University of Zurich^UZH

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

Characterization and Classifications of Blockchains using Softgoal Interdependency Graphs and Machine Learning

Fabian Küffer
Zürich, Switzerland
Student ID: 15-931-421

Supervisor: Eder Scheid, Muriel Franco
Date of Submission: April 12, 2021

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

ifi

Bachelor 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

With the emergence of several thousand Blockchains in recent years, the selection of use-case appropriate Blockchains has become a formidable task. Withal, hitherto no Blockchain standardization body is prevalent, leading to a research gap concerning the characterization and classification of Blockchains. Therefore, this thesis' approach to bridge this gap is by taking inspiration from software engineering principles, namely Functional and Non-Functional Requirements and their characterization using Softgoal Interdependency Graphs. Through decompositions and quantification of relationships, these graphs allow the understanding of how certain Blockchain attributes and aspects are achieved and obtained, and they facilitate the comparison of various Blockchain implementations. Consequently, a Blockchain specific Softgoal Interdependency Graph was designed and evaluated on two relevant use cases. Due to the inherent structure of the graph, a Blockchain characterization relating to four different attributes was achieved, and a Machine Learning evaluation on the use cases resulted in a classification of three different Blockchain clusters. Therefore, this approach can be deemed as successful, and future work can utilize this process and the resulting values to incorporate them into the Blockchain Selection task.

Durch die Popularität von Blockchain in den letzten Jahren ist das Problem entstanden, dass das anwendungsspezifische Auswählen einer passenden Blockchain zunehmend schwierig geworden ist. Umso mehr, Standardisierungsansätze sind bisher noch nicht geläufig, und betreffend der Chrakterisierung und der Klassifizierung von Blockchains existiert bisher eine Forschungslücke. Um diese Lücke zu schliessen, wird in dieser Thesis Inspiration von Software Engineering Prinzipien genommen, nämlich Functional und Non-Functional Requirements und ihrer Charakterisierung durch Softgoal Interdependency Graphs wird erforscht. Durch Dekompositionen und das Quantifizieren von Beziehungen und Werten, erlauben diese Graphen das Verständnis, wie Blockchain Attribute erfüllt werden und ermöglichen gleichzeitig auch die Gegenüberstellung von verschiedenen Blockchain Implementierungen. Infolgedessen, wurde ein Blockchain spezifischer Softgoal Interdependency Graph entworfen und an zwei relevanten Anwendungsfällen evaluiert. Durch die dahinterliegende Struktur des Graphens, wurde eine Charakterisierung von Blockchains anhand vier verschiedener Attribute erreicht, und durch eine Machine Learning Evaluation resultierte eine Klassifizierung in drei verschiedene Blockchain Gruppen. Anhand der Resultate, kann dieser Ansatz als erfolgreich angesehen werden, und zukünftige Arbeiten können die erreichten Werte und den Prozess in das Unterfangen der Blockchain Auswahl inkorporieren.

Acknowledgments

I would like to express my gratitudes to everyone that made this thesis possible, my supervisors, Eder J. Scheid and Muriel Franco; especially Eder J. Scheid, for the continuous support and helpfulness, the Communication Systems Group at the UZH, and last but not least, Prof. Dr. Stiller who offered the opportunity for this thesis. Furthermore, I would also like to thank Pascal Kiechl for his proofreading inputs.

Also, I would also like to thank my friends and family for their support and understanding.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1

1.1 Description of Work 1

1.2 Thesis Outline 2

2 Background 3

2.1 Functional (FR) and
Non-Functional Requirements (NFR) 3

2.2 NFR Framework and SIG 4

2.2.1 SIG Score Calculation 7

2.2.2 Goal Refinement and SIG Design 10

2.3 Blockchain 11

2.3.1 Consensus Mechanism 12

2.3.2 Deployment Models 14

2.3.3 Blockchain Features 15

3 Related Work 17

3.1 SIG 17

3.2 Blockchain Classification 18

3.3 Comparison and Discussion 20

4 Characterizing and Classifying Blockchains **23**

4.1 Blockchain SIG Design 23

4.1.1 Requirements Decomposition 25

4.1.2 Quantification of Values 36

4.2 Implementation 46

4.2.1 Frontend 46

4.2.2 Server API 47

4.2.3 Computation of Scores 51

5 Evaluation **55**

5.1 Use Cases 55

5.1.1 Internet of Things (IoT) 55

5.1.2 Supply Chain Management (SCM) 57

5.2 Performance 58

5.3 Machine Learning 58

5.4 Discussion 65

6 Summary and Future Work **67**

Bibliography **67**

Abbreviations **75**

List of Figures **75**

List of Tables **78**

A IoT Use Case Calculations **81**

B Supply Chain Management Use Case Calculations **85**

C Installation Guidelines **89**

D Contents of the CD **91**

Chapter 1

Introduction

Since the inception of Bitcoin in 2008, and the mining of its first genesis block in early 2009, Bitcoin's popularity started a new era of digital currencies and Blockchain technology, and now, several years later, thousands of Blockchains (BC) have emerged, each with different functionalities and designed as solutions for unique problems [42, 5, 18].

Consequently, due to the plethora of available BC to choose from, and the BC standardizations not being advanced enough, the predicament has arisen, that classification and characterization of BC, but also the selection of BC for specific use cases is no easily accomplishable undertaking [21, 55].

An approach to solve these problems is by taking inspiration from Software Engineering (SE) principles, namely Non-Functional Requirements (NFR) and Functional Requirements (FR) and applying them to BC. Namely, by capturing specific BC implementation subtleties, and devising a Softgoal Interdependency Graph (SIG) according to the NFR Framework, *cf.* [17], a BC characterization and classification can be achieved. In particular, by selecting BC NFRs, such as *Performance* or *Security* and decomposing them into more specific attributes and incorporating design decisions such as *Proof-of-Work*, a graph is synthesized, which includes contributions and interdepenencies between the various selected BC aspects, in which BC implementations can be evaluated based on their fulfillment of user requirements.

Furthermore, by applying a Quantification Extension, *cf.* [1], to a SIG, contributions and values can be quantified and the fulfillment of requirements can be scored and evaluated across different BCs. Therefore, the resulting scores can be processed and they present themselves to be used in various applications, inter alia, BC selection, further motivating the approach of applying the NFR Framework to BC.

1.1 Description of Work

Thus, the goal of this thesis is to approach the characterization and classification of BCs by applying Software Engineering principles such as NFR and FR on BCs by utilizing the NFR Framework and its Quantification Extension. To achieve this, the NFR Framework and its extension have to be fully understood and BC literature review must be undertaken, such that appropriate NFRs and FRs for the SIG can be selected. By designing and proposing a SIG and choosing its parameters, it then serves as a general BC characterization, capturing various BC aspects.

Sequentially, eight BC implementations are selected and the SIG is evaluated on two specific use cases, leading to a quantification of values for the BC aspects. These values are then used for Machine Learning algorithms to evaluate the SIG and to further characterize and classify the BCs.

Moreover, a BC SIG application is devised and implemented to offer a visualization of the SIG and to also enable an accessible way to calculate the scores. Finally, the SIG implementation is evaluated based on calculation process time for given BCs, and possible case studies for applications are discussed.

1.2 Thesis Outline

This thesis has been structured in the following manner: In Chapter 2 the FR and NFR foundations are provided, leading to their appliance in the NFR Framework and its quantification extension, which are thereafter walked through, and hereafter, a concise introduction to BC is given and the necessary BC background is established. In Chapter 3 BC characterization and classification approaches are discussed, and relevant SIG usages are explored. Further, in Chapter 4, a SIG is designed and decomposed, the quantification process is applied and the implementation of a SIG computation system is documented. Subsequently, in Chapter 5, multiple use cases are chosen to evaluate the SIG, a Machine Learning evaluation of the SIG is performed, and further, a performance evaluation of the SIG score computation is given. Finally, a concluding summary is presented and future works are explored in Chapter 6.

Chapter 2

Background

In this chapter, the theoretical background for Requirements, the NFR Framework, the Quantification Extension and Blockchains is explored. This background serves as an overview of the involved concepts and provides the necessary foundations leading to Chapter 4.

2.1 Functional (FR) and Non-Functional Requirements (NFR)

In Software Engineering (SE), requirements are seen as the real-world goals for functionality or constraints on systems, and Requirements Engineering (RE) concerns itself with their precise specification [78]. Besides, the exact elicitation and specification of requirements is not to be overlooked, since these processes have been identified as a major cause for software project failures [1].

Requirements can be classified into Functional Requirements (FR) and Non-Functional Requirements (NFR) [15]. While FRs concern themselves with the functional characteristics of software artifacts and specify what a function of a system must be able to perform, or what a product must do, NFRs are colloquially referred to as the "-ilities" of software, since they take the non-functional perspective [15, 24]. Although NFRs are commonly represented as sentences and listed separately from FRs in the Software Requirements Specification, the requirements engineering community has notably had no consensus over the nature of NFRs and how to specify, elicit, and validate them [15, 24]. In Figure 2.1, depicting a taxonomy of requirements, NFRs are defined as attributes of a system, or constraints on a system [24].

Typical NFR aspects to consider are *usability*, *flexibility*, *interoperability*, or non "-ility" attributes such as *user-friendliness*, *coherence* or *security* [15]. Due to their non-functional nature, NFRs are intrinsically hard to specify and verify [24]. Thus, they can be seen as "soft" [15, 1], which will be important for Section 2.2. In this sense, SE has historically focused more on FRs due to the need and demand for being able to run systems with

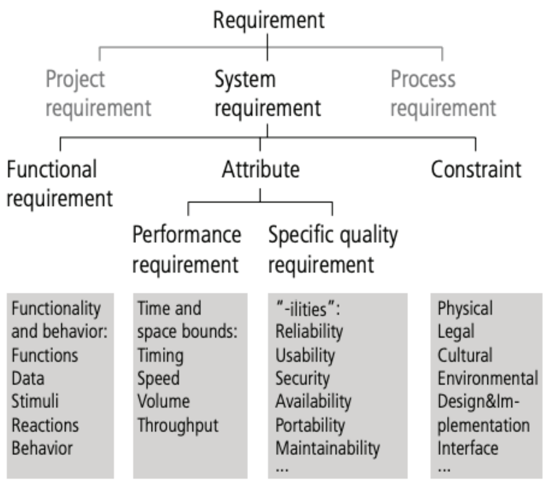

Requirement
Project requirement
System requirement
Process requirement
Functional requirement
Attribute
Constraint
Performance requirement
Specific quality requirement
Functionality and behavior: Functions Data Stimuli Reactions Behavior
Time and space bounds: Timing Speed Volume Throughput
"-ilities": Reliability Usability Security Availability Portability Maintainability ...
Physical Legal Cultural Environmental Design&Implementation Interface ...

Figure 2.1: Taxonomy of requirements: NFR are attributes or constraints on system requirements [24].

the basic functionality as soon as possible [15]. Further, instead of specifying NFRs methodically, an "*ad hoc*" manner has often been applied in development, meaning that NFRs were retrofitted during development and developers relied on their intuition when dealing with NFRs, leading to unsatisfactory user expectations [16]. Additionally, since the non-functional aspects directly complement the functional components, a lack of NFRs leads to inappropriate functionality [16]. Therefore, both sides (*i.e.,* FR and NFR) must be considered for successful software projects [15].

However, despite the previously mentioned predicaments, various frameworks have been developed to address the definition of NFRs [1]. One that stands out is the "NFR framework" [17], which is discussed in Section 2.2, and that is the basis for the Softgoal Interdependency Graph (SIG) [1] described in Section 2.2.

2.2 NFR Framework and SIG

The NFR Framework is a process-oriented approach for NFRs and builds upon decision support systems [16], and offers a precise approach to deal with the decision making process for NFRs [1]. It can be applied in early stages of the design phase, or during late requirements specifications [1].

By utilizing the notions of *goals* and *softgoals (SG)*, the NFR Framework abstracts the conceptions of requirements, *e.g.,* functionality, constraints and attributes, and, thus, SGs are defined as goals, where there are no clear boundaries for their acceptances and accomplishments [15, 44]. Therefore, SGs are considered *satisficed*, when sufficient for, and little indices against, are present to support their conclusion [44]. This notion is used analogous to the meaning of the word *satisficing*, which considers a goal as being sufficiently satisfactory without necessarily being optimal [44, 17]. Further, development techniques, which are design choices to meet the NFR, are termed as *operationalizations (OP)*, and are incorporated equally into the framework [44, 17].

The NFR Framework, as proposed by [17], also includes the concepts of *contributions*, meaning that SGs can contribute to other SGs in various ways:

- MAKE (++): Stands for providing sufficient positive support. A single offspring SG being satisficed "makes" the parent SG satisficed as well.
- HELP (+): Stands for providing partial positive support. A single offspring SG being satisficed leads to its parent being partially positively supported.
- HURT (−): Stands for providing partial negative support. A single offspring SG being satisficed leads to its parent being partially negatively supported.
- BREAK (−−): Stands for providing sufficient negative support. A single offspring SG being satisficed, "breaks" the parent SG and denies it.
- AND: Relates to a group of SG offsprings to their parent. If all offsprings are satisficed, the parent becomes satisficed as well.
- OR: Relates to a group of SG offsprings to their parent. If one offspring is satisficed, the parent becomes satisficed as well.

With these *contributions*, the framework permits NFRs to be conflicting goals, or for them to be synergetic to each other [16]. Thus, due to its modular structure, development alternatives can be considered simultaneously to achieve the same NFR [16].

Consequently, using the aforementioned building blocks, the original NFR framework, proposed by [17], contains the following steps:

1. Acquiring knowledge about the domain, NFR and FR,
2. Identifying NFRs for the domain,
3. Decomposing NFRs,
4. Identifying OPs,
5. Dealing with ambiguities, tradeoffs, priorities and interdependencies among NFRs and OPs,
6. Selecting OPs,
7. Supporting decisions with design rationale,
8. Evaluating the impact of decisions.

These steps can be applied non-sequentially and might need to be iterated over multiple times [17]. Instead of a "top-down" approach, the development process can be viewed as "bottom-up", meaning that one is able to start with certain *OPs* in mind to reach certain *SGs* [17].

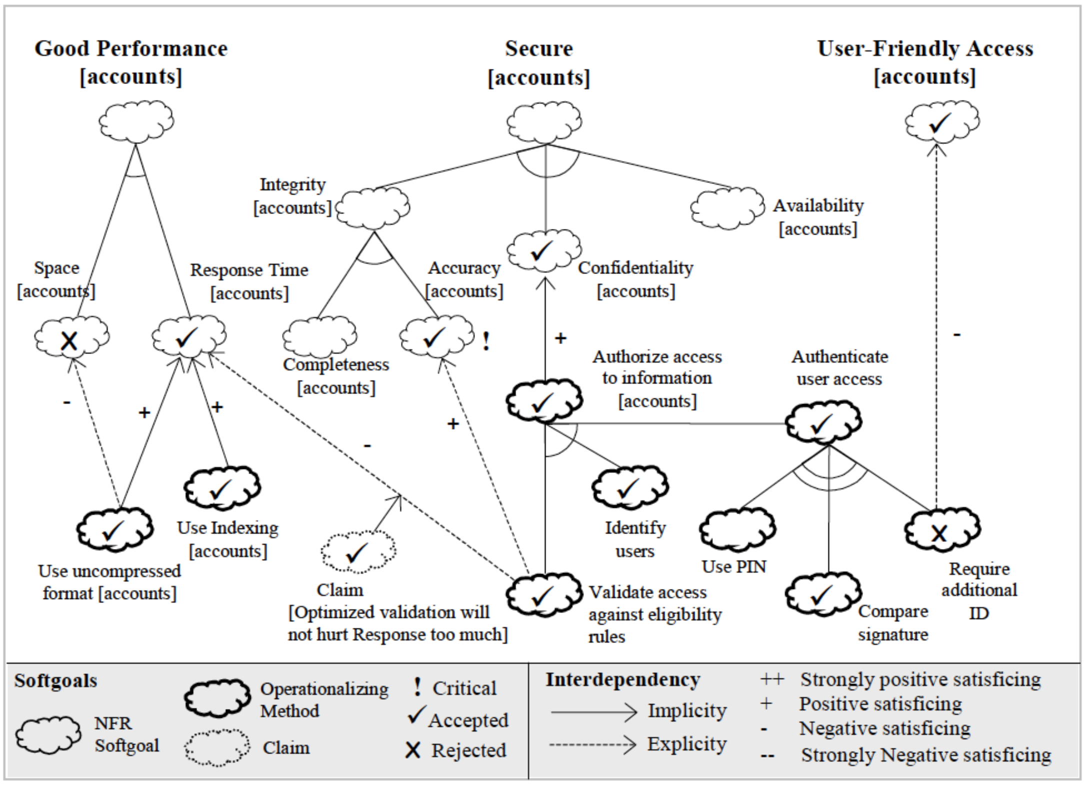


Figure 2.2: A Softgoal Interdepdency Graph using the original NFR framework [44].

Following the definiton of *SGs* and *OPs*, the NFR framework can be visualized in a SIG, which records the design and reasoning process that was acquired through the steps [17]. Figure 2.2 illustrates an example of a SIG for a banking software.

Given the vertices,

- SG: *Softgoals*,
- LSG: *Leaf-Softgoals (LSG)* gained from the refinement process, Section 2.2.2,
- OP: *Operationalizations*,

and the edges,

- $\uparrow^{++}$: Make,
- $\uparrow^{+}$: Help,
- $\uparrow^{-}$: Hurt,
- $\uparrow^{--}$: Break,
- Λ: And,
- Λ: Or,

a SIG is a set defined by [56] as a graph:

$$SIG = (V, E)$$
$$V \in \{SG, LSG, OP\}$$
$$E \in \{\uparrow^{++}, \uparrow^{+}, \uparrow^{-}, \uparrow^{--}, \mathsf{A}, \mathbb{A}\}$$

Although the NFR framework allows reasoning about NFR and is an important cornerstone for NFRs, no quantifications are possible. To address this conundrum, [1] proposes a quantitative extension to the previously mentioned steps. Since the quantification is a fundamental layer of this thesis, the extended NFR framework will be considered next.

2.2.1 SIG Score Calculation

The extended NFR framework, as proposed by [1], is built upon the NFR framework and allows for a quantitative reasoning. To achieve this, the original NFR Framework steps have to be altered, and the resulting modified steps are the following:

- Identify Softgoals,
- Decompose Softgoals,
- Assign Leaf-Softgoal Weights in the range [0.0, 1.0],
- Identify Operationalizations,
- Calculate Operationalization Scores,
- Select Operationalizations,
- Calculate Leaf-Softgoal Scores,
- Calculate Softgoal Scores,
- Calculate Attainment.

To demonstrate its capabilities and inner workings, the next paragraphs will be following the example from Figure 2.2 to calculate the scores and attainment from Figure 2.3 [1]. This example assumes a already decomposed SIG, with the given SG, depicted as white clouds in Figure 2.2, and the contributions $SG_{contrib}$, by which offspring SG relate towards their parents according to Table 2.1.

Table 2.1: Contributions for SIG [1].

Symbol	Name	Contribution
$\uparrow^{++}$	MAKE	1
$\uparrow^{+}$	HELP	[0,1]
$\uparrow^{-}$	HURT	[-1,0]
$\uparrow^{--}$	BREAK	-1
A	AND	$\frac{1}{NumChild}$
$\mathbb{A}$	OR	1

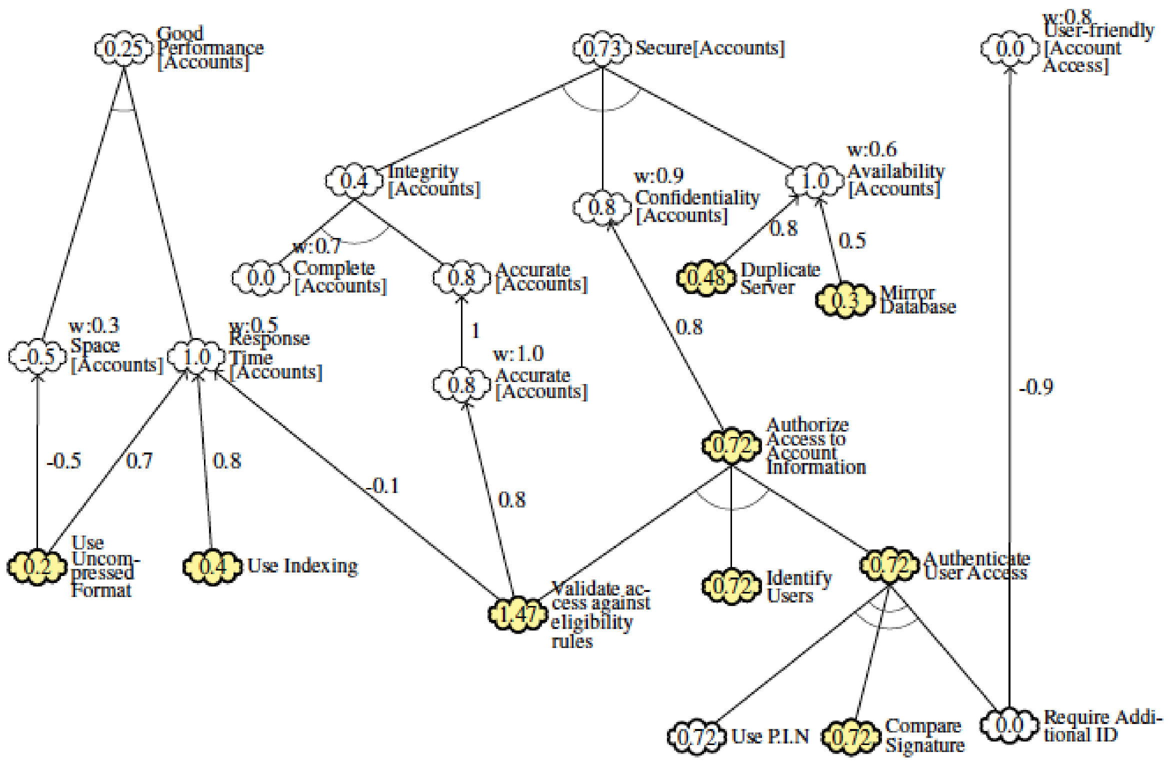

0.25 Good Performance [Accounts]
0.73 Secure[Accounts]
w:0.8 0.0 User-friendly [Account Access]
0.4 Integrity [Accounts]
w:0.9 0.8 Confidentiality [Accounts]
w:0.6 1.0 Availability [Accounts]
w:0.7 0.0 Complete [Accounts]
0.8 Accurate [Accounts]
0.48 Duplicate Server
0.3 Mirror Database
w:0.3 -0.5 Space [Accounts]
w:0.5 1.0 Response Time [Accounts]
w:1.0 0.8 Accurate [Accounts]
0.72 Authorize Access to Account Information
0.2 Use Uncompressed Format
0.4 Use Indexing
1.47 Validate access against eligibility rules
0.72 Identify Users
0.72 Authenticate User Access
0.72 Use P.I.N
0.72 Compare Signature
0.0 Require Additional ID
-0.5
0.7
0.8
-0.1
0.8
1
0.8
0.8
0.5
0.8
-0.9

Figure 2.3: The SIG for the bank example [1].

Consequently, *LSG weights* are assigned to the LSG according to [1], where

$$LSG_{weight} \in [0, 1]$$

and a LSG_{weight} of 0 denotes a *non-critical* SG, while a score of 1 stands for *critical* SGs with significant impact on decisions [1]. In Figure 2.3, the critical SG Accurate [Accounts] was assigned a weight of 1, while Response Time [Accounts] was assigned a weight of 0.5.

In the next step, *Identify Operationalizations*, the relation between OPs and LSGs are gathered, and Table 2.1 can be similarly used to denote the contributions, here *impacts* between OPs and LSGs, termed $impact_{LSG\times OP}$ [1], and OPs that do not contribute towards a SG, feature an impact score of 0, since no relationship exist [1].

