DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Using Smart Contracts for Digital Services: A Feasibility Study based on Service Level Agreements

Stephan Zumkeller

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Using Smart Contracts for Digital Services: A Feasibility Study based on Service Level Agreements

Nutzung von Smart Contracts für Digitale Services: Eine Machbarkeitsstudie über Service Level Agreements

Author:	Stephan Zumkeller
Supervisor:	Professor Dr. Florian Matthes
Advisors:	Ulrich Gallersdörfer, M.Sc.
	Elena Scepankova, Mag. jur.
Submission Date:	August 16th, 2018

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, August 16th, 2018 Stephan Zumkeller

Abstract

The blockchain can facilitate trustworthy business relations by acting as a digital institution for trust. Its features, the immutable ledger of transactions as well as the distributed storage and execution of code (so-called smart contracts), offer a great potential for diverse ways of applications, e.g., the enforced fulfillment of obligations. In spite of the vast potential of the blockchain and smart contracts, they are on a quest for enriching their scope of utilization.
We regard digital services as a prospective application domain for the blockchain. Smart contracts could solve domain-specific challenges, such as the non-fulfillment of obligations, infeasible enforcement of rights, and missing readiness for future demands of cloud computing as well as the Internet of Things.

To gain knowledge about the use of blockchain in the application domain, we study the feasibility of smart contracts for supporting service level agreements (SLA) of digital services.
In this master's thesis, we conduct design science research by designing, developing and evaluating a prototypical, blockchain-based application that aims at solving challenges of SLAs of digital services.

The results of the evaluation show that the developed prototype is a technically viable blockchain-based application, which fulfills the designed requirements. Experts from the digital service domain recognize capabilities of smart contracts to solve challenges of SLAs, as the prototype automates processes and establishes trust among service partners by acting as an escrow. Simultaneously, experts hold concerns against the blockchain technology that decrease their tendency of employing smart contracts. The prototype's practicability is further impaired by the blockchain's high cost and latency.

We conclude that SLAs of digital services present a viable domain for support by smart contracts. The practicability can be increased by improvements of the blockchain technology and a growing acceptance among service partners.

Contents

Abstract **iii**

1. Introduction **1**
1.1. Motivation 1
1.2. Research Questions 2
1.3. Research Method 3
1.4. Outline 5

2. Fundamentals **6**
2.1. Service Level Agreements 6
2.1.1. Types of SLAs 7
2.1.2. Content of SLAs 7
2.2. Blockchain Technology 9
2.2.1. Definition 9
2.2.2. Cryptography 10
2.2.3. Structure of a Blockchain 11
2.2.4. Consensus and Mining 12
2.2.5. Characteristics of Blockchains 13
2.2.6. Blockchain Implementations 14
2.3. Smart Contracts 15
2.3.1. Definition 15
2.3.2. Technical Concept 16
2.4. Oracles 16
2.4.1. Characteristics 17
2.4.2. Limitations 18
2.4.3. Oracle Services 19
2.5. Decentralized Storage 21
2.5.1. On-chain Storage 21
2.5.2. InterPlanetary File System 22
2.5.3. Swarm 22
2.6. State Channels 22

3. Related Work **24**
3.1. Oracles in Academic Literature 24
3.2. Software Patterns 25
3.3. Smart Contract enhanced SLAs 25

4. Theoretical Considerations **28**
4.1. Application Domain 28
4.1.1. People 28
4.1.2. Organizational Systems 29
4.1.3. Technology 31
4.2. Problem identification 32
4.2.1. General Challenges towards Digital Services 32
4.2.2. Challenges of SLAs 34
4.3. Specification of Design Objectives 37
4.3.1. Motivation for Blockchain Usage 37

5. Development of the Artifact **41**
5.1. Scope of the Prototype 41
5.2. Design of the Prototype 41
5.2.1. Functional Requirements 41
5.2.2. Design Decisions 45
5.2.3. Entities of the Prototype 47
5.3. Implementation of the Prototype 47
5.3.1. Structure of the Smart Contracts 47
5.3.2. Implementation of the Oracle as a State Channel 48
5.3.3. Realization of the Web Frontend 49
5.3.4. Technology Selection 49
5.3.5. Fulfillment of Functional Requirements 50

6. Evaluation of the Artifact **52**
6.1. Design of the Evaluation 52
6.2. Functional Evaluation 53
6.2.1. Findings from a Customer's Perspective 53
6.2.2. Findings from a Provider's Perspective 54
6.3. Technical Evaluation 56
6.4. Critical Review of the Application 56
6.4.1. Quantitative Application Evaluation 56
6.4.2. Qualitative Application Evaluation 58

7. Result **60**
7.1. Discussion 60
7.2. Answers to the Research Questions 62
7.3. Conclusion 64
7.4. Implications and Outlook 65

A. Appendix **67**
A.1. Use Cases 68
A.2. Architecture of the Web Frontend 69
A.3. Screenshots of the Web Frontend 70
A.4. Architecture of the Smart Contracts 73
A.5. Sequence Diagrams 74
A.5.1. UC1: Browse Service Offerings 74
A.5.2. UC2: Order a Service 75
A.5.3. UC3: Handle Service Payment 76
A.5.4. UC6: Terminate Service Contract 77

Acronyms **78**

List of Figures **79**

List of Tables **80**

Bibliography **81**

1. Introduction

1.1. Motivation

The blockchain is a technology on a quest for enriching its scope of application that exceeds its initial creation for cryptocurrencies. During the decade since its inception, there has been numerous research and practical experiments to identify potential use cases for the decentralized ledger of transactions, which enables trustworthy interaction among its users and tamper-proof digital assets [93].
The blockchain achieves these features by combining multiple advances in computer science (e.g., Merkle trees, distributed systems, and cryptography). It forms a unique data structure of cryptographically linked blocks that contain verifiable transactions of digital assets. The network employs a consensus mechanism to agree on a single version of the data, thereby preventing manipulation and double-spending of assets. [75]

Newer blockchain implementations enable smart contracts. A smart contract is program code, that is stored and executed on the blockchain. This code can specify contractual obligations and fulfill them by transferring digital assets (e.g., cryptocurrencies), thus allowing entities to form relationships by deploying code on the blockchain. When specified conditions are met, the obligations can automatically be fulfilled. [10]
Researchers from the field of Information Systems (IS) explore the blockchain's potential of distributed data storage and code execution to alter and support existing business processes, enable new business models and facilitate service provision [93, 111].
The possibility of automated fulfillment of obligations enabled by smart contracts attracts legal scholars to the blockchain technology, who research the legal potential of this code as a supplement of traditional contracts or replacement thereof [62].

We are interested in the combination of these two domains (IS and legal) and research the capability of blockchain technology for Service Level Agreement (SLA) in digital services. We chose this topic as we assume a good fit between smart contracts and SLAs. We examine challenges towards SLA of digital services and employ smart contracts to solve them.

To examine the feasibility of this solution, we apply Design Science Research (DSR) [81] by designing and implementing a prototypical, blockchain-based application. It facilitates contractual fulfillment of obligations of SLA and supports the underlying business processes of an exemplary hosting provider. We then evaluate this prototype based on functional and technical expert interviews and gain knowledge about the applicability of blockchain technology for the domain of SLA in digital services.

1.2. Research Questions

In this thesis, we seek to answer the following four research questions.

RQ1: How can smart contracts support SLAs of digital services?

To find possibilities for support, we conduct a literature review on general issues with digital services, analyze challenges of SLAs and examine the capabilities of smart contracts. Based on this fundamental knowledge, we identify possible ways of smart contracts to enable SLAs.

RQ2: How can required information about service performance be made available to smart contracts?

The execution of SLAs relies on information about the service performance, which needs to be transferred to smart contracts on a blockchain. It is an enclosed network that requires particular applications (so-called oracles) to access external information. We perform a literature review on the current state of transferring data to a blockchain, determine the characteristics of oracles, and analyze existing oracles as guidance for our implementation.

RQ3: What are approaches for the design and development of a blockchain-based application which supports SLAs of digital services?

The innovative technological concepts of the blockchain (esp., smart contracts, oracles, state channels) require different approaches to software design and development than established technologies. Its distributed nature demands suitable architectures, tools, and technologies.
We examine the status quo of design and development approaches based on available academic literature and publications of the blockchain community. We apply the gathered information to design and develop a prototypical, blockchain-based application to support SLAs of digital services.

RQ4: How feasible is the prototypical application for supporting SLA of digital services?

We perform a functional and technical evaluation by interviewing experts to evaluate the practicability and technical implementation of the blockchain-based application. Additionally, we critically review the application by measuring system parameters (e.g., latency, cost of use) and identifying limitations of the application.

1.3. Research Method

DSR seeks to answer questions of a problem domain by generating knowledge from the creation and evaluation of an innovative artifact that provides a possible solution [45, 81]. In IS research, typical artifacts are models, methods, and prototypical systems [45]. This thesis follows the Design Science Research Methodology (DSRM) of Peffers et al. [81] (see Figure 1.1). Its nominal process sequence consists of six process steps, beginning with the problem identification and motivation followed by an iterative process of objective definition, design and development, demonstration, evaluation, and communication. The methodology can be entered from different research points. The applied DSRM process of this thesis started with the identification of digital service SLA challenges and their importance, followed by the definition of solution objectives. We iterate three times through the design, development, demonstration and evaluation stages to conceive a prototypical, blockchain-based application and communicate our findings in this thesis.

Problem Identification and Motivation

The activities in this process step overlap with the ones describes for the first research question. We present the problem domain of this DSR and conduct a literature review to identify and motivate problems of digital services and of their SLAs.

Objective of the Solution

We examine the capabilities of smart contracts in this process step and thus perform the other activities of the first research question. Based on this examination, we define the objectives of a solution to the previously identified problems.

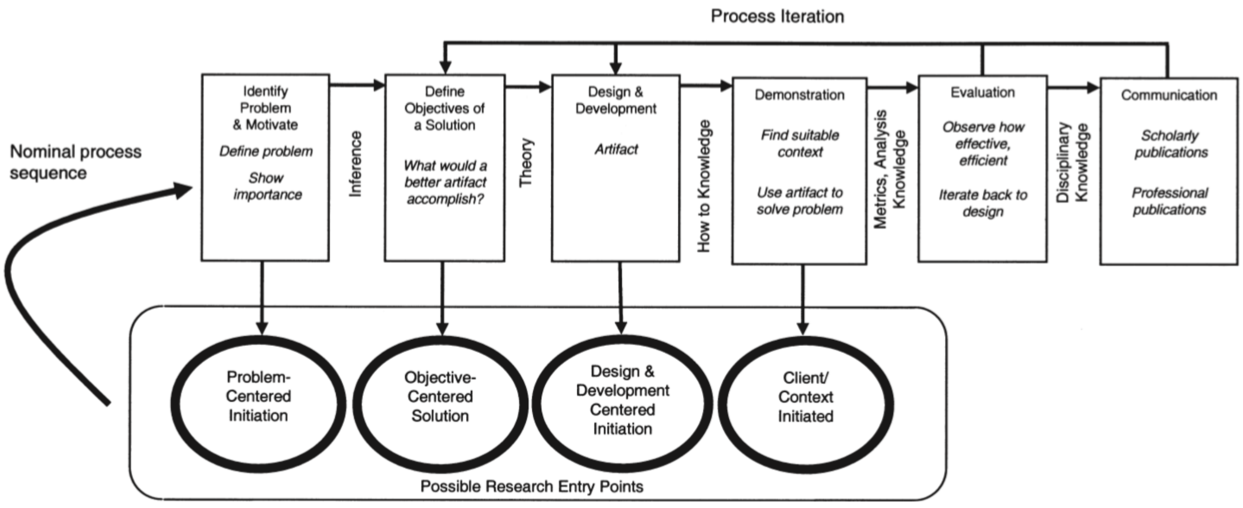

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of a Solution
What would a better artifact accomplish?
Theory
Design & Development
Artifact
How to Knowledge
Demonstration
Find suitable context
Use artifact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points

Figure 1.1.: DSRM process by Peffers et al. [81]

Design and Development

We transfer the objectives of the solution into the application's desired functionality by formulating use cases. This process step relates to the activities of the second and third research question, as we obtain guidance for the design and development of the application from a literature review on blockchain technology. We develop the application and provide the underlying code, Unified Modeling Language (UML) diagrams (class, sequence, components) as documentation for the design and development process.

Demonstration

To demonstrate the developed artifact, we define a simulated environment that exhibits the previously identified problems and apply the artifact to solve those problems.

Evaluation

This process step correlates with the fourth research question, as we evaluate the artifact with expert interviews regarding its functionality and technical implementation. Additionally, we critically review the artifact in two dimensions. We describe its limitations in an qualitative evaluation and quantitatively measure parameters of the system. The evaluations result in two iterations that restart the process at the design and development stage. The first iteration aims at reducing the amount and thus cost of transactions and the second iteration aims at introducing a state channel as a trustworthy oracle.

Communication

We summarize the results of this DSRM process in this thesis.

1.4. Outline

The overall structure of this thesis consists of seven chapters, including this introductory chapter. We build the knowledge base in chapter 2 and introduce the necessary concepts of this thesis, such as SLAs, the blockchain, and its corresponding technologies. Chapter 3 gives an overview of related work with the focus on oracles, patterns for smart contract development and publications regarding the implementation of blockchain technology for SLAs. We state our theoretical considerations in chapter 4 and present the application domain of this design science research, its problems, and we specify design objectives for the artifact. In chapter 5, we summarize the design and implementation process and present the prototypical application. Chapter 6 concerns its evaluation and our critical review of the implementation. We discuss our findings, present answers to the research questions, conclude this thesis and give an outlook in chapter 7.

2. Fundamentals

This chapter introduces the important concepts of this thesis with introductions to SLAs, the blockchain technology, smart contracts, oracles, distributed storage and state channels. It appends the knowledge base of this research. To gain knowledge, we prefer the study of academic literature where possible, yet also refer to websites, discussion boards, and blogs, as the blockchain community shares its knowledge on these mediums.

