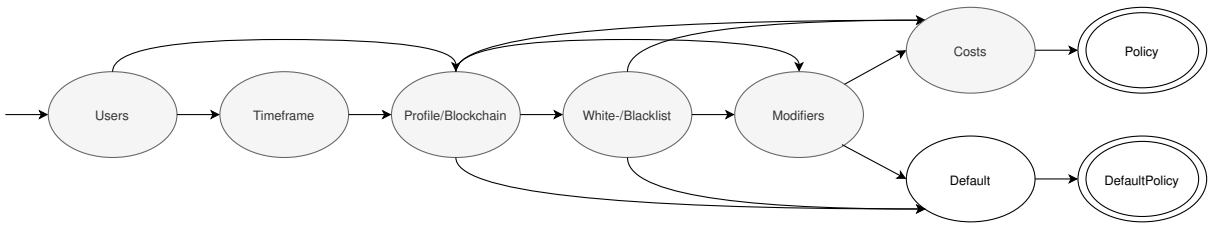

Figure 4.4: State chart overview

The composite users state of the overview is shown in detail in Figure 4.5. The initial starting state is the `ForState`. Therefore, it accepts the token “for”. The `UserState` is a capturing state, *i.e.*, it parses intent options, specifically usernames. Since intents can be defined for more than one user the `UserState` can be visited repeatedly.

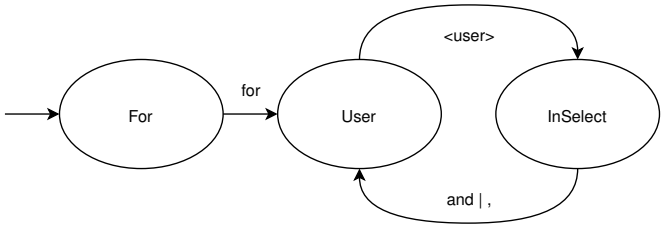

Figure 4.5: User states

Figure 4.6 shows the composite timeframe state of the overview. Since it is an optional state, it can be skipped entirely via the select transition connecting the `InSelectState` with the `TheBlockchainState`. In this part of the state machine, the only capturing state is the `TimeframeState`. It is responsible for parsing and validating the timeframe option.

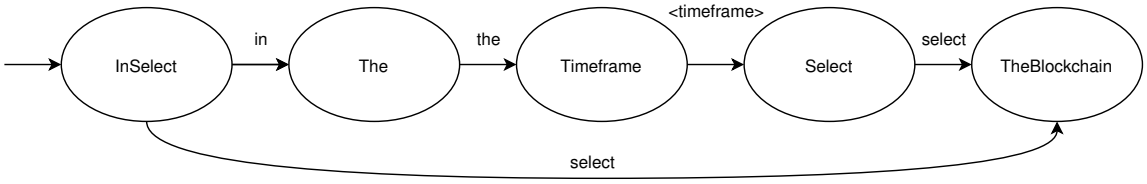

Figure 4.6: Timeframe states

The components of the profile and blockchain states of the overview are depicted in Figure 4.7. As the name suggests, there are two alternative pathways. Usually, an intent specifies a profile that acts as a selection strategy. Alternatively, a single blockchain can be specified directly which will always be selected. Thus, there are two capturing states, the `ProfileState` and the `TheBlockchainState`. Depending on the specified options, the parser state machine ends up in different states. In the profile pathway, there is yet another nested composite state, the filters state.

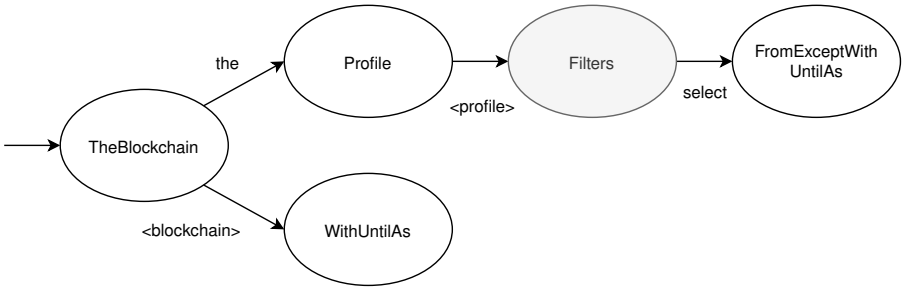

Figure 4.7: Profile and blockchain states

Figure 4.8 illustrates the composite filter states from the profile and blockchain states. There are two capturing states, the `FilterState` and the `FilterBlockchainState`. Both states parse the filter options of the intent. In a single intent, multiple filter options can be specified and therefore, the `FilterState` can be reached repeatedly.

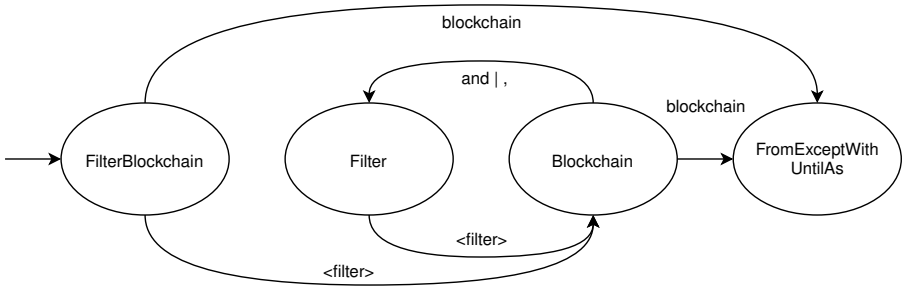

Figure 4.8: Filter states

Figure 4.9 shows the whitelist, blacklist, and modifier states from the overview. There are many alternative pathways. For example, optionally a whitelist or blacklist can be specified. Also, optionally modifiers can be specified. For simplicity, these states are again nested composite states. Depending on the specified options, the parser state machine ends up in different states.

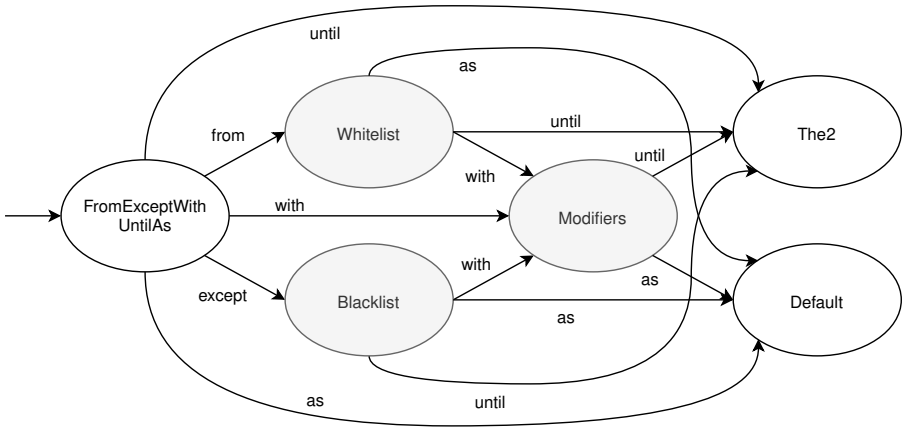

Figure 4.9: Whitelist, blacklist, and modifier states

Figure 4.10 depicts the whitelist and blacklist states from the whitelist, blacklist, and modifier states. Since the whitelist and blacklist accept multiple blockchains, the corresponding states can be reached repeatedly. Also, the `WhitelistState` and `BlacklistState` are capturing states. They parse blockchain names that are either explicitly included, or excluded from the selection process.

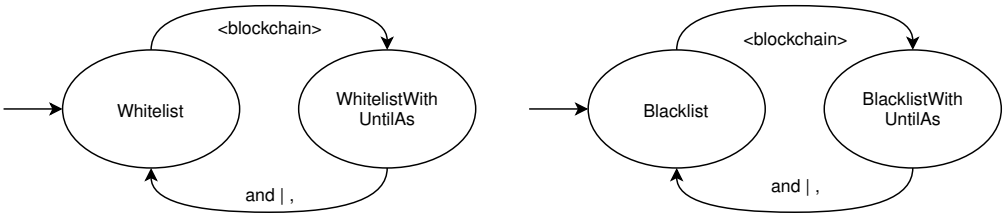

Figure 4.10: Whitelist (a) and blacklist (b) states

The modifiers state from the whitelist, blacklist, and modifier states are shown in Figure 4.11. Again, the specification of modifiers is optional, and therefore the corresponding states can be skipped entirely. The `ModifierState` is a capturing state. It parses modifier options used to alter the behavior of the selection process. Since an intent accepts multiple modifier options, the state can be reached repeatedly.

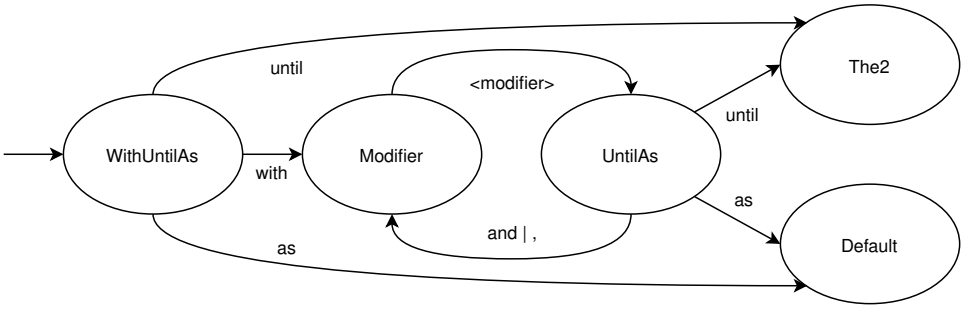

Figure 4.11: Modifier states

Figure 4.12 and Figure 4.13 illustrate the two alternative pathways leading to the accepting states. One pathway corresponds to a non-default policy, and the other corresponds to a default policy. For the non-default policy, cost-related parameters are required. The corresponding capturing states are the `IntervalState`, the optional `CurrencyThresholdState`, and the `ThresholdState`. Only the currency is optional, which, by default is set to US Dollars (USD).

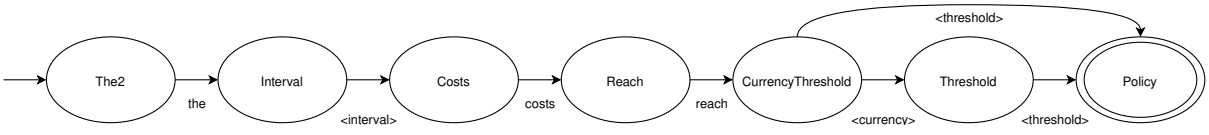

Figure 4.12: Cost states

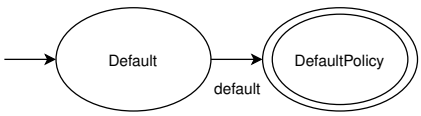

Figure 4.13: Default states

4.1.6 Intent Translation

After an intent has been parsed and validated, it can be translated into low-level policies. The translator component accepts an intent and processes the parsed options into corresponding options of low-level policies. Intents support multiple users, while low-level policies always correspond to a single user. Therefore, a single intent can be translated into a set of low-level policies, *i.e.*, one low-level policy for each user specified in the intent.