Given the $impact_{LSGxOP}$, the OP Scores can be computed in the following way [1]:

$$OP_{score} = OP_{parent.score} + \sum_{LSG} impact_{LSG\times OP} \times LSG_{weight} \tag{2.1}$$

This calculation is performed *top-down*, and OP_{score} denotes the contribution of OPs on the system [1]. Consequently, $OP_{parent.score}$ is 0, if no parent OPs are given [1]. In this example the OP *Use Uncompressed Format* got its score in the following way:

$$\begin{aligned} OP_{score} &= 0 + (-0.5 \times 0.3 + 0.7 \times 0.5) \\ &= 0.2 \end{aligned}$$

In the next step *Select Operationalizations* the OP_{score} is used to determine if operationalizations are to be accepted or rejected [1]:

- $OP_{score} \in (0, \infty)$: Advantageous to the system,
- $OP_{score} = 0$: No effect to the system,
- $OP_{score} \in (-\infty, 0)$: Disadvantageous to the system.

Generally, in the case of *AND-operationalizations*, the developer must take action if the rejection of a child would make the parent rejected as well. In contrast, in the case where OPs are alternatives to each other, they are *OR-operationalizations*, then the OP with the maximum score is selected [1]. The previously calculated score of 0.2 means, that the OP *Use Uncompressed Format* has a low, but beneficial effect on the system.

Afterwards, the LSG Scores LSG_{score} can be calculated using the following equation [1]:

$$LSG_{score} = max(min(\sum_{OP_{accept=true}} impact_{LSG \times OP}, 1), -1) \tag{2.2}$$

The LSG score denotes the satisficing percentage of a SG, and $LSG_{score} \in [-1, 1]$ due to the min and max functions that are necessary to cap the score, since multiple positive or negative contributions are possible [1]. In other words, a SG that was to 100% satisficed will get a LSG score of 1, while a SG that was not satisficed will result in a score of -1 [56].

Using equation 2.2, the LSG_{score} of Response Time [Accounts] is 1.0, meaning that it is fully satisficed, and it can be calculated in the following way [1]:

$$\begin{aligned} LSG_{score} &= max(min(0.7 + 0.8 + (-0.1), 1.0), -1.0) \\ &= max(min(1.4, 1.0), -1.0) \\ &= max(1.0, -1.0) \\ &= 1.0 \end{aligned}$$

After all LSG scores have been calculated, the next objective is to calculate the SGs further up in the graph [56]. In the equation 2.3 for SG_{score}, $SG_{child.contrib}$ denotes the contributions from Table 2.1 [1]:

$$SG_{score} = max(min(\sum SG_{child} \times SG_{child.contrib}, 1), -1) \tag{2.3}$$

Similarly to the LSG_{score}, the SG_{score} is bounded between [-1, 1], and using the *AND-contribution* the score of 0.25 for Good Performance [Accounts] can be calculated in the following way, [1]:

$$\begin{aligned}
SG_{score} &= max(min(((-0.5) \times \frac{1}{2}) + (1.0 \times \frac{1}{2})), 1.0), -1.0) \\
&= max(min((-0.25 + 0.5), 1.0), -1.0) \\
&= max(min(0.25, 1.0), -1.0) \\
&= max(0.25, -1.0) \\
&= 0.25
\end{aligned}$$

In the final step, the optimal and the obtained attainment scores can be calculated, and while the optimal scores denote how much could be potentially obtained by satisficing the SGs to their fullest extent, the obtained score depicts how much was actually obtained during the process [1].

$$attainment_{optimal} = \sum_{LSG} LSG_{weight} \tag{2.4}$$

$$attainment_{actual} = \sum_{LSG} max(LSG_{score} \times LSG_{weight}, 0) \tag{2.5}$$

By comparing both scores and trying several implementations, this metric can be used to select the best implementation and to facilitate decision support or verification [1].

Consequently, from the example it is clear, that this light-weight quantification extension is powerful, and the performed evaluation by the authors found no difference in recommendations compared to the original framework [1]. However, they also note that if little or no tradeoffs are present regarding the design decisions, meaning that a high number of *one-to-one mappings* are prevalent, the extension is less useful, and it is truly able to excel when a high number of trade-offs are available [1].

2.2.2 Goal Refinement and SIG Design

While the previous sections covered the calculation and the steps of the Framework, this section explores the process of designing a SIG.

After studying the requirements and identifying the initial SGs, the predicament can occur, that these SGs are too broad and abstract, and thus, too unspecific to deal with, and as such, these SGs need to be decomposed by the developer into smaller and more specific components [17]. Hence, the SGs are refined into more specific sub-SGs, and the sub-SGs contribute upwards to the decomposed parent SG [17]. At last, this process helps the clearing of ambiguities, since different interpretations of the requirements are possible and misunderstandings are obviated [17]. Simultaneously, the decomposition can also be done for the FR, that are represented by hardgoals and the OPs [64].

During the decomposition step, the developer can choose between the different contributions, which are discussed in more detail in Section 2.2.1, in particular, the developers decide how child SGs relate to their parent SG [64].

Additional to the contributions in the decomposition step, the the two available methods to decompose SGs are by *type* or by *topic* [17]. Decomposing by type means that the sub-SGs cater to the same type, *i.e.,* for performance there are space performance and time performance, as can be seen in Figure 2.2 [17]. The offspring SGs then take a subtype of the parent softgoal type, and also the topic of the SG, such as *[Account]* [17]. On the other hand, decomposing by topic refers to differentiate and distinguish the topic, *i.e.,* accounts can be split into regular accounts and special accounts, while still keeping the type of their parent [17]. These decompositions can be done in a *"divide-and-conquer"* fashion, by trying to solve each component alone the full problem set is tackled [17].

The decomposition step can be seen in Figure 2.2, where the Secure [accounts] SG has been broken down into three sub-SGs [17]:

- integrity,
- confidentiality, and
- availability.

This decomposition was done with the *AND-contribution*, that is to say, that all three sub-SGs must be met to satisfice the parent SG [17].

Additionally, to identify critical SGs, the concept of *priority softgoals* is introduced [17], and in Figure 2.2, the SG Accuracy [accounts] has been denoted a Priority SG. In the SIG, priority SGs produce an offspring, with the same type and topic, as seen for Accurate [Accounts] in Figure 2.3 [17]. This concept was introduced to capture that some SG are vital to the sucess of the system, and tradeoffs among SGs may need to be made [17].

At the end of the refinement process, an initial design of a SIG will be developed. However, to achieve the final design, an iterative process has to be put in place [17].

2.3 Blockchain

The name *Blockchain (BC)* is fitting for this data structure, since a BC is exactly that, a chain of blocks, and each block in the chain contains a cryptographic hash of and pointer to the previous block [76]. Thus, a BC can be seen as an append-only linked-list [73]. Furthermore, every block contains transactions, which are the atomic data structure of a BC, and capture a transfer of digital tokens from a sender to a receiver address as a record [73]. Although these transactions can be of monetary nature, they can also be issued, for instance, to run a program, named *smart contract (SC)*, or can be digital representations of real-world assets [76, 21]. These transactions are created in a

BC network or exchanged with peers directly, all without the need of any intermediate entities [76, 21].

As described by [80, 76, 21], the most prominent BC technology features are the characteristics of being *decentralized* and offering *anonymity*, and in particular, in contrast to centralized systems, there are no central coordinating entities, since all peers in the network can communicate directly with each other, consequently avoiding potential performance bottlenecks. Additionally, since the communication between the nodes happens through generated addresses, and no private information is stored, the decentralization increases the anonymity of users in the network [21]. Furthermore, the data in the blockchain is *immutable*, thus, in the general case the data is *tamper-resilient*, since the transactions are stored in a distributed manner over the network and validated independently, and though possible, a tremendous effort would be needed to alter the data [79, 21]. Furthermore, the transactions are transparently stored and can be verified and traced back to previous transactions, offering auditability opportunities [21, 79].

Withal, these aspects lead to BC allowing mistrusting parties to interact with each other without the need of a Trusted Third Party (TTP) [76]. Emphasizing that there is no need to trust a centralized bank to perform transactions; currencies can be freely traded independently from governments and other authorities [5]. Following, the most notable characteristics that vary in the various BC implementations will be discussed in the next sections.

2.3.1 Consensus Mechanism

Given that the nodes in a BC network do not necessarily trust each other and no central entity is present, how can a consensus amongst the nodes be reached? Consensus mechanism try to solve this conundrum with various strategies [80]. At the same time, consensus mechanism also solve the challenge of double-spending (*i.e.,* the issue where users are able to spend the same tokens twice [71]).

The most commonly used consensus mechanism are listed in the next items. These are a select few from a vast amount, and there are several proposals of consensus mechanism that tackle different problems and provide new features. Extensive surveys of novel approaches have been concluded by [7, 73].

Proof-of-Work (PoW) is notably used by Bitcoin (and Ethereum) [80], and is a competition between the nodes to calculate the hash value of the succeeding block to append to the BC [55, 80]. The challenge lies in finding a hash value that is below a certain target threshold, and once such a value has been found by a node, this value is broadcasted over the network and mutually confirmed by the nodes [55, 80].

In particular, hash algorithms are used, since for an arbitrary-length input, a fixed-length deterministic output is produced, that is always the same for the given input [2]. Furthermore, given an input and the hash algorithm, the verification of the output is easily done, however, the reverse is not true, to find a value below the target threshold one must randomly try until a hashed input fits the criteria [2]. This mechanism is therefore named

PoW, since the found hash value is proof, that a considerable amount of work has been done by the miner [2].

Upon verification, the new block is then added to the BC [80]. This hash calculation process is also called *mining*, and since this process is not only computationally expensive and time consuming, but also energy intensive, incentive mechanisms are needed to reward the winner of the competition and incentivize users to participate [80, 71]. However, Bitcoin's energy consumption already hit an astronomous 121 TWh a year, by comparing it to the energy consumption of various countries, it has already surpassed Argentina and would be in the top 30 energy users worldwide [19].

In Bitcoin's case, this mining process takes on average 10 minutes, and this rate is essentially the heartbeat of Bitcoin, it directs the issuance of currency and also the speed of transactions [2]. In fact, this time is not per chance, but is carefully kept in place since computers are expected to get faster in the future, and the 10 minutes average time is to be kept constant, the difficulty of the PoW target is being constantly and dynamically adjusted [2]. Essentially by comparing the mining time of the last 2016 blocks and its expected 10 minutes per block time, the difficulty can be either increased or decreased to match the expected 10 minutes time [2].

Proof-of-Stake (PoS) is an efficient alternative to PoW that tries to mitigate its considerable energy depletion by randomly selecting a mining block leader for the next block based on the concept of *stake* [73, 80]. This process is also called "virtual mining", since mining does not consume any resources [73]. For this reason, the terms "minting" or "forging" can be used instead of mining [49].

This mechanism follows the idea, that participants with a higher stake in a BC, e.g. owning a greater amount of its currency, have less incentives to attack the network [80]. In particular, in the planned PoS Ethereum upgrade, stake refers to a security deposit that will be forfeited if the miner behaves maliciously [38, 49]. In the similar variant, *Proof-of-Capacity*, stake refers to hard drive space that needs to be allocated [80].

Nonetheless, this essentially means that participants with the highest stake, would dominate the network, and as such various solutions have been proposed that not only take the stake size into account to decide on the miner selection [80]. In Peercoin's case, the originator of this concept, the stake is measured as "coin age", a product of the miners token and the holding time of these tokens, and the coin age increases the probability to mine the next block [73].

However, PoS is vulnerable to the "nothing-at-stake-attack", where conflicting blocks could be generated by the block leader on various forks of the BC, since the creation of these blocks do not cost anything, which could facilitate "double spending" [73, 38]. Furthermore, PoS can also be exploited by the *grinding-attack*, an attack on the leader election process [73].

delegated Proof-of-Stake (dPoS) builds upon the PoS mechanism by making the nodes elect *delegates* (also called *witnesses*), who in turn are permitted to generate and validate blocks [80, 49]. Since dishonest nodes can always be voted out, there are less incentives for dishonest behavior [80]. Finally, dPoS is faster and more scalable than

PoS, as there are only a few delegates, typically 20, and thus blocks can be confirmed quickly [49, 80]. However, it is noteworthy, that due to the small number of validators, this also introduces a considerable amount of centralization [55].

Proof-of-Authority (PoA) features N trusted nodes, the *Authorities*, who stake their identity and are allowed to propose new blocks to achieve distributed consensus [20, 46]. The principle behind this mechanism is, that the trusted nodes are incentivized to not attack the network, since their identity is known, they would harm their reputation by doing so [46]. This consensus works in a round-robin fashion, where in each round one of the authorities, a *mining leader* is in charge [20]. However this mechanism requires at least $N/2 + 1$ nodes to be honest [20].

IOTA takes a different approach to the other consensus mechanism, and considers a *Directed Acyclic Graph (DAG)* instead of blocks to structure the transactions [73]. Therefore, transactions are directly pushed onto the DAG, and each transaction verifies two additional transactions [73, 47]. This approach significantly improves network throughput, since no mining needs to be done [73, 47]. Nonetheless, IOTA is currently under development and features a central coordinator entity, the shift to a fully decentralized system is still underway [48, 47]. However, IOTA is just a variant of nonlinear block organizations, there exist others that also use the underlying pricinples of DAGs [73].

2.3.2 Deployment Models

In essence, a possible BC deployment classification can be done on two axis; permissioned vs. permissionless and public vs. private [55]. The former refers to read and write permissions and the latter denotes the data visibility of a BC [55]. Being able to write to a BC means, that an entity is allowed to add transactions and extend the BC, or in some cases, even validate blocks [76]. On the other hand, a *reader* can not extend the BC, but is allowed to read the content on the BC [76].

Combining the two classifications, the four categories emerge, [55]:

- Public permissionless, open read and write access to all nodes [55].
- Public permissioned, open read but restricted write access [55].
- Private permissionless, a closed BC network [55].
- Private permissioned, restricted access [21] and selection of participants and their rights is handled by a centralized authority [55, 76].

From this classification, it follows, that *permissioned* BCs can only be seen as *partially decentralized*, however, they boast speed advantages compared to *permissionless* BCs [36]. Moreover, [21] notes, that with restricted access and known participants in the network, permissioned or private ledgers do not require a extensive transaction validation schemes, such as *PoW*.

2.3.3 Blockchain Features

The BC implementations have various features and idiosyncrasies that are not shared by all implementations, and therefore, this section gives a short overview on Smart Contracts and Performance metrics for BCs.

Smart Contracts (SC) are an addendum to BCs, allowing developers to add business logic to BCs [33]. In particular, they can be seen as *self enforcing digital contracts*, that grant the BC further capabilities besides transactions [76].

However, SC support strongly depends on the BC, for instance, Ethereum's SC are quite extensive, and in contrast to Bitcoin, they can be developed with the *Turing Complete* programming language Solidity, which allows for many powerful possibilities [33]. Further, although SCs open up many opportunities, such as a *disintermediation* between contract parties, SC do not currently have the necessary legal support, hindering their advancements and usefulnesses [33].

Concerning ***performance***, there are various aspects that can be considered, one of them is *throughput*, defined in *transactions per second* ***(tps)*** [23]. As per [55, 23, 32],

$$TransactionsPerBlock = \frac{BlockSize}{AverageTransactionSize} \tag{2.6}$$

then the maximum theoretical throughput can be calculated as

$$tps = \frac{TransactionsPerBlock}{BlockInterval} \tag{2.7}$$

given the metrics:

- *Transaction Size* refers to the average transaction size in the BC.
- *Block Size* denotes the size of a block in the BC
- *Block Interval* relates to the latency that takes place before blocks are written into the BC, this is directly dependent on the chosen consensus mechanism.

However, [23] show, that performance and security are in some aspects *interlinked*, and increasing performance can have repercussions on security.

Since consensus mechanism function on the basis of incentives, miners are rewarded for their work, they collect ***Transaction Fees*** as well as ***Block Rewards*** [55]. These rewards can be seen as a necessity to secure the network and to avoid attacks on the network, such as flooding the network with transactions [2]. How much transaction fees cost depends on the BC, though in Bitcoin they are subject to market forces, the more one is willing to pay, the higher the probability that their transaction will be included in the next block [55, 2]. Ultimately, although transaction fees are not mandatory, not including them makes it unlikely that the transaction will be processed, as in it might not be included in a block [2].

Furthermore, BC are also able to store arbitrary data with transactions and ***Data Size*** can be used as a criteria to choose between various BC implementations, and for some use cases, enabling to depend on the BC as an immutable and transparent database [55].

Chapter 3

Related Work

To best of the authors knowledge, SIGs or NFRs have not yet been employed in the BC classification context. However, such concepts are not novel and have been successfully applied in different areas of Computer Science. Thus, in Section 3.1 an overview of the SIG applicability is presented, and in Section 3.2 the BC classification work is explored.

3.1 SIG

Influential to this thesis has been the work by [17], who collected early endeavors on NFRs and devised the NFR framework and SIGs. Further, noteworthy for this thesis is the cornerstone laid by [1], to extend the SIGs to allow quantitative reasoning by being able to calculate softgoal scores, which increases the realm of possibilities for the SIG usage. They showed an exemplary usage of the extended NFR framework by going through an early example described by [17].

- In [44], SIGs were adopted to the context of *data warehouses*, to analyze design decisions during the requirements specification phase. They were able to analyze and apply a SIG design to the ongoing development work of a large data warehouse system, and SIGs were used to support implementation decisions.
- [56] proposes a SIG to classify and quantify different aspects, such as security, performance, and privacy, that a Virtualized Network Function (VNF) is able to provide. The values calculated by the SIG are used to select and chain such VNFs in VNF forwarding graphs to satisfy high-level policies input by network operators.
- [13] used a SIG to capture solutions and also characteristics for the novel NFR "Invisibility", meaning the ability to hide technology from users in the context of Ubiquitous Computing and Internet of Things (IoT). They identified 56 specific solutions for the SIG, though they did not consider in their design negative contributions towards other NFRs.

3.2 Blockchain Classification

Regarding the BC Classification, there have been some efforts and endeavors to standardize BC, and a select few are presented here.

- [4] introduced a taxonomy to classify 50 Distributed Ledgers (DL) with 19 attributes in two dimensions, namely attributes concerning the Distributed Ledger Technology (DLT) and aspects concerning the cryptoeconomic design (CED). This taxonomy and the classification were evaluated by contributors to the BC community and via Machine Learning Analysis, Multiple Correspondence Analysis and K-Means, and they were found to be useful and accepted and also extensible for future attributes. Based on the classification, they were also able to propose a design guideline for DLs via Machine Learning. The taxonomy is shown in Figure 3.1.

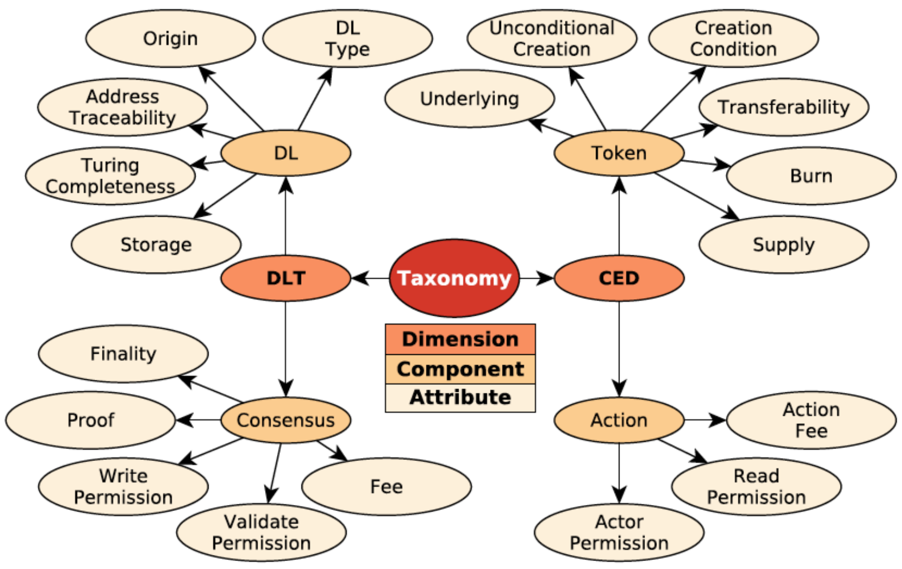


Figure 3.1: Distributed Ledger Taxonomy [4].

- [67] designed a taxonomy tree for DL with 30 attributes that were inferred from an analysis of BC components via a literature review. The taxonomy consists of main, sub and also sub-sub components. They suggest, that this taxonomy could be used, *inter alia*, to compare various BC design decisions or as a regulatory framework. The final taxonomy tree is shown in Figure 3.2.

- [26] devised a BC Applicability Framework (BAF) to determine if a BC is appropriate for an application, and if this is the case, whether the BC should be permissioned/private or permissionless/public, and which one of four selected consensus mechanisms would be suitable. By evaluating 92 controls (*i.e.,* questions) divided in 5 domains by their applicability, the algorithm of the framework recommends one of the 9 aforementioned outcomes based on a percentage distribution, *cf.* Figure 3.3. Controls regarding Transaction Constraints can be seen in Table 3.1.

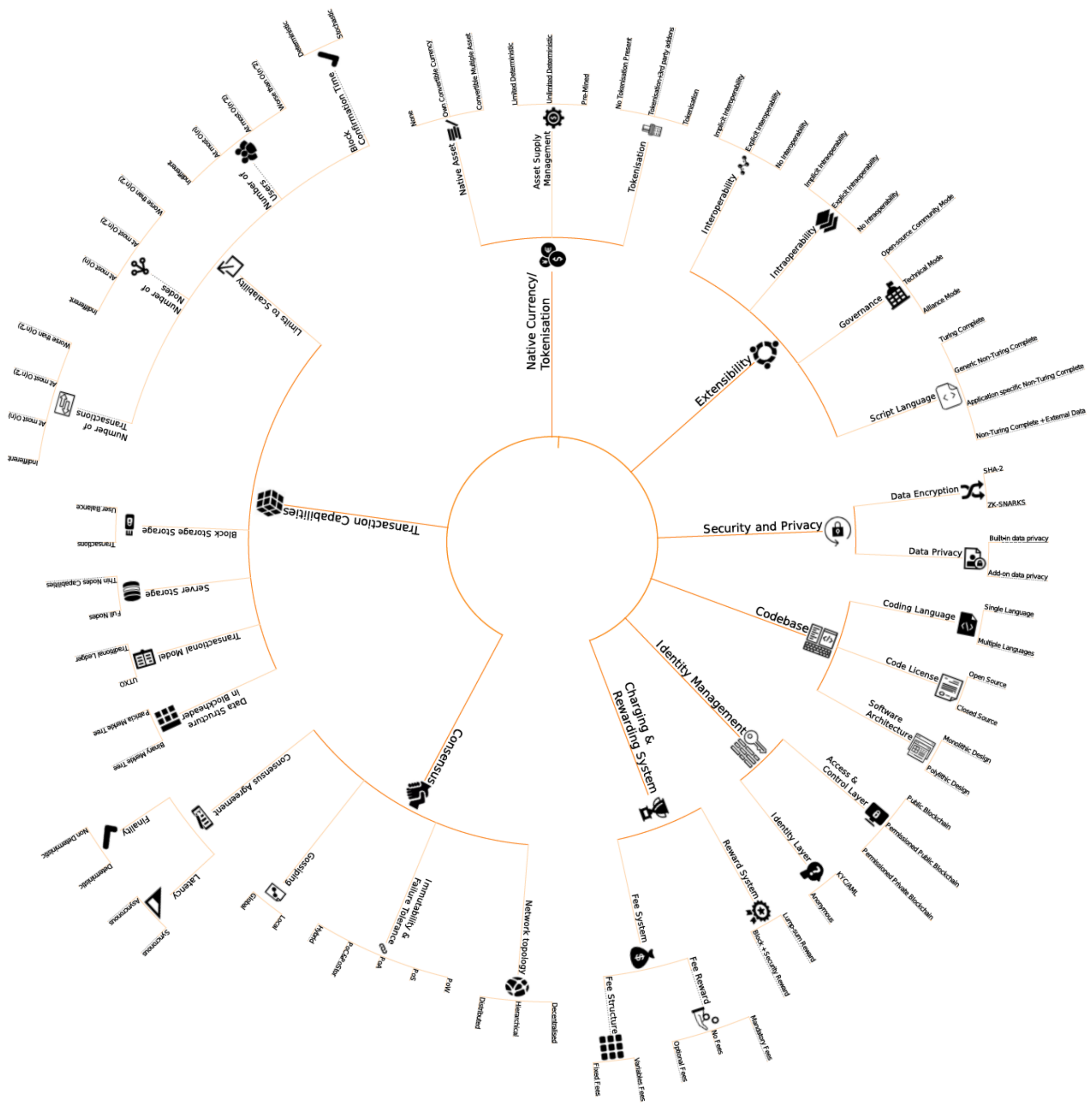


Figure 3.2: Blockchain Taxonomy Tree [67].