2.1. Service Level Agreements

Originating from Information Technology (IT) practice, SLAs are a widely used tool in multiple service sectors (e.g., logistics [94], customer care [91], facility management [8], and healthcare [3]). They are agreements between a service provider and its customer on the content and quality of the provided service and are usually supplementing a service contract [5]. SLAs assist the co-creation of services by declaring roles, responsibilities, and expectations [50]. Service Level Objective (SLO) define objectives for the condition of a service [52], by declaring expressions about SLA parameters as well as their validity period and evaluation characteristics. These parameters are Quality of Service (QoS) metrics, also named Service Level Indicators SLI, that express the quality of a service. This could be the availability of a web hosting service or a call center's count of served customers per hour [91]. Action guarantees describe a party's obligation based on preconditions [65], which connect to SLOs. To give an example, an action guarantee defines that a provider is obliged to reimburse 15% of the service fee when the availability is below 99.9%. Thus, the availability is a QoS metric and the expression of "availability below 99.9%" describes an SLO. In this thesis, the term SLA refers to SLA in digital services. We define digital services in this thesis as services, which are performed and delivered digitally over the internet as well as require little human involvement. We primarily focus on hosting services, such as web application or server hosting services.

2.1.1. Types of SLAs

Depending on the contracting parties, one differentiates between three types of SLAs [5].

External SLAs are agreements between two independent parties, such as two independent companies or private individuals.

Group SLAs are concluded between two companies that are part of the same corporate group. While being legally independent, both companies are economically related.

Internal SLAs constitute agreements between entities of the same corporation. These SLAs are mostly used for internal controlling and coordination without the priority of being legally enforceable.

2.1.2. Content of SLAs

SLAs consist of multiple provisions that belong to four categories: general, legal, service, and management (see Figure 2.1). The provisions of an SLA differ depending on the use case and the negotiation result of the contract parties. We present the elements of each type of SLA provision in the following. The description of the content is based on the publications of [5, 79, 80, 91, 102, 103].

General

The general part of an SLA includes the contracting parties, a service provider, and a service customer. A preface describes the purpose of the SLA and the goals, which the parties aim to achieve with the SLA. Regarding the context of the SLA, its scope, as well as regional and organizational limits, are defined. The effective date and duration of the agreement are set in the general section which also contains document related information such as version number and authors.

Management

The section of an SLA concerning its management describes procedures for a variety of actions. For the evaluation of service level compliance, the frequency, content, and form of reports are defined. There can be different types of reports, such as frequent reports with up-to-date information, reports with aggregated information which summarize service quality over a more extended period and reports that are automatically created on incidents. The SLA contains a section that describes the reporting needs of the customer and duties of the provider in creating these reports. In the case of service level violations, the parties agree on an escalation procedure and

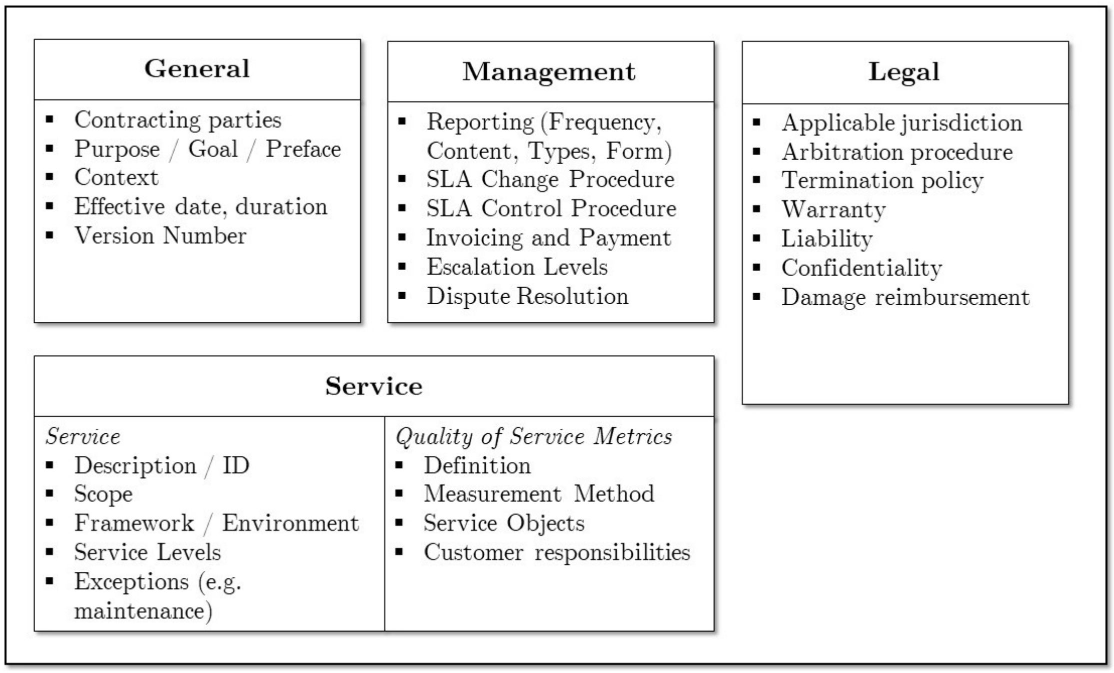


Figure 2.1.: Content of Service Level Agreements [5, 79, 80, 91, 102, 103]

define consequences such as penalty payments. Bonuses for over-performance can also be defined in this section. To efficiently handle disputes, the parties might include a resolution process in the SLA that declares valid dispute types and instances for their resolution. The management section also contains a procedure for SLA control with provisions that regulate audits of the SLA to verify its effectiveness. This enables a reaction to advances in technology or organizational transformations and contributes to the long-term success of the SLA. Possible changes in the context of the SLA make it necessary to include a change procedure that enables the parties to adjust the SLA to new circumstances. The management section includes descriptions of the invoicing and payment modalities.

Service

The central part of an SLA describes the underlying services with their content, SLOs, SLIs and action guarantees. To specify the content of a service, the SLA contains a description, lists required sub-services for its execution in addition to informing about how the service is performed and consumed. The service's scope is defined, and the context of performance is specified by including necessary technical infrastructure, consumer obligations and basic conditions, such as the language for communication. For specifying service quality, metrics for its quantification need to be agreed upon. Those metrics should indicate achievement concerning the customer's goals. It is possible to define multiple metrics that indicate the quality of one service, such as latency and availability to measure the performance of a web hosting service. Each definition of an SLI is appended with a measurement method that specifies its type, location, time and the responsible entity. This is necessary as the QoS value might depend on the type, location and time of its measurement. Both parties need to agree on meaningful QoS metrics and their measurement methods.

After the definition of QoS metrics, the core of the SLA contains SLOs with expressions for these SLI to classify the expected quality of the service. The parties might include exceptions to those SLOs, for example in case of regular maintenance or events that the provider cannot account for.

The cost of the service often relates to the agreed upon SLOs. The service costs are set according to the desired SLO and penalty, or bonus payments are specified to accommodate for variations in the performed service quality.

Legal

Since SLAs are mostly legally binding agreements, they contain sections with legal clauses. In those, the parties specify the applicable jurisdiction, arbitration procedures in case of disputes and termination policies. They also describe confidentiality agreements, warranty and liability clauses as well as provisions for damage reimbursement.

2.2. Blockchain Technology

The goal of this section is the introduction of the blockchain technology. We first define the term blockchain before introducing two essential cryptographic concepts: hash functions and public-key cryptography. These concepts are relevant for the subsequent conception of a blockchain's core processes, namely mining and consensus building. We proceed with an overview of different blockchain technologies, the properties of blockchains as well as potential applications.

2.2.1. Definition

A blockchain is a distributed data structure of a chronological, linked sequence of blocks which contain transactions that are verifiable with public-key cryptography [76, 78, 93]. Participants of the peer-to-peer network validate new transactions, bundle these in a new block and employ a consensus mechanism to append the block to the linked sequence. This chaining of blocks prevents modifications to existing ones and results in an immutable distributed ledger of transactions. As there is no central authority, the network agrees on a set of rules for its governance. These regulate the consensus mechanism, the process for appending new blocks, the validity of transactions and how the network and rules can adapt over time [76]. These rules, the distributed nature of the ledger and cryptographic security allow the blockchain to replace traditional institutions (e.g., banks) by constituting a technical institution on its own.

Similar to traditional institutions, blockchains exist in multiple forms. One differentiates between permissioned and permissionless blockchains [112, 117] as well as private versus public ones [59, 112].

The first property determines the governing entities of a blockchain. Permissionless blockchains (e.g., Bitcoin) allow every participant to validate transactions by appending new blocks and thereby contribute to the network. These blockchains employ consensus mechanisms for appending blocks that ensure a correct state of past activity and prevent manipulation by malicious nodes. [117]
In contrast, permissioned blockchains employ a central authority (e.g., a consortium) that regulates which nodes are allowed in the consensus mechanism to validate transactions by adding blocks and thereby defining the valid state of the chain. While write access is limited, every node is still able to conduct transactions and access the whole data set of the blockchain to retrace the validity of existing blocks. [14]

The second property refers to the access rights to a blockchain's network, as public blockchains allow anyone to join, while private blockchains regulate the access to their networks [117].

2.2.2. Cryptography

Cryptography plays an essential role in its functionality by providing necessary security features [76], especially in signing transactions and chaining blocks. Since the field of cryptography is extending far beyond the requirements of blockchains [26], we focus on introducing hash functions and public-key cryptography as being fundamental to blockchain technology [76].

Hash Functions

Hash functions serve to prove the integrity of data [26]. A hash function $H()$ receives data x as input and transforms its content into the the hash value $H(x)$: the fingerprint of data x. Any modification to data x, regardless of its extent, results in a different fingerprint with $H(x) \neq H(x')$. Thus, the integrity of data is comprehensible by applying the hash function and comparing the resulting fingerprint with the expected one. Narayanan et al. [76] state three requirements of blockchain technology for hash functions: collision-resistance, hiding, and puzzle friendliness. As this exceeds the scope of this thesis, we motivate the interested reader to find definitions and example of these requirements in [76].

Public-Key Cryptography

Public-key cryptography is a method for digital signature. An identity owns a pair of corresponding keys: a privately held signature key and a public verification key. To sign data, one combines it with the signature key in a signing algorithm to receive a digital signature. Anyone is then able to verify the authenticity of data by employing a validation function that receives the data, digital signature, and verification key. When these inputs match, the verification of the signature is positive and proofs the signage of data by the identity that is associated with the public key. [26]

2.2.3. Structure of a Blockchain

The above introduced cryptographic concepts are essential to the data structure of a blockchain, which consists of blocks and therein contained transactions. The following gives an overview of the structural elements of a blockchain, as described by Buterin [10].

Transactions

Transactions are messages with information to change the blockchain's state. Senders cryptographically sign these messages, which contain transfers of cryptocurrency or calls to smart contract functions. When a node initiates a transaction, the node broadcasts the transaction to the entire network to add it to the pool of unprocessed transactions.

Blocks

Mining nodes create blocks. They contain a header, a set of valid transactions from the pool of unprocessed transactions and the most recent state after these transactions were applied. The header includes the block number, the previous block's hash value, the creation timestamp and information for the consensus mechanism. The miner executes the included transactions to migrate the previous block's state to the current state. After creating a block, the miner broadcasts it to the network which then validates and accepts it based on the consensus mechanism.

Blockchain

Cryptographic pointers between blocks create a blockchain. Beginning with the genesis block, which is defined by the founder of a blockchain, each subsequent block includes the hash value of the previous block. These hash pointers result in the fact that the current block implicitly contains information about all previous blocks. This, any manipulation in the blockchain results in a different hash of the current block, virtually making the blockchain tamper-proof. [105]

2.2.4. Consensus and Mining

With the blockchain's ledger of transactions being distributed among the network nodes, they are required to build a consensus on the valid state of the blockchain and its extension with new blocks. Consensus mechanisms are sets of rules to determine which blocks to include in the blockchain.

The most notable one is the *proof-of-work* mechanism, which demands miners to solve cryptographic challenges that require a massive computational effort, yet are easily verifiable. When creating a new block, the miner includes a solution to the challenge and, if the block is incorporated in the blockchain, receives a reward for the conducted effort. The reward consists of transaction fees as well as the gratification for the new block, which constitutes the creation of additional cryptocurrency, thus named mining. The appendix of a block happens when miners accept this block as the most recent one by mining based on it [76]. Since it is possible that multiple, valid blocks are created at the same time and forks of the blockchain develop, the network agrees on the correct version of the blockchain by selecting the longest one, based on the number of blocks weighted by the complexity of their solved challenges. [105]

Other consensus mechanisms include (1) *proof-of-activity*, (2) *proof-of-publication*, as well as (3) *proof-of-stake*, which is a contender to the *proof-of-work* mechanism of the Ethereum blockchain. [105]

The *proof-of-stake* mechanism favors those validating nodes to write the next block, which have a high stake in the success of a blockchain network. The assumption is that nodes with high stakes are motivated to behave correctly not to devalue their own stake. Multiple variants of *proof-of-stake* exist with each defining different stakes or methods, yet regarding a node's cryptocurrency assets is a common notion of a stake. [105] Compared to *proof-of-work* with its cryptographic challenges, the benefits are reduced energy consumption due to missing computational efforts, increased security and less risk of mining centralization [10]. In contrast to these benefits, the mechanism poses some issues: (1) nothing-at-stake leads to validating nodes simultaneously creating (possibly malicious) blocks for competing forks, as mining is effortless; (2) missing transaction finality because validating nodes might secretly build own chains that reverse transactions and (3) the random selection of the validating node, as randomness stems from the blockchain's content and is therefore influenced by the validating nodes [100].

2.2.5. Characteristics of Blockchains

After introducing various vital concepts regarding blockchains, we proceed with the characteristics of blockchain technology. The two main properties of blockchain technology are trust and decentralization. The core of blockchain technology revolves around distributing the ledger of transactions among the network with methods for tamper-proofing and maintaining identical versions on each node. We base the following description of blockchain characteristics on the results of Seebacher and Schüritz [93], that match the publications of [19, 48, 76].