Furthermore, the translator component validates the intent options. For example, it excludes all the blacklisted blockchains from the pool. Depending on the specified options,

Table 4.3: Validation types

Type	Options	Intent
Exclusion	`fastest` and `fast`, `cheapest` and `cheap`	valid
Conflict	`private` and `public`	invalid

it is possible that the resulting pool is empty, *i.e.*, contains no blockchains. In this case the translation is ended with a validation error, to avoid low-level policies with an empty blockchain pool.

4.1.7 Intent Validation

The parser and translator components of the IRTK validate the intent based on the specified options. However, not every state of the parser performs validation. Some states perform basic validation, *e.g.*, check whether an option is within a given range or a member of an enum definition. Other states perform more advanced validation. These complex validations distinguish between exclusions and conflicts. Exclusions correspond to options that have no effect in the current configuration. For example, the `fast` option is redundant if the `fastest` selection strategy has been specified. If an exclusion is encountered, the underlying option is simply ignored. The corresponding policy is still valid and the parsing process continues.

Conflicts are combinations of options that are mutually exclusive. For example, the `public` and `private` filter options conflict with each other. There are no blockchains that are public and private at the same time. Thus, it would result in an empty blockchain pool of the corresponding policy. Whenever a conflict is detected, the parser transitions to the error state and the underlying policy is invalid. The excluding and conflicting options are summarized in Table 4.3.

4.2 Implementation

The prototype of the IRTK is implemented in Python. Python is an object-oriented, interpreted, dynamically and strongly typed language, with a rich ecosystem of third-party packages available. Python is well-suited to implement the system, because other components of the Policy-based Blockchain Agnostic Framework, *e.g.*, Bifrost/OpenAPI (PEP) are implemented in Python already. Therefore, these components can be used in a wrapping framework as libraries directly without the need for implementing complex bindings or exposing a RESTful API. Also, Python allows for faster development compared to other languages, *e.g.*, Java or C++ [27].

The prototype implementation comprises a small number of dependencies. The parser component relies on the Natural Language Toolkit (NLTK) for tokenizing intents. The repository is implemented using sqlalchemy as an Object-Relational Mapper (ORM). An

ORM abstracts the database interactions and is agnostic of the SQL-dialect. Therefore, the repository can easily be extracted and even the underlying database can be switched without affecting the rest of the prototype. Currently, the database is based on PostgreSQL, but it could be replaced with, *e.g.*, MariaDB (MySQL) by simply replacing the database driver. This is possible, because the prototype does not rely on Postgres-specific features. Psycopg2 is a database driver for Postgres. This driver would have to be replaced when switching to a different database such as MariaDB. Finally, to create new policies in the PBS Framework from the refined policies, the requests package is required to interact with the RESTful API from the PBS Framework.

Further, the prototype uses the `unittesting` package for unit testing provided by the Python standard library. The test cases are data-driven and require the `ddt` package. There are almost 200 unit test cases implemented, that verify the functional requirements of the parser, translator and validator.

The remainder of this section is structured as follows. Section 4.2.1 discusses the implementation of the refinement. The implementation of the parsing and the translation are described in more detail in Section 4.2.2 and Section 4.2.3, respectively. Section 4.2.4 describes the implementation of the validation. Finally, the intent and policy data types are described in Section 4.2.5 and Section 4.2.6, respectively.

4.2.1 Intent Refinement

The IRTK exposes a collection of tools to refine blockchain-selection intents to low-level policies. The tools provide different levels of control over the refinement process depending on a user's needs.

```
1 from irtk import refine
2
3 policies = refine(
4     "for client"
5     " select the cheapest blockchain"
6     " until the daily costs reach 30"
7 )
```

Listing 4.2: Intent refinement using the refine method

The most straightforward way to refine an intent, is the `refine` method provided by the IRTK. It implements a function that abstracts the underlying refinement procedure, providing a transparent interaction to the user. Internally, it creates a parser instance which is used to parse the intent. If this intent is valid, it uses a translator to translate the intent into low-level policies. An example to refine an intent using the `refine` method is shown in Listing 4.2.

```python
from irtk import Parser, Translator

parser = Parser()
intent = parser.parse(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)

translator = Translator()
policies = translator.translate(intent)
```

Listing 4.3: Intent refinement using a parser and translator

Alternatively, this process can be achieved with the `Parser` and `Translator` classes. First, a parser instance has to be created. This parser instance can be used to parse an intent. In case, the intent is complete and valid, it will be returned. Then, a translator instance has to be created. This translator instance can be used to translate the parsed intent into low-level policies. An example to refine an intent using a `Parser` and a `Translator` is shown in Listing 4.3.

These two approaches are equivalent in that both produce the same result given the same input. However, the former is simpler and hides all the details but the latter provides more flexibility. For example, it is possible to interact with the parsed intent, *i.e.*, `intent` before the translation.

4.2.2 Intent Parsing

The parser is implemented as a state machine. It defines two interfaces for states. The basic `State` interface and the more advanced `ValidationState` interface. The former does not perform any validation and is shown in Listing 4.4. The latter does perform validation and is shown in Listing 4.5. Both types optionally can be passed a dictionary object of token and successor state pairs defining the transitions for this state upon initialization. Moreover, both interfaces expose a `run` method that takes two parameters. It expects a `token` which is a string and an `intent` which is the intent instance that stores the already parsed options. The default implementation of state, simply checks whether a transition is defined for the token. If there is no transition defined for a token, an `IllegalTransitionError` is raised. Otherwise, the successor state is returned. The default implementation of validation state, additionally calls a validation method after ensuring a transition is defined for the token before returning the successor state. Therefore, concrete implementations that extend the `ValidationState` must override and implement their own `_validate` method.

```
import abc

class State(abc.ABC):
    def __init__(self, transitions=None, accepting=False):
        self._transitions = transitions or {}
        self._accepting = accepting

    def run(self, token, intent):
        if token not in self._transitions:
            raise IllegalTransitionError
        return self._transitions[token]
```

Listing 4.4: State interface and default implementation

```
import abc

class ValidationState(State):
    def run(self, token, intent):
        if token not in self._transitions:
            raise IllegalTransitionError
        self._validate(token, intent)
        return self._transitions[token]

    @abc.abstractmehthod
    def _validate(self, token, intent):
        raise NotImplementedError
```

Listing 4.5: Validation state interface and default implementation

Listing 4.6 shows an example that parses an intent. The result is an instance of the intent data type that stores all the specified options. In this example, the intent is parsed completely.

```
from irtk import Parser

parser = Parser()
intent = parser.parse(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)
```

Listing 4.6: Intent parsing

A parser is implemented as a state machine and stores the current state as an instance attribute. Therefore, it is possible to parse an intent incrementally. The parser is able to

parse every token separately. It is not possible though to pass every character separately. Although, it would be possible to implement a parser that accepts every character the number of states would increase drastically. Therefore, the parser is accepts tokens and not individual characters which also is more sensible with regard to maintainability.

```
from irtk import Parser

parser = Parser()
parser.parse("for client")
parser.parse("select the cheapest blockchain")
intent = parser.parse("until the daily costs reach 30")
```

Listing 4.7: Incremental intent parsing

As an illustration, Listing 4.7 shows the very same intent from the previous example and parses it incrementally. The outcome does not differ from the result when parsing the intent completely, in a single turn. However, this provides users of the library to cover a wider range of use cases, *e.g.*, parsing intents in real-time.

```
from irtk import Parser

parser = Parser()
intent = parser.parse(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)

# Error: the same parser instance cannot
# be used to parse different intents
other_intent = parser.parse(
    "for other_client"
    " select the fastest blockchain"
    " until the weekly costs reach 300"
)
```

Listing 4.8: Parsing multiple intents using a parser

The previous examples focused on parsing a single intent. One can consider a use case where multiple intents must be parsed. Listing 4.8 shows an attempt to parse two individual intents using the same parser instance. However, this approach will only work as intended for the first intent and not for the second one. In fact, the second intent, *i.e.*, `other_intent` raises an `IllegalTransitionError`. After parsing the first intent, the state machine of the parser is in an accepting state. The accepting state does not define any outgoing transitions and therefore does not accept any tokens. If a token is passed to the accepting state anyway, the state machine of the parser transitions to a special error state and raises an `IllegalTransitionError`.

```
from irtk import Parser

parser = Parser()
intent = parser.parse(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)

# Solution: create new parser instance
# before parsing another intent
other_parser = Parser()
other_intent = other_parser.parse(
    "for other_client"
    " select the fastest blockchain"
    " until the weekly costs reach 300"
)
```

Listing 4.9: Parsing multiple intents using multiple parser

Listing 4.9 shows a fixed version of the same example. Rather than using the same parser instance that was used to parse the first intent, a new parser instance is used to create the second intent. This example works as intended for both intents.

4.2.3 Intent Translation

The translator acts as an adapter to adapt the intent data type to the policy data type defined by the PBS Framework. It uses conditional checks and lookup tables to translate intent options to corresponding low-level policy ones.

The intent data type supports multiple users, the policy data type only a single one. Therefore, an intent is translated to one policy for each user. The intent data type has a timeframe field, the policy data type expects a start and end point specified in hours and minutes. Therefore, the timeframe is translated to a start and end point based on values defined in a configuration file. This configuration file allows users to customize the translation for their particular use case. Further, both data types support cost related parameters such as interval, currency, and threshold. The intent data type supports a blockchain instead of a profile. The policy data type does not support this, it always requires a profile. Therefore, the translation restricts the blockchain pool to only contain the specified blockchain so that it will always be selected, regardless of the profile.

The intent and policy data type support a number of filter options. However, not all options are supported by both data types. Specifically, the intent data type supports cheap, popular, and stable as filter options which are not supported by the policy data type. The policy data type supports maximum block time, minimum data size, and turing complete as filter options which are not supported by the intent data type. Both

data types support a whitelist parameter. The intent data type additionally supports a blacklist parameter. During translation the blockchains in the blacklist are simply mapped to the whitelist of the policy data type by adding the remaining blockchains. Both data types support splitting of transactions as a modifier. The intent data type additionally supports redundancy and encryption as modifiers.

```
1 from irtk import Translator
2
3 intent = ...
4 other_intent = ...
5
6 translator = Translator()
7 policies = translator.translate(intent)
8 other_policies = translator.translate(other_intent)
```

Listing 4.10: Intent translation using a translator

Listing 4.10 shows an example that translates multiple, already parsed intents. It does not show how these intents have been parsed though. Section 4.2.2 provides examples for for parsing intents. In contrast to the parser, the implementation of the translator allows to translate multiple intents. There is no need to create a new translator instance for every intent to be translated.