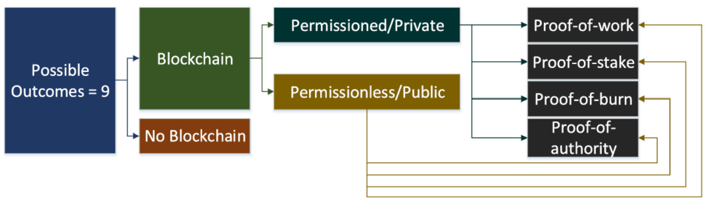


Figure 3.3: The 9 possible BAF outcomes [26].

Table 3.1: Controls regarding Transaction Constraints, F/L refer to the states Fully or Partially Applicable and P/N refer to Partially or Not Applicable [26]. The entries in the table are abbreviated in the following way: Blockchain (Y), No Blockchain (N), Permissioned/Private (V), Permissionless/Public (U), PoW (W), PoS (S), Proof-of-Burn *(PoB)* (B), and PoA (A) [26].

Controls	F/L	P/N
29: Are exchanges and transactions involved?	Y	N
30: Do the blockchain need to first provide the nodes with the rights to view the transactions?	V	U
31: Is there a requirement to get authorization to validate transactions in the blockchain? (see the Appendix)	VA	UWBS
32: Is there a need for the transactions to be validated by votes/consensus?	Y	N
33: Is the transactional fee required to carry out transactions very small (or null)? (see the Appendix)	V	U

3.3 Comparison and Discussion

Concerning the BC classification, [4] notes, that their work stands out from related works since their results and choice of attributes are evaluated and validated. Further, they observe that in [67], it is not clear how the components relate to each other, and that their amount of attributes is much higher than in other comparable related works.

Compared to the other related works, the BAF by [26] takes a different approach, although it takes some of the same attributes into account with the controls, the classification is not done on a BC level, but on an application-level to determine which type of BC would be appropriate for an use case.

Comparing the aspects, for instance, they all consider Fees, in [67] this can be seen in the sub component Fee System, in [4] there is the differentiation between Action Fees, fees that must be paid to perform an action and are unrelated to consensus mechanism, and Consensus Fees, in [26] there is the control: 'Is the transactional fee required to carry out transactions very small (or null)?'. However, the considered attributes are not in all cases the same, *e.g.,* [26] does not consider Turing Completeness, however, this choice is reasonable, since the recommendations are not as granular, and the consensus mechanism does not relate with the smart contract capabilities.

Consequently, the proposed SIG from this thesis differs from the aforementioned taxonomies, although graph-like structures were given in some of the related works, the use of the SIG notably allows to focus on interdependencies between the attributes and to quantify BC aspects. Further, a total of 74 nodes and 88 edges are present in the proposed SIG, and although Fees and Turing Completeness have been covered by the SIG design, the work by [4] went into more detail concerning the Token attributes. However, the SIG is able to show how certain aspects are achieved, *e.g.,* the proposed SIG covers various Performance and Security aspects that were not used in either [4] or [67].

Furthermore, unlike the design guidelines proposed by [4], design choices with the SIG are based on the quantified results for a specific use case, and the BC selection can be done based on the quantified values.

Chapter 4

Characterizing and Classifying Blockchains

This chapter describes the design and implementation of an approach based on SIGs to characterize and classify different BC implementations based on the functions they provide. Thus, Section 4.1 presents the design of the complete SIG, Section 4.1.1 goes over its design and the meaning of its values, next, in Section 4.1.2 the computation of the values are described and finally, in Section 4.2 an implementation is introduced, to be able to calculate and visualize BC SIGs.

4.1 Blockchain SIG Design

The full design of the proposed generic BC SIG is depicted in Figure 4.1, and the next section describes how the design and decomposition of such a SIG from the requirements was achieved, and lastly the quantification of its values is presented in Section 4.1.2.

To design the SIG, the aforementioned NFR Framework by [17] was utilized in conjunction with the quantification extension devised by [1] for the quantification of its values, *cf.* Section 4.1.2.

Consequently, to achieve this SIG, an iterative process was employed to refine from the initial SGs the proposed SIG according to Section 2.2.2. Due to the SIG topic of this thesis focusing on BCs *per se*, it meant that the decompositions were done by *type*, and therefore, the *topic*, *[Blockchain]*, was omitted from the SIG visualizations, since it belonged to all nodes of the SIG, hence, its omnipresence did not contribute any additional useful information to the SIG, and further, the chosen decomposition contributions, according to Table 2.1, are explained in detail in the next Section.

Due to the design and the nodes of the SIG, it can be used as a characterization of BCs, and due to its design it is able to showcase how certain BC attributes are achieved and how different BC aspects are interdependent to each other.

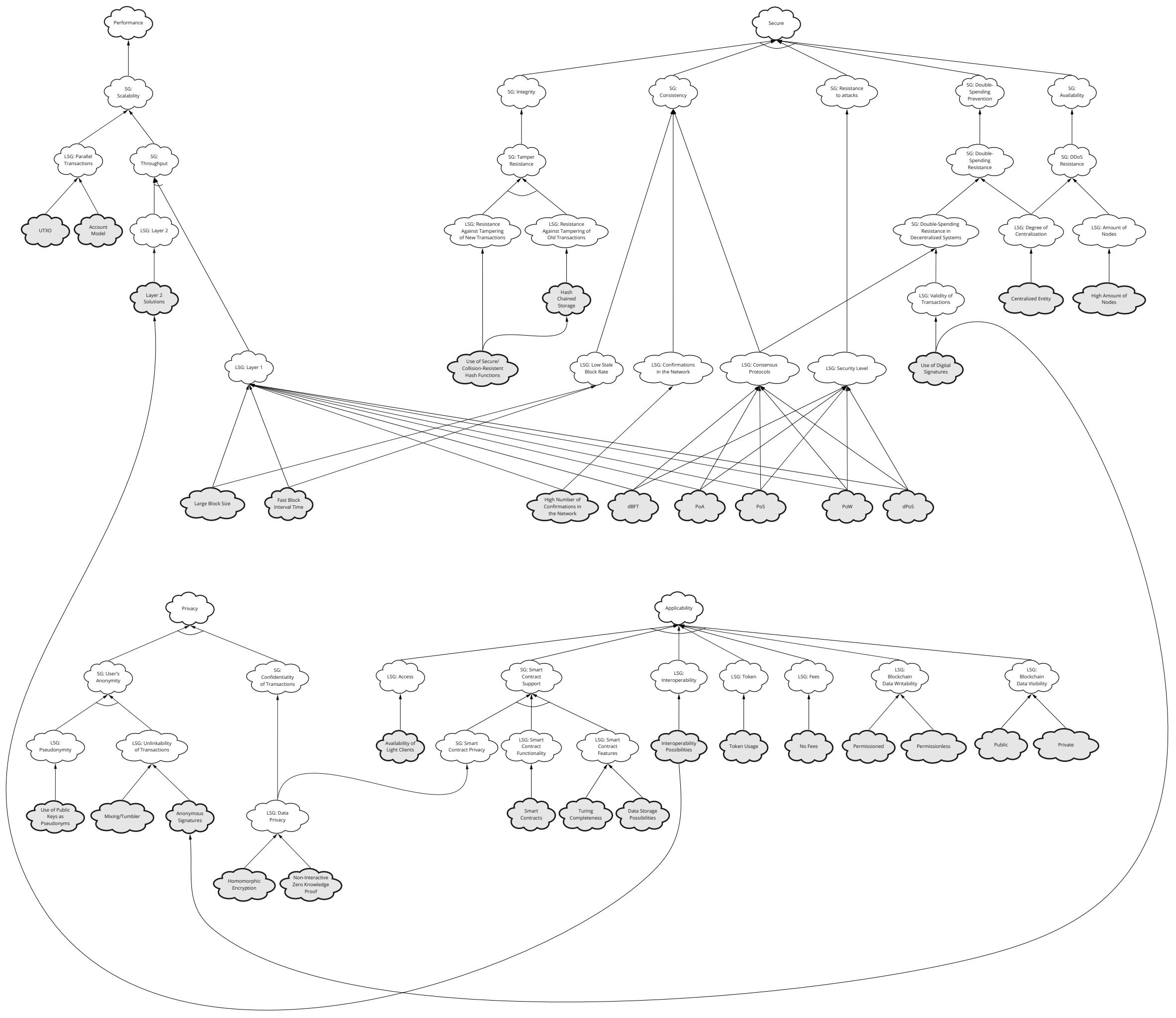

Performance
SG: Scalability
LSG: Parallel Transactions
SG: Throughput
UTXO
Account Model
LSG: Layer 2
Layer 2 Solutions
LSG: Layer 1
Large Block Size
Fast Block Interval Time
Secure
SG: Integrity
SG: Consistency
SG: Resistance to attacks
SG: Double-Spending Prevention
SG: Availability
SG: Tamper Resistance
LSG: Resistance Against Tampering of New Transactions
LSG: Resistance Against Tampering of Old Transactions
Hash Chained Storage
Use of Secure/ Collision-Resistent Hash Functions
LSG: Low Stale Block Rate
LSG: Confirmations in the Network
LSG: Consensus Protocols
LSG: Security Level
SG: Double-Spending Resistance
SG: DDoS Resistance
LSG: Double-Spending Resistance in Decentralized Systems
LSG: Degree of Centralization
LSG: Amount of Nodes
LSG: Validity of Transactions
Centralized Entity
High Amount of Nodes
Use of Digital Signatures
High Number of Confirmations in the Network
dBFT
PoA
PoS
PoW
dPoS
Privacy
SG: User's Anonymity
SG: Confidentiality of Transactions
LSG: Pseudonymity
LSG: Unlinkability of Transactions
Use of Public Keys as Pseudonyms
Mixing/Tumbler
Anonymous Signatures
LSG: Data Privacy
Homomorphic Encryption
Non-Interactive Zero Knowledge Proof
Applicability
LSG: Access
SG: Smart Contract Support
LSG: Interoperability
LSG: Token
LSG: Fees
LSG: Blockchain Data Writability
LSG: Blockchain Data Visibility
Availability of Light Clients
SG: Smart Contract Privacy
LSG: Smart Contract Functionality
LSG: Smart Contract Features
Interoperability Possibilities
Token Usage
No Fees
Permissioned
Permissionless
Public
Private
Smart Contracts
Turing Completeness
Data Storage Possibilities

Figure 4.1: Complete SIG. Source: The author.

4.1.1 Requirements Decomposition

The proposed SIG has been modeled with four initial NFRs, *Performance*, *Security*, *Privacy* and *Applicability* that form the SGs and core of the SIG. These NFRs have been chosen, since they are major talking points of BCs, for instance, Performance has been selected, since Bitcoin has been criticized in the past for scalability issues and having low tps, making its performance a considerable bottleneck to its usability compared to more traditional options [32]. The aspect of security and privacy are also considered, since they are both imperative to the context of transactions and payment solutions [79], thus, the survey by [79] was closely followed and applied to the SIG to capture these significant details. Furthermore, Applicability was added to recognize that various BCs have different features, that may either limit or expand the BC's functionality for a given use case, thus, this is also highlighting the fact, that not all BCs are suitable for all use cases, as will be further evaluated in Section 5.1. In total, the four aforementioned SGs lead to a decomposition into 32 OPs, and 42 LSGs and SGs, meaning that the proposed SIG spans 74 nodes and 88 contributions.

Performance

Firstly, the NFR Performance is looked at, and in Section 2.3.3 *Performance* was measured in throughput, defined as *tps*, although other metrics exist as well [55]. Therefore, in this SIG *Performance* directly relates to the NFR *Scalability*, this stems from the choice of viewing *Scalability* as a limitation of *tps* [28], and using *tps* as a performance metric.

SG: **Scalability** is described by [28] as a serious limitation of BC based systems, and is therefore subject to ongoing research. Subsequently, a wider adoption of BC systems, *e.g.,* IoT use cases, is hindered by the current Scalability and Performance of BC systems [47]. Moreover, due to the importance of Scalability, novel BC are being designed to tackle this very problem [47], *e.g.,* the IOTA Tangle [48].

LSG: **Parallel Transactions**, it can be said that a BC utilising the *UTXO* accounting standard in contrast to an *Account Model*, enables *Parallel Transactions* to a degree [60, 65]. This in turn gives the advantage of reducing computational load [60].

OP.: **UTXO** refers to 'Unspent Transaction Output', and new transactions use the output from previous transactions [60]. This OP. enables *Parallel Transactions*, since Transactions can be processed in parallel [60]. However, their parallel functionality is detrimental to stateful Smart Contracts [60], though this connection has not been made in the SIG, since Extended UTXO (EUTXO) models exist [14], which for example are used by Cardano, enabling the writing of Smart Contracts, though in a more functional approach, *cf.* [9]. The EUTXO models are commonly referred to as hybrid models, enjoying the advantages of parallel processing without losing the functionality of smart contracts [60]. For reasons of complexity, this distinction has not been considered here.

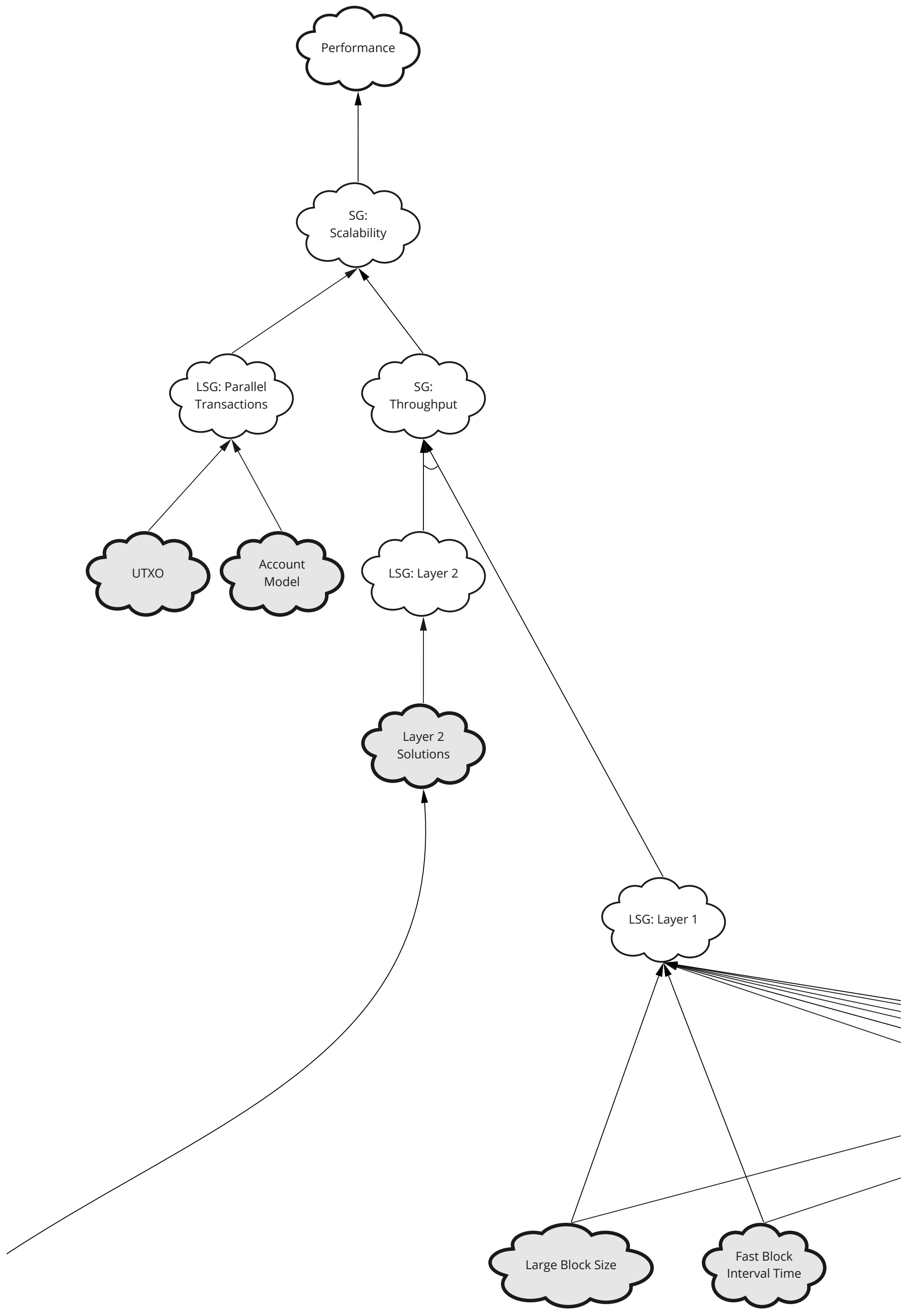


Figure 4.2: SG Performance. Source: The author.

OP.: **Account Model**, in comparison to *UTXO*, this model works similarly to debit card accounts, and BC addresses hold an account with the balance [60]. The connection to Smart Contracts, as described by [60], was also not considered here, to avoid giving an advantage of Account Models over UTXO models towards Smart Contract Functionality.

SG: **Throughput**, this is also a feature that various greatly within the BC world, thus, novel BC's and so-called *Layer 2 Solutions* are emerging to specifically tackle the problem of *Scalability*, and therefore, *Performance* in the bigger picture [28]. Additional to the *Layer 2*, throughput is typically also split into *Layer 0* and *Layer 1*, where each layer has different approaches to increase *tps* [28]. Despite that, the *Layer 0* approaches, which concern the network infrastructure [28], were considered out of scope for this thesis.

- **LSG:** **Layer 2** revolves around off-chain solutions to solve the problem of *Scalability* [28]. Unsurprisingly, since this is an important topic, there are various *Layer 2* approaches in development. For Ethereum the reader can refer to [22], and for Bitcoin, [35].
  - **OP.:** **Layer 2 Solutions** relates to the BC's availability of such aforementioned Layer 2 solutions.
- **LSG:** **Layer 1** refers to the points related to the consensus protocols and general BC data structures [28]. Additionally to the consensus protocols and High number of confirmations in the network, which will be be considered under the NFR Secure, the aspects of Block Size and Block Interval Time were added to the SIG, which were discussed in Section 2.3.3.
  - **OP.:** **Large Block Size**, Block Size has been defined in Section 2.3.3 as the size of a block in the BC, thus, given the formulae in Equations 2.6 and 2.7, it automatically follows, ceteris paribus, that a *larger* Block Size leads to an increase in tps. However, they introduce security risks, since larger blocks reduce the propagation speed in the network, leading to a higher *Stale Block Rate* [23].
  - **OP.:** **Fast Block Interval Time**, Block Interval Time has been defined in Equation 2.3.3 as the time it takes to add a new block to the BC. Here it also follows, with the same reasoning as for *Large Block Size*, that, ceteris paribus, given the formula for *tps* in Equation 2.7, a faster Block Interval Time leads to an increase in tps. Nevertheless, a faster Block Interval Time increases the probability of stale blocks, naturally, since for PoW consensus mechanisms this can be pictured by lowering the overall mining difficulty [23].

Security

Security of BC systems stands at the center of debate regarding its applications [79], especially FinTech applications require high security that needs to be fulfilled to become viable [73], so to highlight its importance, the NFR **Secure** was added to the SIG. Since Privacy and Security are similar in nature. [26, 79] handled Privacy together with Security, however, they have been split for the SIG to allow for a better quantification of the individual SG values and to provide a better overview. Moreover, it has to be mentioned, that [23] also showed that increasing the block reward also leads to an increase in security, though for complexity reasons this was not added to the SIG. The SGs *Consistency, Integrity*, *Availability* and *Double-Spending Prevention* were identified by [79] as Secu-

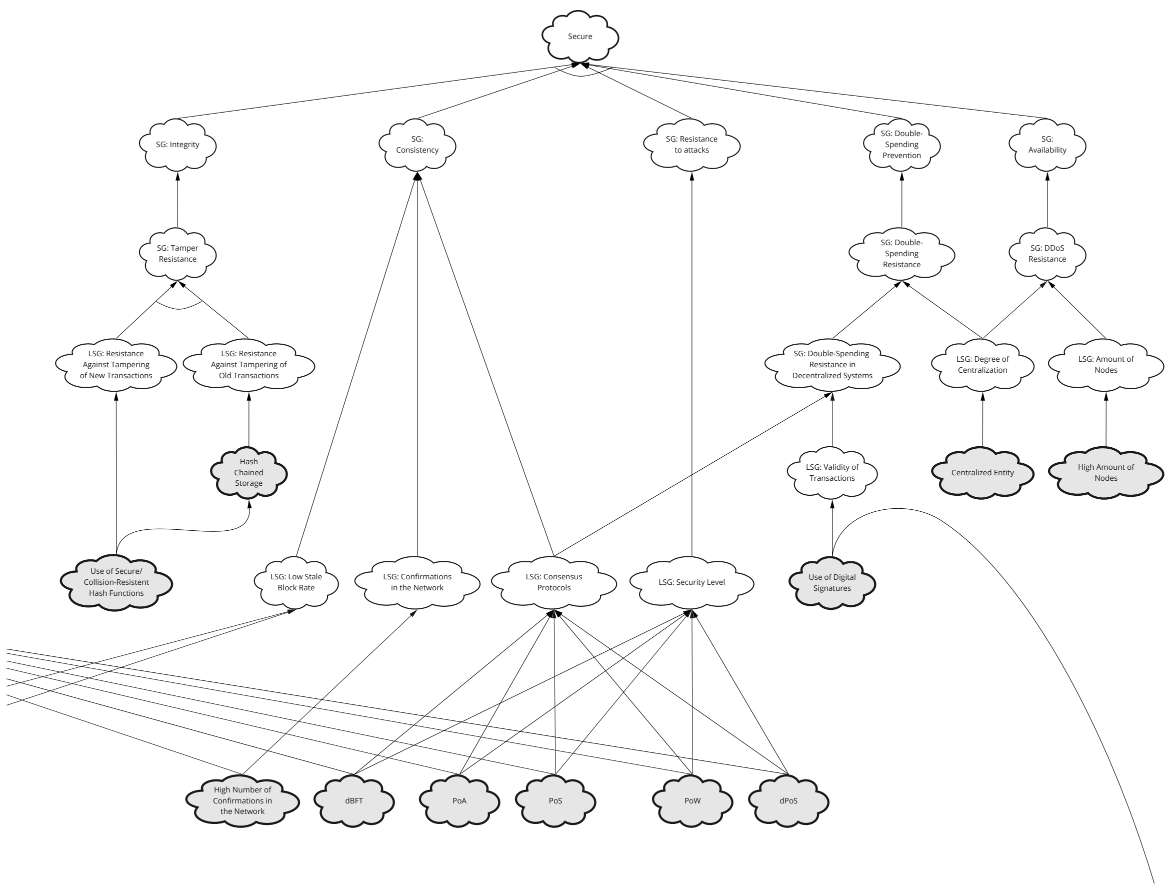

Secure
SG: Integrity
SG: Consistency
SG: Resistance to attacks
SG: Double-Spending Prevention
SG: Availability
SG: Tamper Resistance
SG: Double-Spending Resistance
SG: DDoS Resistance
LSG: Resistance Against Tampering of New Transactions
LSG: Resistance Against Tampering of Old Transactions
SG: Double-Spending Resistance in Decentralized Systems
LSG: Degree of Centralization
LSG: Amount of Nodes
Hash Chained Storage
LSG: Validity of Transactions
Centralized Entity
High Amount of Nodes
Use of Secure/ Collision-Resistent Hash Functions
LSG: Low Stale Block Rate
LSG: Confirmations in the Network
LSG: Consensus Protocols
LSG: Security Level
Use of Digital Signatures
High Number of Confirmations in the Network
dBFT
PoA
PoS
PoW
dPoS

Figure 4.3: SG Secure. Source: The author.

rity and Privacy Requirements, and subsequently incorporated into the SIG as Security decompositions.