Three features establish the core principle of trust. The first one is *transparency*, as transactions in the network are public, shared with all nodes and thereby simplify the collection of information, such as the whole blockchain or individual blocks and transactions. The *integrity of data* is the second feature and aims at validating this information by employing cryptographic functions for digital signatures and manipulation identification. As multiple nodes perform validations of the information, the integrity of data is further secured. Related to the distributed validation is the application of a consensus mechanism to share a common view of the blockchain's state and tamper-proof the data structure against malicious actors. Thus the *immutability* is the third trust providing feature.

The second core principle of blockchain technology, its decentralized form, generates three main characteristics. Blockchain technology employs public-key cryptography to identify entities in the network, thus providing the public key as a pseudonym and increasing *privacy* for participants. A second aspect of decentralization is *reliability*. Miners provide the whole data structure without interruption by partial network losses, and since the blockchain is a digital technology, it offers the potential of automated actions to remove manual interaction and human error. A third characteristic is the versatility of blockchain technology which allows the integration with other applications to develop new systems.

Both core principles, trust, and decentralization supplement and require each other. Immutability and proof-of-data integrity establish the trust required by participants to interact and form the decentralized network cooperatively. At the same time, the decentralization offers reliability and thereby complements trust by providing information sharing and distributed verification of a blockchains correctness.

2.2.6. Blockchain Implementations

Since the introduction of the Bitcoin blockchain, there has been ongoing development of new blockchain technologies that offer additional features (e.g., smart contracts) or target other audiences (e.g., businesses). We subsequently present the most important implementations of the blockchain principle with their core features.

Bitcoin

Bitcoin [75] is the first blockchain and the foundation of this technology. It is designed as a "peer-to-peer electronic cash systems" to solve the double spending problem of digital currencies and created the Bitcoin cryptocurrency. The public network is open for participation and governed by the rules of its protocol (e.g., proof-of-work). While offering the ability to execute scripts for controlling transactions, Bitcoin does not provide an environment for complex smart contracts.

Ethereum

The Ethereum blockchain [11] was designed to enable smart contracts with a Turing-complete bytecode language that is compiled from high-level languages (e.g., Solidity). These smart contracts are on-chain accounts with the ability to compute, manipulate data and transfer ether, which is the cryptocurrency of Ethereum. The blockchain is open to public participation and relies on the proof-of-work algorithm while transitioning to proof-of-stake. Mining nodes validate new transactions and execute smart contracts during the creation of a new block.

Corda

Corda [85] is a blockchain implementation with smart contracts similar to Ethereum. They differ twofold: (1) Corda employs Java as the high-level language of smart contracts, which run in a Java virtual machine and (2) the blockchain is permissioned instead of public. The goal of Corda is the development of a blockchain following business' needs.

Hyperledger Fabric [14]

Hyperledger Fabric [14] implements a blockchain with smart contracts (Go language). To address business needs, it contains certificate authorities to increase security and provide identity proofs. These certificate authorities allow for identity management and therefore limit the blockchain's participants, resulting in a permissioned blockchain.

2.3. Smart Contracts

With the concept of blockchains introduced in the previous section, we build upon this knowledge by describing the blockchain's feature of smart contracts in the subsequent section, beginning with their definition and followed by their opportunities and challenges.

2.3.1. Definition

While the blockchain is fundamentally defined by a shared consensus of its correct state, the research community and practitioners have yet to agree on a shared definition of smart contracts [16, 22]. We approach a definition by taking into account the historical, technical and legal perceptions of smart contracts.

The first formal mentioning and definition of smart contracts is attributed to Szabo [98], who describes smart contracts as a "set of promises, specified in digital form, including protocols within which the parties perform on these promises" [98].

While this definition mentions the digital form of smart contracts, Szabo [98] did not specify their actual form. Since the development of the Ethereum blockchain, which included smart contracts as first-world citizens, their definitions gained a more technical perspective. Several authors regard smart contracts as software code on a distributed ledger [22, 29, 41, 62, 64, 93, 100, 117] which is executed and computed on the blockchain [57, 64, 69, 95, 111, 117]. Further, the processing of this code is automated [16, 19, 22, 58, 111]. Other authors define smart contracts as distributed state machines [27, 40] or software agents, that control digital assets, fulfill obligations and exercise rights [16, 19].

These technical definitions are complemented by other authors, including some from the legal domain, who apply a more legal view on smart contracts. Similar to their paper-based predecessors, smart contracts formalize elements of a relationship [19, 93, 95, 98, 113] and constitute codified agreements [95]. The agreement is not only enforceable Clack, Bakshi, and Braine, however, automatically executes the included terms [15, 19, 22, 111, 113, 116] without human intervention [116] and thus represents a self-enforcing contract [57, 113, 116]. In contrast to this definition of a self-enforcing legal contract, Cuccuru [19] regard smart contracts not as binding agreements, but rather as means to execute agreements. According to Frantz and Nowostawski [40], it is possible that smart contracts do not include contractual obligations at all.

The bandwidth of definitions ranges from a technical perspective to legal notions. Similar to the form of written text being defined by its content, as lyric verses form a poem and obligations form a contract, the embodiment of programming code is defined by its content as well. Thus, we argue that the inclusion of formalized contractual terms of an agreement is essential to regard code on a distributed ledger as a smart contract. From the first definition by Szabo [98], recent technical developments [10] and the different definitions of smart contracts presented above, we derive the following definition:

> A smart contract is distributively stored and executable code that contains formalized contractual terms to perform agreements of a relationship.

2.3.2. Technical Concept

Following the above definition of a smart contract, we further introduce the technical concept of their implementation on the Ethereum blockchain [10]. Being first-class citizens, smart contracts are contract accounts on the blockchain. Compared to externally owned accounts (controlled by users with private keys), they share the inclusion of a transaction counter and their balance. Contract accounts additionally contain contract code and storage. Validating nodes execute smart contracts in their Ethereum virtual machine (EVM) by processing transactions with a contract account as the recipient. These transactions contain data for the smart contract and more importantly gas, which is required for each computational step of a smart contract. The concept of gas was introduced to limit the code execution since every mining node in the network is processing these smart contracts, which leads to a high computational effort. Thus, when calling a smart contract function, it is necessary to include Ether (Ethereum's native cryptocurrency) in the transaction, which is then spent during the run-time of the smart contract. When the provided gas exhausts during run-time, the computation stops, and the transaction is not processed anymore. A transaction to a smart contract initializes its transition from the previous contract state to the new one, with data being written to the smart contract's storage. During the run-time, the smart contract performs computations according to its Turing-complete language and can interact with other accounts by sending messages (to call other smart contracts) or transfer Ether (to externally owned and contract accounts). The computation and data storage features of smart contracts allow distributed applications on the Ethereum blockchain.

2.4. Oracles

Smart contracts enable complex use cases by acting based on information that is processed within their computational logic. Therefore, the availability of this information is crucial to the potential of smart contracts [43]. This is challenging since smart contracts can only access and write information that is stored on the blockchain [41], which is an enclosed network without direct interfaces to the real world. Oracles bridge the gap between the blockchain and the real-world by feeding data from outside the blockchain to smart contracts. They are used to report events and data after the smart contract has been programmed to allow the smart contract to react to future information. As is the case with regular applications, the usefulness largely depends on the available data and oracles enable smart contracts to query data similar to an API. Common other terms for oracle include *data feed* [29], *data carrier* [77] or *trigger* [111].

2.4.1. Characteristics

While all oracles transfer data across the border of the blockchain network, they can inhibit different characteristics depending on their use case.

Push / Pull Principle

Two principles exist regarding the initiation of a data transfer. With the *pull* principle, an oracle reacts to a query by a smart contract and responds with the required data. This is useful when the point in time for the demand of this data is unknown to the smart contract developer as it allows the smart contract to query data on-demand. When oracles apply the *push* principle, they provide data to the smart contract without on-demand requests. Oracles can periodically send information to the smart contract or notify it after an event happened. Smart contracts utilize oracles with the push principle to react to information as soon as it is available.

Data Source

An important feature of the blockchain is its generation of trust. Since the blockchain immutably stores all transactions, one can trust the absence of manipulation of those transactions. Oracles provide data to smart contracts and thereby influence the computations and the resulting data that is written on the blockchain. Hence, the correctness of the data provided by an oracle is the basis for an accurate state of the blockchain. With use cases such as digital twins, business process automation or automatic value transfers, malicious information on the blockchain can lead to irreversible loss of value. The data source and its trustworthiness is therefore at the core of oracles. We subsequently describe three different types of data sources that offer different levels of trustworthiness.

Single-sourced oracles obtain their information from only one source, which needs to be trusted not to perform maliciously. A single source can, for example, be an information service, such as for weather data or flight data, an IoT device sending sensor data to the blockchain or a person providing information. Stakeholders of a blockchain application that implements a single-sourced oracle need to have a high degree of trust in the source as the provided information can lead to irreversible transactions and potential losses.

Multi-sourced Instead of relying on a single source of information, oracles can query multiple sources for the same data. When a majority of sources reports the same data, the oracle sends this data to the blockchain. The distribution of information retrieval is designed to increase the trustworthiness of data. This agreement on the correct information is similar to blockchain nodes agreeing on the correct state of the blockchain. Multi-sourced oracles are also called consensus-based oracles [7].

Crowd-sourced A particular form of a census-based oracle is a crowd-sourced oracle where human participants input information. A consensus mechanism then uses these inputs to create a single point of data about a particular subject. Crowd-sourced oracles are useful for retrieving information that is not readily available in a digitized form.

Architecture

Similar to regular applications, oracles exist in a centralized or decentralized architecture. Oracles running on a single node are centralized. While requiring fewer resources and complexity, they are prone to targeted attacks and in case of failure cease to function completely. Decentralized oracles [28] meanwhile employ multiple nodes with connections to the blockchain and are therefore able to provide their service even when a single node is not available. The fault-tolerance against outages increases when the multiple nodes of an oracle access different data sources and thereby reduce the dependency on individual data sources.

2.4.2. Limitations

While oracles are the core infrastructure for complex smart contract use cases, several limitations exist. First and foremost, the information delivered by an oracle needs to be correct. The processing of false information in smart contracts is a high risk because the result of the processing can lead to the false on-chain representation of real-world events, thus resulting in losses by transactions that are not reversible. Oracle providers use different methods to establish trustworthiness. One method is by providing their code open source for increasing transparency into the operating mode of the oracle. Another method is an incentive system that motivates oracles to provide correct data

Name	Architecture	Data Source	Special features
Chainlink	Decentralized	Multiple Sources	Reputation system
Eventum	Decentralized	Crowd-sourced	
Oraclize	Centralized	Single Source	TLSNotary proof for data authenticity
RealityKeys	Centralized	Single Source	Supports Bitcoin
SchellingCoin	Decentralized	Crowd-sourced	No off-chain components
Town Crier	Centralized	Single Source	Intel SGX proof for authenticity

Table 2.1.: Overview and characteristics of oracle services

by introducing financial rewards and penalties. A third option is the deployment of technical measures to prove that the data is authentic and has not been manipulated by the oracle [99, 118]. Aside from the oracle, the source of information needs to be trustworthy as well. That is because even if the oracle proofs the authenticity of data, the source is still able to influence smart contract computation by delivering false information. The use of multiple sources decreases the required trust in a single source but does not guarantee the correctness of data as multiple concluding sources can form a census on incorrect information [12]. Kothapalli and Cordi [56] show the vulnerability of multiple sources to bribery attacks that efficiently change the outcome to their preferences.

2.4.3. Oracle Services

In the following, we present multiple oracle services (see Table 2.1). The main characteristics of oracles are their architecture and the source of information. Additional characteristics are technologies for increased security and trustworthiness as well as reputation or reward systems that motivate honest behavior. Common to these oracle services is the pull-principle: they provide data on demand instead of autonomously pushing it to the blockchain.

Chainlink

Chainlink is a decentralized, multi-sourced oracle with nodes that retrieve information from API and forward this data to Chainlink's Ethereum smart contract. These nodes are operated by individuals that provide access to API and are paid for their service with a special LINK token. A smart contract's request for data is handled by multiple nodes that query multiple different sources to decrease the possibility of false information. SLA and a reputation system are incentives for Chainlink Nodes to relay valid information reliably. [28]

Eventum

Eventum is a decentralized, crowd-sourced oracle for the Ethereum blockchain. Users provide information to the Eventum validation nodes, which build a consensus on the valid data. The validation nodes relay this information to the Eventum smart contract and store necessary data on Swarm [35]. Developers who require real-world data in their smart contracts register this request on the Eventum smart contract and add a reward for data providers. Eventum positions itself as a crowd-sourced oracle that is suitable to provide all capturable real-world data. [70]

Oraclize.it

Oraclize.it is a centralized, single-sourced oracle service for web API. Smart contracts query the Oraclize Contract for information and can verify that the returned data is authentically from the source. This authenticity proof rests upon TLSNotary [101], a technique using the TLS protocol to proof that data persisted from interaction with a certain server. [77]

Reality Keys

Reality Keys is a centralized, single-sourced oracle service that queries information from web API. They offer their service to the Ethereum and the Bitcoin blockchain. The provider implemented multiple data sources to enable queries that expect binary Yes or No answers (e.g., Did France win the 2018 World Cup?), with a public-private key par representing each answer. After the creation of such a question, the public keys for its answer options are published. When the answer is available, Reality Keys will publish the private key of either the Yes or No answer, which can be used to sign a multi-sig wallet. In case of disputes about the validity of an API's information, they offer the service of manually checking the data to correct the answer. [88]

SchellingCoin

SchellingCoin is a decentralized, crowd-sourced oracle with users providing the necessary information. SchellingCoin builds a consensus on the information to derive one data point and rewards those users, that contributed to that data. [11]

Town Crier

Town Crier is a centralized, single-sourced oracle system that queries HTTPS-enabled websites. Its main proposition is the serving of authenticated data from trusted websites. By utilizing hardware conclaves (Intel SGX [49]) in their system, Town Crier attests to tamper-free data transfer between smart contract and website. [118]

2.5. Decentralized Storage

The previous sections introduced the concepts of blockchain, smart contracts, and oracles. These technologies cover the protocols for the distributed system, execution of code therein and querying of information. While the blockchain is a database, its distributed characteristics of data redundancy and slow processing times prevent it from being a data storage solution, and smart contracts can offer only basic storage of data. There are currently multiple solutions for decentralized data storage in development to overcome the boundaries of on-chain storage since centralized file servers contradict the nature of a distributed ledger and are therefore not applicable. We present three main storage solutions for the Ethereum blockchain in the following: on-chain storage, IPFS, and Swarm.