4.2.4 Intent Validation

Intent validation is implemented as part of the parser and the translator. Whenever the state machine of the parser encounters a token that it does not expect, *i.e.*, a token for which no transition is defined in the current state, an `IllegalTransitionError` is raised.

```
1 class ErrorState(State):
2     def run(self, token, intent):
3         return self
```

Listing 4.11: Error state implementation

The parser catches this error and transitions to a special error state before reraising the `IllegalTransitionError`. The implementation of the error state is shown in Listing 4.11. It does not define any outgoing transitions and therefore, once this error state is reached, there is no recovery and the underlying intent stays invalid. Some states also perform validation of the encountered token. For example, some states check whether the token is a member of an enum. Other states, such as the `ThresholdState` check whether the token is in fact a number that can be converted to a floating point value. Finally, there are also states that perform more complex validation. In these cases, exclusions and conflicts can be distinguished. An exclusion is encountered, whenever the current token corresponds to an option that has no effect in combination with an already specified option. An

example would be the `fast` filter option with the `fastest` profile. In case, an exclusion is encountered, it is simply ignored. On the other hand, a conflict occurs when the current token corresponds to an option that is mutually exclusive with an already specified option. For example, the `private` and `public` filter options are mutually exclusive, because a blockchain can be either private or public but not both at the same time. Therefore, if these options are specified together, the blockchain pool would always be empty. In case, a conflict is encountered, a `ValidationError` is raised. The parser will catch this error and transition to a special error state.

The translator also validates the intent. During translation of intent options to low-level policy ones, it can occur that the resulting blockchain pool will be empty. For example, if an intent blacklists all the available blockchains the pool will be empty. To avoid this, the translator checks whether the pool will be empty. In case, it detects a violation it will raise a `ValidationError`.

4.2.5 Intent

The IRTK implements an intent data type which is shown in Listing 4.12. It is the intermediate result of the refinement process. The parser parses an intent and creates a new instance of the intent data type. It is used to store all the parsed options. These abstract, high-level options must be translated to low-level ones. For example, the intent data type has a timeframe attribute. The allowed values are `DAY`, `NIGHT`, `MORNING`, and `AFTERNOON`. However, the low-level policy defined by the PBS Framework expects a `timeframe_start` and `timeframe_end` in the format of `"HH:MM"`. Therefore, the translator has to translate the timeframe value to a corresponding `timeframe_start` and `timeframe_end`. Moreover, some of the options from the intent data type are not supported by the low-level policies of the PBS Framework.

```
1  from dataclasses import dataclass, field
2
3  @dataclass
4  class Intent:
5      users = field(default_factory=set)
6      timeframe = None
7      blockchain = None
8      profile = None
9      filters = field(default_factory=set)
10     whitelist = field(default_factory=set)
11     blacklist = field(default_factory=set)
12     modifiers = field(default_factory=set)
13     interval = None
14     currency = None
15     threshold = 0.0
```

Listing 4.12: Intent data type

4.2.6 Policy

The policy data type shown in Listing 4.13 corresponds to the policy model that is used by the PBS Framework. It is the final result of the refinement process. The translator translates a parsed intent and returns a set of instances of the policy data type. The IRTK implements a number of enum types such as `Time`, `CostProfile`, or `BlockchainType`. Their implementation is not shown here, but it can be looked up in the source code. Also, the policy data type defines an `asdict` method that returns a dictionary object of its options. Analogously, the `asjson` method returns a JSON representation of the to policy instance. These formats use the exact same names and export only the options that are compatible with the PBS Framework. Therefore, these methods can be used to interact with the PBS Framework, *e.g.*, to create a new policy.

```
1  from dataclasses import dataclass, field
2
3  @dataclass
4  class Policy:
5      user = ""
6      timeframe_start = Time.DEFAULT
7      timeframe_end = Time.DEFAULT
8      cost_profile = CostProfile.ECONOMIC
9      min_tx_rate = 4
10     max_block_time = 600
11     min_data_size = 20
12     turing_complete = False
13     blockchain_pool = field(default_factory=set)
14     blockchain_type = BlockchainType.INDIFFERENT
15     split_txs = False
16     interval = None
17     currency = None
18     threshold = 0.0
```

Listing 4.13: Policy data type

In contrast to the intent data type, the policy data type supports `turing_complete` as an option. This option acts as a filter on the blockchain pool, if it is specified only blockchains that are turing complete are considered. The policies refined by the IRTK will always have this flag disabled, and therefore allow both turing-complete and non-turing-complete blockchains. If a user wants to only consider turing-complete blockchains in their selection via the IRTK, they must manually enable this flag after the translation. Similarly, the policy data type supports maximum block time and minimum data size which are not supported by the intent data type.

Chapter 5

Evaluation

This chapter presents the results of the evaluation of the performance of the IRTK as well as the results of the survey on blockchain usage requirements. The procedure and the results of the performance testing of the IRTK prototype are described in Section 5.1. Section 5.2 formulates a set of evaluation questions and hypotheses regarding the blockchain usage requirements which are compared against the results of the survey.

5.1 Performance Testing

Performance testing is used to assess non-functional requirements, such as the running time of a software system. Software microbenchmarks are a type of performance tests which focus on performance of small isolated components, *e.g.*, functions. Section 5.1.1 describes the hardware and settings used to execute the performance benchmarks. The performance tests and procedure are described together with the results in Section 5.1.2. Finally, Section 5.1.3 discusses the results.

5.1.1 Test System

All the benchmarks were conducted on the same bare-metal machine. The system uses an i5-3570K CPU @ 3.40GHz and has 16GB of RAM. The operating system is Arch Linux using the Linux desktop 5.4.1-arch1-1 kernel. Python is a garbage-collected programming language. The benchmarks were performed with garbage-collection enabled deliberately. This assures that the measured performance values are accurate and resemble a real-life example. Disabling the garbage collection and switching to manual garbage collection has improved the stability of the results. However, the key findings remained the same for automatic and manual garbage collection.

Table 5.1: Overview of intent categories

Intent category	Parameters	Currency	Examples
Simple	required	default	Listing 5.1, Listing 5.2
Intermediate	required	non-default	Listing 5.3, Listing 5.4
Complex	required and optional	non-default	Listing 5.5, Listing 5.6

Table 5.2: Overview of intent variations

Intent variation	Number of users	Examples
Single	1	Listing 5.1, Listing 5.3, Listing 5.5
Multi	> 1	Listing 5.2, Listing 5.4, Listing 5.6

5.1.2 Results

There are a number of benchmarks to assess the performance of the IRTK prototype. Specifically, the performance of the refinement, the parsing and the translation is tested. The performance of the parsing is compared to an alternative approach based on regular expressions. Moreover, there are different categories of intents that are used as input data for the benchmarks which are summarized in Table 5.1. There are simple, intermediate and complex intents. Simple intents only specify the required parameters and use the default currency. In contrast, intermediate intents use a non-default currency. Complex intents specify any number of parameters including optional ones and also use a non-default currency. Furthermore, the categories exist in two variations which are summarized in Table 5.2. The single-user variation specifies a single-user and the multi-user variation specifies multiple users. Specifically, in the example benchmarks three users are used for the multi-user variations. Finally, each group and variation pair consists of a set of intents that satisfy the group and variation criteria.

```
1 for client
2 select the cheapest blockchain
3 until the daily costs reach 10
```

Listing 5.1: Simple intent with a single user

```
1 for client_a, client_b and client_c
2 select the cheapest blockchain
3 until the daily costs reach 10
```

Listing 5.2: Simple intent with multiple users

```
for client
select the cheapest blockchain
until the daily costs reach CHF 10
```

Listing 5.3: Intermediate intent with a single user

```
for client_a, client_b and client_c
select the cheapest blockchain
until the daily costs reach CHF 10
```

Listing 5.4: Intermediate intent with multiple users

```
for client
select the cheapest public, stable and popular blockchain
except Bitcoin and Ethereum
with splitting, redundancy and encryption
until the daily costs reach CHF 10
```

Listing 5.5: Complex intent with a single user

```
for client_a, client_b and client_c
select the cheapest public, stable and popular blockchain
except Bitcoin and Ethereum
with splitting, redundancy and encryption
until the daily costs reach CHF 10
```

Listing 5.6: Complex intent with multiple users

In the first part of the performance testing, the refinement, parsing and translation are tested. The measurements are performed over 1000 iterations. In each iteration a random intent from the particular group is chosen. Finally, the raw measurements are collected in a file. Moreover, statistics such as the mean and the standard deviation are computed for each group and variation combination separately. These statistics are also written into a file. Finally, a bar chart is plotted using the mean values from the measurements for every benchmark. The bar chart displays error bars which correspond to the 95% confidence interval. The formula used to calculate the confidence interval is shown in Equation (5.1), where $\bar{x}$ denotes the mean, σ the standard deviation, and n the number of samples. It is important to note, that the calculation of confidence interval assumes a normal distribution of the measurements.

$$CI_{95} = \bar{x} \pm 1.96 \times \frac{\sigma}{\sqrt{n}} \tag{5.1}$$

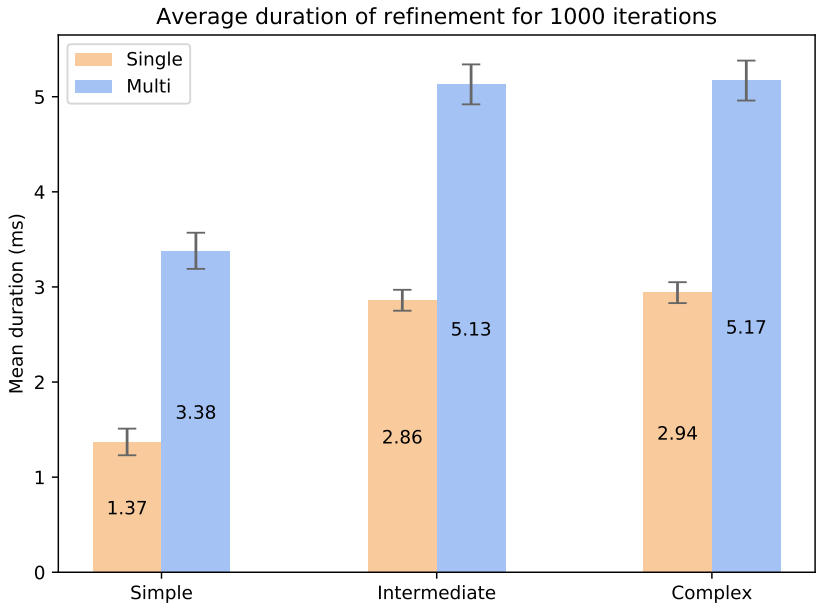

Figure 5.1: Average duration of refinement

Figure 5.1 shows the average duration of the refinement. The results for the intermediate and complex intents are similar, indicating that the number of parameters does not affect the performance of the refinement. There is a measurable difference in performance when comparing the simple intents with the intermediate or complex ones, indicating that the specification of a non-default currency affects the performance of the refinement. The duration to refine intermediate and complex intents increases compared to simple ones. Finally, there is a difference in performance for all categories of intents between the single- and multi-user variations, indicating that the specification of multiple users affects the performance of the refinement. Specifically, the multi-user variations have a higher duration for all categories, than the single-user variations.