SG: **Consistency** was outlined by [79] as a basic Security property, denoting that all the nodes hold the same BC at the same time. Furthermore, [79] points out, that it is up for debate whether BC systems fulfill *eventual consistency* or *strong consistency*. In Summary, *eventual consistency* means in the distributed computing context, that the data will be eventually consistent, while *strong consistency* means that the same ledger is shared by all nodes, and the amount of confirmations in the BC network could be seen as a *parameterized strong consistency* [79], however, this distinction was not incorporated into the SIG.

LSG: **Consensus Protocols** have been identified by [79] as the appropriate technique to ensure the Security and Privacy requirement of Consistency. Regarding the consensus protocols, that were looked at in Section 2.3.1, five are considered here.

OP.: **PoW**, to ensure security, the mechanics behind the PoW consensus mechanism limit the throughput greatly [70, 20]. For instance, Bitcoin is only able to achieve 7 tps [70]. Regarding *Security Level*, *PoW* based consensus mechanism are susceptible to *Majority Attacks* [51], more details can be found under the LSG *Degree of Centralization*.

- **OP.:** **PoS**, regarding the *Security Level*, it is susceptible to *nothing-at-stake* and *grinding-attack* [73]. Additionally, [25] notes that *PoS* does not work well with *permissioned* BC, since miners with the most stake could add fradulent transactions and 'rich' nodes tend go get richer [25].
- **OP.:** **dBFT**, referring to delegated Byzantine Fault Tolerance, offers a very high throughput since they are able to handle up to thousand of tps [25]. However, regarding the LSG *Security Level*, the missing 'Commit' Phase has security implications and additionally, BFT protocols only support up to $\frac{n}{3}$ byzantine nodes [72].
- **OP.:** **PoA** is a hybrid consensus protocol, combining the Byzantine Fault Tolerance (BFT) property with *PoS* [25]. Therefore, it also offers a very high performance, though hardware limitations are at play [25]. Regarding *Security Level*, it requires $\frac{N}{2} + 1$ nodes to be honest, though the real identities behind the nodes can also be seen as a security mechanism [25]. More details can be found in Section 2.3.1 and [25].
- **OP.:** **dPoS**, is referring to delegated Proof of Stake. From Section 2.3.1 and [79], it offers higher scalability than *PoS*, though, similar security levels.

LSG: **Low Stale Block Rate**; due to conflicts or concurrency, not all blocks get appended to the longest chain, they are denoted as *stale blocks* [23]. Since they lead to chain forks, which is an *inconsistent state* of the BC, they offer considerable advantages to attackers and simultaneously increase the bandwidth in the network, thus, a lower Stale Block Rate increases Security [23].

LSG: **Confirmations in the Network**, refers to the additional blocks that have been added on top of the BC [28]. Therefore, transactions are only accepted after a specific number of confirmations have passed [23].

- **OP.:** **High Number of Confirmations in the Network**, [23] showed that to offset security issues introduced by a stale blocks, a higher number of confirmations is needed to ensure Security. However, it is also another restriction on tps, since to confirm a single transaction, additional blocks are needed [28].

SG: **Integrity** is another Security and Privacy Requirement by [79] relating to the *Integrity of Transactions*. This is also an important point, since different intermediaries might access the transaction data and a risk of falsification or forging exists, hence, a *Tamper Resistance* guarantee is needed [79].

SG: **Tamper Resistance** in the context of BC means that the BC data cannot be tampered with, either during the block creation process, or after it has already been written to the BC [79].

LSG: **Resistance Against Tampering of New Transactions**, this LSG contains the OP. *Use of secure/collision resistent Hash Functions*, though the LSG *Validity of Transactions* with the OP. *Use of Digital Signatures* is related, since Digital Signatures are also used against tampering [79]. Since for both, the process is the same [79], it was not added here a second time.

- **OP.:** **Use of secure/collision resistent Hash Functions**, to solve the issue of tampering of new transactions, secure hash functions are used, such as SHA-256 [79].

LSG: Resistance Against Tampering of Old Transactions, this relates to the tampering of historical data that has already been written to the BC [79]. This resistance is ensured by using *Hash Chained Storage* [79].

OP.: Hash Chained Storage make use of *Hash Pointers*, which use *hash functions* on the data and pointers to show its location; and *Merkle Trees*, a binary search tree data structure [79]. *Merkle Trees* use *Hash Pointers* to link nodes together, and parent nodes contain the hash values of their children [79]. With these two techniques, tampering can be prevented, since tampering causes parent nodes to not match anymore, and an adversary would need to change all nodes in the path up to the top [79]. Compared to simply using *Hash Pointers*, due to the binary search tree data structure of *Merkle Trees*, one can verify its integrity in logarithmic time [79].

SG: Availability relates to *Availability of System and Data* by [79]. It means, that for online systems, that they should be accessible to their users at any time, even in case of attacks [79].

SG: DDoS Resistance, was chosen as a suitable *Availability* decomposition, since it relates to the network's resistance against *Distributed Denial-of-Service*, abbreviated as DDoS, attacks [79]. Accordingly, DDoS attacks flood the network with traffic to make the server unavailable to its intended use, consequently, this kind of attack is typically very difficult to deal with [79]. However, for decentralized systems, like BCs, this is not the case, since attackers would need to attack a large portion of the network simultaneously, even if some nodes become unavailable, the network is able to function properly [79]. Nonetheless, [51] notes, that a DDoS attack can also manifest itself by a Majority Attack, also leading to a denial of the service, though this case was also covered in the SIG under *Security Level*.

LSG: Amount of Nodes indicates the amount of nodes in the network.

OP.: High Amount of Nodes, this stems from the fact, that the larger the network is, the harder it is to simultaneously attack all nodes of the network with a DDoS attack, and consequently, a large network provides better DDoS resistance [79].

LSG: Degree of Centralization specifies to which degree the BC network is centralized. Having a centralized network directly imposes huge security implications, since mining pools or centralized entities controlling more than 50% of the resources can take over the network [34]. Additionally, following the deduction for the DDoS resistance for *High Amount of Nodes*, it becomes clear, as outlined by [79], that this is only the case for *fully decentralized systems*, resulting in centralized systems and services having a single point of failure.

OP.: Centralized Entity refers to the question whether centralized entities are prevalent in the network. [34] showed that in 2019, in Ethereum the largest mining pool mined 28.2% of all blocks, and in Bitcoin 18.2%. In 2014, *GHash.IO*, a Bitcoin mining pool went even above 51% of the total hash rate [51, 41]. [34] points out, that these numbers are concerning. However, for Consortium or Private Blockchains, where the participants of the network are known and pre-selected [79], naturally, this does not impose a security risk, since the network was designed that way.

SG: Double-Spending Prevention, has been added to the SIG, since for digital currencies double-spending is a challenge which needs to be addressed, because digital information can be easily reproduced [79]. Double-spending essentially means, that a user is able to spend a coin more than once [79].

- **SG: Double-Spending Resistance** has been split into *Double-Spending Resistance in Decentralized Systems* and *Degree of Centralization. Degree of Centralization* was chosen, since in a fully centralized system, the centralized Trusted Third Party is responsible for the verification of the transactions, and double-spending can be prevented in that way [79].
  - **SG: Double-Spending Resistance in Decentralized Systems**, in *Decentralized Systems*, the consensus algorithms are used to prevent Double-Spending [79]. Therefore, the LSG *Consensus Protocols*, discussed at the SG *Consistency*, is connected here.
    - **LSG: Validity of Transactions** denotes that the transaction's validity must be verifiable, to prevent Double-Spending [79].
    - **OP.: Use of Digital Signatures**, by digitally signing a transaction, its authenticity and validity of the data is verifiable [79]. An example for such an algorithm is the *Elliptic Curve Digital Signature Algorithm (ECDSA)*, that is used by Bitcoin [79].

SG: Resistance to Attacks, for BCs a multitude of attacks exist, some on the BC application level, some on the Peer-to-Peer System, and more details were collected and can be found under [72]. For reasons of complexity, only consensus specific attacks were considered here and aggregated into an overall LSG *Security Level.*

- **LSG: Security Level**, this LSG was added to highlight the fact, that the multiple consensus mechanisms offer different attack surfaces, according to the explanation given above, and the details can be found under the specific Operationalizations.

Privacy

Privacy has been outlined as a relevant property for DL by [76] and subsequently been chosen as a NFR, since due to the importance of Privacy, it has been in the focus of research, and it can also been seen as an enabler for novel BC applications [79]. There have also been BCs designed to specifically deal with Privacy [55]. *Privacy* was decomposed into *User's Anonymity* and *Confidentiality of Transactions*, which are Security and Privacy Requirements described by [79], and an 'AND' contribution has been chosen here, since these are all important aspects.

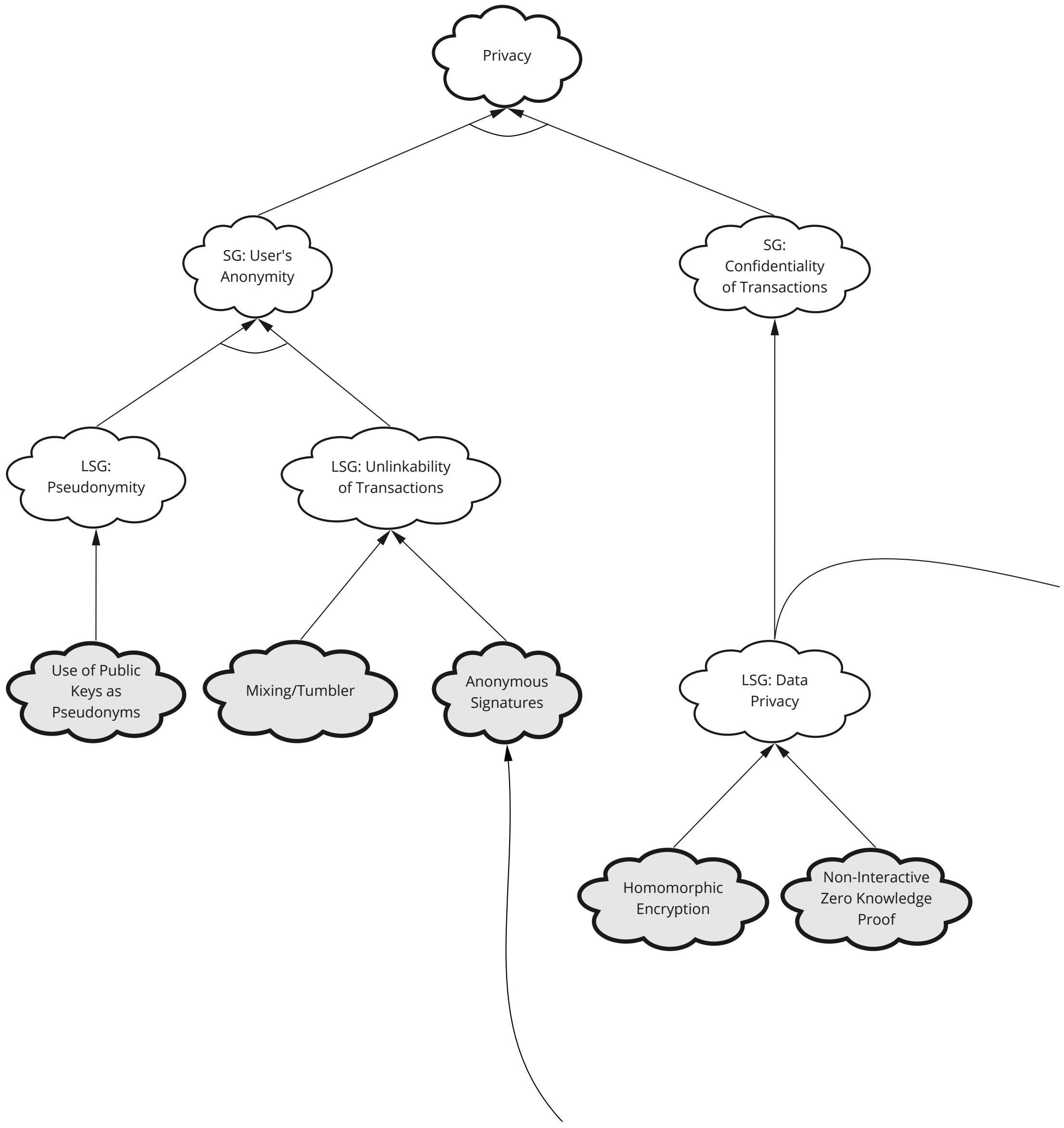

Privacy
SG: User's Anonymity
SG: Confidentiality of Transactions
LSG: Pseudonymity
LSG: Unlinkability of Transactions
Use of Public Keys as Pseudonyms
Mixing/Tumbler
Anonymous Signatures
LSG: Data Privacy
Homomorphic Encryption
Non-Interactive Zero Knowledge Proof

Figure 4.4: SG Privacy. Source: The author.

SG: User's Anonymity has been described by [79] as the notion that the user's anonymity is ensured to prevent accidental leakage of user's data to potential intermediaries, or further, parties might not even want to engage in a transaction if their real identities are known to the other party. To ensure Anonymity, both *Pseudonymity* and *Unlinkability of Transactions* need to be fulfilled [79], thus, leading to an 'AND' Contribution.

LSG: Pseudonymity is the appropriate Security and Privacy Property by [79] for the Requirement *User's Anonymity*, and has been added as a LSG here.

OP.: Use of Public Keys as Pseudonyms was added as an OP to *Pseudonymity*. Since BC users generate their key pairs, consisting of the public and private key, themselves, and the hashes of public keys are used as addresses to interact in the network, the public keys act de facto as pseudonyms, and no personal identity information is shared [79].

LSG: Unlinkability of Transactions was noted as a requirement for online transactions by [79]. This requirement implies that it is not possible to link multiple transactions of a user together, denying any possible deductions about the user, such as his account balance or transaction behavior [79]. It is also noteworthy, that through statistical analysis, this data could even be used to infer the real identities of users [79], and Unlinkability of Transactions is therefore an important LSG for Privacy. Additionally, if addresses are reused for transactions, it is trivial to link them together, greatly decreasing the Privacy requirement, however, since this can be avoided by not reusing the addresses [2], this has not been added as an OP here.

OP.: Mixing/Tumblers were identified as an OP, since the Unlinkability of Transactions is not in every case given, especially when addresses are reused [79]. The idea behind Mixing or Tumblers is to obfuscate the coin ownership by randomly exchanging coins with other users, and as such, outsiders cannot link transactions together anymore [79].

OP.: Anonymous Signatures refer to Digital Signature variants that provide anonymity to the signer, these variants include *Group Signatures* or *Ring Signatures* [79]. These techniques obfuscate the sender of a transaction, since not only the sender, but any node in a group or a ring can sign the transaction [79]. For example, using *Ring Signatures* the probability of guessing the real sender of a transaction is $1/n$, given n participating nodes in a ring [79].

SG: Confidentiality of Transactions as a security and privacy requirement traditionally means that transactions only disclose as little information as possible, and that user's information cannot be accessed by others [79]. However, for public BCs, this is not something that is easily achievable, since the Smart Contract Data and transaction content are public [79].

LSG: Data Privacy This LSG was chosen to signify the importance of data privacy, and is linked to *Confidentiality of Transactions*. Due to existing inference attacks, *Pseudonymity* alone is not enough [79]. Therefore, *Homomorphic Encryption* and *Non-Interactive Zero Knowledge Proof* were selected as possible solutions and OPs, though other techniques have also been outlined by [79]. Data Privacy is also linked to the SG *Smart Contract Privacy cf.* **Applicability**.

OP.: Homomorphic Encryption denotes a cryptographic technique where computations can be done directly on the encrypted data [79]. This can be relevant where not only the privacy of the user, but also the privacy of the data is important [77]. There are Partial and Fully Homomorphic Encryptions techniques available [79, 77]. By storing encrypted data on the BC, one would thus be able to still do computations on the data without leaking any information about the data, in particular, this can be applied to data stored in Smart Contracts [79].

OP.: **Non-Interactive Zero Knowledge Proof (NIZK)** is another noteworthy cryptographic technique, that allows to verify an assertion without leaking any information about it, since the information that the user gives to a verifier does not disclose anything else [79]. This technique can be used to prove that an user's balance is sufficient for a transaction, without the need to reveal the balance publicly [79].

Applicability

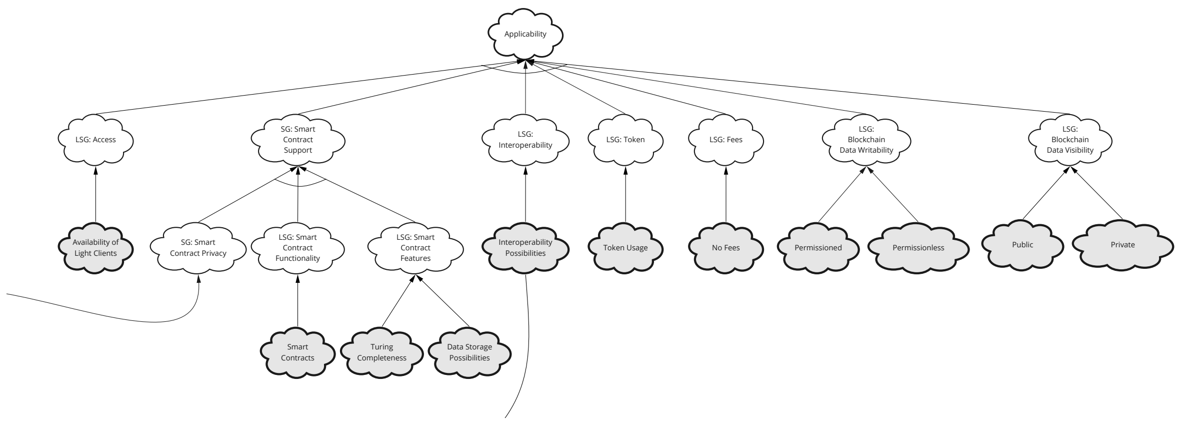

Applicability
LSG: Access
SG: Smart Contract Support
LSG: Interoperability
LSG: Token
LSG: Fees
LSG: Blockchain Data Writability
LSG: Blockchain Data Visibility
Availability of Light Clients
SG: Smart Contract Privacy
LSG: Smart Contract Functionality
LSG: Smart Contract Features
Interoperability Possibilities
Token Usage
No Fees
Permissioned
Permissionless
Public
Private
Smart Contracts
Turing Completeness
Data Storage Possibilities

Figure 4.5: SG Applicability. Source: The author.

The SG **Applicability** relates to the selected requirement, that a BC should be applicable to various use cases, and various of the selected attributes help extend the BC's usability. Thus, *Access*, *Interoperability*, *Token*, *Fees*, *Blockchain Data Writability* and *Blockchain Data Visibility* have been added as LSGs with their OPs each, and *Smart Contract Support* as a SG that has been further decomposed.

The need for the LSGs *Access*, *Interoperability*, *Token* and *Fees* arose, since the quantification of values by [1] does not consider the case where OPs and LSGs or SGs are simultaneously children to a parent SG. Thus, these LSGs only contain a single OP. In the following items the SIG's structure and decomposition regarding *Applicability* is explained.

LSG: **Access** stems from the need of facilitating communication between devices, especially IoT devices, that have several limitations [50]. This is an important point, and as such this use case will be evaluated in Section 5.1. An example of BC that was specifically designed to be applicable in the IoT context is IOTA [50].

OP.: **Availability of Light Clients**; the Light Clients allow interaction with the BC by connecting the client to full nodes, which verify the blocks and require all transactions [68]. In contrast to the full nodes the Light Clients are very lightweight and can be used to allow IoT communication with the BC to satisfy the need of Access [50].

SG: **Smart Contract Support**; various BCs feature the functionality of Smart Contracts, but they are not in every case equal and can differ greatly, as seen in Section 2.3.3. The Smart Contract functionality has been decomposed into three aspects by an 'AND' contribution:

- **SG:** **Smart Contract Privacy**, has been linked to **Data Privacy**, as the OP. **Homomorphic Encryption** and **Non-Interactive Zero Knowledge Proof** can be used in conjunction to Smart Contracts to ensure additional Privacy on the BC, and Privacy based applications [79].

- **LSG:** **Smart Contract Functionality** has been separated from the other points and only relates to the question if Smart Contracts *per se* are supported by a particular BC, and does not consider their functionality and features.

  - **OP.:** **Smart Contracts**, if the functionality of Smart Contracts is supported and offered by the BC.

- **LSG:** **Smart Contract Features** is about the available features of Smart Contracts, namely to which degree are Smart Contracts supported by a particular BC?

  - **OP.:** **Turing Completeness**, this OP denotes whether the Smart Contract Scripting Language can execute Turing machines [4]. This allows for complex logic flows and also loops in Smart Contracts [2], and Turing Incomplete Script Languages are thus less complex, though, also less vulnerable to Smart Contract attacks in a network, since e.g. an infinite loop that would be executed by all nodes would not be possible and can therefore not be exploited [2].
  - **OP.:** **Data Storage Possibilities**, if data that is unrelated to transactions can also be stored on a BC [4]. As an example, this would allow for applications where IoT devices function on top of the underlying BC network [31]. Since most *On-chain* storage is extremely expensive, *Off-chain* data storage, like the InterPlanetary Filesystem (IPFS) can be used by storing only the identifying hash of a file on the BC [63]. Furthermore, to enhance the Applicability of the SIG and to capture a broader spectrum of BC characteristics, this OP also includes data storage that is unrelated to Smart Contracts, *e.g.,* Ethereum's 46 kByte Maximum String Size from Transactions [54] would likewise allow for storage applicabilities.

LSG: **Interoperability** further enables the possibilities for applications, that can also make use of the synergies of different BC [6]. This interoperability could in some cases also be leveraged to further increase scalability [6], which has also been considered here in the SG **Performance**, at **Layer 2 Solutions**.

- **OP.:** **Interoperability Possibilities**, in the context of BC, is about whether different BC can interoperate with each other to make use of the aforementioned advantages.

LSG: **Token**, can belong to either of Utility Tokens, Asset Tokens or Payment Tokens [4].

OP.: **Token Usage** denotes whether the BC employs one of the aforementioned types of Tokens. Although Tokens can enable Applicability functionalities, some BC like R3's Corda or Fabric do not use tokens [45].

LSG: **Fees**; in [4] the distinction has been made, whether Validators are paid a fee and if participants in the network need to pay a fee for actions, since both cases are not dependent on each other, though for simplicity both cases have been merged together here. Paying the validators fees for their work incentives their work [2]. However, within a private BC deployment (*e.g.,* a BC for IoT), fees are not required as all the participants are known and the validators pre-selected. Thus, there is no need for incentives in such deployment.

OP.: **No Fees** denotes that no kind of Fees exist in the BC.

LSG: **Blockchain Data Writability**, as in Section 2.3.2 and [55] BC can be decomposed into four categories, based on *Data Writability* and *Data Visibility*. This LSG is therefore about who can write on the BC. Since the options are mutually exclusive, the contribution `HELP` has been chosen here.

OP.: **Permissioned** in comparison to *Permissionless*, this attribute restricts the write functionality to only some select nodes [55].

OP.: **Permissionless**, represents the functionality that enables all nodes to write on the blockchain [55].

LSG: **Blockchain Data Visibility**, referring to the accessibility of the data, *cf.* Section 2.3.2 and [55].

OP.: **Private**, only some selected nodes are allowed to view the BC data [55].

OP.: **Public**, the BC is public and its data can be read by all peers [55].

4.1.2 Quantification of Values

In this section the proposed generic SIG from Section 4.1.1 is extended with predefined values, and [1]'s quantification of values is used to compute the OP, LSG and SG scores of the generic SIG.

Although the choice of predefined values was arbitrary, they were discussed during meetings with the supervisors, and selected choices are shortly justified and explained here. Additionally, for reasons of complexity, only steps of 0.25 were considered. To reiterate, according to the quantification framework of [1], a LSG weight of 1.0 denotes a critical SG, while a weight of 0 denotes a SG without influence on decisions, and a an OP impact of 0 shows, that an OP has no association with a LSG.