2.5.1. On-chain Storage

Storing data accessible to smart contracts on the Ethereum blockchain is possible in two ways: (1) as data included in a transaction or (2) as part of a smart contract.
The first alternative allows to include arbitrary binary data into a transaction, as the technical structure of a transaction includes a data field for smart contract calls. This data field can be independently used for adding data to be stored in the blockchain as part of this transaction. The retrieval of information is complex as it requires the knowledge of the transaction identifier, or multiple ones if the data is distributed among multiple transactions.
The second alternative employs the storage of contract accounts. Compared to data in transactions, smart contracts allow for getter- and setter-functions to ease data access and manipulation while also enabling rights management for data access. Another advantage are variable types that indicate whether data is a string, integer, list or bytecode. Hitchens [46] offers patterns for storage in smart contracts.
The advantage of on-chain storage is its low complexity as data is included on-chain and thus trivially accessible by smart contracts.
The decentralized nature of the blockchain and its data redundancy dictates the limitations, as every node stores a copy of the data structure. Any data stored in transactions

or smart contracts is thus replicated among all participants of the network, leading to huge storage demand. Didil [23] exemplify the resulting cost thereof and calculates, that saving one kilobyte of data to the Ethereum blockchain in October 2017 would have cost about 5 United States Dollars (USD) and one gigabyte amounts to approximately five Million USD.

2.5.2. InterPlanetary File System

A proposed solution to the challenges of on-chain storage is the InterPlanetary File System (IPFS). The peer-to-peer system allows computers to share a common file system that is accessible similar to the current web while being distributed on participating nodes instead of central servers. Files are split into multiple blocks that are identifiable by their fingerprint (i.e., hash value). Each node in the network stores and provides those blocks, that are relevant for it, and offers indexes of its storage to the network. To access a file, IPFS searches its network to find the nodes holding the necessary blocks which then distributively deliver the file similar to BitTorrent technology. Files are stored with their version history (comparable to Git) to provide past versions of the content. In order to point to recent files, IPFS includes a naming system that allows static links to changing content, similar to "https://www.tum.de" leading to the most recent web content. The static links of this changing data can be included in smart contracts to access the distributed data of IPFS in distributively performing smart contracts on the blockchain. [60]

2.5.3. Swarm

Similar to IPFS, the Swarm project aims at developing a distributed storage solution based on peer-to-peer services. Swarm distributes content among its nodes and employs an incentive system, integrated with Ethereum blockchain, for redundant and fault-tolerant content provisioning. The incentives are paid for storage and distribution services and act as a "content availability insurance". The integrated naming systems provide static links to content, which can thereby be referenced in smart contracts or used as a traditional uniform resource identifier. [35]

2.6. State Channels

Transactions and smart contract computations on the blockchain require a fee to process, which reimburses miners for their effort of creating new blocks. Depending on the network's utilized capacity and demanded fees by miners, these costs vary significantly, with for example Bitcoin transactions costs reaching a high of 36 USD in January

2018 and current (July 2018) prices of around 0.19 USD [6]. The current price for Ethereum transactions is 0.11 USD [30]. While these fees do not severely impact single transactions, they become relevant for blockchain-based payment systems and distributed applications, as it is necessary to issue a high volume of transactions in both cases, which would be impaired by the costs. A proposed solution to the high costs of on-chain transactions are state channels [20], also called payment channels, which allow off-chain transactions with the security features of the blockchain (i.e., prevention of double-spending) by employing the same cryptographic functions. These off-chain transactions work similar to banking cheques, which are a signed paper with a value and the reference to a bank.
When two parties, Alice and Bob, agree on using a uni-directional state channel for transactions, they initialize a smart contract with necessary logic for the channel, and Alice transfers a deposit. To send an off-chain transaction, Alice signs the hashed combination of a reference to the smart contract and a value with her private key and sends the signature to Bob. Each time Alice wants to send more value (e.g., due to a subscription to Bob), she increases the value, signs it with the contract reference and sends the signature to Bob. When Bob decides to retrieve the value on-chain, he sends the received signature and his signature of contract reference and value to the smart contract, which then transfers the funds to Bob as both parties signed the value.
These state channels enable distributed applications and payment methods by reducing the number of on-chain transactions. Aside from the above stated smart contract based uni-directional state channel, they also exist in bi-directional and multi-participant forms. Implementations of such state channels are the Raiden network [87] and the Bitcoin Lightening Network [82].

3. Related Work

The goal of this section is to present academic literature that relates to the application of smart contracts in the domain of SLAs. We first introduce academic works regarding the implementation of an oracle, then depict academic contributions of Solidity software patterns and conclude with a review of publications about smart contract enabled service level management.

3.1. Oracles in Academic Literature

Weber et al. [111] utilize the blockchain for business process monitoring and execution. The blockchain stores business logic, execution state, partial data and participant accounts. They regard oracles as triggers that keep track of the executing process as well as its representation on the blockchain. Weber et al. [111] implement these triggers not as oracles that provide a wide variety of information, but as agents of an organization which interact with specific external APIs and send updates to smart contracts in the context of the monitored process. Triggers allow communication in both ways and therefore inhibit the push and the pull principle.
Murkin, Chitchyan, and Byrne [73] follow a similar approach. In their peer-to-peer electricity trading system, oracles receive energy production and consumption amounts from electric meters and push this information to a registry on the blockchain. Households trade electricity by transferring tokens over the registry smart contract. The oracle in this system constitutes an interface for an IoT device (smart meter) to allow their participation in the electricity market.
Another oracle implementation following the push-principle is PriceGeth developed by Eskandari et al. [29]. An off-chain application tracks several exchange prices every second and updates the on-chain smart contract with each new block. Eskandari et al. [29] note that the advantage of near real-time information on the blockchain comes at the cost of gas, rendering this an inefficient oracle.

3.2. Software Patterns

The most common programming language for Ethereum smart contracts is Solidity. As with other programming languages, code snippets are shared on various websites, yet there is a lack of a solid knowledge base with best practices and proven designs. Few publications To generate knowledge about developing Solidity, there are a few publications that address this gap [2, 63, 109, 115].

Liu et al. [63] identified eight design patterns of the categories: creational patterns, structural patterns, inter-behavioral patterns and intra-behavioral patterns. These patterns include amongst others interfaces to mask contract functions, observers to update on data changes, factories to create instances, decorators to enable the upgrade of contracts and a multi-signature pattern to implement co-signage contracts.

Bartoletti and Pompianu [2] describe functional patterns (e.g., tokens, oracles) of smart contracts and did not publish code examples.

Volland [109] employed the grounded theory method to derive a catalog of software patterns from academic literature, existing smart contracts and online communication platforms. The catalog contains 14 patterns in four sections: behavioral, security, upgradeability and economical. Each pattern is thoroughly presented with, amongst others, a motivation, its applicability, a sample, consequences and known usages. The corresponding code repository contains these patterns and their examples [110].

Wöhrer and Zdun [115] recognize the consequences of error-prone smart contracts due to their permanent transfer of digital assets and thus infer the demand of software patterns for Solidity. They contribute six security-related patterns that offer solutions to smart contract vulnerabilities. A corresponding code repository contains code snippets of these security patterns as well as additional patterns for Solidity [114].

3.3. Smart Contract enhanced SLAs

There have been few works on facilitating blockchain technology for SLAs. The following describes related work which either directly implements SLAs or researches collaborative process execution with smart contracts.

Di Pascale et al. [22] research smart contracts to support growing networks of small cell-service providers, as mobile network operators increasingly use these providers to gain access to radio capacities.

To approach the problem of increasing negotiation effort in the growing network, they propose smart contract as a solution that offers automatic contract enforcement, a trustless environment, decreased legal costs as well as quicker and cheaper transactions.

They developed a smart contract that pays the service fee to cell-service providers based on their delivered performance [21]. The smart contract penalizes SLA violations by reducing the withdraw-able service fee. Acting as an oracle, the mobile network operator supplies all relevant information to the smart contract, including the data of providers, SLA terms, and QoS measurements. Di Pascale et al. [22] integrate a database of cell-service providers with QoS-based service fee calculation and payment thereof. Manual interaction by the mobile network operator is still required for the smart contract to perform the calculations and allow providers to access their payments.

While they aim at establishing a trustless environment, the authors provide an implementation where trust is bundled in mobile network operators as they control the QoS monitoring, the information provided to the smart contract and thereby the payout for providers. The implementation thus lacks the goal of balancing the trust between service participants.

Klems et al. [54] developed a prototype decentralized service marketplace with trustless intermediation between co-creators of IT services. They regard the dependency on intermediaries and entry-barriers of centralized marketplaces as challenges to IT services and propose a blockchain-based market that offers service-related processes without a trusted third party. Their motivation for using blockchain stems from its decentralized architecture and the potential to program rules of interaction between service participants into smart contracts, which then execute to enforce these rules.

In their implementation [53], a service registry contract facilitates market activities for match-making of a service provider and consumer while a service contract supports transaction settlement with SLA calculation, payment and refund operations. IPFS stores additional service metadata. Supporting actors monitor the Quality of Service, relay this information to a monitoring contract which notifies the service contract to trigger refunds on SLA violations. To decrease the amount of trust required in supporting actors, Klems et al. [54] chose to pseudo-randomly select the single or multiple agents which monitor a service. The implementation offers a decentralized service marketplace that replaces intermediaries with smart contracts to automatically enforce the rules of a service.

This thesis differs from the publication as we research digital services with the aim of enforcing SLAs and evaluate our DSR artifact with members of the digital service domain.

Weber et al. [111] research the blockchain's potential for collaborative business process monitoring and execution. While they do not specifically implement SLAs, their work is important due to the collaborative conduction of SLA processes. Instead of integrating a third party to build trust between process participants, they utilize the blockchain to control the process flow, create an immutable audit trail and store shared data. This way, neither process party bundles disproportional trust.

Weber et al. [111] developed a translator that uses BPMN diagrams as input to create Solidity smart contracts containing the logic of the process. The proposed system of smart contracts is twofold, as (1) a *choreography monitor* watches messages and stores the process execution state while (2) an *active mediator* is actively engaging in the process execution by performing data transformations as well as sending and receiving messages.

To interact with off-chain systems, Weber et al. [111] implement oracles, called triggers, which relay information, query external API and update the process state of the blockchain. These oracles are interfaces for off-chain systems with the blockchain and are not general oracle solutions that query and API on a smart contracts demand. The limited functionality of these triggers is justified due to their specific use case of supporting a specific process.

The authors demonstrate the usage of blockchain technology to establish trust in collaborative processes with mutually distrustful participants that would otherwise require a trusted third party. They also indicate that such a use case could be limited to public blockchains due to their high latency, cost, and privacy issues.

4. Theoretical Considerations

This chapter first introduces the application domain according to the structure of A. Hevner [44]. Then, we identify and motivate challenges within that domain to accomplish the first step of the DSRM process. In the third section, we define design objectives for the DSR artifact to resolve these challenges, according to the second process step of the applied DSRM.

4.1. Application Domain

The domain of SLAs for digital services consists of multiple actors that interact within organizational systems with the support of technical systems. To narrow down the scope of this thesis, we assume that a service is delivered by a single provider and does not depend on other services or providers. The following description of the application domain draws from [4, 5, 80, 104].

4.1.1. People

The people within this domain belong to two different parties: the provider and the customer. While people within these parties inhibit different roles, responsibilities, and skills, we forgo this detailed description and combine these characteristics on the party level.

Provider

The people belonging to this party cooperate to provide a service. They are typically employees of one company, such as a hosting provider. Their roles mainly are service providing and SLA management. Their capabilities are special skills in providing such a service, e.g., the knowledge and required skills to maintain a data center. Providing the service is part of the provider's business model.

Consumer

The consumer party consists of one or more people, such as a private individual or employees of a company. Consumers lack the knowledge and skills of providing the

service on their own or to the same cost as offered by a provider. The consumer holds multiple roles, such as service usage and SLA management. The economically minded consumer desires low service costs while being willing to pay more for a better service. When a digital service is at the core of a company's business model, they are motivated to pay extra for increased service quality. When consumers lack the knowledge to provide a service on their own, they might be disadvantages in SLA negotiations with a knowledgeable provider.
In the course of this thesis, we interchangeably use the term 'consumer' and 'customer' to increase textual variety.

4.1.2. Organizational Systems

Life cycle of an SLA

There are multiple stages in the life cycle of SLAs. [4, 5, 67, 92] each sort the SLA life cycle activities into multiple stages with different levels of detail, yet their content is similar. A brief overview of the stages and activities according to Berger [5] is given below.

1. Definition During the first phase, the contracting parties identify suitable service characteristics according to customer needs. This leads to the definition of the service, its scope and the development of QoS metrics that measure the fulfillment of customer requirements and goals. The contracting parties negotiate service levels depending on QoS values and set penalties for the violation of service levels. For adherence or over-performance of service levels, the parties might agree on bonuses. At the end of the definition state, an SLA describing a service and its performance metrics have been negotiated and agreed upon.

2. Implementation The second stage is the preparation for the service delivery and consists of organizational, technical and staff activities that lead to the implementation of the service according to the SLA. This includes the allocation of resources, employee training and setting up processes for monitoring and reporting the service quality.

3. Application The consumption of the service is the third phase of the SLA life cycle. While the service is performed and consumed, the main activities are the monitoring of QoS metrics, the evaluation of service level compliance and the processing of violations. Violations of SLOs lead to action guarantees as specified in the SLA. These action guarantees depict obligations of a party. The application of an SLA is a continuous cycle with the goal of controlling the service to ensure its performance according to the SLA.