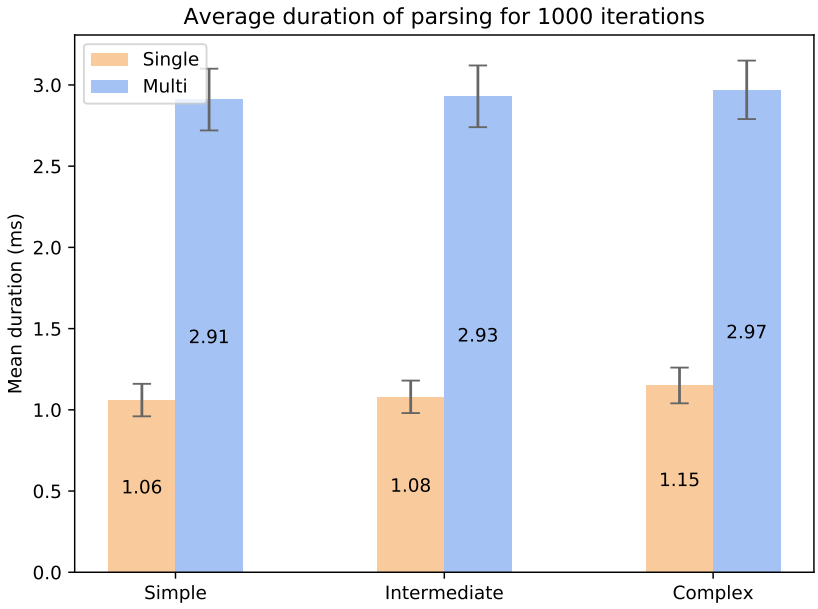

Figure 5.2: Average duration of parsing

Figure 5.2 depicts the average duration of the parsing. The results for all categories of intents are similar. The performance of the parsing is neither affected by the specification of a non-default currency nor by the number of parameters. However, a difference in performance can be observed between the single- and multi-user variations of all intent categories. The parsing of intents that specify multiple users takes longer than intents specifying only a single user. The relationship is not linear, but close to linear, since the multi-user variations specify three users instead of a single one.

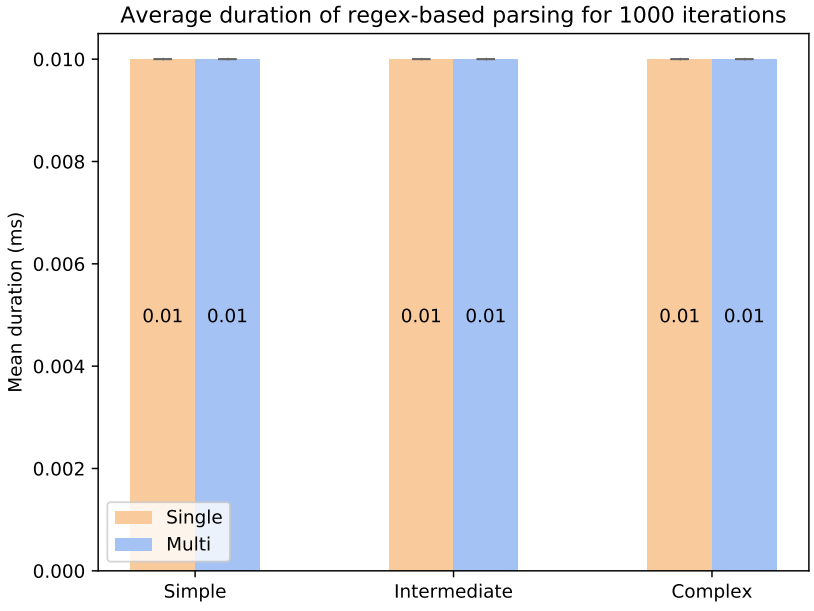

Figure 5.3: Average duration of regex-based parsing

Figure 5.3 shows the average duration of the parsing based on regular expressions which is shown in Listing 5.7. The regex-based parsing outperforms the state-machine-based parsing for all categories and variations. Moreover, the regex-based approach is neither affected by the number of parameters, by the specification of a non-default currency nor by the specification of multiple users. It is important to note, that the state-machine-based parsing performs validation and the regex-based parsing does not.

```
1 "^for (?P<users>[\w ,]+?) "
2 "(?:in the (?P<timeframe>\w+) )?"
3 "(?:select the (?P<profile>\w+)(?P<filters>[\w ,]+)? blockchain(?: (?:
  from (?P<whitelist>[\w ,]+)|except (?P<blacklist>[\w ,]+)))?|select (?P
  <blockchain>\w+)) "
4 "(?:with (?P<modifiers>[\w ,]+) )?"
5 "(?:until the (?P<interval>\w+) costs reach (?P<currency>\w+ )?(?P<
  threshold>[0-9.]+)|as default)$"
```

Listing 5.7: Regular expression for intent parsing

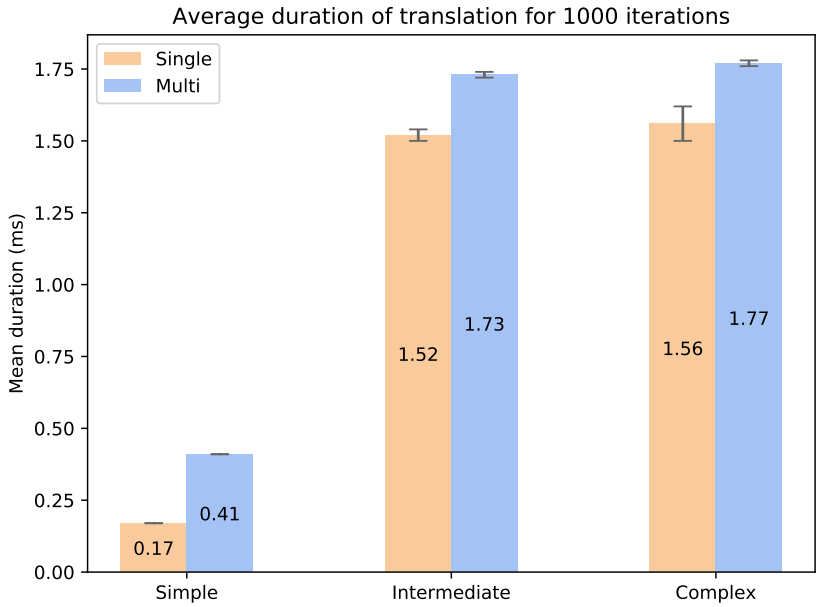

Figure 5.4: Average duration of translation

Figure 5.4 illustrates the average duration of the translation. The results for the intermediate and complex intents are similar. Once again, there is a difference in performance between the simple intents and the intermediate or complex ones, as already seen in the refinement. The translation of intermediate and complex intents takes longer than the simple ones, indicating that the specification of a non-default currency affects the performance of the translation. There is a slight difference in performance between the single- and multi-user variations of all intent categories. However, it is not as significant as for the parsing.

In the second part of the performance testing, the refinement benchmark was repeated for different number of intents, *i.e.*, different number of iterations. Specifically, in the range from 10 iterations up to 10'000 iterations. These benchmarks were executed for all intent categories and variations.

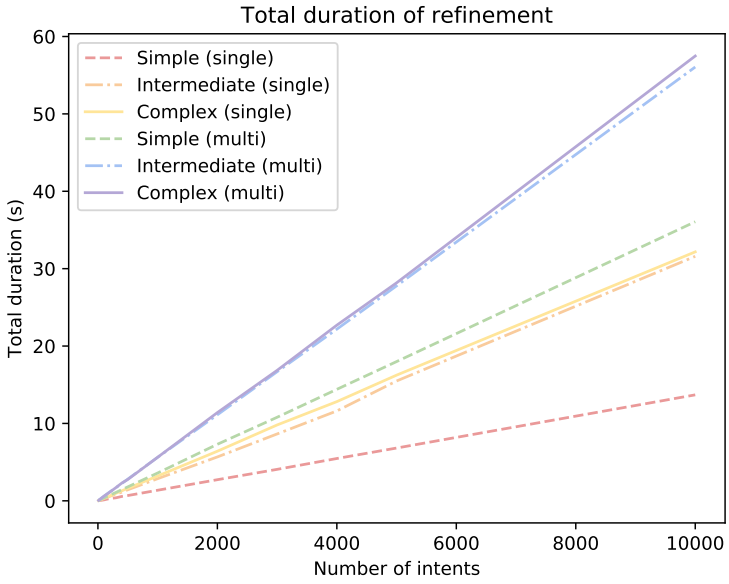

Figure 5.5: Total duration of refinement

Figure 5.5 depicts the total duration of the refinement. The results clearly show a linear relationship between the number of intents and the total duration for the refinement. Increasing the number of intents increases the total duration of the refinement process. The duration of the refinement of simple intents in the single-user variation increase the slowest. The duration of the refinement of intermediate and complex intents in the single-user variation increase slightly faster. The duration of the refinement of simple intents in the multi-user variation increases similar to intermediate and complex intents in the single-user variation. The duration of the refinement of intermediate and complex intents in the multi-user variation increases the fastest.

5.1.3 Discussion

In summary, the performance of the refinement is affected by the specification of multiple users and a non-default currency. It is not affected by the number of parameters. The performance of the parsing decreases when specifying multiple users. It is not clear what exactly causes this performance penalty in the parsing. The performance of the translation decreases when specifying a non-default currency. In case a non-default currency is specified, the translation performs a database query to obtain the exchange rate to convert the threshold value. It is to expect that database interactions introduce latency and some variance in performance.

The performance of the refinement of simple intents is dominated by the parsing for single- and multi-user variations. The performance of the refinement of intermediate and complex intents is dominated by the translation for single-user variations. However, the performance of the refinement of intermediate and complex intents is dominated by the parsing for multi-user variations.

The regex-based parsing outperforms the state-machine-based parsing. However, the regex-based approach does not support validation of the intents. Moreover, the maintainability is more difficult compared to the state-machine-based parsing. Therefore, the gain in performance does not seem to be worth it, especially considering that the state-machine-based parsing is still in the milliseconds range.

Finally, all intent categories and variations exhibit a linear relationship between the number of intents and total duration of the refinement.

5.2 Survey on Blockchain Usage Requirements

In total the survey had 19 respondents from roughly 100 recipients of the email which corresponds to a response rate of 19%. The questionnaire is included in Appendix A. Some of the Questions (Q) are evaluated for different categories of respondents separately, while others are evaluated for all respondents. Respondents are categorized into technical and non-technical individuals based on their answers to Q2 of the questionnaire. Furthermore, respondents are also categorized into computer-science-related, business-related, and other backgrounds based on their answers to Q1 of the survey.