For the quantification of values, the Tables 4.1, 4.4, 4.7, 4.10 denote the impact between the OPs to either a LSG or to a parent OP. In Tables 4.3, 4.6, 4.9, 4.12 the contributions are shown from a node towards the parent. The chosen LSG weights are shown in Tables 4.2, 4.5, 4.8, 4.11.

Performance

The quantification for Performance are denoted in Tables 4.1, 4.2 and 4.3.

Table 4.1: Performance: Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
UTXO	Parallel Transactions	1.0
Account Model	Parallel Transactions	0.0
Layer 2 Solutions	Layer 2	1.0
Interoperability Possibilities	Layer 2 Solutions	1.0
Large Block Size	Layer 1	0.5
Fast Block Interval Time	Layer 1	0.5
High Number of Confirmations in the Network	Layer 1	-0.5
dBFT	Layer 1	0.75
PoA	Layer 1	0.25
PoW	Layer 1	-0.75
PoS	Layer 1	0.25
dPoS	Layer 1	0.5

Table 4.2: Performance: LSG Weights

LSG	**Weight**
Parallel Transactions	0.5
Layer 2	1.0
Layer 1	1.0

Table 4.3: Performance: Contributions

Node	**Parent**	**Type**	**Contribution**
Scalability	Performance	MAKE	1
Parallel Transactions	Scalability	AND	0.25
Throughput	Scalability	AND	0.75
Layer 2	Throughput	AND	0.5
Layer 1	Throughput	AND	0.5

OP.: **UTXO**, in Section 4.1.1, it was mentioned that UTXO enables Parallel Transactions, so this value was chosen as 1.

OP.: **Account Model**, since in Section 4.1.1, it was mentioned that Account Model does not allow for parallel computations, the impact to Parallel Transactions was chosen as 0.

LSG: **Layer 1** and **Layer 2**, the AND contribution has been chosen, since both layers have been deemed as equally important, furthermore, they are both in the focus of research *cf.* Section4.1.1.

OP.: **Large Block Size** and **Fast Block Interval Time**, both have been given an equal impact of 0.5, since from the Equations 2.6 and 2.7, higher tps can be achieved through either one of those.

OP.: **High Number of Confirmations in the Network**, from Section 4.1.1, more confirmations lead to less tps, thus, it can be said that a high number of confirmations affects Layer 1 negatively.

LSG/SG: **Throughput** has been given more weight than **Parallel Transactions**, since in Section 4.1.1 it has been noted, that Throughput directly relates to Scalability.

OP.: The consensus protocol **dBFT** was given an impact towards Layer 1 of 0.75, since it is able to handle up to thousand of tps, *cf.* Section 4.1.1. Since Visa is on average at 1700tps [32], this value was taken as the maximum and a linear interpolation [74] between the minimum impact value of -1 and the maximum of 1 was done. However, since the dBFT's and Visa's tps value are rather outliers, the linear interpolation approach lead to a poor choice of values, since PoW, PoS and PoA would not be distinguishable anymore due to the large difference in scale. Thus, the choice was done arbitrarily, and an impact of value of 0.75 was chosen for dBFT. **PoA** was given a score of 0.25, since in Parity, it only achieves up to 80 tps [25]. Equally, **PoS** was also given a score of 0.25, since similar tps can be achieved [12]. **PoW** was given an impact rating of -0.75, due to the inner workings of the mechanism mainly targeting security over performance, *cf.* Section 4.1.1. In comparison to PoS, **dPoS** was given an impact of 0.5, since a higher scalability can be achieved *cf.* Section 2.3.1.

Secure

The chosen values for the SG Secure can be found in Tables 4.4, 4.5 and 4.6.

LSG: both **Low Stale Block Rate** and **Confirmations in the Network** were given a score of 0.5, since as per [23], the amount of confirmations in the network can offset security issues, and a Low Stale Block Rate is desirable as well. Regarding **Consensus Protocols**, they were given a value of 1.0, since from [79], they are the main technique to achieve Consistency in the network.

OP.: Regarding **Security Level**, **PoW** was given a score of 1.0, due to the work by [51, 25, 73, 79] and also *cf.* 4.1.1, it was deemed as very secure, since it boasts a high security by being mostly only susceptible to Majority attacks. **PoS** was given a score of 0.25, since by not being applicable to permissioned BCs and being susceptible to various attacks *cf.* Section 4.1.1, its security was deemed lower. Equally, **dBFT** was only given an impact of 0.25 due to the missing 'Commit' Phase and by having a weakness towards majority attacks, *cf.* Section 4.1.1. In comparison, **PoA** was given an impact score of 0.75, since its honest nodes requirement is higher than *e.g., dBFT* and by staking the real identity, the repercussions can be high if a node acts dishonest, acting as an incentive to act honest and equally as a security mechanism, *cf.* Section 4.1.1. Similarly to PoS, **dPoS** was given the same impact score, since it hosts a similar tolerated adversary power and further, dishonest nodes can be voted out [80].

LSG: **Resistance Against Tampering of New Transactions** and **Resistance Against Tampering of Old Transactions** were both given a LSG weight of 1.0 and an 'AND'

Table 4.4: Secure: Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
Use of Secure/ Collision Resistent Hash Functions	Resistance Against Tampering of New Transactions	0.75
Use of Secure/ Collision Resistent Hash Functions	Hash Chained Storage	1.0
Hash Chained Storage	Resistance Against Tampering of Old Transactions	0.75
Large Block Size	Low Stale Block Rate	-0.5
Fast Block Interval Time	Low Stale Block Rate	-0.5
High Number of Confirmations in the Network	Confirmations in the Network	1.0
dBFT	Consensus Protocols	1.0
PoA	Consensus Protocols	1.0
PoS	Consensus Protocols	1.0
dPoS	Consensus Protocols	1.0
PoW	Consensus Protocols	1.0
dBFT	Security Level	0.25
PoA	Security Level	0.75
PoS	Security Level	0.25
dPoS	Security Level	0.25
PoW	Security Level	1.0
Use of Digital Signatures	Validity of Transactions	1.0
Centralized Entity	Degree of Centralization	1.0
High Amount of Nodes	Amount of Nodes	1.0

Table 4.5: Secure: LSG Weights

LSG	**Weight**
Resistance Against Tampering of New Transactions	1.0
Resistance Against Tampering of Old Transactions	1.0
Low Stale Block Rate	0.5
Confirmations in the Network	0.5
Consensus Protocols	1.0
Security Level	1.0
Validity of Transactions	1.0
Degree of Centralization	0.75
Amount of Nodes	0.5

contribution, since Tamper Resistance is an important feature of BC [79], and both have been deemed as equally important and also critical SGs.

OP.: **dBFT**, **PoA**, **PoS**, **dPoS**, **PoW** have all been given an OP. impact of 1.0, since they all satisfice the LSG Consensus Protocols equally.

SG/LSG: Since Double-Spending Resistance has been decomposed into decentralized and cen-

Table 4.6: Secure: Contributions

Node	**Parent**	**Type**	**Contribution**
Integrity	Secure	AND	0.2
Consistency	Secure	AND	0.2
Resistance to Attacks	Secure	AND	0.2
Double-Spending Prevention	Secure	AND	0.2
Availability	Secure	AND	0.2
Tamper Resistance	Integrity	MAKE	1.0
Resistance Against Tampering of New Transactions	Tamper Resistance	AND	0.5
Resistance Against Tampering of Old Transactions	Tamper Resistance	AND	0.5
Low Stale Block Rate	Consistency	HELP	0.5
Confirmations in the Network	Consistency	HELP	0.5
Consensus Protocols	Consistency	HELP	0.5
Consensus Protocols	Double-Spending Resistance in Decentralized Systems	HELP	0.75
Validity of Transactions	Double-Spending Resistance in Decentralized Systems	HELP	0.25
Security Level	Resistance to Attacks	MAKE	1.0
Double-Spending Resistance in Decentralized Systems	Double-Spending Resistance	HELP	0.75
Degree of Centralization	Double-Spending Resistance	HELP	0.75
Degree of Centralization	DDoS Resistance	HURT	-0.75
Amount of Nodes	DDoS Resistance	HELP	0.5
Double-Spending Resistance	Double Spending Prevention	MAKE	1.0
DDoS Resistance	Availability	MAKE	1.0

tralized, and the latter being denoted by the LSG Degree of Centralization, both were given an equal contribution towards Double-Spending Resistance.

Privacy

The chosen values for the SG Privacy are in the Tables 4.7, 4.8 and 4.9.

SG: The SG Privacy was decomposed into **User's Anonymity** and **Confidentiality of Transactions** with an 'AND' contribution, leading to both SGs having the contribution towards the SG Privacy.

OP.: The OP. **Homomorphic Encryption** and **Non-Interactive Zero Knowledge Proof** were equally given an OP impact of 0.5, since both were identified by [79] as techniques to achieve the Security and Privacy property Confidentiality. However, [79] also considered other techniques, which were not added to the SIG.

OP.: **Mixing/Tumbler** and **Anonymous Signatures** were also equally given the same OP impact, since both techniques can be used to achieve Unlinkability of Transactions, as noted by [79].

LSG: Furthermore, **Pseudonymity** and **Unlinkability** were both also given an equal contribution towards User's Anonymity, as both were decomposed with an 'AND' contribution, *cf.* Section 4.1.1.

LSG: **Unlinkability of Transactions** was given a smaller LSG weight compared to the LSG **Pseudonymity**, since Unlinkability needs to be enhanced, and is not fully supported so far [79].

Table 4.7: Privacy: Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
Use of Digital Signatures	Anonymous Signatures	1.0
Use of Public Keys as Pseudonyms	Pseudonymity	1.0
Mixing/ Tumbler	Unlinkability of Transactions	0.75
Anonymous Signatures	Unlinkability of Transactions	0.75
Homomorphic Encryption	Data Privacy	0.5
Non-Interactive Zero Knowledge Proof	Data Privacy	0.5

Table 4.8: Privacy: LSG Weights

LSG	**Weight**
Pseudonymity	1.0
Unlinkability of Transactions	0.75
Data Privacy	0.75

Table 4.9: Privacy: Contributions

Node	**Parent**	**Type**	**Contribution**
Pseudonymity	User's Anonymity	AND	0.5
Unlinkability of Transactions	User's Anonymity	AND	0.5
Data Privacy	Confidentiality of Transactions	MAKE	1.0
User's Anonymity	Privacy	AND	0.5
Confidentiality of Transactions	Privacy	AND	0.5

Applicability

The quantifications for the SG Applicability can be found in the Tables 4.10, 4.11 and 4.12.

LSG/SG: Smart contract Support was decomposed into **Smart Contract Privacy**, **Smart Contract Features** and **Smart Contract Functionality** with an 'AND' contribution, leading to a contribution value of 1/3 for each. To incorporate all three aspects when considering Smart Contracts, the 'AND' contribution has been chosen. Consequentially, as can be seen in the case of Bitcoin, Turing Incompleteness [2] hurts and limits the Smart Contract Support in the SIG.

LSG: **Interoperability**, **Token** and **Fees** were all given a LSG weight of 0.5, since, they can be useful for Applications, but are not supported by all BC, *cf.* Section 4.1.1, and therefore not always compulsory.

OP.: **Data Storage Possibilities** has been given a smaller OP. impact than **Turing Completeness**, since due to the functionality gains by using a Turing Complete Scripting Language more applications are deemed possible, *cf.* Section 4.1.1.

Table 4.10: Applicability: Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
Availability of Light Clients	Access	1.0
Smart Contracts	Smart Contract Functionality	1.0
Turing Completeness	Smart Contract Features	0.75
Data Storage Possibilities	Smart Contract Features	0.25
Interoperability Possibilities	Interoperability	1.0
Token Usage	Token	1.0
No Fees	Fees	1.0
Permissioned	Blockchain Data Writability	1.0
Permissionless	Blockchain Data Writability	1.0
Public	Blockchain Data Visibility	1.0
Private	Blockchain Data Visibility	1.0

Table 4.11: Applicability: LSG Weights

LSG	**Weight**
Access	0.25
Smart Contract Functionality	1.0
Smart Contract Features	1.0
Interoperability	0.5
Token	0.5
Fees	0.5
Blockchain Data Writability	1.0
Blockchain Data Visibility	1.0

Table 4.12: Applicability: Contributions

Node	**Parent**	**Type**	**Contribution**
Data Privacy	Smart Contract Privacy	MAKE	1.0
Access	Applicability	AND	0.14
Smart Contract Support	Applicability	AND	0.14
Interoperability	Applicability	AND	0.14
Token	Applicability	AND	0.14
Fees	Applicability	AND	0.14
Blockchain Data Writability	Applicability	AND	0.14
Blockchain Data Visibility	Applicability	AND	0.14
Smart Contract Privacy	Smart Contract Support	AND	0.33
Smart Contract Functionality	Smart Contract Support	AND	0.33
Smart Contract Features	Smart Contract Support	AND	0.33

Computation of Values

The values from the tables above were steps A, B and C from the quantification framework [1]. Hence, they can then be used to compute the OP Scores, LSG Scores and SG Scores following [1] quantification extension. As an example, given Equation 2.1 from Section 2.2.1 and the values from Tables 4.1, 4.2, 4.4 and 4.5 the OP Score of *Large Block Size* is calculated in the following way:

$$\begin{aligned}
OP_{Score} &= impact_{Layer1 \times LargeBlockSize} \times Layer1_{weight} \\
&\quad + impact_{LowStaleBlockRate \times LargeBlockSize} \times LowStaleBlockRate_{weight} \\
OP_{Score} &= 0.5 \times 1.0 + (-0.5) \times 0.5 \\
OP_{Score} &= 0.5 - 0.25 \\
OP_{Score} &= 0.25
\end{aligned}$$

Although in the next step the OPs are to be selected, *cf.* [1], this was not done for the generic SIG, since it serves as a representation and example only. As will become clear from the calculations, this also leads to unfavorable LSG and SG scores due to all OPs being selected, and in some cases contradictory values are reached as well. However, for the use cases in Section 5.1 this step is crucial and will naturally be applied.

Subsequently, using Equation 2.2, the LSG scores can be calculated, *cf.* Table 4.14.

Table 4.13: Computed Operationalization Scores

Operationalization	**Operationalization Score**
UTXO	0.5
Account Model	0.0
Layer 2 Solutions	1.0
Large Block Size	0.25
Fast Block Interval Time	0.25
Use of Secure/Collision-Resistent Hash Functions	1.5
Hash Chained Storage	0.75
High Number of Confirmations in the Network	0.0
dBFT	2.0
PoA	2.0
PoS	1.5
dPoS	1.75
PoW	1.25
Use of Digital Signatures	1.56
Centralized Entity	0.75
High Amount of Nodes	0.5
Use of Publilc Keys as Pseudonyms	1.0
Mixing/Tumbler	0.56
Anonymous Signatures	0.56
Homomorphic Encryption	0.38
Non-Interactive Zero Knowledge Proof	0.38
Availabililty of Light Clients	0.25
Smart Contracts	1.0
Turing Completeness	0.75
Data Storage Possibilities	0.25
Interoperability Possibilities	1.5
Token Usage	0.5
No Fees	0.5
Permissioned	1.0
Permissionless	1.0
Public	1.0
Private	1.0

Table 4.14: Computed LSG Scores

LSG	**LSG Score**
Parallel Transactions	1.0
Layer 2	1.0
Layer 1	1.0
Resistance Against Tampering of New Transactions	0.75
Resistance Against Tampering of Old Transactions	0.75
Low Stale Block Rate	-1.0
Confirmations in the Network	1.0
Consensus Protocols	1.0
Security Level	1.0
Validity of Transactions	1.0
Degree of Centralization	0.75
Amount of Nodes	1.0
Pseudonymity	1.0
Unlinkability of Transactions	1.0
Data Privacy	1.0
Access	1.0
Smart Contract Functionality	1.0
Smart Contract Features	1.0
Interoperability	1.0
Token	1.0
Fees	1.0
Blockchain Data Writability	1.0
Blockchain Data Visibility	1.0

The LSG Score for Unlinkability of Transactions was computed in the following way:

$$\begin{aligned}
LSG_{score} &= max(min(0.75 + 0.75, 1.0), -1.0)\\
LSG_{score} &= max(min(1.5, 1.0), -1.0)\\
LSG_{score} &= max(1.0, -1.0)\\
LSG_{score} &= 1.0
\end{aligned}$$

Given Equation 2.3, the SG Scores from Table 4.15 are then computed. As an example, the SG Score for Consistency was calculated in the following way:

$$\begin{aligned}
SG_{score} &= max(min(((-1.0 \times 0.5) + (1.0 \times 0.5) + (1.0 \times 0.5)), 1.0), -1.0)\\
SG_{score} &= max(min((-0.5 + 0.5 + 0.5), 1.0), -1.0)\\
SG_{score} &= max(min(0.5, 1.0), -1.0)\\
SG_{score} &= max(0.5, -1.0)\\
SG_{score} &= 0.5
\end{aligned}$$

Table 4.15: Computed SG Scores

SG	**SG Score**
Performance	1.0
Scalability	1.0
Throughput	1.0
Secure	0.6
Integrity	0.75
Tamper Resistance	0.75
Consistency	0.5
Resistance to Attacks	1.0
Double-Spending Prevention	1.0
Double-Spending Resistance	1.0
Double-Spending Resistance in Decentralized Systems	1.0
Availability	-0.25
DDoS Resistance	-0.25
Privacy	1.0
User's Anonymity	1.0
Confidentiality of Transactions	1.0
Applicability	1.0
Smart Contract Support	1.0
Smart Contract Privacy	1.0

4.2 Implementation

In this section an application is documented that has been built to specifically handle the BC SIG. A user is able to view a visualization of the graph, can select use cases (*cf.* Section 5.1) and also select OPs. A short overview of the client facing user interface of the application is described in Section 4.2.1, the REST API between the client and the server is given in Section 4.2.2 and the calculation of the scores is documented in Section 4.2.3. The sequence and flow between the different components has been sketched in Figure 4.6.

4.2.1 Frontend

For the frontend, React has been used with the additional libraries:

- axios, a HTTP library for node.js [3].
- react-beforeunload, a library to add functionality to unmounting components in the client [10].
- react-copy-to-clipboard, a library to copy text to the user's clipboard [11].
- material-ui, an User Interface (UI) design framework for React [39].

Upon opening the BC SIG page, the user is presented with the SIG visualization, *cf.* Figure 4.7, and can select OPs, see the computed SG scores from the graph and also copy

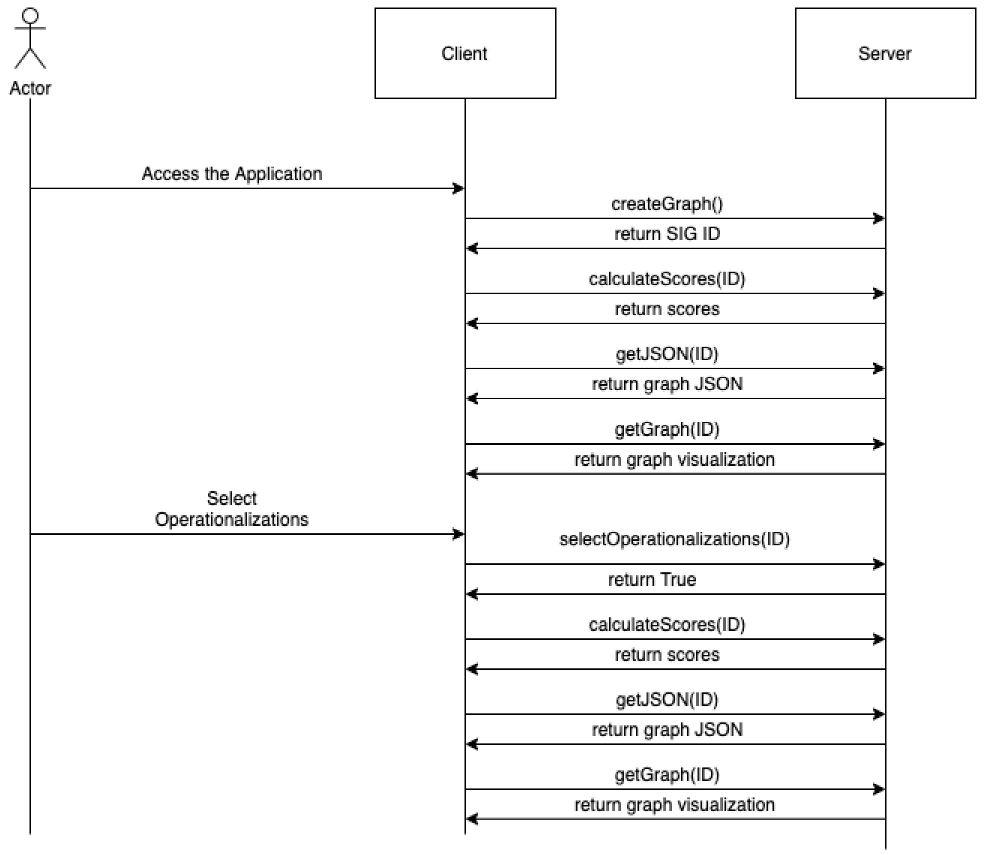


Figure 4.6: The sequence between the components. Source: The author.

the SIG's JSON, to *e.g.,* store it elsewhere. All of this is being sent from the server to the client to display. Directly below the header, the two different evaluation use cases from Section 5.1 were added, and the use case 'IoT' has been preselected as default.

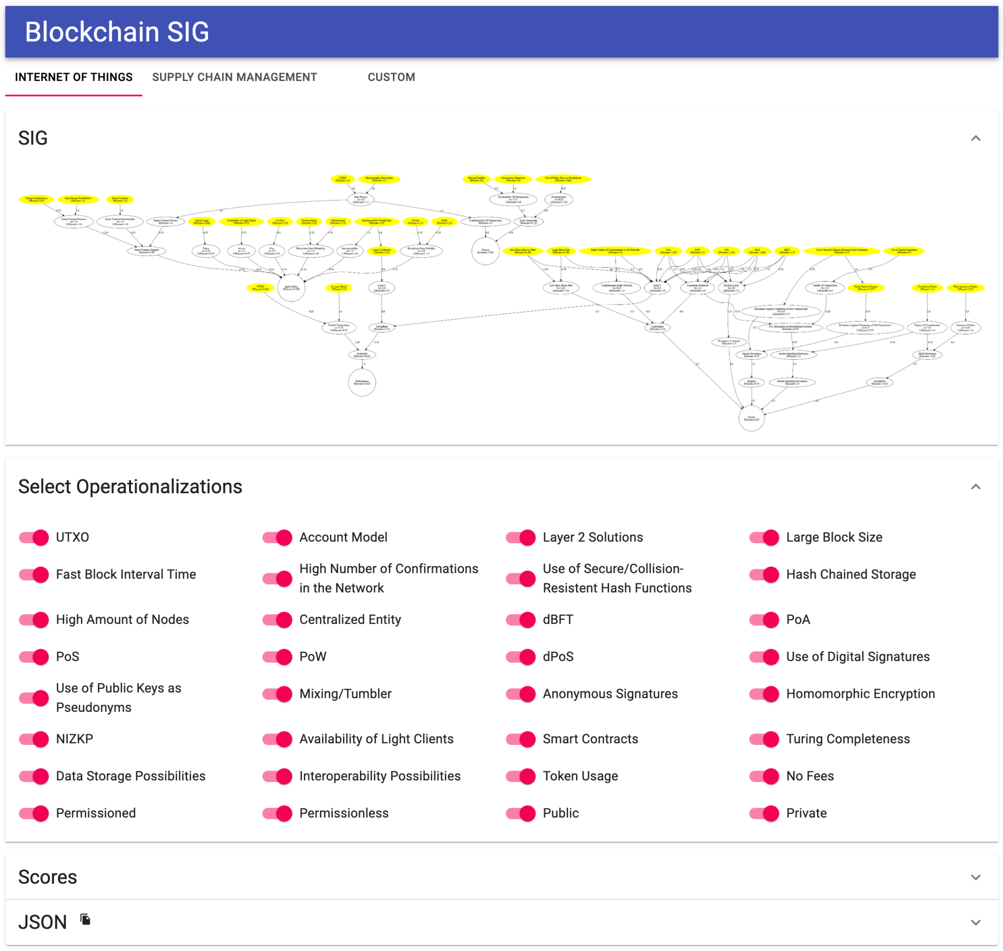
Upon clicking on the tab 'Custom', the UI changes, *cf.* Figure 4.8, and the user can enter a custom SIG, which in turn gets sent and processed by the server to create a visualization of the graph, and also to compute the scores. However, the application requires the custom SIG that was entered to be a networkX graph encoded as a JSON.