4. Control The last phase of the SLA life cycle is the evaluation of the effectiveness and goal achievement of the SLA itself. The customer satisfaction is determined, and both parties analyze the economic feasibility of the SLA. This is done to analyze the efficiency and effectiveness of the defined QoS metrics, the assumptions made in the definition phase and the impact of unforeseen changes in the environment of the SLA. The control phase marks the end of the SLA term and could lead to an extension of the SLA, an extension with changes or the termination of the business relationship between the contracting parties.

Strategies

During the life cycle of an SLA, the consumer and provider cooperate to co-create the value of the service [107]. In this cooperation, both parties pursue different strategies to achieve their objectives.

Its business goals influence the provider's strategy. The provider aims at delivering the required quality of service with the lowest possible cost in order to maximize its profit. It might even be economically viable to violate an SLA when the cost savings exceed the contract penalty and possible subsequent damages, such as loss of the public image. This profit-oriented strategy might lead to specific behavior patterns, including the negotiation of vague SLAs with loopholes to avoid penalties and intentionally delivering an overload of reporting information that exceeds the consumer's ability to process, which results in unrecognized SLA violations. The provider strives to get around penalty payments by interpreting the service delivery in compliance with requirements

A consumer aims at maximizing the proportion of the benefit of service quality and its cost. The consumer applies SLAs to maintain that cost-to-benefit ratio. When the quality of a service decreases and violates a defined SLO, the resulting action guarantee simultaneously lowers the cost (e.g., by reimbursement) or offers additional benefits. During the service delivery and application of the SLA, the consumer aims at lowering its cost by filing complaints with the provider to receive indemnification. The consumer improves its proportion of benefit and cost by interpreting the service delivery as violating certain service levels. [45]

Legal context

Digital services operate in the legal context of their jurisdiction. This context describes laws, regulations and best practices for digital services. Laws define the validity of SLA terms, customer rights and procedures for arbitration and litigation. The delivery of digital services across borders enhances the legal context for international laws and the legislation of the respective states. This increases the complexity for arbitration and litigation as it involves the consideration of multiple legislation. [116]

4.1.3. Technology

In the application domain of SLAs for digital services, the technical systems comprise of the infrastructure required for the service, applications for interaction and communication systems for the exchange of information.

Communication Systems

Websites, electronic mail, telephony and video conferences are common technologies for communication. A provider's website contains information about the company and its services, aiding the consumer in discovering a suitable provider and service. Providers also publish nonnegotiable SLAs on their website for consumers with small business volume which do not warrant the effort of SLA negotiations (e.g., [1, 24]). Electronic mail, telephony, and video conference technology are utilized for personal communication in stages such as contract negotiation, SLA application, and control.

Infrastructure

The underlying infrastructure of the communication systems is a part of this application domain. More important however is the required infrastructure for the service delivery and consumption. Digital services are by their nature performed in conjunction with electronic devices, thereby rendering electrical power and its delivery systems as the most important underlying infrastructure. The second crucial infrastructure is the Internet with its connections, hubs, and protocols. Many digital services (e.g., server hosting, video or music streaming, web API or communication services) rely on the Internet to work or to add value, as a video streaming service commonly does not work without the Internet, and a web server in a data center does not offer much value without an Internet connection. For hosting services, in particular, a well-connected and steadily working data center is essential. Incidents with the data center directly affect the service provided by it and result in wrong performance measurements even though the service (e.g., a server) works without problems.

Applications

Regarding the financial processes of invoicing and payment, the applications in the domain of digital services equal those of other business relationships and mainly resort to online banking and usage of payment providers, such as Visa or PayPal. For ordering the services, the providers typically utilize web applications with store systems, mobile applications or RESTful API when targeting software developers [24].

Monitoring the performance of services requires more complex applications. Infrastructure monitoring applications (e.g., Nagios [74]) observe multiple performance indicators of applications, servers, and networks. Other forms of monitoring are possible as well, for example, servers regularly updating an application with their current status or frequent bandwidth tests to determine the throughput of Internet connections. The provider often conducts the monitoring, e.g., for media streaming or hosting services, due to the proximity of the provider to its infrastructure and its superior know-how. A consumer who is lacking the necessary skills to generate the service on its own is not capable of setting up the monitoring for such a digital service.

When the consumer is not monitoring the service on its own, it requests a performance report from the provider. The applications for this purpose are data sheets, dashboards, web applications or notifications that attest to the compliance of service levels.

4.2. Problem identification

In the following section, we present several challenges in the domain of SLAs of digital services. We identified these challenges from literature. First, we describe challenges that are generally apparent in digital services. Second, we describe challenges concerning SLAs of digital services.

4.2.1. General Challenges towards Digital Services

A general issue with business relationships is the necessary trust between partners. Providers rely on their customers to pay the invoices while customers trust on receiving the service in return. Uncertainties such as insolvency of a contracting party further complicate these relationships. In case of such conflicts, SLAs and service contracts define clauses with obligations, arbitration procedures and penalties. However, the enforcement of these rights is not trivial.

Challenge	**Description of the challenge**
C1: *Online fraud*	Barrier to business relationships due to uncertainty of returned performance
C2: *Small disputes*	Enforcement of small disputes is waived due to misaligned cost-to-benefit ratio
C3: *Cross-border enforcement*	International relationships hindered by complexity of foreign law and uncertainty of enforcement
C4: *SLA cost-to-benefit ratio*	Effort in SLA management might not be economically viable
C5: *Cloud services*	Dynamic digital services challenge the current static SLA
C6: *IoT*	Machine2Machine relationships require machine-usable SLA
C7: *Fragmented systems*	Integrated customer view is missing due to fragmented customer-oriented processes
C8: *Unclear SLA specifications*	Defined SLI and SLO leave room for interpretation
C9: *Ineffective problem solving activities*	SLA describe effort to solve incident and not the desired result
C10: *SLA governs set of services*	SLA are not transparent enough to analyze individual service performance and optimize cost
C11: *Heterogeneity of SLAs challenges providers*	Complexity of overload of different SLA overburdens providers processing ability
C12: *SLA management complexity for consumers*	SLAs effectively useless as consumers are overburdened by management
C13: *Performance reports depend on provider*	Reporting on their own performance allows for fraud
C14: *SLA as dead-end document*	SLA tend to be filed away and not used

Table 4.1.: Challenges in the domain of digital service SLA

C1: Online fraud

Online fraud is an issue with digital services, especially because of the global nature of the Internet that enables transactions between unacquainted individuals. The contractual obligations, such as payment and returned service, are separately executed and thus exposed to fraud due to voluntary performance [19]. There is no certainty in receiving compensation for prior performance. This uncertainty and the risk of online fraud impede the formation of business relationships.

C2: Small disputes

An example of enforcement challenges are disputes with small claims [97]. Their disputed amounts are less than the legal costs and the profit for lawyers is not attractive enough for their engagement, resulting in one contractual party not receiving fair reimbursement while the other party continues violations with no enforcement to fear. This is observable in the flight industry. Airlines deny their customers delay compensations and gamble on them waiving their rights as the effort for enforcement is not proportional to the small claim amount [38]. Similar behavior is possible when the indemnification for QoS violations is not worth the trouble of enforcing it [57].

C3: Cross-border enforcement

International services comprise another challenge to the enforcement of contracts. This challenge intensifies with the prevalence of globalization and increasing international trade that is enabled by the Internet [57]. Enforcing contracts in other countries requires knowledge about the applicable law, which is mostly not available in digital service relationships. Similar to the issue of small claims, cross-border enforcement is impaired by the unprofitable cost-to-benefit ratio of litigation [57].

4.2.2. Challenges of SLAs

In the following section, we present problems with SLAs as described in the literature. We structure this section into general challenges with SLAs and those per step in an SLA's life cycle.

General challenges

C4: Cost-to-benefit ratio A general problem with SLAs is their cost-to-benefit ratio. [50] presents examples showing that exorbitant effort in drafting and applying an SLA does not lead to benefits in the same order, but might impede the potential of SLAs.

Berger [5] supports this view and states, that SLAs might not be economically viable in relation to their development, application and control costs.

C5: Cloud services The importance of a useful cost-to-benefit ratio is especially apparent in today's cloud services [61]. One of the virtues of cloud services is the high variability of service volume to enable on-demand scaling. Manually drafting an SLA before receiving additional service volume requires an effort that contradicts the dynamic nature of cloud services. Buyya [13] states the need for dynamic negotiations of SLA. Aside from the negotiation of an SLA, its application for cloud services is also challenged by the variation of service volume. As this changes during the service contract, the monitoring of service levels and enforcement of the SLA needs to adapt to the variation to enable compliance for each point in time and each service [96]. The dynamic characteristics of cloud service challenge the manual efforts in the monitoring, reporting, and compliance checking processes [68].

C6: IoT Another modern technological trend is the IoT (IoT). Devices, such as machines, sensors, utilities, are connected and communicate with each other. This machine to machine (M2M) communication allows self-organizing capabilities among the devices [71]. When machines not only communicate but also cooperate in service relationships, they require methods to utilize SLAs without manual human intervention to ensure QoS as well as to enforce the SLA [66]. The current static, legal-text oriented SLA are not suitable for IoT and M2M technology.

C7: Fragmented systems prevent integrated customer view When providers deploy multiple applications to communicate with customers, send invoices, handle incidents and report service levels, they receive an imperfect view of the customer. This missing integration of SLA management systems into customer relationship management is a deficiency with providers [86].

Challenges in the life cycle of SLA

Multiple challenges arise during the life cycle of an SLA, which are subsequently specified.

Definition

C8: Unclear specifications During the definition of an SLA, the involved parties might define unclear specifications [104] that lead to interpretation disputes. As an example, a common metric for Internet-connected services is availability. Defining that the availability of the underlying service should exceed 99% is not sufficient. The SLA should also describe what 100% means in this context, how and during which time the measurement is made and if the outage of 1% is allowed for a single incident or accumulated over multiple incidents with each limited to a maximum duration. Without these specifications, the availability metric would be measured and interpreted differently by the contracting parties and conflicts about SLA compliance would arise during service performance.

C9: Ineffective problem-solving activities SLAs often specify an effort by the provider instead of a certain result in case of service interruptions [104, 106]. Problem-solving activities are declared successful if a provider invests the specified effort rather than measuring the effectiveness of that effort. For example, when a provider restores a network, the services on that network might need time to restart their operation, and the problem is only effectively solved when the supported business processes are restored. The gap between the provider's effort and the actual problem solving is underrepresented in SLAs [104, 106]. [5] adds that important QoS metrics and service levels are required to address the supported business process instead of the effort by the provider.

Application

C10: Definition for a set of services SLAs are often part of contracts for a set of services. The associated service cost is due for the whole set, and it is not possible to map the costs to individual service's quality. This makes it difficult for the customer to calculate the optimal price-performance ratio for individual services and hinders cost management [104].

C11: Heterogeneity of SLAs challenges providers Providers offer their different services to numerous customers with varying SLA. This heterogeneity in QoS requirements and management of SLAs poses a complex challenge for providers which is not solvable manually [5, 79]

C12: Consumer is overburdened by SLA As with the provider's heterogenous collection of SLAs, consumers face a similar challenge. They often use different services from multiple providers which requires them to keep track of multiple SLAs [106]. Not only do consumers face the challenge of managing multiple SLA; private consumers are also most likely to lack the technical knowledge to understand technical content of SLAs, as consumers often demand services that they cannot perform themselves. The overburdening amount of services and lack of domain knowledge indirectly prevents consumers to enforce the compliance of SLAs, therefore rendering these agreements effectively useless for consumers.

C13: Performance reports depend on provider In order to control SLA compliance, customers need the service monitoring data which is often generated and controlled by the provider [102]. The customer, therefore, depends on the grace of the provider to receive this data timely and with correct information [106] which complicates the customer's ability to enforce SLA compliance.

Control

C14: SLA as dead-end document After signing an SLA during service performance, the document only becomes relevant again in case of issues, such as non-compliance. When that happens, the content of the SLA with its technical terminology and concepts might be difficult to understand for non-technical managers and hinder its application [104]

4.3. Specification of Design Objectives

Based on the identified challenges, we subsequently describe design objectives for the artifact. The main goals of the artifact are the reduction of manual effort for service participants, the enforced fulfillment of obligations by automating SLA actions and the enabling of machine-to-machine contract interaction. The reduction of manual effort supports the scalability of SLAs for cloud computing. We also aim at creating a common worldview among the service co-creators to increase transparency, establish trust and facilitate interaction.

4.3.1. Motivation for Blockchain Usage

We motivate the use of the blockchain technology for the following design objectives with its promising features: Weber et al. [111] indicate an increased efficiency of collaborative processes by employing a blockchain. This is supplemented by its "streamlined data sharing" ability that creates a shared view on the service among service partners [47]. The use of the blockchain as an intermediary could reduce costs of third-party agents [108]. Kruijff and Weigand [58] describe the reduction of manual efforts with smart contracts compared to traditional contracts. This could increase the efficiency, speed and performance of contracts [37]. Enforcement of contractual obligations by smart contracts could reduce the need for disputes and litigation [19].

Design Objective	Description of design objective
DO1: *Synchronize obligation fulfillment*	Reduce risk of advanced performance by coupling obligation fulfillments
DO2: *Remove enforcement costs*	Reduce costs for court litigation with smart contract enforcement
DO3: *Guarantee enforcement*	Employ smart contract enforcement to reduce foreign law unambiguity
DO4: *Re-usable SLA patterns*	Software patterns oriented SLA smart contract programming
DO5: *On-demand SLAs*	Smart contracts enable SLA scaling with dynamic demand
DO6: *Enable SLA interaction for machines*	Smart contracts allow devices to sign and utilize SLAs
DO7: *Integrate customer-oriented processes*	Smart contracts integrate multiple aspects of customer-oriented processes
DO8: *Code unambiguous metrics*	Smart contract's need for coded service levels forces unambiguous metrics
DO9: *Measure incident solution effectiveness*	The monitoring service measures meaningful metrics
DO10: *Modularize SLA to increase detail*	A service in a set is represented by an own smart contract to allow specific SLA and detailed analysis
DO11 & DO12: *Automate SLA management*	Smart contracts are deployed to trustfully monitor compliance of the SLA and automate contract actions
DO13: *Externally monitor performance*	By implementing a monitoring service, required trust is more evenly distributed
DO14: *Revive SLA by deploying as smart contract*	Smart contracts execute without user interaction thus keeping the SLA alive

Table 4.2.: Design objectives for the IT artifact

DO1: Synchronize obligation fulfillment To counter fraud, we propose to synchronize the fulfillment of obligations thus eliminating the need for one party to risk going into advance. In a deal with an asset exchange for payment, the transfer of the asset and the payment happen at the same time. This consideration is translated to services by introducing a turn-by-turn system. The fulfillment of obligations approximates to a synchronized transfer by increasing the frequency of transfers and thus reducing the potential loss of each turn in case of non-compliance. Smart contracts are employed to increase the frequency of transfers without increasing the effort for the contract parties.