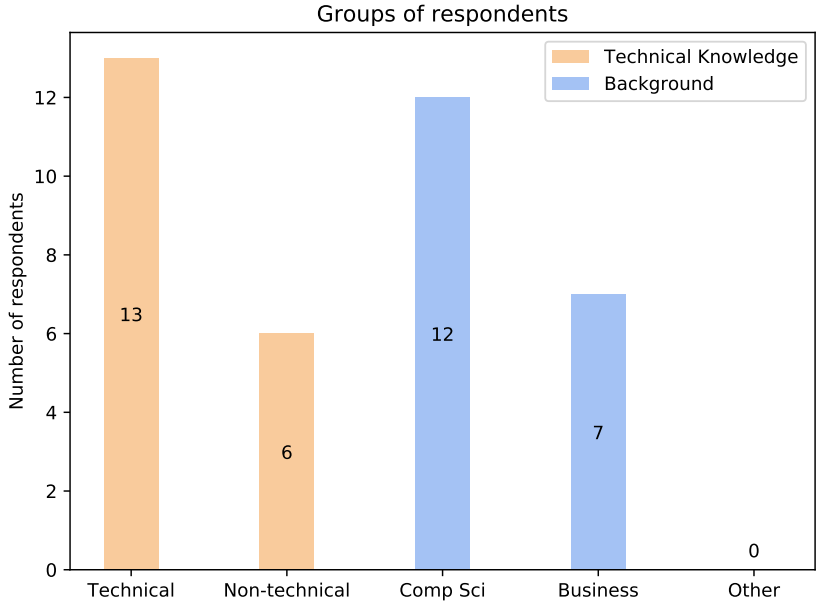

Figure 5.6: Groups of respondents

Figure 5.6 shows the categories of respondents based on their level of technical knowledge of blockchains, and background respectively. 13 respondents are considered to have a

technical knowledge of blockchain technology, and 6 respondents are considered to have a non-technical understanding of blockchain technology. The respondents who selected the first answer of Q2, stating "I know technical details, such as different consensus mechanisms, address formats, node types (e.g., miner and peer), and I have used more than one blockchain implementation." are considered technical. The remaining respondents that selected a different answer of Q2 are considered non-technical. Similarly, 12 respondents who selected either "Computer Science" or "Information Systems" as answer to Q1 have a computer-science-related background. The remaining 7 respondents selected either "Banking and Finance" or "Business Administration" as answer to Q1 and therefore have a business-related background. No respondent stated to have another background in Q1 and therefore the corresponding category is empty.

The following Evaluation Questions (EQ) are answered with the responses of the survey. Moreover, a Hypothesis (H) was defined for a subset of the evaluation questions.

EQ1: Do non-technical individuals consider fewer attributes (compared to technical ones) of a blockchain implementation when selecting a blockchain?

H1: On average, non-technical individuals consider fewer attributes, e.g., deployment type and costs, when selecting a blockchain implementation, because they lack the technical knowledge. On the opposite side, technical individuals tend to consider more attributes, such as throughput, latency, and consensus mechanisms.

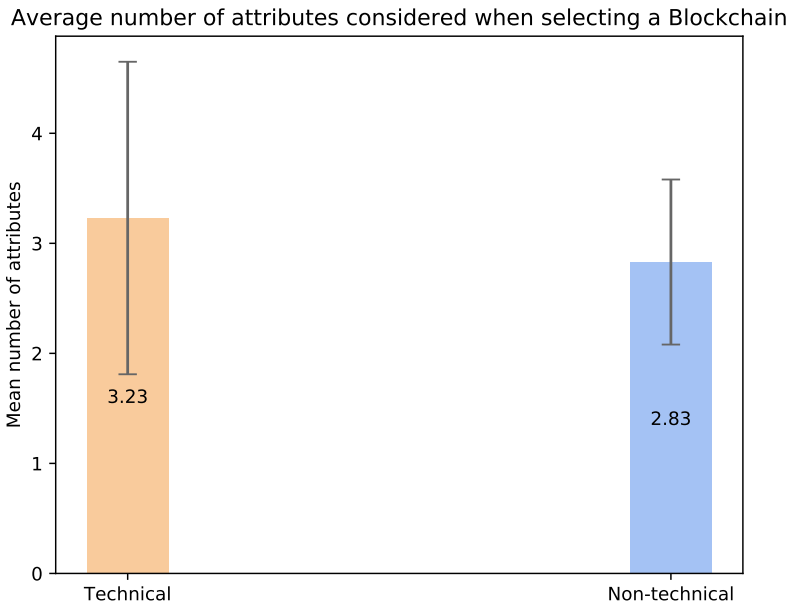

Figure 5.7: Average number of attributes considered when selecting a blockchain

Figure 5.7 depicts the average number of attributes considered when selecting a particular blockchain implementation by technical and non-technical individuals. On average, technical respondents consider 3.23 attributes, and non-technical respondents consider 2.83 attributes. Therefore, the hypothesis holds because technical individuals in fact consider more attributes than non-technical individuals.

Table 5.3: Overview of unlisted responses

Response	Type	Number of respondents
Smart contract language capabilities	Attribute	1
Sharding	Feature	1

EQ2: Which attributes of a blockchain implementation (that were not listed as an option in Q4) do respondents consider? What ratio of the respondents considers these attributes?

Table 5.3 lists the only attribute that was mentioned by a respondent that was not provided as an option to the question. The attribute is "smart contract language capabilites". This option corresponds to the turing complete filter already available in low-level policies.

EQ3: Which is the feature (non-native to blockchains) that most respondents would like an application (*e.g.*, that stores data in a blockchain) to provide?

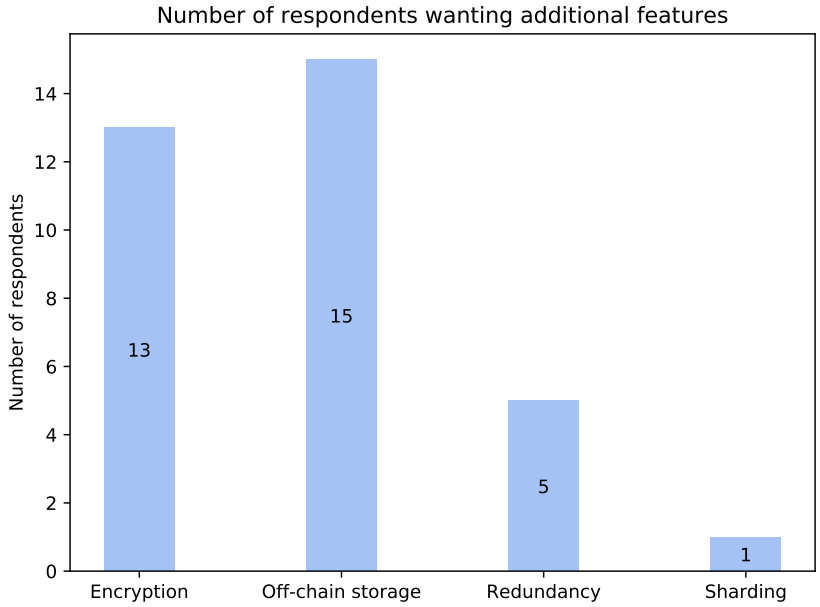

Figure 5.8: Number of respondents wanting an additional feature

Figure 5.8 illustrates for each feature the number of respondents that selected it in Q5. Respondents were able to select multiple features and could even add other features. The results show that the most wanted feature is "off-chain storage", *i.e.*, an automated computation of the checksum of the transaction data which can be stored in a blockchain. This feature is especially useful for large data, *e.g.*, a large number of measurements from sensors to reduce the size of the data. An implementation of this feature could then store the actual data in conventional databases or filesystems and only store the checksum in a blockchain to reduce the transaction costs. The second most selected feature is "encryption". An implementation of this feature could encrypt the transaction data before

storing it in a blockchain and would have to keep track of the encryption key. This feature is considered especially useful, when dealing with sensitive data. "Redundancy" means storing of a redundant copy of the transaction data in a conventional database is only selected by 5 respondents and "sharding" only by 1 respondent.

EQ4: Which additional features that was not listed would respondents like an application that stores data in a blockchain to provide? What ratio of the respondents considers these features?

Table 5.3 summarizes all the features that were not explicitly listed as an option in Q5 but were added by respondents via the custom field. The only unlisted feature that was mentioned by respondents was "sharding" which was mentioned only by 1 respondent.

EQ5: Do respondents perceive complex queries (Example C) (in a particular representation) as less intuitive than simple queries (Example A or B), *i.e.*, does the complexity of the query affect the intuitiveness of the query?

H2: Simple queries (either written in pseudo-code or natural language) are perceived as more intuitive than complex ones.

In Q6 and Q7 two alternative representations for blockchain selection statements are presented. A pseudo-code representation is used in Q6, and a natural language formulation is used in Q7. The examples used in both representations correspond to the same selection criteria. Each representation provides examples A, B, and C. These examples can be categorized based on their complexity. Examples A and B specify only required parameters and are considered simple. Example C specifies the required parameters and a optional parameters as well and is therefore considered complex. Respondents were asked to rate these examples in the different representations based on their intuitive understanding. A respondent could select a score from 1 to 5 or select N/A for not applicable. This rating scale corresponds to a Likert scale.

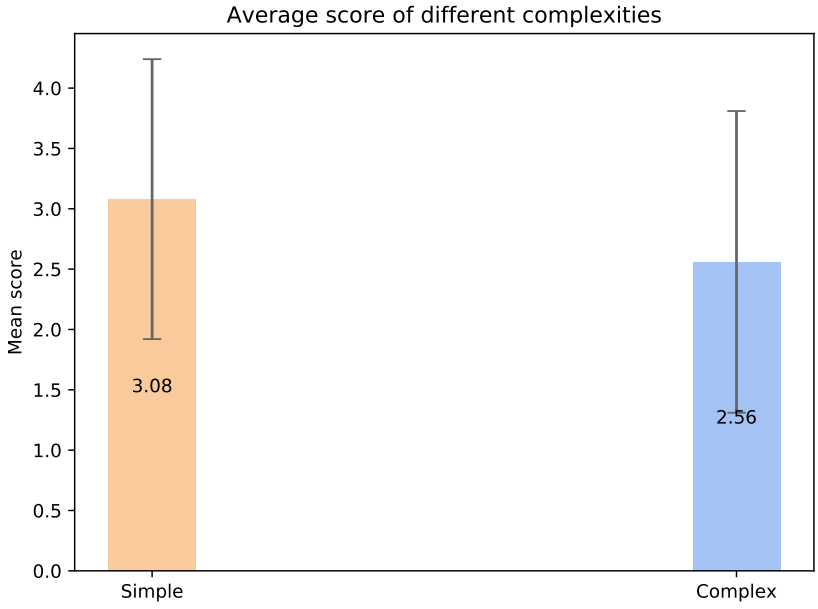

Figure 5.9: Average scores of different complexities

Table 5.4: Missing values based on complexity

Complexity	Missing values
Simple	5
Complex	2
Total	**7**

Figure 5.9 shows the average scores of the simple examples and the complex examples. On average, simple examples were rated as more intuitive than complex ones with a mean score of 3.08 and 2.56, respectively. Therefore the hypothesis holds, and the complexity in fact influences the perceived intuitiveness. Table 5.4 shows the missing values based on the complexity of the examples. In total 7 values from Q6 and Q7 were missing because the respondents selected the N/A option. These missing values were excluded from the evaluation, *i.e.*, the statistics such as the mean are computed without considering these values.