4.2.2 Server API

For the implementation of the server that calculates the SIG values and creates the graph, Python has been used with the following additional libraries:

- NetworkX, a network library for Python [43].
- Sanic [53] and Sanic_cors [62], a framework to build Python servers.

To make the code modular, a SIG class has been defined that can be used to handle different SIG with various user-defined contributions and impacts. These SIGs are stored in the backend, the Sanic Python Server, which communicates with the various clients.



Blockchain SIG

INTERNET OF THINGS | SUPPLY CHAIN MANAGEMENT | CUSTOM

SIG

Select Operationalizations

UTXO	Account Model	Layer 2 Solutions	Large Block Size
Fast Block Interval Time	High Number of Confirmations in the Network	Use of Secure/Collision-Resistent Hash Functions	Hash Chained Storage
High Amount of Nodes	Centralized Entity	dBFT	PoA
PoS	PoW	dPoS	Use of Digital Signatures
Use of Public Keys as Pseudonyms	Mixing/Tumbler	Anonymous Signatures	Homomorphic Encryption
NIZKP	Availability of Light Clients	Smart Contracts	Turing Completeness
Data Storage Possibilities	Interoperability Possibilities	Token Usage	No Fees
Permissioned	Permissionless	Public	Private

Scores

JSON

Figure 4.7: The user interface of the application. Source: The author.

The server logic has been set up in a way, that when a client connects to the server, it calls *createGraph* to create an unique *id* (*cf.* line 3 in Listing 4.1) that is associated with the SIG and stored in a dictionary data structure (*cf.* line 5 in Listing 4.1), and used throughout the session as an identifier for the various clients and their SIGs.

```python
1 @api.route('/createGraph', methods=['POST'])
2 async def createGraph(request):
3     id = uuid.uuid4().hex
4     sig = bc_SIG_iot(id)
5     sigs[id] = sig
6     return response.json({'success': True, 'id': id})
```

Listing 4.1: Create Graph API Endpoint

This function also creates the default 'IoT' SIG (*cf.* line 4 in Listing 4.1), which has arbitrarily been chosen as the default SIG to present to the user.

The server then returns this id to the client (*cf.* line 6 in Listing 4.1), and after the client receives this id, it calls in succession the following three functions. In a way, these three functions are the backbone of the application, if a user changes the use case or deselects

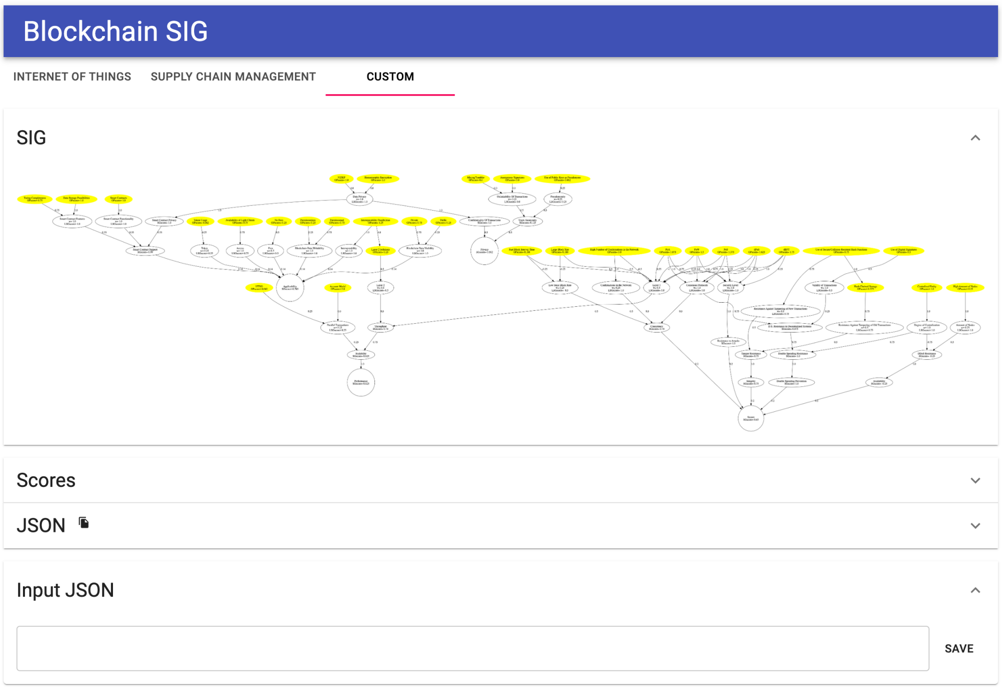


Figure 4.8: The ‘Custom’ UI View which allows the user to enter a custom SIG as a JSON. Based on the information, the system calculates the SIG scores. Source: The author.

an OP, these functions are always those that compute and return the values and create the visualization for the client to display. They all have in common, that they are ‘GET’ functions that receive the unique id from the client, (*cf.* line 4 in Listing 4.2).

The first function is to calculate all the SIG scores, namely the OP, LSG and SG scores and to return them to the client in a JSON format. The ‘GET’ functions make use of a ‘getter’ function that returns the client’s SIG, which is identified by the unique id, from the dictionary (*cf.* line 4 in Listing 4.2). Line 5 of Listing 4.2 calls the function in Listing 4.8.

```
1 @api.route('/calculateScores', methods=['GET'])
2 async def calculateScores(request):
3     try:
4         sig = getSIGfromId(request.args['id'][0])
5         scores = sig.calculateScores()
6         return response.json(scores)
7     except:
8         return response.json({'success': False})
```

Listing 4.2: Calculate Scores API Endpoint

After the scores have been calculated, the server outputs the graph in a JSON format, containing all the nodes, edges and scores from the SIG and returns it subsequently to the client (*cf.* line 5 in Listing 4.3). This JSON can be used later, *i.e.*, entered in networkX as input to display the very same SIG again.

```
1  @api.route('/getJSON', methods=['GET'])
2  async def getJSON(request):
3      try:
4          sig = getSIGfromId(request.args['id'][0])
5          return response.json(sig.createJSON())
6      except:
7          return response.json({'success':False})
```

Listing 4.3: Get JSON API Endpoint

And subsequently, the client asks for the visualization of the SIG with the scores. The server tries to write a 'PNG' image file (*cf.* line 6 in Listing 4.4) and to return it to the client to display (*cf.* line 8 in Listing 4.4). To avoid errors and exceptions, a check is made before sending the file, to make sure whether writing the file was truly successful and the file exists (*cf.* line 7 in Listing 4.4).

```
1  @api.route('/getGraph', methods=['GET'])
2  async def getGraph(request):
3      try:
4          id = request.args['id'][0]
5          sig = getSIGfromId(id)
6          sig.draw(id)
7          if os.path.isfile(id+'graph.png'):
8              return await response.file(id+'graph.png')
9          else:
10             return response.json({'success': False})
11     except:
12         return response.json({'exception': True})
```

Listing 4.4: Get Graph API Endpoint

After the computation of the previously mentioned functions has completed, the client's UI is fully loaded and the frontend is presented in full to the user.

Should the user decide to change the OP., the following function is called. It gets the id from the client, and based on the 'tabChoice' the user made, that is, if the user is currently working with the 'IoT' or 'Supply Chain Management' SIG, a new SIG is created (*cf.* lines 7, 9, 11 in Listing 4.4). Based on the OPs that were selected by the user, the deselected OPs are removed from the graph (*cf.* line 16 in Listing 4.4). This in turn makes the client ask the server anew to calculate the scores, to get the JSON and to get the SIG visualization.

Should the user want to change the 'tabChoice' instead, this function gets called as well and the currently selected OPs are taken into consideration and get applied to the different use case as well. This enables the user to compare the use cases easily with each other.

```
1  @api.route('/selectOperationalizations', methods=['POST'])
2  async def selectOperationalizations(request):
3      body = request.json
4      try:
5          id = body['id']
6          if body['tabChoice'] == 'IOT':
7              sig = bc_SIG_iot(id)
8          elif body['tabChoice'] == 'SCM':
```

```
            sig = bc_SIG_scm(id)
        elif body['tabChoice'] == 'Custom':
            pass
        else:
            raise Exception('Content Body Invalid')
        sig.selectOperationalizations(body['OP'])
        sigs[id] = sig
        return response.json({'success': True})
    except:
        return response.json({'success': False})
```

Listing 4.5: Select Operationalizations API Endpoint

If a user chooses to input his or her custom SIG, then the server attempts to create a graph from the given graph JSON (*cf.* line 7 in Listing 4.6). This will then also trigger the functions to calculate the scores, get the JSON and create the visualization.

```
1  @api.route('/readJSON', methods=['POST'])
2  async def readJSON(request):
3      try:
4          input = json.loads(request.json['inputJSON'])
5          id = request.json['id']
6          sig = getSIGfromId(id)
7          sig.createGraphFromJSON(input)
8          sigs[id] = sig
9          return response.json({'success':True})
10     except:
11         return response.json({'success':False})
```

Listing 4.6: Read JSON API Endpoint

The final function in the API, is the ‘cleanup’ function. Since every client creates a visualization graph, this would clutter the server with the image files and it would sooner or later run out of memory. This function gets called when a client unmounts its components and it deletes the computed SIG from the dictionary to free up memory, and also removes the ‘PNG’ file from the server, (*cf.* line 8 in Listing 4.7).

```
1  @api.route('/cleanup', methods=['POST'])
2  async def cleanup(request):
3      id = request.json['id']
4      if id and sigs.get(id):
5          del sigs[id]
6      if id and os.path.exists(id + 'graph.png'):
7          # check if file exists, then delete
8          os.remove(id + 'graph.png')
9      return response.json({'success':True})
```

Listing 4.7: Cleanup API Endpoint

4.2.3 Computation of Scores

The calculation of the scores is done in a procedural manner, and the SIG calculation score code was adapted from the code of [56]. Based on the predefined LSG weights and OP impacts, firstly the OP scores are computed, secondly the LSG scores, and lastly the SG scores. Therefore, for the SIG class the computation has been split into three functions as well.

```
1 def calculateScores(self):
2     self.__calculateOperationalizationScores()
3     self.__calculateLSGScores()
4     self.__calculateSGScores()
5     return self.printSGScore()
```

Listing 4.8: Calculate Scores Function

After the computation of the scores is done, this function also prints the computed SG scores in a JSON to return the client, which is done with the function *printSGScore()* (*cf.* line 5 in Listing 4.8).

For the OP score calculation, it is important, that the calculation is done in a *top-down* manner, since parent's OP score are included in the computation of the child node's OP score (*cf.* Equation 2.1 in Section 2.2.1), given that this constellation of OP edges exists in the graph. Therefore, to ensure the *top-down* calculation, all the nodes in the graph are *topologically sorted* (*cf.* [75]) and then the order is reversed (*cf.* line 4 in Listing 4.9). The computation gets all nodes from the graph that were tagged as OPs (*cf.* line 2 in Listing 4.9), and for every node the algorithm loops through all its edges and makes the case distinction whether the parent node is a LSG, or a OP (*cf.* lines 12, 15 in Listing 4.9). The algorithm applies Equation 2.1 then.

```
1  def __calculateOperationalizationScores(self):
2      operationalizations = list(nx.get_node_attributes(
3                              self.sig, 'operationalization').keys())
4      topologicallySorted = list(reversed(list(
5                              nx.topological_sort(self.sig))))
6      for node in topologicallySorted:
7          if node in operationalizations:
8              impact = 0
9              for edge in self.sig.edges(node, data=True):
10                 edgeImpact = edge[2]['impact']
11                 parent = self.sig.nodes[edge[1]]
12                 if(parent.get('leafSoftGoal')):
13                     weight = parent['w']
14                     impact += edgeImpact * weight
15                 else:
16                     impact += parent['opScore']
17             impact = round(impact, 2)
18             self.sig.nodes[node]['opScore'] = impact
19             self.sig.nodes[node]['label'] =
20                                     node + '\nOPscore= '+ str(impact)
```

Listing 4.9: Calculate Operationalization Scores Function

To compute the LSG scores, the Equation 2.2 can be simply applied to all LSG nodes in the graph, taking into account the impact of the OPs (*cf.* lines 5-8 in Listing 4.10). Since the label attribute is used to show the weight and also the score of the node, the label gets overwritten here (*cf.* lines 9, 11-12 in Listing 4.10).

```python
def __calculateLSGScores(self):
    lsg = list(nx.get_node_attributes(self.sig, 'leafSoftGoal').keys())
    for node in lsg:
        lsgScore = 0
        for edge in self.sig.in_edges(node, data=True):
            lsgScore += edge[2]['impact']
        lsgScore = min(max(lsgScore, -1.0), 1.0)
        lsgScore = round(lsgScore, 2)
        weight = self.sig.nodes[node]['w']
        self.sig.nodes[node]['lsgScore'] = lsgScore
        self.sig.nodes[node]['label'] =
            node + '\nw= ' + str(weight) + '\nLSGscore= ' +str(lsgScore)
```

Listing 4.10: Calculate LSG Scores Function

Lastly, the SG scores themselves are computed. This algorithm follows a similar logic to the one to compute the OP scores, *cf.* Listing 4.10. The nodes from the graph are again topologically sorted, but this time the order is not reversed, leading to a *bottom-up* approach of the computation (*cf.* line 3 in Listing 4.11). This step is necessary, since SG nodes can have parent SG nodes, and the computation must follow the right order in that case. The algorithm also makes a case distinction to check if the children is a LSG or a SG, and the Equation 2.3 is then applied (*cf.* lines 9-15 in Listing 4.11).

```python
1  def __calculateSGScores(self):
2      sg = list(nx.get_node_attributes(self.sig, 'softGoal').keys())
3      topologicallySorted = list(nx.topological_sort(self.sig))
4      for node in topologicallySorted:
5          if node in sg:
6              sgScore = 0
7              for edge in self.sig.in_edges(node, data=True):
8                  children = self.sig.nodes[edge[0]]
9                  contribution = 1.0
10                 if edge[2].get('contribution') is not None:
11                     contribution = edge[2]['contribution']
12                 if children.get('lsgScore') is not None:
13                     sgScore += contribution*children.get('lsgScore')
14                 elif children.get('sgScore') is not None:
15                     sgScore += contribution*children.get('sgScore')
16                 else:
17                     raise Exception
18             score = round(sgScore, 2)
19             self.sig.nodes[node]['sgScore'] = score
20             self.sig.nodes[node]['label'] =
21                                 node + '\nSGscore= ' + str(score)
```

Listing 4.11: Calculate SG Scores Function

Chapter 5

Evaluation

In this chapter, in Section 5.1, to show the flexibility of this approach, the proposed SIG is firstly evaluated on different use cases, that are both relevant and contemporary for BC applications. Consequently the use cases suggest each different parameter choices for the SIG and lead to a different evaluation and outcome. Sequently, in Section 5.2 the computational performance of the SIG score calculation is evaluated. Following, the results from the use cases are evaluated through Machine Learning in Section 5.3. Ultimately, the results from the evaluation are discussed in Section 5.4.

5.1 Use Cases

For the evaluation of the proposed SIG, the use cases **Internet of Things (IoT)** and **Supply Chain Management (SCM)** have been chosen, since they are major use cases for BC, *cf.* [8, 29, 52]. To undertake the evaluations, the key aspects for applying BC on these use cases were researched and evaluated, and the SIG values were adjusted to properly reflect these aspects. However, even though the use case specifc aspects were weighted, at the core, these values are still arbitrary, though they were discussed with the supervisors. The BC that were chosen for the use case evaluation were based on [54], and due to its popularity [72], *NEO* was added to the evaluation as well.

The chosen values for the SIG are displayed in the Appendix in Tables A.1, A.2 for IoT, and for SCM in Tables B.2, B.1. However, the contributions between LSG to SG or SG to SG from the generic SIG were not changed, since they were part of the SG decomposition step and therefore part of the SIG foundation. The aforementioned contribution values from the proposed SIG are found in Tables 4.3, 4.6, 4.9, 4.12 in Chapter 4.1.2.

5.1.1 Internet of Things (IoT)

The first use case that was selected is *IoT*, a technology that has seen enormous growth and impact lately, and its name stems from a network of connected devices, hence *things* [29].

Table 5.1: Evaluated IoT SG Scores. The scores are capped between [-1, 1] [1].

BC	Performance	Secure	Privacy	Applicability
Bitcoin	-0.31	0.76	0.06	0.40
Ethereum	-0.19	0.76	0.56	0.49
Stellar	0.19	0.14	0.56	0.51
EOS	0.38	0.3	0.06	0.58
Multichain	0.44	0.48	0.06	0.61
HyperLedger Sawtooth	0.44	0.12	0.56	0.70
IOTA	0.25	0.27	0.06	0.42
NEO	0.44	0.3	0.56	0.56

In fact, earlier predictions assumed that by 2020 more than *50 billion* connected devices are reached [29]. However, given its potential, open issues and limitations are also present [29]. In particular, IoT devices are limited by their processing power and storage, and unfortunately, the rapid development has lead to security issues not being fully addressed so far [31]. Nonetheless, BCs utilizing SCs can be expected to solve key challenges of IoT and additionally, IoT is also seen as a major use case for BC technology [29, 30].

Various BC aspects that are of interest to IoT and were considered and applied to the SIG:

- ***Smart Contracts***, can be used, amongst others, for authorization and to administer devices [29], and also for autonomous transactions [30].
- ***Data Privacy***, can be ensured via SCs to set access rules and conditions [29].
- ***Decentralization***, a decentralized BC can overcome the centralized clouds to prevent a single point of failure and provide availability [30].
- ***Security***, IoT security has been a predominant issue, compromised IoT devices have in the past been used to launch DDoS attacks [30], and also double-spending must be avoided [29].
- ***Interoperability***, is seen currently as an open issue for IoT, namely how different security protocols can interact with each other [29].
- ***Scalability*** and ***Efficiency***, are both ongoing challenges for BC based IoT solutions, which must also be considered in conjunction with the limitations that IoT devices are both low-cost and low-power devices [29].
- ***Immutability***, is important for identity and access management systems [30].
- ***Private***, this refers to private BC, allowing to store device's cryptographic hashes [30].
- ***Data Storage***, enabling *e.g.,* the storage of sensor data on the BC [31].

Thus, the SIG values were adapted to take these aspects into consideration, and the results of this evaluation are noted in Table 5.1, while as the impact values and LSG weights are noted in Tables A.2 and A.1.

5.1.2 Supply Chain Management (SCM)

SCM is in its nature about creating value to the customer, enabled by planning and controlling activities and by simultaneously integrating and coordinating processes across companies [8]. Nonetheless, this is not easily accomplishable, since supply chains range across continents, heterogenous regulatory policies and cultural differences make this endeavor very complex to manage [52]. As a result, supply chain problems can lead to inefficient transactions, fraud, and a lack of traceability can lead to safety and health hazards when issues arise and problematic goods are not traceable, as has happened in the past [52].

Furthermore, consumers desire to know the *provenance* and the real *value* of certain items, and they want to be able to verify those pieces of information as well [52]. It is clear, that a lack of transparency prevents that, also leading to companies endangering their reputation [52]. Consequently, supply chain problems lead to a lack of trust between parties and SCM requires better information sharing and verifiability, and importantly a fundamental traceability [52].

The four key pillars that BCs provide for SCM were identified by [8]:

- ***Transparency***, the BC is shared and its data originates from various data sources.
- ***Validation of information***, enabled by immutability and consensus-based verification.
- ***Automation***, refers to the SC functionality of BCs that enables 'ex-post' contract enforcability and also non reversibility.
- ***Tokenization***, denotes to the BC's ability to create tokens, each referring to an asset claim in this context, and to exchange them between parties.

Additionally, in comparison to other BC applications, a ***closed***, *i.e.,* a private and permissioned BC is seen as adequate for SCM [52]. Therefore, the values for the SIG were adjusted to reflect these aspects. The results of this evaluation are noted in Table 5.2.

Table 5.2: Evaluated SCM SG Scores. The scores are capped between [-1, 1] [1].

BC	**Performance**	**Secure**	**Privacy**	**Applicability**
Bitcoin	-0.31	0.73	0.13	0.43
Ethereum	-0.28	0.73	0.13	0.48
Stellar	0.09	0.19	0.13	0.55
EOS	0.38	0.4	0.13	0.50
Multichain	0.34	0.53	0.06	0.61
HyperLedger Sawtooth	0.25	0.16	0.06	0.65
IOTA	0.16	0.21	0.06	0.42
NEO	0.44	0.4	0.06	0.55

5.2 Performance

For the performance evaluation, the BC SIG application has been evaluated, namely, the computation of the calculation time for the *OP*, *LSG* and *SG* scores were timed and noted, and finally compared to each other, *cf.* Figure 5.1. The code of the respective functions can be seen in Listings 4.9, 4.10 and 4.11. It is noteworthy, that the computation of these values happen in the backend, and as such, the latency to return the scores to the clients is not included in the evaluation results, only the score computation time.

The evaluation has been performed 10 times on a MacBook Pro (15-inch, 2017) with a 2.8 GHz Quad-Core Intel Core i7 Processor and 16 GB of memory, and the resulting average computation times are shown in Figure 5.1.

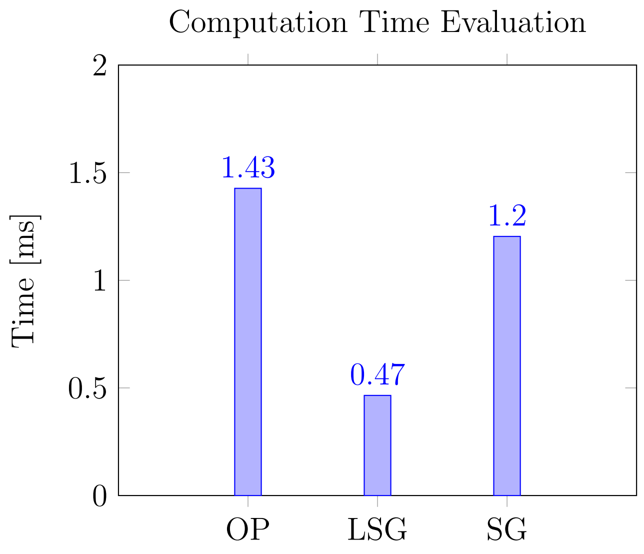


Figure 5.1: Comparison of the average computation times for the calculation of the *OP*, *LSG* and *SG* scores in ms *(n=10)*. Source: The author.

From the Listing 4.9 and Listing 4.11 it is clear, that the *topological sorting* of the nodes in the graph takes a performance hit for the computation of the OP. and SG scores. In comparison with Listing 4.10, where no graph preprocessing is needed, the computation is naturally much faster.

Nonetheless, the difference in results are not peculiarly problematic, since the time frame is in the scope of *milliseconds*. These results ensure, that the application can be used in *real-time* and further extenstions, *e.g.,* a larger SIG is not a problem.

5.3 Machine Learning

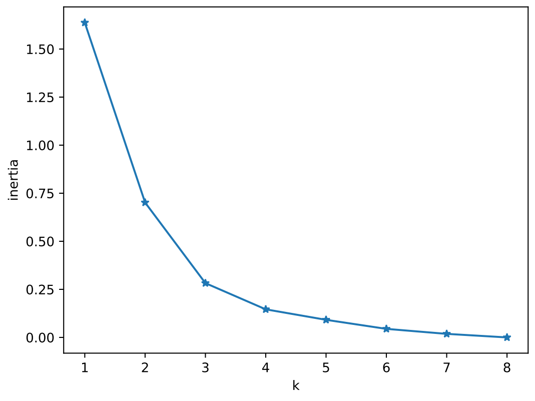
For the Machine Learning evaluation, the data from the use cases *IoT* and *SCM*, in Tables 5.1 and 5.2, were used for clustering with the *Scikit-Learn* Python Library [57, 58]

Table 5.3: Computed Mutual Information between SGs and Cluster Label.