DO2: Remove enforcement costs Smart contracts can increase efficiency in litigation by automating the enforcement of contractual obligations. Their programmed contractual logic judges the present information to enforce the actions of an SLA. Employing smart contracts saves the need for costly arbitration and thus lowering the barrier to contract enforcement. We introduce the required logic for enforcement into smart contracts.

DO3: Guarantee enforcement Similar to decreasing the costs for litigation as stated above, smart contracts provide a shared basis of legal logic that is enforceable without the need of international courts. As SLA partners define their obligations and actions in a smart contract, they rely on the enforcement of their contract which is executed independently from the cross-border law. We include provisions in the smart contract, which reduce the need for cross-border litigation.

DO4: Re-usable SLA patterns Programmable SLAs decrease the complexity of drafting SLAs by offering re-usable patterns. Comparable to software patterns, these SLA patterns offer standardization for routine use cases. As an example, the same pattern could be applied to monitor the availability of a web service and server hosting. We aim at developing re-usable patterns for SLAs.

DO5: On-demand SLAs For today's dynamic cloud services, we propose smart contracts that scale with the service volume. The SLA terms are programmed, but the monitoring and enforcement of obligations dynamically adapts to the number of services.

DO6: Enable SLA interaction for machines To enable devices in the IoT to act on their own, the smart contract SLA offers interfaces to machines [4, 89]. These interfaces support the transfer of SLA information and allow the devices to use services that are enforceable by them. We propose a smart contract that offers service interactions to IoT devices including indemnification without human intervention.

DO7: Integrate customer-oriented processes For a holistic view of a customer [86], we integrate multiple processes into the artifact by supporting sales, invoicing, performance reporting and enforcement workflows.

DO8: Code unambiguous metrics The design objective for countering this challenge is implied in the nature of programming SLA terms. In order to develop code that monitors the compliance of service levels, it is necessary to define the metrics and measurement methods unambiguously without any lack of definition.

DO9: Measure incident solution effectiveness We propose that the smart contract validates the effectiveness of incident solutions by measuring performance indicators that reflect the business process and not an underlying infrastructure.

DO10: Modularize SLA to increase detail For a set of services, such as combined web hosting and electronic mail offers, we yield the modularization of these individual services and their SLA. The SLA enforcement of individual services offers a more detailed analysis of each service that increases transparency for the consumer.

DO11 & DO12: Automate SLA management In order to remove complexity for both the provider (DO11) and the consumer (DO12), we automate SLA actions and decrease required manual tasks. Providers and consumers thereby reduce their effort with monitoring and enforcing multiple SLAs.

DO13: Externally monitor performance By including an independent monitoring service, or enabling a service to report its own performance, we automate the reporting of performance data. Consumers will be able to transparently access this information to validate the correct enforcement of an SLA [9].

DO14: Revive SLA by deploying as Smart Contract In order to increase the usage of SLAs, we develop them as smart contracts that are executable on the blockchain. This way, they provide the benefit of regulating the service even if they lose attention after their negotiation.

5. Development of the Artifact

In the last chapter, we demonstrated the relevance of this artifact by identifying challenges of this problem domain and designed objectives for a solution to its challenges. We follow these goals in the subsequent sections, where we draw from the knowledge base of chapter 2 and chapter 3 to design and develop a prototypical, blockchain-based application as the design science artifact.

5.1. Scope of the Prototype

The prototypical application is developed for a particular environment to confine its magnitude to the capacity of this master's thesis. We selected server hosting as an appropriate representative for a digital service, as it matches the identified problems of uncertain SLA enforcement, untrustworthy contract partners, the on-demand nature of cloud computing and required service interaction between devices. Further, we define the availability of a server as the metric for its performance. As we are IS scholars, we do not assume to hold the legal knowledge required to assess the legal validity of the prototype. Therefore, we exclude legal considerations in the design and development of this artifact.

5.2. Design of the Prototype

First, we define the functional requirements for the system by describing use cases. We then describe design decisions, construct an architecture and specify components of the system.

5.2.1. Functional Requirements

The application has two different types of users: providers and customers. Typically for digital services, a provider (e.g., a data center operator) offers at least one service, provides the service and manages billing and monitoring of SLA compliance. Customers browse the offerings, buy services and pay for consumption.
We reason that SLAs and the financial aspect of their actions (e.g., violation reimbursements) are entwined with other processes in the service provisioning (e.g., billing) and

cannot exist in an application on their own. We include the user activities as mentioned above in the application and derive the following use cases (see Figure A.1) from the life cycle of SLAs and the specified design objectives (see Table 4.2).

UC1: Browse service offerings

Name	Browse service offerings
Goal	Offered services are presented in the application
Description	A provider generates a service portfolio by adding service offerings to the system. These offerings are presented in a human and machine-readable form that allows a consumer to gather information on services and their SLAs.
Pre-condition	No pre-conditions
Connection to other use cases	No connections
Design Objectives	• DO4: Data model allows SLA to be re-usable • DO8: Machine-readable form requires unambiguous SLA metrics • DO10: Modularized service offerings with own SLAs

UC2: Order a service

Name	Order a service
Goal	The system allows customers to order a service
Description	Customers are able to order a service via the application. The system processes the order, initiates a service contract and relays necessary information to the provider.
Pre-condition	The portfolio of UC1 contains service offerings.
Connection to other use cases	No connections

Design Objectives	• DO5: SLA are connected to on-demand orders • DO6: System allows machines to place orders • DO14: Order creates smart contract that implements SLA

UC3: Handle service payment

Name	Handle service payment
Goal	Payment for the service is handled by the system
Description	The application acts as an escrow by holding consumer funds and transferring them to the provider. It reduces the payments by the reimbursement amount, which is derived from enforcing the SLA. Thereby, the payment is connected to the fulfillment of contract obligations. A customer interacts with the system to deposit the service fee and retrieve refunds. Providers are able to withdraw their payment from the application.
Pre-condition	Ongoing service contract between provider and consumer
Connection to other use cases	Includes Enforce SLA (UC4)
Design Objectives	• DO1: Fulfillment of obligations is connected • DO7: Billing, enforcement and performance validation are integrated • DO11/12: The system automated the SLA enforcement and payment

UC4: Enforce SLA

Name	Enforce SLA
Goal	Actions defined in the SLA are automatically executed to enforce the contract

Description	The system evaluates the service performance data in comparison with the SLA to check compliance. Actions due to SLA violations are executed and the system utilizes the herein calculated reimbursements in the payment process to refund the consumer.
Pre-condition	System contains service performance data
Connection to other use cases	Included by Handle service payment (UC3)
Design Objectives	• DO2: The application handles enforcement without extra cost • DO3: Enforcement is guaranteed by the application • DO6: Contracts between devices are enforced • DO11/12: No manual trigger required • DO14: SLA actively 'lives' as part of the application

UC5: Validate service performance

Name	Validate service performance
Goal	Actors and the system are able to validate the service performance
Description	The application stores service performance data and provides it to internal processes as well as external actors. Consumers and providers are thereby able to verify the service performance.
Pre-condition	Service performance data is monitored
Connection to other use cases	Included by Enforce SLA (UC4)
Design Objectives	• DO7: Performance data is integrated with other customer information • DO9: Monitoring includes measurement of incident solution effectiveness • DO13: External agents monitor the service's performance

UC6: Terminate service contract

Name	Terminate service contract
Goal	The system terminates service contract at the end of its duration or upon request by an actor
Description	According to the specified end date of the service contract, the system ends the contract and its supporting processes. This can also be initiated by a provider or consumer.
Pre-condition	Ongoing service contract
Connection to other use cases	No connections
Design Objectives	• DO11/12: Ending of service contract is automated

UC7: Consume service

Name	Consume service
Description	This use case is independent from the application. It is included in the diagram as it is the core activity of a service between consumers and providers.

5.2.2. Design Decisions

The characteristics of blockchain-based applications demand a specific design process. We used the design process of Xu et al. [117] as an orientation to derive the subsequent decisions.

On-chain / Off-chain Data Storage

The decentralized nature of the blockchain favors data integrity and reliability over storage space, as nodes replicate the entire state of the blockchain. This affects the design of our blockchain-based application as the storage of data on public blockchains is limited and requires costly transactions.
The prototype's data is twofold: (1) static data contains information about the contract, actors, and the SLA and (2) a frequently increasing dataset includes the performance measurements.

For the first type of data (static), we propose on-chain storage. Storing data on the blockchain requires an initial transaction that writes the metadata to the blockchain. There are no numerous modifications afterward, and by saving this information on-chain, smart contracts can access the data without requests to external data sources, thereby increasing the speed of their functions. The storage of data in smart contract also allows to include access management.
The second type of data in this prototype, the set of performance measurements, proliferates over time. This data is required by consumers and providers to verify the service performance, and by the prototype to check compliance with the SLA. The continuous service monitoring in this prototype generates regular updates (e.g., per second), which are small in size but the amount of these updates poses a high cost at 0.11 USD per transaction (July 2018 [30]).
Off-chain storage provides a possible solution to the high amount of transactions, yet introduces complexity into the application as data is not readily available for processing in smart contracts.
Therefore, we decide on a compromise and employ on-chain storage of service performance data by concentrating the set of measurements into a single data point for a specified period (e.g., one day). Instead of sending multiple transactions (86,400 for a day with measurements every second) to the blockchain, we design the monitoring agent to aggregate all measurements over a reasonable time span into a single value. The benefit of this single value is its small size and the reduced frequency of transactions (e.g., one per day).

Identities and Roles

This application contains data and functions whose access should be limited to certain roles and identities. The Ethereum blockchain employs cryptographic keys so that a public key represents an individual. While this does not reveal the identity of a person, it is an adequate way to identify and authorize individual blockchain accounts. Therefore, we employ the integrated role and identity management of the blockchain to enable access management in smart contracts by restricting access to specified identities.

Oracle for Monitoring Data

The application requires SLIs to facilitate the validation of service performance in accordance to the SLA. Blockchain utilize oracles to retrieve information from outside their network. Services, such as Chainlink or Oraclize.it, operate smart contracts that can be called to query specific web APIs and return the gathered data. For the development of this prototype, we decided to design an own oracle that provides QoS measurements to smart contracts.

5.2.3. Entities of the Prototype

The functionalities of the system are distributed among multiple entities, which include the blockchain, a frontend and an agent for service monitoring.

Blockchain

The blockchain is the central part of the system as it contains the data and the program logic. Smart contracts implement the functionality of the use cases, control access rights of data and transfer funds between consumers and providers.

Web Frontend

The user interface of this prototype is a lightweight web frontend that interacts with the blockchain. The frontend does not perform complex functions but merely displays the blockchain's data, enables submission of performance data and transactions to the blockchain to control the smart contracts.

Monitoring Agent

The monitoring agent is a standalone application without a graphical user interface. It only sends information to the blockchain, the provider, and the consumer and does not perform program logic of use cases aside from its monitoring function. The agent can also be part of the service itself, such as a daemon that runs in the background of a hosted server.

5.3. Implementation of the Prototype

In this section, we describe the structure of the developed smart contracts, our implementation of the integrated oracle and state channel, and the views of the web frontend. We follow with presenting the selection of the employed technology and list the implementation status of the use cases.

5.3.1. Structure of the Smart Contracts

We developed the smart contracts based on existing patterns [63, 72, 109, 114] and separated the functionality into multiple contracts: controller, database, and logic. Controller contracts provide constructors as well as getter and setter functions. They

inherit logic smart contracts, which include the application's logic and themselves inherit the database smart contracts. The database smart contracts contain variables, modifiers, and events.
An overview of the developed smart contracts is included in Figure A.6.
The *Provider.sol* smart contract implements the logic and data, that is necessary for representing a provider on the blockchain. It contains the offered services, customers, and their contracts as well as functions to add service offerings and to buy a service.
The "buyService" method of *ProviderLogic.sol* creates new instances of the *Service.sol* smart contract, which represents a service contract between a provider and consumer. This smart contract offers functions and data for contract management, verification of service performance, SLA execution and payment.
The *Hosting.sol* contract includes the data model (as structs) for SLAs and service offerings.

5.3.2. Implementation of the Oracle as a State Channel

A first implementation of the monitoring agent acted as an oracle by frequently (1-second intervals) sending the service status to the smart contract. An evaluation of the costs of these numerous transactions revealed that this approach is financially not viable.
Instead, we integrated the oracle function into a state channel to enable verifiable off-chain transactions that contain information on the service performance. The monitoring agent generates an off-chain transaction for each turn (e.g., one day) to verifiably send the performance data to the customer and the provider. These parties validate the data and, when approved, sign the off-chain transaction with their identity and send it to the other party. If the data does not match the expectations, the disapproving party does not sign the transaction and closes the state channel with the last approved transaction instead of entering the next turn.
The state channel closes when the signed off-chain transactions of both parties are transmitted to the Service smart contract. This triggers the smart contract to determine the SLA compliance, possibly execute SLA actions and transfer the resulting service fee to the provider.

Our implementation of a uni-directional state channel differs from existing solutions (e.g., [82, 87]), that employ state channels for off-chain cryptocurrency transfers and offer multi-participant, bi-directional channels. The state channel in this prototypical application enables the verifiable off-chain broadcast of monitoring data and acts as a multi-sourced, push-principled oracle.