EQ6: Which of the two presented representations (*i.e.*, pseudo-code and natural language representations) do respondents perceive as more intuitive?

H3.1: Non-technical respondents find the natural language representation more intuitive.

H3.2: In contrast, technical respondents find the low-level policy more (or equally) as intuitive as the natural language representation.

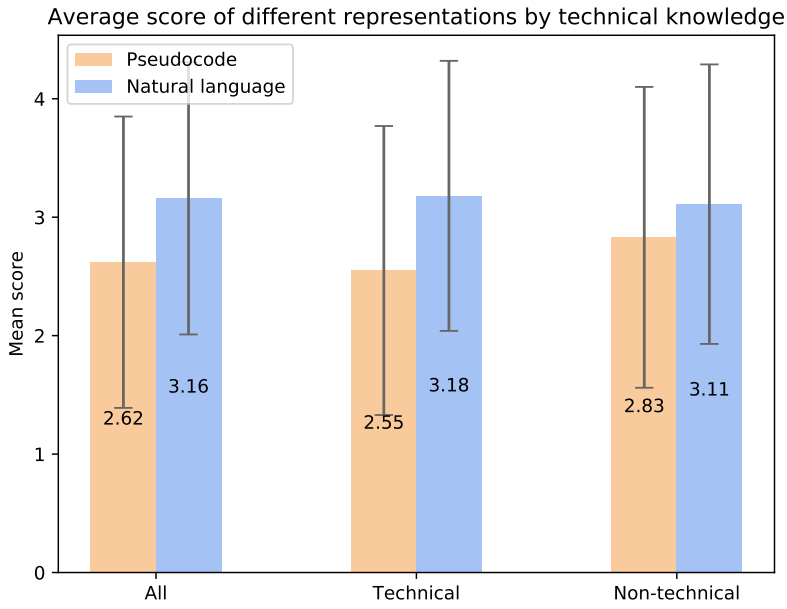

Figure 5.10: Average scores of different representations by technical knowledge

Figure 5.10 shows the average scores of the pseudo-code and natural language representations. An example intent specified in pseudo-code is shown in Listing 5.8 and in natural

Table 5.5: Missing values based on representation

Category	Representation	Missing values
Technical	Pseudo-code	1
	Natural language	0
Non-technical	Pseudo-code	6
	Natural language	0
Total	**Pseudo-code**	**7**
	Natural language	**0**

language in Listing 5.9. On average, the natural language representation was perceived as more intuitive than the pseudo-code representation for all categories. Therefore, H3.1 does in fact hold, because non-technical individuals perceive the natural language representation as more intuitive. However, the results also show that even technical individuals perceive the natural language representation as more intuitive which invalidates H3.2. Therefore, an intent specified in natural language instead of low level configurations such as pseudo-code will benefit both technical and non-technical individuals. Table 5.5 shows the missing values based on the representation of the examples.

```
{
  "clients": ["B"],
  "profile": "fastest",
  "filters": ["private"],
  "interval": "daily",
  "currency": "CHF",
  "threshold": 30
}
```

Listing 5.8: Pseudo-code representation

```
For client B select the fastest private blockchain until the daily
costs reach CHF 30.
```

Listing 5.9: Natural language representation

EQ7: Is there a correlation between the priority of aspects for choosing a blockchain (over a traditional database) and the area of specialization of a respondent? For example, do computer science specialists prioritize "security" higher than other specialists, or do business administration specialists prioritize "low costs" higher than other specialists?

H4.1: Individuals with informatics-related background prioritize "security" higher than other specialists.

H4.2: Individuals with business-related background prioritize "low costs" higher than other specialists.

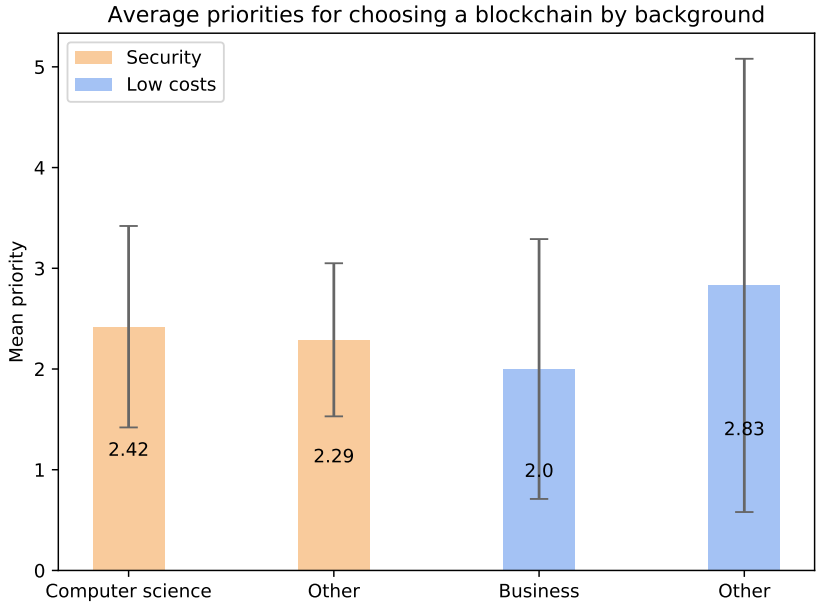

Figure 5.11: Average priorities for choosing a blockchain

Figure 5.11 depicts the average priorities assigned to the "security" and "low costs" aspects by background. In fact, respondents with informatics-related background prioritize "security" slightly higher than respondents with a different background, on average. Therefore, H4.1 actually holds. However, respondents with a business-related specialization prioritize "low costs" lower than respondents with another specialization, on average. Therefore, H4.2 does not hold.

5.2.1 Discussion

The survey was conducted over a period of 1 month and had 19 respondents. The respondents are categorized into technical and non-technical users based on their knowledge of blockchain technology. More than two third of the respondents are considered technical. Therefore, the findings for technical users have a higher significance than the findings related to non-technical individuals. Furthermore, the participation in the survey was voluntary. Therefore, it is possible that the results suffer from a self-selection bias.

The results show that in fact technical users consider more attributes of a blockchain implementation in the selection process than non-technical users. Therefore, a solution supporting non-technical users in the selection process might compensate the lack of the technical knowledge. Moreover, the complexity of intents does affect the perceived comprehension of the intent. Simple intents with fewer parameters are perceived as more intuitive than complex ones with many different parameters. Overall, intents specified in natural language are perceived as more intuitive than intents specified in pseudo-code.

Further, there is only a slight difference when comparing the results from technical and non-technical respondents separately. Therefore, it is assumed that technical and non-technical users benefit from intents specified in natural language equally.

Chapter 6

Summary and Future Work

This thesis introduces the Intent Refinement Toolkit (IRTK). The IRTK refines intents specified in natural language into low-level policies. As part of the refinement, the options specified in the intent are parsed and then translated into low-level policies. Moreover, the intents are validated both during parsing and translation.

The IRTK is designed to be used in a Policy-based Blockchain Agnostic Framework (PBBAF) [30]. It acts as an abstraction layer for low-level policies, by allowing the specification without knowing the underlying technical details. Currently, not all options that are available in the intent specification language are supported by the Policy-based Blockchain Selection (PBS) framework.

The performance evaluation of the IRTK has uncovered several findings. The first finding is that the performance is affected by the specification of a non-default currency and the number of users for which an intent is specified. It is not affected by the number of optional parameters that are specified. The specification of a non-default currency introduces a database access during the translation to retrieve the corresponding exchange rate. In fact, this database access causes a decrease in performance. Similarly, the specification of multiple users decreases the performance of the parsing. The performance evaluation also showed that the duration and the number of intents have a linear relationship. Therefore, refining more than one intent takes linearly more time than parsing a single intent. Finally, an alternative implementation of the parsing based on regular expressions exhibits better performance. However, it is more difficult to maintain and does not allow intent validation.

As part of this thesis, a survey was conducted to assess the blockchain usage requirements of technical and non-technical users. The results showed that on average non-technical users consider less attributes of a blockchain implementation than technical ones. This fact may be attributed to the fact, that non-technical users lack the technical understanding of these attributes. The most desired features by respondents were off-chain storage and encryption. Off-chain storage computes a checksum of the data and only stores this checksum in a blockchain which can reduce the size of the data and price, *e.g.*, when storing large amounts of sensor measurements. Storing sensitive data in a blockchain might not be possible due to regulation. However, encryption of the data might solve this

issue under some circumstances. Finally, the results indicate that both technical and non-technical individuals perceive intents specified in natural language as more intuitive than low-level policies. Therefore, even technical individuals might benefit from an abstraction layer as provided by the IRTK.

6.1 Future Work

The prototype of the IRTK focuses on refining intents into corresponding low-level policies. These low-level policies can be used with the PBS framework which acts as PDP. However, currently not all parameters provided by the IRTK are actually supported by the PBS framework. Specifically, these parameters are the `encryption` and `redundancy` modifiers, and the `stable`, `popular`, and `cheap` filters. Therefore, the implementation of the PBS framework has to be extended to support these parameters, otherwise they will be ignored. Alternatively, encryption and redundancy could also be implemented in a separate component that intercepts incoming transactions. However, an implementation of this approach might result in a lot of overhead and redundancy. Furthermore, to quantify the stability and popularity of a blockchain a model has to be defined for the calculation of these values.

The IRTK relies on a database to retrieve exchange rates for non-default currencies such as Swiss Francs. These exchange rates are used to convert cost thresholds into the default currency which is US Dollars. Exchange rates for currencies are dynamic and therefore have to be updated over time. One possible approach fetches the exchange rates from an external service on every incoming request. However, this approach might result in a lot of network traffic. A better solution might be to implement monitor services that periodically fetch these exchange rates.

Finally, the IRTK is designed as a library. Although, it can be used directly with the Python interpreter this might not be an optimal solution. Instead, it could be integrated in an implementation of the PBBAF which would also expose a web interface.

Bibliography

[1] M. H. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. E. Carpenter, S. Jiang, and L. Ciavaglia, *Autonomic Networking: Definitions and Design Goals*. RFC 7575, June 2015.