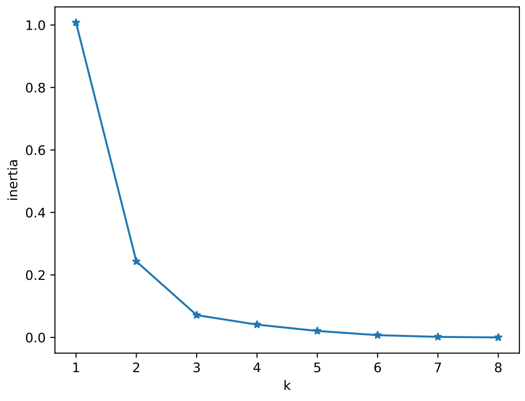
Use Case	Performance	Secure	Privacy	Applicability
IoT	0.39	0.57	0.82	0.00
SCM	0.21	0.35	1.01	0.01

and the visualizations were made with *Matplotlib* [40] and *Seaborn* [59]. The goal of this evaluation is to apply the *k-Means clustering* algorithm [37] to find clusters from the results that group different BCs together for a specific use case. Consequently, based on their scores, the clustering results lead to a classification of the BCs, and similarities between the BC implementations can be found.

For clustering, the right amount of clusters have first to be found, thus, the *elbow* technique [66] was applied, and by plotting the different amount of clusters, for both use cases $k = 3$ was chosen as appropriate, *cf.* Figures 5.2a and 5.2b. The idea of this technique is to evaluate the Sum of Square Errors (*inertia* [58]), against the amount of clusters to find a k value after a drastic decrease in the Sum of Square Errors [66].



(a)



(b)

Figure 5.2: Computing the best amount of clusters for the k-means clustering (a) IoT Use Case and (b) SCM Use Case. Source: The author.

Although all the computed values are bounded in [-1, 1], the distribution of the values is not balanced, *cf.* the diagonal entries in Figure 5.3 denoting the distributions. Therefore, the values have been *Min-Max Normalized* to rescale them into the range of $[0, 1]$ [27]. Most importantly, this transformation preserves the relationship between the values [27], and is therefore applicable to clustering.

However, if all four SGs are clustered, this approach is not without issues, *cf.* *Mutual Information* in Table 5.3. In particular, Mutual Information captures how much information the variables share [69], and the high *Privacy* value implies that it already predicts the Cluster Label, and *Applicability* has no, or an extremely low impact.

Indeed, this can be best seen in Figure 5.4, it is noticeable that the clustering algorithm grouped the BCs with low *Performance* together (*Cluster 1*), since in those cases their

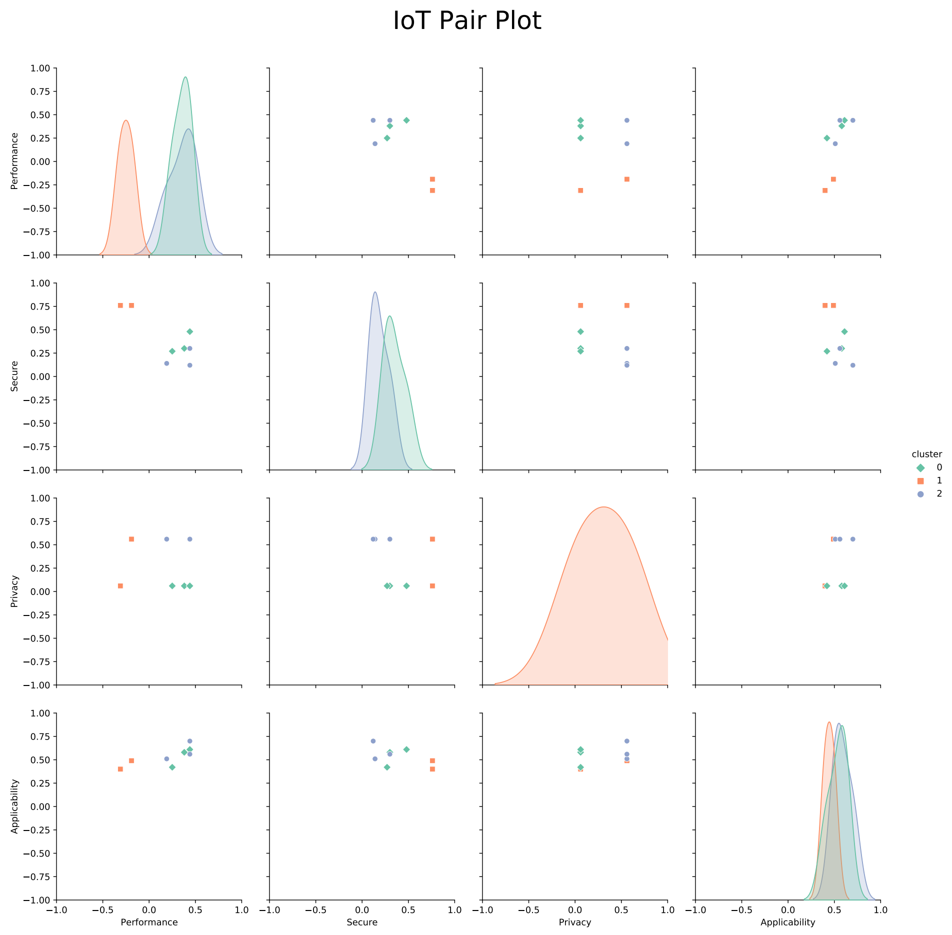


Figure 5.3: IoT: Clustered Pair Plots (without Normalization). Source: The author.

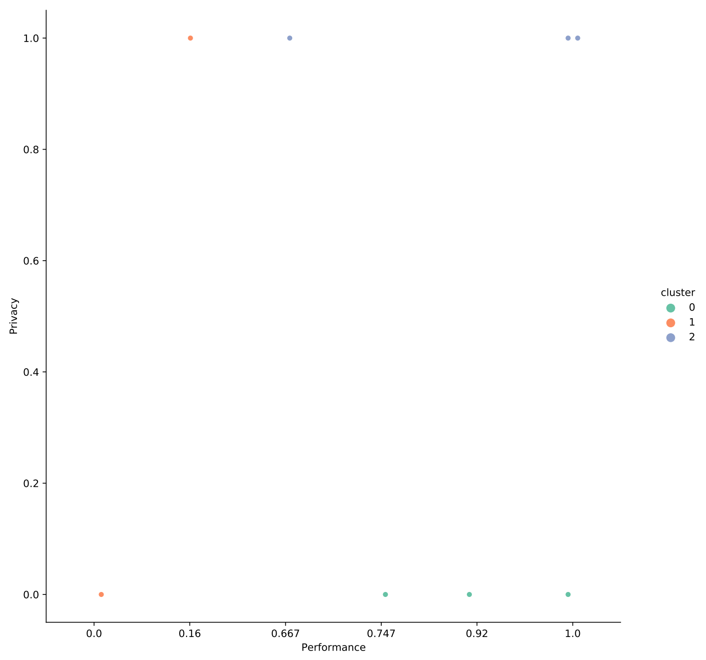


Figure 5.4: IoT Performance/Privacy Clustering Results (normalized). Source: The author.

Secure score is also high, and they are therefore further away from the other BCs, *cf.* Figure 5.3. In comparison, *Cluster 0* and *Cluster 2* have similar *Performance* results, and the algorithm has separated them mostly by their *Privacy* score.

Consequently, the SG *Privacy* has not been considered for further analysis, since it distorts the clustering strongly, which is problematic in this case, due to the proposed SIG not capturing the differences between the BCs well enough relating *Privacy* for the chosen use cases. This is further amplified by the chosen use cases not having a focus on Privacy, *cf.* Section 5.1.

Finally, the result of the clustering are visualized in Figure 5.5 for *IoT* and 5.6 for *SCM*. The algorithm ran with the value of *n_init* = 50, meaning that the resulting clustering was the best output from a random centroid initialization that ran 50 times, which is much higher than the default value of 10 [58]. This has been done, since clustering results are susceptible to the initial centroid locations [66]. Additionally, to obtain significance, the experiment was repeated 10 times, with each run resulting in the same clustering outcome.

For **IoT**, *cf.* Figure 5.5, the clustering results look promising, the *intra-cluster distance* [61] is low, all BCs forming a cluster are close together, meaning that they all have similar values, and additionally, the *inter-cluster distance* [61] is high, the clusters are far away from each other, and thus, not similar. By fulfilling these two clustering performance metrics, the clustering has been effective [61].

- **Cluster 0** captured *Multichain*, *NEO*, *EOS* and *HyperLedger Sawtooth*. All of them

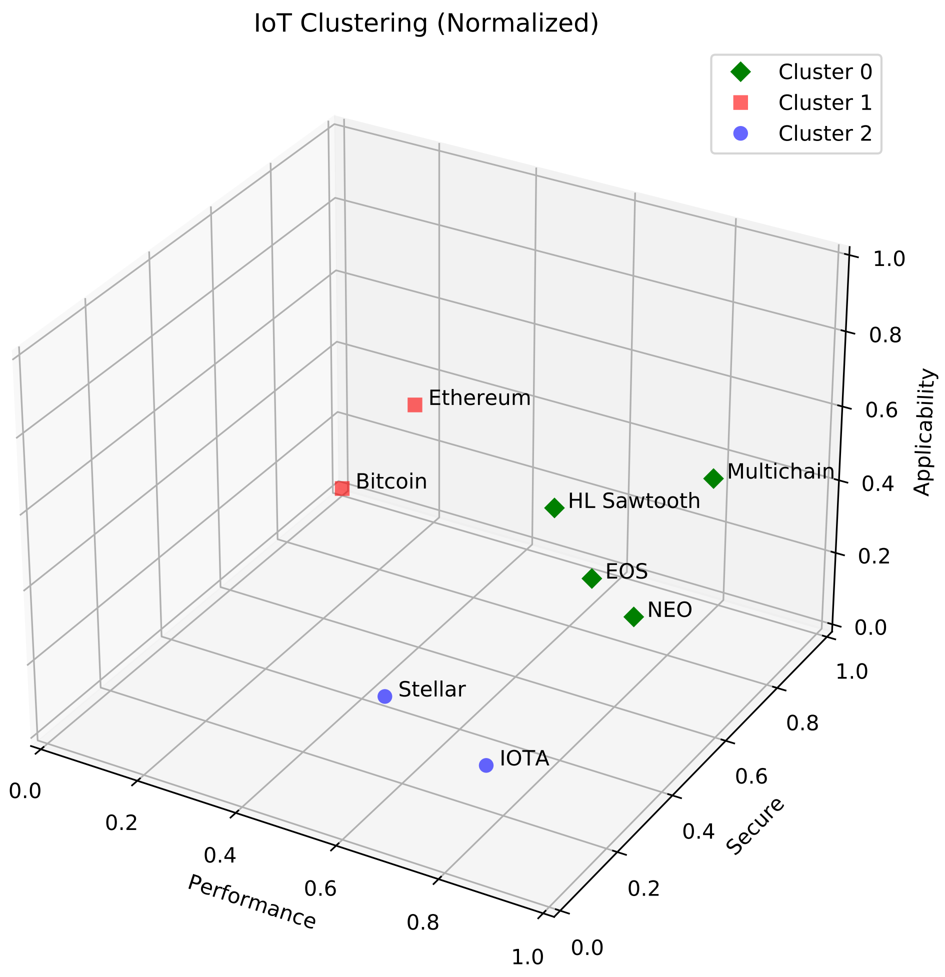


Figure 5.5: IoT k-means clustering results with normalized values. Source: The author.

scored the highest Performance scores, though their Secure scores were low to average and their Applicability scores were average to very high.

- **Cluster 1** includes *Ethereum* and *Bitcoin*, both have been clustered together, which can be explained by their use of *PoW* (*cf.* Section 2.3.1 and both have therefore similar attribute scores, due to PoW influencing both Performance and Secure SGs.
- **Cluster 2** contains *Stellar* and *IOTA*, although they scored high Performance values, both their Secure and their Applicability scores were comparatively low.

For **SCM**, *cf.* Figure 5.6, the clusters are not as neatly grouped, and the *intra-cluster distance*, *cf.* [61] is higher than in the *IoT* case. The visualization is also deceptive, since *HyperLedger Sawtooth* appears extremely close to *Bitcoin* and *Ethereum*, even though they are not close together, *HyperLedger Sawtooth's* high Performance and Applicability scores makes it simply appear in front of the others.

- **Cluster 0** includes *HyperLedger Sawtooth*, *Multichain*, *NEO* and *EOS*, which all scored highly in the Performance metric and they all have rather high Applicability scores, though NEO scored a bit lower than the average. Regarding Secure, their values differ greatly. In fact, the cluster spans a wide area and leads therefore to not a homogenous grouping.
- **Cluster 1** is equal to the IoT use case, and only includes *Ethereum* and *Bitcoin*. Again, they boast the highest Secure scores, but lack in Performance.
- **Cluster 2** captured *Stellar* and *IOTA*. Again, both have comparable Performance and Secure scores, though the clustering did not consider the Applicability differences.

Consequently, it is noteworthy, that due to the SIG design, both use cases resulted in the same clusters, even though the values differed between the use cases. This can be partly explained by the fact, that both use cases had an overlap in their requirements, leading to similarities between the relative values, and partly, this can be attributed to the inherent selection of BC specific Operationalizations, *i.e.,* BC implementations will not change their attributes.

In summary, based on the results of the clusterings, a BC classification can be made, and the following three categories present themselves:

- BCs that focus highly on Security, but on the other hand have low Performance and Applicability.
- BCs that at their core have *very* high Performance values, though their Security tends to be on the lower side. Additionally, they reach various degrees of Applicability.
- BCs that score high on the Performance side, but have low Applicability *and* Security.

Hence, for a BC selection point of view and a specific BC use case, it depends on the use case's priorities and requirements to select a BC from a resulting category.

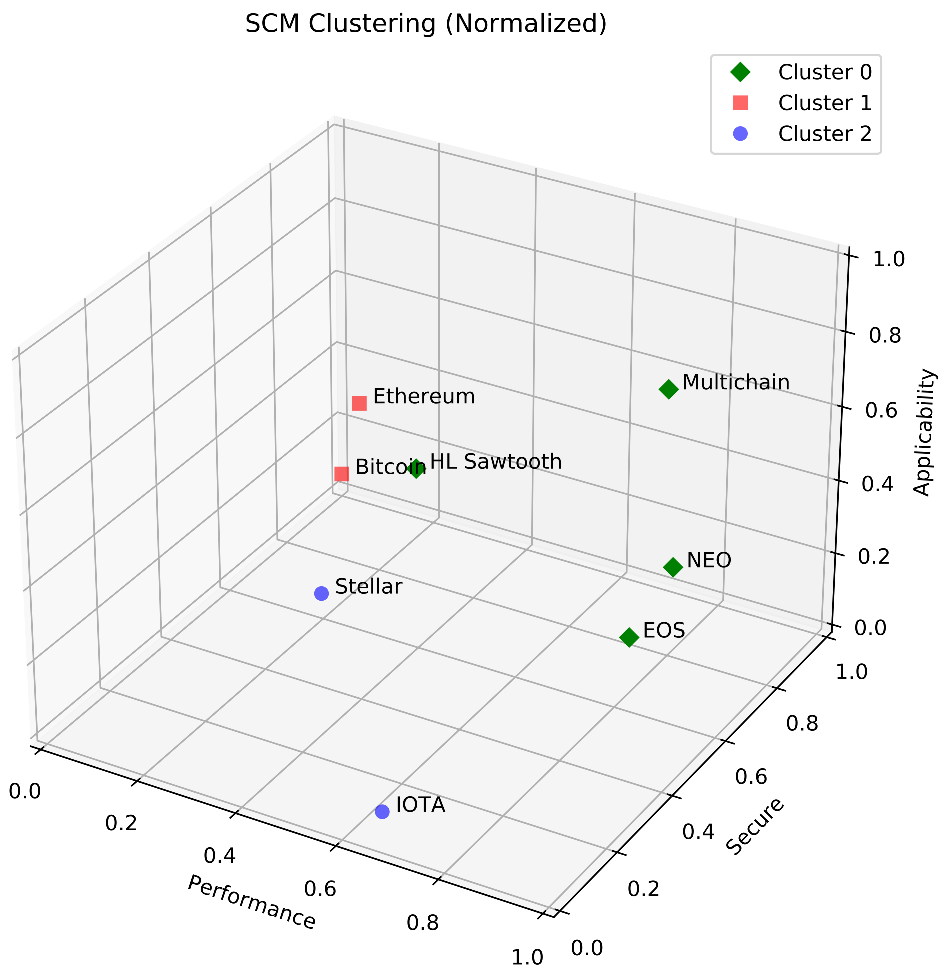


Figure 5.6: SCM k-means clustering results with normalized values. Source: The author.

5.4 Discussion

Based on the clustering results, *cf.* Section 5.3, the proposed SIG was able to successfully capture the fine grained differences between various BC implementations. However, for complexity reasons, the use case evaluations only covered *Mainnet* BC deployments and did not consider *private* BC deployments, *e.g.,* a private Ethereum BC deployment, hence, this would naturally influence the resulting scores in various dimensions. Further, even though the graph spans 74 nodes and 88 edges, for complexity reasons, many BC implementation specific characteristics were not covered by the SIG, which in some cases also lead to perhaps surprising results, *e.g., IOTA* not scoring highly in the IoT use case (*cf.* Section 5.1.1), even though IoT can be seen as one of IOTA's main use cases [48].

Withal, the contributions between the edges of the SIG offer additional layers of intricacies, since requirements and BC aspects are regularly related and interdependent in near unfathomable ways. Furthermore, the authors of the quantification extension also pointed out, [1], that the extension's value rises from having numerous *many-to-many* mappings and tradeoffs. Consequently, only by capturing these small, but influental details an exhaustive SIG can be achieved. Essentially, due to complexity constraints, this directs a paramount limitation on the design of the SIG and the ultimate goal would be then to achieve the highest expressivity with the smallest SIG as possible. However, by doing so, while the most important differences between various BCs would be captured, and common, but important attributes would be left out, not a full BC characterization would be obtained. Nonetheless, since this thesis set out to capture the characterization aspect of various BC implementations, the focus was thence laid on this very aspect.

Additionally, there is an inherent difficulty in proposing operationalization impacts and leaf softgoal weights, since the framework does not only not allow for half satisficing values, the quantification of differences is elaborate and occasionally hard to justify. While in some cases linear interpolation could be applied, at other times this was not as straightforward, and the values could be more seen as *arbitrary.* In spite of that, given from the clustering results, the relative differences between the values are more important than the values per se, and the SIG was still able to capture the aforementioned characteristics.

Part of these limitations can be attributed to the fact, that the NFR framework was devised to be mainly used during the design stage of an application to, inter alia, facilitate the selection of design decisions (*cf.* Section 2.2), whereas it was applied in this thesis to already realized BC implementations. Consequently, the Softgoal Scores were the main focus, and operationalization scores and attainment were not further used. Nonetheless, based on the BC characterization and classifications that resulted from the use of SIGs, its BC appliance can be seen as successful.

Chapter 6

Summary and Future Work

The recent emergence of a great multitude and variety of blockchains in conjunction with no blockchain standardization body being prevalent led to a research gap concerning the classification and characterization of blockchains, further hinting at complicacies in selecting a suitable blockchain for an application.

Therefore, in this thesis inspiration has been taken from Software Engineering principles, in particular, the Non-Functional Requirements Framework was exercised and applied to blockchains, and accordingly, a multitude of blockchain characteristics were examined and collected to design and decompose a specific blockchain Softgoal Interdependency Graph. This graph decomposition process led to an incorporation of a total of 74 Nodes and 88 Edges. By further applying a framework extension on the Softgoal Interdependency Graph that permitted the quantification of values, specific and timely blockchain use cases (*e.g.,* Internet of Things (IoT) and Supply Chain Management (SCM)) were researched and then evaluated through Machine Learning, resulting in a high-level blockchain classification.

Simultaneously, an application in a client server architecture was designed and established to allow for a convenient visualization of the graph and the calculation of scores. Further, the application was also devised to grant its users the ability to select design choices in the form of operationalizations, to evaluate distinct blockchain implementations, and to enter their own graph to process and to visualize.

Future work should include the expansion of the Softgoal Interdependency Graph to add more blockchain aspects, and to focus further on considering and researching more interdependencies and impacts between the nodes of the graph. Besides, through the quantification of values, the calculated scores data offers the opportunity to further explore Machine Learning techniques for various use cases, inter alia, to also approach the blockchain selection use case.

Bibliography

[1] Amy Affleck and Aneesh Krishna. Supporting Quantitative Reasoning of Non-Functional Requirements: A Process-Oriented Approach. In *2012 International Conference on Software and System Process (ICSSP)*, pages 88–92. IEEE, 2012.

[2] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain.* O'Reilly Media, Inc., 2017.

[3] Axios. Axios Github Repository, 2021. `https://github.com/axios/axios`, Last visit March 1, 2021.

[4] Mark C. Ballandies, Marcus M. Dapp, and Evangelos Pournaras. Decrypting Distributed Ledger Design - Taxonomy, Classification and Blockchain Community Evaluation, 2018.

[5] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to Better-How to Make Bitcoin a Better Currency. In *International Conference on Financial Cryptography and Data Security (FC12)*, pages 399–414, Bonaire, 2012. Springer.

[6] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *arXiv preprint arXiv:2005.14282*, 2020.

[7] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. A Vademecum on Blockchain Technologies: When, Which, and How. *IEEE Communications Surveys & Tutorials*, 21(4):3796–3838, 2019.

[8] Gregor Blossey, Jannick Eisenhardt, and Gerd Hahn. Blockchain Technology in Supply Chain Management: An Application Perspective. In *Proceedings of the 52nd Hawaii international conference on system sciences*, 2019.

[9] Lars Brünjes and Murdoch J. Gabbay. UTxO-vs account-based smart contract blockchain programming paradigms. In *International Symposium on Leveraging Applications of Formal Methods*, pages 73–88. Springer, 2020.

[10] Jacob Buck. React-beforeunload Github Repository, 2021. `https://github.com/jacobbuck/react-beforeunload`, Last visit March 1, 2021.

[11] Nikita Butenko. React-Copy-To-Clipboard Github Repository, 2021. `https://github.com/nkbt/react-copy-to-clipboard`, Last visit March 1, 2021.

[12] Bin Cao, Zhenghui Zhang, Daquan Feng, Shengli Zhang, Lei Zhang, Mugen Peng, and Yun Li. Performance analysis and comparison of PoW, PoS and DAG based blockchains. *Digital Communications and Networks*, 6(4):480–485, 2020.

[13] Rainara M. Carvalho, Rossana M.C. Andrade, Káthia M. Oliveira, and Christophe Kolski. Catalog of invisibility requirements for ubicomp and iot applications. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 88–99. IEEE, 2018.

[14] Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The Extended UTXO Model. In *International Conference on Financial Cryptography and Data Security*, pages 525–539. Springer, 2020.

[15] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On Non-Functional Requirements in Software Engineering. In *Conceptual Modeling: Foundations and Applications*, pages 363–379. Springer, 2009.

[16] Lawrence Chung and Brian A. Nixon. Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In *Proceedings of the 17th International Conference on Software Engineering*, pages 25–37, Seattle, Washington, USA, 1995. IEEE.

[17] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional Requirements in Software Engineering*, volume 5. Springer Science & Business Media, 2000.

[18] CoinMarketCap. Coinmarketcap Market Capitalizations, 2020. `https://coinmarketcap.com/`, Last visit October 25, 2020.

[19] Cristina Criddle. BBC: Bitcoin consumes 'more electricity than Argentina', 2021. `https://www.bbc.com/news/technology-56012952`, Last visit April 7, 2021.

[20] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In *Proceedings of the Second Italian Conference on Cyber Security, Milan, Italy, February 6th - to - 9th, 2018*, volume 2058 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[21] Advait Deshpande, Katherine Stewart, Louise Lepetit, and Salil Gunashekar. Distributed Ledger Technologies/Blockchain: Challenges, Opportunities and the Prospects for Standards. *Overview Report The British Standards Institution (BSI)*, 40:40, 2017.

[22] Ethereum Documentation. Layer 2 scaling, 2020. `https://ethereum.org/en/developers/docs/layer-2-scaling/`, Last visit Febraury 8, 2021.

[23] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.