5.3.3. Realization of the Web Frontend

The web frontend's base is the App.js component. It imports the views, web3.js and smart contract ABI. Its state contains the fundamental data for the web frontend, such as the service offerings, active service contracts and the address for the Provider smart contract. The App.js component connects to a blockchain node via the web3.js API and executes calls to receive data from the Provider smart contract and Service smart contracts based on the selected account in MetaMask. A footer displays the currently selected account in MetaMask as well as the address of the Provider smart contract. The navigation bar offers access to the three views, which are child components of App.js: Home, Store, and Billing.
The Home view (see Figure A.3) is the web frontend's landing page and contains a brief introduction to the usage of the web application.
The Store view (see Figure A.4) gives an overview of the provider's service offerings and allows consumers to buy services. The displayed service offerings and their corresponding SLAs originate from the Provider smart contract. Consumers input the desired duration for the contract and buy the service via the "Buy"-button, that shows the price for the selected duration and effects the "buyService" transaction of the Provider smart contract.
The Billing view (see Figure A.5) contains information on individual service contracts and offers the submission of monitoring data for the state channel. The top area shows contract related information and enables the user to extend or shorten the contract duration. The area 'Bill Calculation' displays the compliance with the SLA for the selected time span and lists the associated costs and refunds due to SLA actions. The subjacent collapsible area provides input fields for the submission and validation of off-chain transactions that include monitoring data. This is the frontend for the state channel, which integrates the oracle.

5.3.4. Technology Selection

In this section, we present the employed technologies of this prototype for each entity.

Ethereum as Blockchain technology

We chose Ethereum [10] as a blockchain technology as it allows the deployment of smart contracts for distributed execution of code. We selected Solidity as programming language for smart contracts, since it is the prevailing language for Ethereum [33]. As a development blockchain, we used the Truffle Framework that includes a simulated Ethereum blockchain [18]. To simulate the behavior of the prototype on the Ethereum

Mainnet, we employed the Ropsten test network with a light-syncing geth node [31].

React as Web Frontend

We developed the web frontend with the React library [36]. It allows re-usable components and uncomplicated state management for data storage that refreshes page content without reloading. Being based on JavaScript, the React library easily integrates the web3js API [32] to directly communicate with Ethereum nodes or interact with the MetaMask browser extension that handles interactions with blockchain nodes. A Node.js server [39] hosts the React-based single-page application. Figure A.2 displays the architecture of the web frontend with its three views (Home, Store, Billing) and components.

Node.js as Monitoring Agent

Another use for the Node.js JavaScript runtime [39] is for the entity of the monitoring agent. This Node.js application integrates web3js [32] to communicate with an Ethereum node and runs without a graphical user-interface while continuously monitoring the service performance. The monitoring agent produces simulated measurements, as the implementation of suitable measurement methods is beyond the scope of this thesis.

5.3.5. Fulfillment of Functional Requirements

We implemented the prototype according to the functional requirements specified in 5.2.1. The core functionality of enforcing the specified SLA by deducting a reimbursement from the service fee is implemented as part of a smart contract. To focus on this core functionality, we decided to only partially implement the use cases "UC1: Browse service offering" and "UC2: Order a service" as their missing features do not essentially limit the potential of the prototype. The table 5.8 gives an overview on the implemented use cases.

	Use Case	**Implementation status**
	UC1: Browse service offerings	Partially implemented: lacks interface for provider to add service offerings
	UC2: Order a service	Partially implemented: lacks provider app that handles orders
✓	*UC3: Handle service payment*	Fully implemented
✓	*UC4: Enforce SLA*	Fully implemented
✓	*UC5: Validate service performance*	Fully implemented
✓	*UC6: Terminate service contract*	Fully implemented

Table 5.8.: Status of the implementation of use cases

6. Evaluation of the Artifact

In this chapter, we proceed to step five in the DSRM process with the evaluation of the developed prototypical application. The following sections present the design of the evaluation and its results. Insights from evaluations led to two subsequent iterations of the design and development process.

6.1. Design of the Evaluation

We design the evaluation of the artifact according to the framework of Pries-Heje, Baskerville, and Venable [84]. Their strategic DSR evaluation framework includes three characteristics to formulate a strategy: the timing of the evaluation (ex-ante or ex-post), the type of the artifact (product or process) and the setting of the evaluation (artificial or naturalistic). The artifact in this thesis is a software application, which constitutes a product type. To evaluate the application, we first need to develop it; therefore we choose an ex-post evaluation. Due to the simulated setting of the artifact, with its exemplary hosting provider, the artifact needs to be evaluated in an artificial context. Therefore, we apply an artificial ex-post strategy by evaluating in a simulated setting after the artifact's creation.

We base the evaluation on the criteria of Prat, Comyn-Wattiau, and Akoka [83] to measure the artifact's goal achievement, its utility, the applicability in its environment and the validity of its design and development.

There are multiple methods for the evaluation of DSR artifacts, including simulations, theoretical arguments, case studies and satisfaction surveys [42, 45, 81, 84]. Due to the innovative nature of the blockchain-based artifact, we concede us a flexible selection of methods to gather diverse results. This selection includes expert interviews with demonstrations of the application [42], a review of the design and development process [83] and our critical review that includes quantitative system measurements and a qualitative evaluation [81].

6.2. Functional Evaluation

We evaluate the artifact by interviewing experts in the domain of digital services. The following section depicts the interviews with an expert in the role of a customer, and an expert in the role of a provider of digital services.

6.2.1. Findings from a Customer's Perspective

We evaluate the artifact from the customer's viewpoint by interviewing Mr. Kauk, an IT Business Analyst in a global industrial company with around 1,900 employees, where he oversees IT service contracts. The following information is based on the interview [51].

We first validated our theoretical considerations (see Table 4.1) and found support for multiple challenges, which are indicated in the following with their number in parentheses. During service delivery, the customer is monitoring the performance of services for two reasons: (1) to quickly identify and solve possible problems and (2) to verify the performance of the service, as those measurements are otherwise held by the provider (C13). When performance issues occur, the customer decides on whether to enforce contractual rights from an SLA against the provider or not to. This consideration is necessary especially for small disputes (C2), as the effort to enforce contractual rights might not be viable in comparison to the expected reimbursement (C4). This effort is increased as SLA documents are often forgotten after signing and thus need to be found, understood (C8) and applied to the prevailing issue (C14). The management of multiple SLA for multiple different services further complicates their usage (C11).

The concept of the artifact and the demonstrated prototype offer the potential to reduce manual effort in SLA management as the smart contract-based SLA verifies service performance and enforces obligations on its own (DO12, DO14). This artifact thus removes the need to consider enforcement (DO3). As the enforcement of SLA provisions runs according to its predefined terms, legal support for individual incidents is not required and the artifact thus decreases enforcement costs (DO2). The implemented turn-by-turn based service delivery and payment reduce the risk of not receiving the service when paying in advance (DO1) and offers an escrow-like system that establishes trust in the provider. Another perceived benefit of the prototypical application is its support of business growth, as the automation of SLA enforcement enables scaling of employed digital services. This scaling is supported as the administrative efforts of automated SLAs does not grow on the same scale as the use of digital services with SLAs.

Despite of these benefits, the interviewee raises concerns about the practicability of the artifact. Decision makers in companies lack the technical knowledge regarding blockchains and associate the technology with financial gambling and marketplaces for illegal goods. This leads to concerns about the reliability, effectiveness, efficiency and legality of this artifact. To gain the trust-establishing benefits of the blockchain, they would rather commission an external escrow service and employ bank transfers instead of cryptocurrency transactions. The SLA of this prototype, with its reimbursements as SLA actions, limits the usability of the artifact. These reimbursements do not cover damages that arise from ineffective digital services that fail to support their business process and thus impact the customer's business. Therefore, the artifact is limited to digital services with low associated risks.

Aside from the practicability of the artifact, its web frontend poses a suitable interface that intuitively allows to order services and verify their performance. The area where customers submit monitoring data is too complicated for users due to the required input value. Aside from these complex inputs, the knowledge requirements of using a blockchain-based application could be supplied to users with appropriate training.

6.2.2. Findings from a Provider's Perspective

We evaluated this artifact from a provider's perspective with Mr. Dillinger, a self-employed, small provider of root servers, virtual servers and web hosting space with about 200 customers. The following information is based on the interview [25].

Several identified challenges (see Table 4.1) received recognition in the interview. Disputes about service performance require an administrative and legal effort, that does not always relate to the acclaimed value. Thus, the enforcement of an SLA might not be economically viable for the provider (C4), which especially holds true for services with a small marginal return (C2). Some customers are technically not knowledgeable enough to monitor the performance of their service and depend on measurements of the provider. This delivery of information is an additional effort for the provider (13). To ease the own administrative effort, the provider employs similar SLAs for the offered services (C11). While there has not been a severe legal dispute with international customers yet, the provider operates under legal uncertainty about his rights and enforcement possibilities abroad (C3).

The provider positively regards the automatic SLA enforcement of the artifact, as this could remove enforcement costs and enable the application of SLAs for service with small marginal returns (DO2). The on-demand nature of the prototype's SLA enforcement could enable short-term services, such as dynamic cloud computing (DO5), as the automatic application of SLA provisions reduces the manual effort for SLA management (DO11). As the use case of ordering a service is implemented within a smart contract, it allows devices to order services autonomously (DO6), which could further support automatic scaling of services. The prototype's integration of billing and SLA reimbursement supports the unification of these data points into a single view (DO7).

Aside from the implementation of the design objectives as described above, the interview showed other benefits of the artifact. Some of the provider's customers appreciate privacy and would prefer the payment with cryptocurrencies instead of PayPal or bank transfers. The blockchain-based application could reduce administrative work of the provider in regards to customers, that tend to abuse the SLA by repeatedly filing unsubstantiated claims of lacking service performance with the provider. These claims need to be addressed by the provider with proof of service performance. This effort could be reduced by the artifact, as it holds the monitoring data, automatically verifies SLA compliance and thereby proves the service performance.
Our interview partner recognized the artifact's trust-enabling features of automated SLA enforcement. Employing these features into the business model could allow a novel value proposition for offering SLAs that are guaranteed by smart contracts.
The implemented web frontend with its store and billing view suitably offers the functionality of a provider's website.

The interview also highlighted issues of the blockchain-based prototype. The artifact's turn-by-turn principle leads to a manual effort for the provider, as he must validate the monitoring data for each service every turn (day). Instead, the provider would prefer to hide the state channel functionality behind an interface that automatically handles these transactions, as the current implementation is too complicated and existing monitoring data might be unavailable to the web frontend. In Germany, the turn-by-turn principle of service delivery and payment is restricted by laws that prohibit providers to directly shut down servers on missed payments. An automated stop of service delivery by smart contracts is thus legally not valid.
A second legal obstacle are data protection laws as data is distributed on the blockchain and thus out of control of the provider.
Similar to the findings of the interview with a customer, the provider noted the reputation of cryptocurrencies as another issue, as these currencies are often associated with criminal activities, financial gambling and security vulnerabilities. The skepticism against cryptocurrencies impedes the feasibility of blockchain-based applications, especially in Germany where customers prefer traditional payment methods (e.g., cash, bank transfers).
Another problem with the blockchain is the complexity of its technology. The provider stated missing trust in the reliability and security of the blockchain as reasons against its use. The exchange rate fluctuations of cryptocurrencies increase the entrepreneurial risk and lead to additional effort for the provider.

6.3. Technical Evaluation

We obtained an expert review of the technical aspects of the artifact, to evaluate the design, development, and implementation of the prototypical, blockchain-based application. The following insights stem from a review by a Mr. König, a blockchain developer for a software company with about 400 employees [55].
The use of blockchain technology in this artifact presents an innovative development that allows new functionality when compared to existing IT solutions. The innovative functionality of the artifact is the trustworthy recordkeeping of contract signing and service performance on the blockchain, which prevents the manipulation of service performance data. Therefore, the architecture of the artifact, with the blockchain for data storage and application logic, and a web frontend as a user interface, present itself suitable for this functionality. A possible improvement to the architecture could be distributed storage (e.g., IPFS, Swarm) for storing the data. This leads to less and smaller transactions to the Ethereum blockchain; thus reducing the costs for using the prototype.
The selection of technology, with Ethereum as blockchain, the React framework for the web frontend and MetaMask to interact with the blockchain, conforms to the current state of blockchain-based applications. While MetaMask is not trivial to use, its utilization is reasonable for this artifact. The Angular framework would have been an equally suitable choice, allowing personal preference as a deciding factor between both frameworks. Ethereum is the proper blockchain technology for the artifact, as it allows smart contracts as well as tokens. Bitcoin supports neither, and the Hyperledger blockchain allows smart contracts but lacks provisions for tokens. These tokens could be used for an improvement of the artifact by implementing subscription-based payment (see ERC-948 [17]) instead of requiring the user to provide a deposit.

6.4. Critical Review of the Application

The following critical review of the application consists of a qualitative and quantitative evaluation which has been performed by the research team.

6.4.1. Quantitative Application Evaluation

Since we research the feasibility of smart contracts in this thesis and are interested in the implications of blockchain as an architectural design choice, we focus on measuring blockchain-related metrics. The characteristic differences of blockchains, compared to traditional programs and databases, are the occurring monetary cost of transactions as well as the high latency of processing due to the creation time of blocks [90]. Hence, we measure the cost and the latency of the prototypical application to execute the use cases, based on the interactions presented in the sequence diagrams of section A.5. We measure the developed prototype on the Ethereum test-network Ropsten, which offers real-world conditions similar to the Ethereum Mainnet without real costs. As a premise, we set the price of a single unit of gas to the current average of the Ethereum Mainnet of 3.5 Gwei (July 2018 [34]).