[2] M. Belotti, N. Božić, G. Pujolle, and S. Secci, *A Vademecum on Blockchain Technologies: When, Which and How*, in IEEE Communications Surveys Tutorials, 2019, pp. 1–47.

[3] Bitinfocharts, *Bitinfocharts: Blockchain statistics*, 2019. `http://bitinfocharts.com`, last visited October 13, 2019.

[4] T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, *Blockchains Everywhere - a Use-Case of Blockchains in the Pharma Supply-Chain*, in IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017), Lisbon, Portugal, May 2017, pp. 772–777.

[5] V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform*, 2014. `https://github.com/ethereum/wiki/wiki/White-Paper`, last visited October 13, 2019.

[6] R. Chadha, H. Cheng, Y.-H. Cheng, C.-Y. Chiang, A. Ghetie, G. Levin, and H. Tanna, *Policy-based Mobile Ad Hoc Network Management*, 07 2004, pp. 35–44.

[7] W. Chao and S. Horiuchi, *Intent-based cloud Service Management*, in Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN 2018), Feb 2018, pp. 1–5.

[8] M. Charalambides, P. Flegkas, G. Pavlou, A. K. Bandara, E. C. Lupu, A. Russo, N. Dulav, M. Sloman, and J. Rubio-Loyola, *Policy Conflict Analysis for Quality of Service Management*, in IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden, Sweden, June 2005, pp. 99–108.

[9] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, and N. Dulay, *Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management*, in IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), Vancouver, BC, Canada, January 2006, pp. 294–304.

[10] S. Chen, J. Zhang, R. Shi, J. Yan, and Q. Ke, *A Comparative Testing on Performance of Blockchain and Relational Database: Foundation for Applying Smart Technology into Current Business Systems*, in Distributed, Ambient and Pervasive Interactions (DAPI 2018), Las Vegas, NV, USA, July 2018, pp. 21–34.

[11] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, *Intent-Based Networking - Concepts and Overview*. Work in Progress, `https://www.ietf.org/id/draft-clemm-nmrg-dist-intent-03.txt`, last visited December 10, 2019.

[12] V. Clincy and H. Shahriar, *Blockchain Development Platform Comparison*, in 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol. 1, Jul 2019, pp. 922–923.

[13] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, *On Scaling Decentralized Blockchains*, in Financial Cryptography and Data Security (FC 2016), Christ Church, Barbados, February 2016, pp. 106–125.

[14] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, *The Ponder Policy Specification Language*, in Proceedings of the International Workshop on Policies for Distributed Systems and Networks, POLICY '01, London, UK, UK, 2001, Springer-Verlag, pp. 18–38.

[15] S. Davy, B. Jennings, and J. Strassner, *The Policy Continuum - A Formal Model*, in Proc. of the 2nd IEEE International Workshop on Modelling Autonomic Communications Environments, MACE, (2007), pp. 65–79.

[16] Y. Elkhatib, G. Coulson, and G. Tyson, *Charting an Intent Driven Network*, in 2017 13th International Conference on Network and Service Management (CNSM), Nov 2017, pp. 1–5.

[17] S. D. Foundation, *Fees | Stellar Developers*, 2019. , `https://www.stellar.org/developers/guides/concepts/fees.html`, last visited November 10, 2019.

[18] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong, *An Intent-based Network Virtualization Platform for SDN*, in International Conference on Network and Service Management (CNSM 2016), Oct 2016, pp. 353–358.

[19] T. Hegnauer, *Design and Development of a Blockchain Interoperability API*, Master's thesis, Zürich, Switzerland, February 2019.

[20] A. Jacobs, R. Pfitscher, R. Ferreira, and L. Granville, *Refining Network Intents for Self-Driving Networks*, 08 2018, pp. 15–21.

[21] J. Kang, J. Lee, V. Nagendra, and S. Banerjee, *LMS: Label Management Service for intent-driven Cloud Management*, in IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017), May 2017, pp. 177–185.

[22] D. Lakic, *Design and Implementation of a Policy-based Blockchain Selection Framework*, Master's thesis, Zürich, Switzerland, June 2019.

[23] C. Li, O. Havel, P. Martinez-Julia, J. Nobre, and D. Lopez, *Intent Classification*. Work in Progress, `https://www.ietf.org/id/draft-li-nmrg-intent-classification-02.txt`, last visited December 10, 2019.

[24] E. C. Lupu and M. Sloman, *Conflicts in Policy-based Distributed Systems Management*, in IEEE Transactions on Software Engineering, vol. 25, November 1999, pp. 852–869.

[25] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2009. `http://www.bitcoin.org/bitcoin.pdf`, last visited October 10, 2019.

[26] N. W. Paton and O. Díaz, *Active Database Systems*, ACM Comput. Surv., 31 (1999), pp. 63–103.

[27] L. Prechelt, *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program.*, (2000).

[28] D. Puthal, N. Malik, S. Mohanty, E. Kougianos, and G. Das, *Everything You Wanted to Know About the Blockchain: Its Promise, Components, Processes, and Problems*, in IEEE Consumer Electronics Magazine, vol. 7, July 2018, pp. 6–14.

[29] A. Riekstin, G. Januário, B. Rodrigues, V. Nascimento, T. Carvalho, and C. Meirosu, *A Survey of Policy Refinement Methods as a Support for Sustainable Networks*, IEEE Communications Surveys & Tutorials, 18 (2015), pp. 1–1.

[30] E. Scheid, B. Rodrigues, and B. Stiller, *Toward a Policy-based Blockchain Agnostic Framework*, in IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019), April 2019, pp. 609–613.

[31] E. J. Scheid, T. Hegnauer, B. Rodrigues, and B. Stiller, *Bifröst: a Modular Blockchain Interoperability API*, in IEEE Conference on Local Computer Networks (LCN 2019), Osnabrück, Germany, October 2019, pp. 1–9. Accepted. To be published.

[32] E. J. Scheid, D. Lakic, B. Rodrigues, and B. Stiller, *PleBeuS: a Policy-based Blockchain Selection Framework*, in IEEE/IFIP Network Operations and Management Symposium (NOMS 2020), Budapest, Hungary, April 2020, pp. 1–9. Accepted. To be published.

[33] E. J. Scheid, C. C. Machado, R. L. dos Santos, A. E. Schaeffer-Filho, and L. Z. Granville, *Policy-based Dynamic Service Chaining in Network Functions Virtualization*, in IEEE Symposium on Computers and Communication (ISCC 2016), June 2016, pp. 340–345.

[34] E. J. Scheid, C. C. Machado, M. F. Franco, R. L. dos Santos, R. P. Pfitscher, A. E. Schaeffer-Filho, and L. Z. Granville, *INSpIRE: Integrated NFV-based Intent Refinement Environment*, in IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017), May 2017, pp. 186–194.

[35] R. S. Scowen, *Extended BNF - A Generic Base Standard*, in Proceedings 1993 Software Engineering Standards Symposium, Brighton, UK, August 1993, IEEE, pp. 25–34.

[36] M. Sloman, *Policy driven Management for Distributed Systems*, vol. 2, Springer, 1994, pp. 333–360.

[37] Q. Sun, W. Liu, and K. Xie, *An Intent-driven Management Framework*. Work in Progress, `https://www.ietf.org/id/draft-sun-nmrg-intent-framework-00.txt`, last visited December 10, 2019.

[38] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, *Automatic Intent-based Secure Service Creation Through a Multilayer SDN Network Orchestration*, IEEE/OSA Journal of Optical Communications and Networking, 10 (2018), pp. 289–297.

[39] K. Teja, M. Shravani, C. Y. Simha, and M. R. Kounte, *Secured voting through Blockchain technology*, in 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), April 2019, pp. 1416–1419.

[40] D. C. Verma, *Simplifying Network Administration Using Policy-Based Management*, in IEEE Network, vol. 16, March 2002, pp. 20–26.

[41] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, *A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks*, IEEE Access, 7 (2019), pp. 22328–22370.

[42] K. Wüst and A. Gervais, *Do you Need a Blockchain?*, in 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), June 2018, pp. 45–54.

Abbreviations

API	Application Programming Interface
BNF	Backus-Naur Form
CBR	Case-based Reasoning
CLI	Command Line Interface
DFA	Deterministic Finite Automaton
dPoS	Delegated Proof-of-Stake
EBNF	Extended Backus-Naur Form
ECA	Event Condition Action
GUI	Graphical User Interface
IBN	Intent-based Networking
IETF	Internet Engineering Task Force
IRTK	Intent Refinement Toolkit
JSON	JavaScript Object Notation
MIB	Management Information Base
NLTK	Natural Language Toolkit
NV	Network Virtualization
ORM	Object-Relational Mapper
PBBAF	Policy-based Blockchain Agnostic Framework
PBNM	Policy-based Network Management
PBS	Policy-based Blockchain Selection
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIB	Policy Information Base
PMT	Policy Management Tool
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
REST	Representational State Transfer
RFC	Request for Comments
SDN	Software Defined Networking
SLA	Service Level Agreement
TPS	Transactions per Second
VN	Virtual Network
VNF	Virtual Network Function

Glossary

Blockchain A distributed append-only immutable ledger.

Conflicts Conflicting intent or policy options that are invalid in the current configuration.

Exclusions Intent or policy options that are ignored in the current configuration.

Intent An abstract, high-level policy specified in natural language.

Interoperability A characteristic of a system, whose interfaces are completely understood to work with other systems without any restrictions.

Likert Scale A psychometric scale used in research that employs questionnaires. It is a widely used approach to scaling responses in survey research.

Parsing Parsing of intent options specified in natural language into an instance of intermediate data structure. Intent options are validated as part of the parsing process.

Policy A sequence of condition-action rules.

Refinement Refinement of intents specified in natural language into a set of low-level policies. High-level policies are parsed into an intermediate data structure and then translated into low-level policies.

Translation Translation of intent options stored in intermediate data structure into corresponding low-level policy ones. The outcome of the intent translation is a set of low-level policies. Policy options are validated as part of the translation process.

Turing Completeness A programming language is said to be turing complete if it can be used to simulate any turing machine. It is used as a way to express the power of programming languages. The concept is named after English mathematician and computer scientist Alan Turing.

Validation Intent and policy options are validated as part of the refinement, *i.e.*, as part of the parsing and translation. The validation detects conflicts and exclusions which are handled as part of the refinement.