[24] Martin Glinz. On Non-Functional Requirements. In *IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[25] Isitan Görkey, Chakir El Moussaoui, Vincent Wijdeveld, and Erik Sennema. Comparative Study of Byzantine Fault Tolerant Consensus Algorithms on Permissioned Blockchains, 2020.

[26] Sri Nikhil Gupta Gourisetti, Michael Mylrea, and Hirak Patangia. Evaluation and Demonstration of Blockchain Applicability Framework. *IEEE Transactions on Engineering Management*, 67(4):1142–1156, 2019.

[27] T. Jayalakshmi and A. Santhakumaran. Statistical Normalization and Back Propagation for Classification. *International Journal of Computer Theory and Engineering*, 3(1):1793–8201, 2011.

[28] Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. SoK: A Taxonomy for Layer-2 Scalability Related Protocols for Cryptocurrencies. *IACR Cryptol. ePrint Arch.*, 2019:352, 2019.

[29] Minhaj Ahmad Khan and Khaled Salah. IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[30] Nir Kshetri. Can Blockchain Strengthen the Internet of Things? *IT professional*, 19(4):68–72, 2017.

[31] Yeşem Kurt Peker, Xavier Rodriguez, James Ericsson, Suk Jin Lee, and Alfredo J. Perez. A Cost Analysis of Internet of Things Sensor Data Storage on Blockchain via Smart Contracts. *Electronics*, 9(2):244, 2020.

[32] Kenny L. The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed, 2019. `https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44`, Last visit February 6, 2021.

[33] Roy Lai and David LEE Kuo Chuen. Blockchain–From Public To Private. In *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2*, pages 145–177. Elsevier, 2018.

[34] Nikos Leonardos, Stefanos Leonardos, and Georgios Piliouras. Oceanic Games: Centralization Risks and Incentives in Blockchain Mining. In *Mathematical Research for Blockchain Economy*, pages 183–199. Springer, 2020.

[35] Jian-Hong Lin, Kevin Primicerio, Tiziano Squartini, Christian Decker, and Claudio J. Tessone. Lightning Network: a second path towards centralisation of the Bitcoin economy. *New Journal of Physics*, 22(8):083022, 2020.

[36] Manlu Liu, Kean Wu, and Jennifer Jie Xu. How Will Blockchain Technology Impact Auditing and Accounting: Permissionless versus Permissioned Blockchain. *Current Issues in Auditing*, 13(2):A19–A29, 2019.

[37] James MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[38] Julian Martinez. Understanding Proof of Stake: The Nothing at Stake Theory, 2018. `https://medium.com/coinmonks/understanding-proof-of-stake-the-nothing-at-stake-theory-1f0d71bc027`, Last visit November 21, 2020.

[39] Material-UI Developers. Material-UI, 2021. `https://material-ui.com/`, Last visit March 1, 2021.

[40] Matplotlib Developers. Matplotlib, 2021. `https://matplotlib.org/`, Last visit March 22, 2021.

[41] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 680–691, 2015.

[42] Satoshi Nakamoto. A Peer-to-Peer Electronic Cash System, 2008. `https://bitcoin.org/bitcoin.pdf`, Last visit October 25, 2020.

[43] NetworkX Developers. NetworkX, 2020. `https://networkx.org/`, Last visit March 1, 2021.

[44] Fábio Rilston Silva Paim and Jaelson Castro. Enhancing Data Warehouse Design with the NFR Framework. *WER*, 2:40–57, 2002.

[45] Stephanie Perez. Does a Blockchain Need a Token?, 2017. `https://medium.com/swlh/does-a-blockchain-need-a-token-66c894d566fb`, Last visit February 23, 2021.

[46] POANetwork. Proof of Authority: consensus model with Identity at Stake, 2017. https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256, Last visit February 1, 2021.

[47] Jorge Pena Queralta and Tomi Westerlund. Blockchain for mobile edge computing: Consensus mechanisms and scalability. *arXiv preprint arXiv:2006.07578*, 2020.

[48] Daniel Ramos and Gabriel Zanko. Review of iota foundation as a moving force for massive blockchain adoption in different industry sectors, 2021. https://static1.squarespace.com/static/5d9d1a861b3b6f17755e5151/t/5eee80107be4e71d5400d32a/1592688657686/IOTA_SciPaper.pdf, Last visit April 9, 2021.

[49] Shaan Ray. Difference Between Traditional and Delegated Proof of Stake, 2018. https://bit.ly/2JWIn3f, Last visit November 21, 2020.

[50] Elizabeth Reilly, Matthew Maloney, Michael Siegel, and Gregory Falco. A Smart City IoT Integrity-First Communication Protocol via an Ethereum Blockchain Light Client. In *Proceedings of the International Workshop on Software Engineering Research and Practices for the Internet of Things (SERP4IoT 2019), Marrakech, Morocco*, pages 15–19, 2019.

[51] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. Exploring the Attack Surface of Blockchain: A Systematic Overview. *arXiv preprint arXiv:1904.03487*, 2019.

[52] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research*, 57(7):2117–2135, 2019.

[53] Sanic Developers. Sanic Framework, 2018. https://sanicframework.org/en/, Last visit March 1, 2021.

[54] Eder J. Scheid, Timo Hegnauer, Bruno Rodrigues, and Burkhard Stiller. Bifröst: a Modular Blockchain Interoperability API. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, pages 332–339. IEEE, 2019.

[55] Eder J. Scheid, Daniel Lakic, Bruno B. Rodrigues, and Burkhard Stiller. PleBeuS: a Policy-based Blockchain Selection Framework. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2020)*, pages 1–8, Budapest, Hungary, 2020. IEEE.

[56] Eder J. Scheid, Cristian C. Machado, Muriel F. Franco, Ricardo L. dos Santos, Ricardo P. Pfitscher, Alberto E. Schaeffer-Filho, and Lisandro Z. Granville. INSpIRE: Integrated NFV-based Intent Refinement Environment. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017)*, pages 186–194, Lisbon, Portugal, 2017. IEEE.

[57] Scikit-Learn Developers. Scikit-Learn, 2020. https://scikit-learn.org/, Last visit March 22, 2021.

[58] Scikit-Learn Developers. Scikit-Learn Cluster Documentation, 2020. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans, Last visit March 22, 2021.

[59] Seaborn Developers. Seaborn Statistical Data Visualization, 2020. https://seaborn.pydata.org/, Last visit March 22, 2021.

[60] Seba Bank. A Beginner's Guide to Blockchain Accounting Standards, 2020. https://www.seba.swiss/research/A-Beginner-s-Guide-to-Blockchain-Accounting-Standards, Last visit Febru-

ary 8, 2021.

[61] Julien Soler, Fabien Tencé, Laurent Gaubert, and Cédric Buche. Data Clustering and Similarity. In *The Twenty-Sixth International FLAIRS Conference*. Citeseer, 2013.

[62] Ashley Sommer. Sanic-CORS Github Repository, 2021. `https://github.com/ashleysommer/sanic-cors`, Last visit March 1, 2021.

[63] Mathis Steichen, Beltran Fiz, Robert Norvill, Wazen Shbair, and Radu State. Blockchain-Based, Decentralized Access Control for IPFS. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1499–1506. IEEE, 2018.

[64] Nary Subramanian, Steven Drager, and William McKeever. Recovering Software Design from Interviews Using the NFR Approach: An Experience Report. *Advances in Software Engineering*, 2014, 2014.

[65] Flora Sun. UTXO vs Account/Balance Model, 2018. `https://medium.com/@sunflora98/utxo-vs-account-balance-model-5e6470f4e0cf`, Last visit Febraury 8, 2021.

[66] M.A. Syakur, B.K. Khotimah, E.M.S. Rochman, and B.D. Satoto. Integration K-MEANS Clustering Method and Elbow Method For Identification of The Best Customer Profile Cluster. In *IOP Conference Series: Materials Science and Engineering*, volume 336. IOP Publishing, 2018. Art. no. 012017.

[67] Paolo Tasca and Claudio J. Tessone. Taxonomy of Blockchain Technologies. Principles of Identification and Classification. *arXiv preprint arXiv:1708.04872*, 2017.

[68] Sardan Thibaut. What is a light client and why you should care?, 2018. `https://www.parity.io/what-is-a-light-client/`, Last visit February 22, 2021.

[69] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.

[70] Marko Vukolić. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

[71] Mark Walport et al. Distributed Ledger Technology: Beyond Blockchain. *UK Government Office for Science*, 1:1–88, 2016.

[72] Qin Wang, Jiangshan Yu, Zhiniang Peng, Van Cuong Bui, Shiping Chen, Yong Ding, and Yang Xiang. Security Analysis on dBFT Protocol of NEO. In *International Conference on Financial Cryptography and Data Security*, pages 20–31. Springer, 2020.

[73] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access*, 7:22328–22370, 2019.

[74] Wikipedia. Linear Interpolation, 2021. `https://en.wikipedia.org/wiki/Linear_interpolation`, Last visit March 6, 2021.

[75] Wikipedia. Topological sorting, 2021. `https://en.wikipedia.org/wiki/Topological_sorting`, Last visit March 3 2021.

[76] Karl Wüst and Arthur Gervais. Do you need a Blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[77] Sharath Yaji, Kajal Bangera, and B. Neelima. Privacy Preserving in Blockchain ased on Partial Homomorphic Encryption System for AI Applications. In *2018 IEEE 25th*

International Conference on High Performance Computing Workshops (HiPCW), pages 81–85. IEEE, 2018.

[78] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321, 1997.

[79] Rui Zhang, Rui Xue, and Ling Liu. Security and Privacy on Blockchain. *ACM Computing Surveys (CSUR)*, 52(3):1–34, 2019.

[80] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

Abbreviations

BAF	Blockchain Applicability Framework
BC	Blockchain
BFT	Byzantine Fault Tolerance
CED	Cryptoeconomic Design
DAG	Directed Acyclic Graph
dBFT	delegated Byzantine Fault Tolerance
DDoS	Distributed Denial-of-Service
DL	Distributed Ledger
DLT	Distributed Ledger Technology
dPoS	delegated Proof-of-Stake
ECDSA	Elliptic Curve Digital Signature Algorithm
FR	Functional Requirements
IoT	Internet of Things
IPFS	InterPlanetary Filesystem
LSG	Leaf Softgoal
NIZK	Non-Interactive Zero Knowledge Proof
NFR	Non-Functional Requirements
OP	Operationalization
PoA	Proof-of-Authority
PoB	Proof-of-Burn
PoC	Proof-of-Capacity
PoS	Proof-of-Stake
PoW	Proof-of-Work
SC	Smart Contract
SCM	Supply Chain Management
SE	Software Engineering
SG	Softgoal
SIG	Softgoal Interdependency Graphs
tps	Transactions per second
TTP	Trusted Third Party
UI	User Interface
VNF	Virtualized Network Function

List of Figures

2.1 Taxonomy of requirements: NFR are attributes or constraints on system requirements [24]. 4

2.2 A Softgoal Interdepdency Graph using the original NFR framework [44]. . 6

2.3 The SIG for the bank example [1]. 8

3.1 Distributed Ledger Taxonomy [4]. 18

3.2 Blockchain Taxonomy Tree [67]. 19

3.3 The 9 possible BAF outcomes [26]. 19

4.1 Complete SIG. Source: The author. 24

4.2 SG Performance. Source: The author. 26

4.3 SG Secure. Source: The author. 28

4.4 SG Privacy. Source: The author. 32

4.5 SG Applicability. Source: The author. 34

4.6 The sequence between the components. Source: The author. 47

4.7 The user interface of the application. Source: The author. 48

4.8 The 'Custom' UI View which allows the user to enter a custom SIG as a JSON. Based on the information, the system calculates the SIG scores. Source: The author. 49

5.1 Comparison of the average computation times for the calculation of the *OP*, *LSG* and *SG* scores in ms *(n=10)*. Source: The author. 58

5.2 Computing the best amount of clusters for the k-means clustering (a) IoT Use Case and (b) SCM Use Case. Source: The author. 59

5.3 IoT: Clustered Pair Plots (without Normalization). Source: The author. . . 60

5.4 IoT Performance/Privacy Clustering Results (normalized). Source: The author. 61

5.5 IoT k-means clustering results with normalized values. Source: The author. 62

5.6 SCM k-means clustering results with normalized values. Source: The author. 64

List of Tables

2.1 Contributions for SIG [1]. 7

3.1 Controls regarding Transaction Constraints, F/L refer to the states Fully or Partially Applicable and P/N refer to Partially or Not Applicable [26]. The entries in the table are abbreviated in the following way: Blockchain (Y), No Blockchain (N), Permissioned/Private (V), Permissionless/Public (U), PoW (W), PoS (S), Proof-of-Burn *(PoB)* (B), and PoA (A) [26]. . . . 20

4.1 Performance: Operationalization impacts on parent LSG or OP. 37

4.2 Performance: LSG Weights 37

4.3 Performance: Contributions 37

4.4 Secure: Operationalization impacts on parent LSG or OP. 39

4.5 Secure: LSG Weights 39

4.6 Secure: Contributions 40

4.7 Privacy: Operationalization impacts on parent LSG or OP. 41

4.8 Privacy: LSG Weights 41

4.9 Privacy: Contributions 41

4.10 Applicability: Operationalization impacts on parent LSG or OP. 42

4.11 Applicability: LSG Weights 42

4.12 Applicability: Contributions 43

4.13 Computed Operationalization Scores 44

4.14 Computed LSG Scores 45

4.15 Computed SG Scores 46

5.1 Evaluated IoT SG Scores. The scores are capped between [-1, 1] [1]. 56

5.2 Evaluated SCM SG Scores. The scores are capped between [-1, 1] [1]. . . . 57

5.3 Computed Mutual Information between SGs and Cluster Label. 59

A.1 IoT use case. LSG weights on parent LSG. 82

A.2 IoT use case. Selected Operationalization impacts on parent LSG or OP. . 83

B.1 Supply Chain Management use case. LSG weights on parent LSG. 86

B.2 Supply Chain Management use case. Selected Operationalization impacts on parent LSG or OP. 87

Appendix A

IoT Use Case Calculations

Table A.1: IoT use case. LSG weights on parent LSG.

LSG	**Weight**
Blockchain Data Visibility	1.0
Blockchain Data Writability	1.0
Fees	0.5
Token	0.25
Interoperability	1.0
Smart Contract Features	1.0
Smart Contract Functionality	1.0
Access	1.0
Data Privacy	1.0
Unlinkability of Transactions	0.0
Pseudonymity	0.25
Amount of Nodes	0.25
Degree of Centralization	1.0
Validity of Transactions	1.0
Security Level	1.0
Consensus Protocols	1.0
Confirmations in the Network	0.25
Low Stale Block Rate	0.25
Resistance Against Tampering of Old Transactions	0.5
Resistance Against Tampering of New Transactions	0.5
Layer 1	1.0
Layer 2	1.0
Parallel Transactions	0.25

Table A.2: IoT use case. Selected Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
Private	Blockchain Data Visibility	0.75
Public	Blockchain Data Visibility	0.25
Permissionless	Blockchain Data Writability	0.25
Permissioned	Blockchain Data Writability	0.75
No Fees	Fees	0.50
Token Usage	Token	0.25
Interoperability Possibilities	Interoperability	1.0
Interoperability Possibilities	Layer 2 Solutions	1.0
Data Storage Possibilities	Smart Contract Features	1.0
Turing Completeness	Smart Contract Features	0.75
Smart Contracts	Smart Contract Functionality	1.0
Availability of Light Clients	Access	0.75
Non-Interactive Zero Knowledge Proof	Data Privacy	1.0
Homomorphic Encryption	Data Privacy	1.0
Anonymous Signatures	Unlinkability of Transactions	0.0
Mixing/Tumbler	Unlinkability of Transactions	0.0
Use of Public Keys as Pseudonyms	Pseudonymity	0.25
High Amount of Nodes	Amount of Nodes	1.0
Centralized Entity	Degree of Centralization	1.0
Use of Digital Signatures	Validity of Transactions	0.5
Use of Digital Signatures	Anonymous Signatures	1.0
dPoS	Security Level	0.25
PoW	Security Level	1.0
PoS	Security Level	0.25
PoA	Security Level	0.75
dBFT	Security Level	0.25
dPoS	Consensus Protocols	1.0
PoW	Consensus Protocols	1.0
PoS	Consensus Protocols	1.0
PoA	Consensus Protocols	1.0
dBFT	Consensus Protocols	1.0
dPoS	Layer 1	0.75
PoW	Layer 1	-1.0
PoS	Layer 1	0.25
PoA	Layer 1	0.25
dBFT	Layer 1	1.0
High Number of Confirmations in the Network	Confirmations in the Network	1.0
High Number of Confirmations in the Network	Layer 1	-0.5
Fast Block Interval Time	Layer 1	0.5
Large Block Size	Layer 1	0.5
Fast Block Interval Time	Low Stale Block Rate	-0.25
Large Block Size	Low Stale Block Rate	-0.25
Use of Secure/Collision-Resistent Hash Functions	Resistance Against Tampering of New Transactions	0.75
Use of Secure/Collision-Resistent Hash Functions	Hash Chained Storage	1.0
Hash Chained Storage	Resistance Against Tampering of Old Transactions	0.75
Layer 2 Solutions	Layer 2	0.5
Account Model	Parallel Transactions	0
UTXO	Parallel Transactions	0.25

Appendix B

Supply Chain Management Use Case Calculations

Table B.1: Supply Chain Management use case. LSG weights on parent LSG.

LSG	**Weight**
Blockchain Data Visibility	1.0
Blockchain Data Writability	1.0
Fees	0.25
Token	1.0
Interoperability	0.75
Smart Contract Features	1.0
Smart Contract Functionality	1.0
Access	0.5
Data Privacy	0.0
Unlinkability of Transactions	0.0
Pseudonymity	0.0
Amount of Nodes	0.25
Degree of Centralization	0.5
Validity of Transactions	1.0
Security Level	0.75
Consensus Protocols	1.0
Confirmations in the Network	0.25
Low Stale Block Rate	0.5
Resistance Against Tampering of Old Transactions	1.0
Resistance Against Tampering of New Transactions	1.0
Layer 1	0.25
Layer 2	0.25
Parallel Transactions	0.25

Table B.2: Supply Chain Management use case. Selected Operationalization impacts on parent LSG or OP.

Operationalization	**Parent**	**Impact**
Private	Blockchain Data Visibility	0.75
Public	Blockchain Data Visibility	0.25
Permissionless	Blockchain Data Writability	0.25
Permissioned	Blockchain Data Writability	0.75
No Fees	Fees	0.25
Token Usage	Token	1.0
Interoperability Possibilities	Interoperability	0.75
Interoperability Possibilities	Layer 2 Solutions	1.0
Data Storage Possibilities	Smart Contract Features	1.0
Turing Completeness	Smart Contract Features	1.0
Smart Contracts	Smart Contract Functionality	1.0
Availability of Light Clients	Access	0.5
Non-Interactive Zero Knowledge Proof	Data Privacy	0.0
Homomorphic Encryption	Data Privacy	0.0
Anonymous Signatures	Unlinkability of Transactions	0.0
Mixing/Tumbler	Unlinkability of Transactions	0.0
Use of Public Keys as Pseudonyms	Pseudonymity	0.25
High Amount of Nodes	Amount of Nodes	0.25
Centralized Entity	Degree of Centralization	0.5
Use of Digital Signatures	Validity of Transactions	1.0
Use of Digital Signatures	Anonymous Signatures	1.0
dPoS	Security Level	0.25
PoW	Security Level	1.0
PoS	Security Level	0.25
PoA	Security Level	0.75
dBFT	Security Level	0.25
dPoS	Consensus Protocols	1.0
PoW	Consensus Protocols	1.0
PoS	Consensus Protocols	1.0
PoA	Consensus Protocols	1.0
dBFT	Consensus Protocols	1.0
dPoS	Layer 1	0.75
PoW	Layer 1	-1.0
PoS	Layer 1	0.25
PoA	Layer 1	0.25
dBFT	Layer 1	1.0
High Number of Confirmations in the Network	Confirmations in the Network	1.0
High Number of Confirmations in the Network	Layer 1	-0.5
Fast Block Interval Time	Layer 1	0.25
Large Block Size	Layer 1	0.25
Fast Block Interval Time	Low Stale Block Rate	-0.25
Large Block Size	Low Stale Block Rate	-0.25
Use of Secure/Collision-Resistent Hash Functions	Resistance Against Tampering of New Transactions	1.0
Use of Secure/Collision-Resistent Hash Functions	Hash Chained Storage	1.0
Hash Chained Storage	Resistance Against Tampering of Old Transactions	1.0
Layer 2 Solutions	Layer 2	0.25
Account Model	Parallel Transactions	0.0
UTXO	Parallel Transactions	0.25

Appendix C

Installation Guidelines

Frontend

The frontend can be launched by running the following commands in the project directory:

```
yarn install
yarn build
npm start
```

Additionally, this project's packages require a node.js version of 10, 12 or 14. The version can be configured with the following command, using the node version manager *nvm*[1]:

```
nvm use 10
```

This application has been tested with *Firefox* and *Chrome*, latest versions.

Server

This implementation has been tested with Python 3.8. The author recommends using Anaconda[2] or Miniconda[3] to install the necessary packages and to manage the python environment.

In particular, this stems from the package pygraphviz[4] appearing to be problematic to install on OS X using pip[5] as a package manager, however, no issues were had with pip on a Ubuntu installation. Pygraphviz installation errors will lead to the server not being able to draw the graph visualizations, though it can still calculate the scores.

Additionally, the server runs per default on localhost with the port *8000*. If this should not be the case, this must be changed for the frontend, in the file src/components/config.js., such that the frontend is able to find the server.

[1]https://github.com/nvm-sh/nvm
[2]https://docs.anaconda.com/anaconda/install/
[3]https://docs.conda.io/en/latest/miniconda.html
[4]https://pygraphviz.github.io/documentation/stable/pygraphviz.pdf
[5]https://pip.pypa.io/en/stable/

Installation using Anaconda/Miniconda:

```
conda create -n conda-server-test python=3.8 networkx=2.5 sanic=20.12.1
    sanic-cors=0.10.0.post3 pygraphviz=1.7
```

Proceed: ‘y’

```
conda activate conda-server-test

conda install matplotlib
```

Proceed: ‘y’

```
python3 main.py
```

Installation using pip:

```
pip3 install networkx sanic sanic-cors
sudo apt install graphviz libgraphviz-dev
pip3 install pygraphviz matplotlib
```

Appendix D

Contents of the CD

```
thesis
├── data
│   └── performance_evaluation.xlsx
├── evaluation
│   └── ml_eval.py
└── figures
    ├── background
    │   ├── affleck2012.png
    │   ├── glinz2007non.png
    │   └── paim2002.png
    ├── clustering
    │   ├── eval_iotperfprivacy.pdf
    │   ├── IOT3d.pdf
    │   ├── iotpairplot.pdf
    │   ├── k_iot.pdf
    │   ├── k_sc.pdf
    │   └── SCM3d.pdf
    ├── other
    │   ├── AND.png
    │   ├── OR.png
    │   └── client-server-sequence.png
    ├── related work
    │   ├── ballandies2018.png
    │   ├── gourisetti2019evaluation.png
    │   ├── gourisetti2019outcomes.png
    │   └── tasca2017.png
    ├── sig
    │   ├── sig_applicability.pdf
    │   ├── sig_complete.pdf
    │   ├── sig_performance.pdf
    │   ├── sig_privacy.pdf
    │   └── sig_secure.pdf
    └── ui
```

```
│   ├── ui.png
│   └── ui-custom.png
├── implementation
│   ├── frontend
│   │   ├── public
│   │   ├── src
│   │   │   ├── components
│   │   │   │   ├── App.js
│   │   │   │   ├── config.js
│   │   │   │   ├── defaultOperationalizations.js
│   │   │   │   └── operationalizationToolTips.js
│   │   │   └── index.js
│   │   ├── package.json
│   │   ├── README.md
│   │   ├── yarn.lock
│   │   └── package-lock.json
│   └── server
│       ├── main.py
│       ├── sig.py
│       └── readme.md
├── thesis
│   ├── fabian_kueffer_thesis.pdf
│   └── fabian_kueffer_thesis.zip
└── presentation
    └── midterm_presentation.pdf
```