List of Figures

2.1 PBNM architecture 6

2.2 Policy continuum 7

2.3 Generic refinement process 8

2.4 Blockchain Agnostic Framework architecture 9

4.1 Refinement process 15

4.2 IRTK architecture 20

4.3 State chart legend 21

4.4 State chart overview 21

4.5 User states 22

4.6 Timeframe states 22

4.7 Profile and blockchain states 22

4.8 Filter states 23

4.9 Whitelist, blacklist, and modifier states 23

4.10 Whitelist (a) and blacklist (b) states 23

4.11 Modifier states 24

4.12 Cost states 24

4.13 Default states 24

5.1 Average duration of refinement 38

5.2 Average duration of parsing 39

5.3 Average duration of regex-based parsing 39

5.4 Average duration of translation 40
5.5 Total duration of refinement 41
5.6 Groups of respondents 42
5.7 Average number of attributes considered when selecting a blockchain . . . 43
5.8 Number of respondents wanting an additional feature 44
5.9 Average scores of different complexities 45
5.10 Average scores of different representations by technical knowledge 46
5.11 Average priorities for choosing a blockchain 48

List of Tables

3.1 Overview of related work 13

4.1 EBNF notation 17
4.2 Classification of configuration parameters 17
4.3 Validation types 25

5.1 Overview of intent categories 36
5.2 Overview of intent variations 36
5.3 Overview of unlisted responses 44
5.4 Missing values based on complexity 46
5.5 Missing values based on representation 47

Appendix A

Questionnaire

The following email was sent in English and German to students of Blockchain CAS, and the Blockchains and Overlay Network course on November 1, 2019. When clicking the link to the survey, which is below indicated by [...], participants were presented the questions stated in this section.

Subject: *Survey on Blockchain Usage Requirements*

[German version below]

****

Dear Researcher, Student, User,

Have you ever asked yourself which is the optimal blockchain to use for a particular use case? The wide variety of different blockchains and cryptocurrencies (over 2900 according to CoinMarketCap [1]) makes the selection of a suitable blockchain difficult, especially for non-technical users.

Thus, the Communication Systems Group CSG of the Department of Informatics IfI explores approaches that automate and support users in the process of selecting a suitable blockchain for a particular use case. These approaches allow users to comprehensibly specify requirements in natural language. Therefore, users need not know the technical implementation details.

In this sense, the goal of this survey is to evaluate your possible or potential requirements of blockchains and to compare it to those approaches developed at CSG so far. If you could spend approximately 10 minutes of your valuable time, it is recommended to use a desktop computer or laptop to fill in the survey for an optimized presentation.

Please use this link to the survey: [...]

We would really appreciate, if you could run through this survey. All data collected in this questionnaire is anonymized and applied for research purposes only.

Thank you and best regards,
Patrick Widmer

Eder John Scheid
Burkhard Stiller
Communication Systems Group CSG, Institut für Informatik IfI

[1] CoinMarketCap. "Cryptocurrencies by Market Capitalization". Available at `https://coinmarketcap.com/`

Survey on Blockchain Usage Requirements

According to CoinMarketCap [1], more than 2900 cryptocurrencies exist. Most of them run their own blockchain implementation focusing on specific features, such as privacy or performance. With this wide variety of different blockchain implementations, it is difficult for non-technical users to understand key or even minor differences. Thus, even a selection of a blockchain that suits a dedicated use case is difficult.

This questionnaire collects insights on how requirements defined by non-technical users can be potentially translated into low-level (*i.e.*, covering the necessary technical details) blockchain-selection rules.

Please answer the questions in the presented order. If you have any difficulties understanding a question or answer, simply skip it and leave a remark at the end of the questionnaire.

This questionnaire contains 7 questions and takes approximately 10 minutes to complete. All data collected in this questionnaire is anonymized and applied only to research purposes.

[1] `https://coinmarketcap.com/`

Q1: What is your area of specialization? [Single choice]

- Banking and Finance
- Business Administration
- Information Systems
- Computer Science
- *<Other>*

Q2: What do you know about blockchains? [Single choice]

- I know technical details, such as different consensus mechanisms, address formats, node types (*e.g.*, miner and peer), and I have used more than one blockchain implementation.
- I only know how to use a blockchain, *e.g.*, use a wallet application to send/receive cryptocurrencies.
- I only heard of the blockchain concept from media.

- Nothing

Q3: Prioritize the following technical and measurable reasons (from most important (1) to least important (6)) for choosing a blockchain instead of a conventional database? [Ranking]

Please note that no two options can have the same priority.

1. Immutability
2. Decentralization
3. Scalability
4. Security
5. Performance
6. Low costs

Q4: Which of the following attributes of a blockchain implementation do you consider when selecting a blockchain? [Multiple choice]

- Deployment type (private, consortium, public)
- Consensus mechanism
- Transaction costs
- Latency (block time)
- Throughput (transaction rate)
- Transaction size
- *<Other>*

Q5: Suppose you have access to an application that stores and retrieves data in a blockchain for you. Which additional functionality or feature (not provided by the blockchain) should such an application provide? [Multiple choice]

- Encryption (encrypt transaction data before storing in a blockchain)
- Redundancy (store a copy of transaction data in a conventional database)
- Off-chain storage (only store hash of transaction data in a blockchain)
- *<Other>*

Suppose you have access to an application that helps you to choose the most suitable blockchain implementation for a client with specific requirements. Questions 6 and 7 present two possible ways of representing these requirements (one using pseudo-code and one using natural language). Each representation contains three examples on how to specify clients, blockchain attributes, features, and costs.

Q6: How intuitive do you consider the following statements to specify what type of blockchain should be selected for the client? Please state your answer on a scale from 1 to 5. [**Likert scale**]

```
{
  "clients": ["A"],
  "profile": "cheapest",
  "default": true
}
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

```
{
  "clients": ["B"],
  "profile": "fastest",
  "filters": ["private"],
  "interval": "daily",
  "currency": "CHF",
  "threshold": 30
}
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

```
{
  "clients": ["A", "B", "C"],
  "profile": "fastest",
  "filters": ["private", "stable", "popular"],
  "blacklist": ["Multichain"],
  "modifiers": ["encryption", "redundancy"],
  "interval": "daily",
  "currency": "CHF",
  "threshold": 30
}
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

Q7: How intuitive do you consider the following alternative representation of the same statements as in Question 6? Please state your answer on a scale from 1 to 5. [Likert scale]

```
For client A, select the cheapest blockchain as default.
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

```
For client B, select the fastest private blockchain until the daily
costs reach CHF 30.
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

```
For client A, B and C, select the fastest private, stable, and popular
blockchain, except Multichain, then apply encryption and redundancy
until the daily costs reach CHF 30.
```

(1) not at all (2) not very (3) neutral (4) somewhat (5) very (6) N/A

Q: Do you have any remarks that should be considered? [Open ended]

Appendix B

Installation Guidelines

The IRTK prototype is implemented in Python. It requires Python version 3.7 or later. Other dependencies are the `nltk`, the Natural Language Toolkit to tokenize intents, `sqlalchemy` as ORM, `psycopg2` as database driver and `requests` for interacting with the PBS API. During the development to run the tests `ddt` is required for data-driven testing.

B.1 Dependencies

Create a virtual environment:

```
$ python -m venv venv
```

Activate the virtual environment:

```
$ source venv/bin/activate
```

Install the dependencies:

```
(venv) $ pip install -r requirements.txt
```

Deactivate the virtual environment:

```
(venv) $ deactivate
```

B.2 Database

First, start (or alternatively enable) the database service:

```
# systemctl start posgresql.service
```

Switch to the postgres user:

```
$ sudo -iu postgres
```

or without sudo:

```
$ su
# su -l postgres
```

Create a new database:

```
[postgres]$ createdb irtk
```

Restore the database dump:

```
[postgres]$ psql irtk < db_dump
```

This will seed the conversion table of the irtk database with exchange rates for CHF and EUR which are needed for the intent translation. Specifically, exchange rates are required only for intents specifying a non-default currency such as Swiss Francs.

B.3 Configuration

The IRTK configuration file is located under `irtk/config.py`. The URIs of the PDP API and the database can be configured in this file. Moreover, it allows the customization of various options.

```
PDP_API_ENDPOINT_URI = ""
DATABASE_URI = ""

MIN_TX_RATE = ...
MAX_TX_COST = ...
MIN_POPULARITY = ...
MIN_STABILITY = ...

TIME_DAY_START = ""
TIME_AFTERNOON_START = ""
TIME_NIGHT_START = ""
```

B.4 Tests

The IRTK defines unit test cases for the parser and translation. To run these test cases, we can simply use the discover functionality provided by the unittest package as shown below.

```
$ python -m unittest discover irtk
```

B.5 Usage

The IRTK exposes a refine function and a Parser and Translator classes. The refine function acts as a facade for the parsing and translation. However, it provides no control over the parsing and translation. Parser and Translator allow to control the parsing and translation (*e.g.*, incremental parsing).

Refinement

```
from irtk import refine

policies = refine(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)
```

The above example shows how to refine an intent into low-level policies. The refinement parses the intent and translates it into low-level policies.

Parsing and Translation

Alternatively, the parsing and translation of the intent can be done manually.

```
from irtk import Parser, Translator

parser = Parser()
intent = parser.parse(
    "for client"
    " select the cheapest blockchain"
    " until the daily costs reach 30"
)

translator = Translator()
policies = translator.translate(intent)
```

This example shows the basic usage of the IRTK to parse an intent and then translate it into low-level policies.

B.6 Troubleshooting

LookupError: Resource punkt not found

```
LookupError: Resource punkt not found.
Please use the NLTK Downloader to obtain the resource:

>>> import nltk
>>> nltk.download('punkt')

Attempted to load tokenizers/punkt/PY3/english.pickle
```

This error message indicates that the punkt resource from the nltk data is missing. It is required for tokenizing the intents. To resolve the issue run:

```
$ python -m nltk.downloader punkt
```

sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) could not connect to server: Connection refused

```
sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) could not
connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 5432?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and
accepting
        TCP/IP connections on port 5432?
```

This error message occurs when the database cannot be reached. There are many possible causes. Make sure that the database is actually running under the specified URI. The database URI can be specified in the configuration file, *i.e.*, `irtk/config.py`.

ConnectionRefusedError: [Errno 111] Connection refused

```
requests.exceptions.ConnectionError: HTTPConnectionPool(host='localhost
', port=3000): Max retries exceeded with url: /api/policies (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at 0
x7f3711fb13d0>: Failed to establish a new connection: [Errno 111]
Connection refused'))
```

This error message occurs when the API endpoint of the Policy-based Blockchain Selection Framework cannot be reached. There are many possible causes. Make sure that the endpoint is actually running under the specified address. The URI of the PDP API endpoint can be specified in the configuration file, *i.e.*, `irtk/config.py`.

Appendix C

Contents of the CD

The contents of the CD is organized as follows:

Code The source code of the IRTK prototype as a git repository.

Data The measurements collected from the performance evaluation and the responses from the survey. Also contains the scripts to run the benchmarks and the evaluation of the survey responses as well as the scripts to generate the corresponding figures. It also contains the generated figures. Also a git repository including all the version information.

Protocols The protocols of the weekly meetings documenting the progress and discussions.

Slides The slides of the midterm presentation.

Thesis The latex source code and other files of this thesis. Also contains an already compiled version of this thesis. A git repository including all the version information.