Cost of Execution

We observe the execution costs in units of gas, as each programmatical operation of smart contracts costs a fixed amount of gas, which is independent of the underlying Ethereum network. This gas amount is then multiplied by the gas price (3.5 Gwei) to estimate the current execution cost in Ether on the Ethereum Mainnet. We finally convert that value to USD by applying the current exchange rate (461 USD/ETH, July 2018 [34] to derive a financial value. We observe the highest costs when instantiating new smart contracts by deploying the Provider smart contract or buying a service. The functions of adding a new service offering to the provider smart contract and handling the payment by adding monitoring data also incur relatively high costs due to the data attached to their transaction. Extending or shortening the contract duration are relatively cheap functions. Table 6.1 lists the costs per use case. These costs relate to the characteristics of the Robsten test-network, which is publicly accessible and employs a work-intensive proof-of-work consensus mechanism.

System Latency

We define the system latency of a use case as the duration between issuing the transaction to the network and receiving the receipt of the transaction being included in a block. This is the earliest possible moment to provide feedback to a user of the system. The deployment of a new provider smart contract instance presents the highest latency of almost 2 minutes. Handling the service payment also display a high latency of 97 seconds. Since this use case includes two transactions, each from the provider and consumer, its latency per transaction is comparable to the functions of extending and shortening a service contract (53 seconds, 42 seconds). The measurement results are displayed in Table 6.1.

Use Case	Consumed gas	Cost in ETH	Cost in USD	Latency in seconds
UC1: Deploy provider instance	5,147,875	0.01802	$ 8.31	119.7
UC1: Add service offering	487,929	0.00171	$ 0.79	18.7
UC2: Order a service	2,729,743	0.00955	$ 4.40	24.7
UC3: Handle service payment	1,455,864	0.0051	$ 2.35	97.0
Extend contract duration	624,168	0.00218	$ 1.01	53.8
UC6: Shorten contract duration	492,154	0.00172	$ 0.79	42.2

Table 6.1.: Cost and latency measurement results of the prototype

6.4.2. Qualitative Application Evaluation

Limitations of the Prototype

This section presents limitations of the prototypical application, that arise due to the defined scope of the artifact as well as the inherent characteristics of the blockchain technology.

Limitations of SLA Integration

The prototype limits possible SLA actions to a refund of the service fee, which depends on the reached service level. This is a minor, financial action while SLA in practice can contain other actions (e.g., increasing workforce to resolve incidents). This action is further limited by the employed data model which only allows the specification of three SLOs (high, middle, low) and action guarantees (as refunds) for the two lower SLOs (middle, low).

The applications form of SLAs limits them to the specification of the service levels mentioned above. They do not contain the typical clauses (see Figure 2.1), such as legal or managerial provisions.

Limitations of the Blockchain

During development, the prototypical application was deployed to a development blockchain based on the truffle framework [18]. This Ethereum blockchain instantly creates new blocks on transactions without the latency of public blockchains. Transactions and the processing of smart contract costs gas, which is available in an unlimited capacity in this development blockchain. The low latency and missing costs are unrealistic parameters for a distributed application.

Limitations with User Interaction

The first limitation of the prototype regarding user interaction is a core issue of the blockchain's use of public-key cryptography. Since individuals identify themselves with their private key, the loss if this key is irreversible and thus restricts their access to their assets or smart contracts. If a consumer or provider loses their private key, the application remains in the current state, and the individual cannot control it anymore.

The implemented state channel distributed the power of providing the monitoring data among the participants and adds safeguards so that neither provider nor consumer can manipulate the application to their gain. While this is an advantage, the complicated handling of the state channel limits the ease-of-use of the prototype.

Limitations of the Monitoring Agent

The implemented monitoring agent produces simulated measurements of service performance based on the metric of server availability. This simulation of measurements limits the practical validity of this prototype, primarily since the availability metric offers only limited expressiveness on the performance of a server.

7. Result

In this chapter, we discuss the findings of this thesis, give answers to the research questions, conclude this thesis and give an outlook for future work.

7.1. Discussion

We found multiple challenges in the problem domain of SLAs of digital services (see Table 4.1). These relate to the possible non-fulfillment of obligations, the uncertainty of enforcement of fulfillment, the high effort for SLA processes, the low cost-to-benefit ratio, the lack of scalability for cloud computing and the lack of effectuation by IoT devices. These identified challenges lead to objectives of a solution that establishes trust among the service partners by enforcing the fulfillment of obligations in an automated way, thus reducing the manual effort and costs associated with SLAs. The automation of the SLA process supports the scalablility of cloud computing. Implementing an SLA as a smart contract is a possible solution to enable contractual interaction among IoT devices.

The implemented prototypical application utilizes smart contracts to create a permanent record of service performance and fulfill obligations, thereby supporting SLA of a digital service.

We implemented a prototypical application that effects service contracts and SLAs. Being IS scholars, we forwent the legal validity of these contracts and agreements as we do not presume to have substantiated legal knowledge. The uncertain legal soundness of smart contracts as service contracts and SLAs impedes the realization of such a blockchain-based application for service partners [25].

While the prototype demonstrates the possibility of implementing SLAs as smart contracts, it does so in a simulated setting with a very basic SLA that neglects provisions from real SLAs. To implement richer SLAs as smart contracts, additional research is required to transfer legal agreements to smart contract code (such as Surden [97]) and find unambiguous performance metrics that are interpretable by smart contracts.

With both application logic and data storage distributed in smart contracts on the blockchain, the application withdraws from the central control of a single service party and thus establishes trust in the execution of SLA actions and the validity of the service

performance data [25, 51]. The reduced risk of obligation non-fulfillment lessens the required trust to enter into service contracts when service performance and the chance of legal rights enforcement are uncertain. We regard the blockchain and its smart contracts as a useful technology to support services, where multiple parties interact with mutual distrust. Weber et al. [111] support this assessment by indicating the usefulness of the blockchain to support collaborative business processes.
Additionally, the prototypical application demonstrates the possibility of IoT devices to autonomously effect service contracts that employ SLAs to enforce service quality [25, 51]. The underlying smart contracts enable this business process without manual input.

The blockchain's benefits of establishing trust between service partners come at a cost. As shown in Table 6.1, the execution of each use case necessitates fees to pay the blockchain network for its service. As an example, ordering a service in our application would incur a transaction fee of $ 4.40, which the consumer pays. While we did not measure the execution cost of these use cases on a cloud-based application, Rimba et al. [90] showed that the execution of their business process is two times of magnitude more expensive with a blockchain when compared to cloud services. The calculated cost of the prototypical application relates to its execution on a public, proof-of-work Ethereum network. As the technical review indicates [55], other consensus mechanisms and the use of private or commissioned blockchain networks could lower these costs and render the application economically more feasible.
Aside from possible improvements by employing a different blockchain technology, the design of the prototype and its implementation with Ethereum as blockchain and the React framework as web frontend are suitable for blockchain-based applications [55]. The use of the MetaMask browser extension for interactions with the blockchain follows current practice [55]. It also demonstrates a flaw of the application due to its complex handling that expects users to possess the blockchain-related knowledge, thus adversely influencing user experience for uninformed users [25, 51].

A second complex concept of the prototype is the implemented state channel, which is a useful tool in need of an improved implementation with a user interface, that does not expose the technical functionality to the user [25, 51]. The state channel requires users to input arbitrary looking data to enable verifiable off-chain transactions. This concept decreases the amount of required on-chain transaction to save transaction fees on the payment handling use case of the prototype.
The state channel also serves as an oracle by transmitting off-chain transactions to the blockchain, as these transactions include the service performance data and thus transfer off-chain information to smart contracts. By integrating the oracle as part of the state channel, we save on transaction fees that are usually required for oracle services. A disadvantage of this push-based oracle is the lack of data in the web application when desired information has not been transferred to the smart contract [25]. Using a

distributed storage (e.g., IPFS, Swarm) could improve the artifact in this relation and lower the application's usage cost due to storing less data on the blockchain [55].

Another limiting factor of our developed application is its high latency on feedback for user interactions. Table 6.1 shows the long duration of about 25 seconds until a consumer receives a confirmation on buying a service. This latency stems from the blockchain's proof-of-work consensus mechanism and presents improvement opportunities for (1) accelerating the consensus mechanism and block creation as well as (2) developing mitigation strategies to decrease the impact of latency on the user experience.

Next to these two technical limitations, the current maturity of the blockchain technology impedes the use of smart contracts. The expert interviews [25, 51] show that providers and customers are not ready to embrace the technology. They lack the technical understanding and trust in the distributed system in addition to citing legal uncertainty and currency fluctuations as issues.

These disadvantages of our blockchain-based application show existing barriers to the practicability of employing blockchains and smart contracts for digital services.

7.2. Answers to the Research Questions

RQ1: How can smart contracts support SLAs of digital services?

Based on challenges identified within a literature review, we determined three central concepts on how smart contracts can support the execution of SLAs. Firstly, smart contracts can include obligations and corresponding actions in their program code. They execute these actions based on predefined conditions. Secondly, smart contracts serve as a data storage, that immutably stores service level information on the blockchain for secure record keeping and traceability of service performance. Thirdly, the digital nature and interface of smart contracts support its application for modern technologies: the automated execution of SLAs supports the scalability needed for cloud computing, and enables IoT devices to enter into service contracts, due to ensured fulfillment of obligations.

Chapter 4 describes the theoretical considerations that lead to these three concepts.

RQ2: How can required information about service performance be made available to smart contracts?

We researched information from academic literature and publications of the blockchain community to find oracles as a technology to provide information to smart contracts (see section 2.4). Current oracle solutions differ in characteristics of their data source, their architecture and their use of the push or pull principle. The oracle implementation of Weber et al. [111] is described as an agent of a process participant with a predefined flow of information to specific smart contracts. This selective, push-based solution is suitable for the requirements from a service partner's perspective. Thus, we follow Weber et al. [111] and implement an oracle in our application that transmits information to the application as ordered by a service partner (see subsection 5.3.2).

RQ3: What are approaches for the design and development of a blockchain-based application which supports SLAs of digital services?

We evaluated multiple blockchain technologies and related work to derive information for the design and development of our artifact (see chapter 2). We design the application's architecture with smart contracts for data storage and program logic in addition to a web frontend as a user interface. This offers the benefits of distributed data storage to create an immutable record of service performance and the distributed program execution to ensure the exercise of an SLA. For the implementation, we select Ethereum as blockchain technology, Solidity for smart contracts and React as the framework for the web frontend. We enable off-chain transactions with a state channel that integrates a push-principled, consensus-based oracle. The technical evaluation regards this selected approach as suitable for the design and development a blockchain-based application (see section 6.3). Chapter 5 describes the design and development of the artifact.

RQ4: How feasible is the prototypical application for supporting SLA of digital services?

The prototypical application is feasible for supporting SLAs with smart contracts by enabling their enforced execution, an immutable record of service performance and a turn-by-turn system for obligation fulfillment (see subsection 6.2.2). The technological implementation of the prototype demonstrates sound design decisions and presents a viable blockchain-based application, that includes the innovative concepts of smart contracts, oracles, and state channels (see section 6.3). The effectiveness of the prototypical application is reduced by its missing consideration of the legal validity of smart contract-based SLAs in addition to the high costs and high latency of the blockchain-based application. Further, the lack of knowledge about blockchains, fluctuations of cryptocurrency and distrust of the blockchain further impede the practicability of the developed artifact.

7.3. Conclusion

This thesis set out to determine the feasibility of smart contracts for supporting SLAs of digital services. We applied DSR according to the methodology of Peffers et al. [81]. Founded on a knowledge base generated from literature reviews, we identified the following three most relevant challenges in the problem domain: (1) the uncertainty regarding the fulfillment of obligations and enforcement of rights, (2) the low cost-to-benefit ratio of SLAs, and (3) the lacking support of SLAs for modern technologies (e.g., cloud computing, IoT).
The blockchain technology provides new prospects for the secure transfer of digital value and smart contracts that offer the enforced fulfillment of therein programmed obligations, interfaces to machines and are more flexible than regular textual contracts.

We designed and developed a prototypical, blockchain-based application as DSR artifact to generate knowledge on employing smart contracts for supporting SLAs of digital services. Our viable implementation draws on the current practice of blockchain-based applications and presents a technologically innovative artifact, which employs a state channel as a multi-sourced oracle to enable turn-by-turn service delivery.

With our prototype, we demonstrated a possible application domain for the blockchain technology. Our solution achieved two objectives: the first one is the automated, enforced fulfillment of SLA obligations to reduce the uncertainty of contract performance and to improve process efficiency. The second objective is to equip SLAs with the required functionalities for supporting cloud computing and the IoT, meaning to improve scalability of SLAs and enabling of contractual device interaction. Thus, the blockchain in our prototype acts as a digital institution to facilitate service relationships in digital service co-creation.

Additionally, the evaluation of our artifact highlighted issues with employing smart contracts for supporting SLAs of digital services. The practicability of our blockchain-based application is restricted by its high usage costs and its high latency, as well as user concerns against the technology.

In conclusion, this thesis implies that the feasibility of utilizing smart contracts for supporting SLAs of digital services is limited by the inadequate maturity of blockchain technology and user's scepticism. The presented use cases are valid solutions to the problem domain. Overcoming the limitations requires (1) the improvement of blockchain technology, (2) the training of users to generate an understanding of the technology as well as (3) the reduction of risks of legal uncertainty and cryptocurrency fluctuations.

7.4. Implications and Outlook

The following section presents five implications for research and practice. It finished with an outlook on the future progress of blockchain technology.

Implications

Sales argument for service providers The developed artifact in this thesis demonstrates a unique sales argument for providers. By incorporating the trust-enabling features of the blockchain technology and automated fulfillment of obligations of SLAs into his business model, a provider could offer the guaranteed enforcement of SLAs as a value proposition.

Suitable domain for future research We developed a prototypical application within a simulated context to research the feasibility of smart contracts. To gain more insights into the practicability of such an application, we recognize the necessity for real-world tests. The server hosting domain of this thesis appears as a suitable testing domain for future research as it contains technologically minded people that might hold knowledge about blockchain technology and are interested in this innovative technology.

Further development of the artifact In this thesis, we designed and developed a viable blockchain-based application. While our prototype is limited in its scope and functionality (see chapter 5), its code is extensible for additional features and use cases. The generated artifact, its design and development approach can guide researchers as well as practitioners in their efforts to develop blockchain-based applications. Therefore, the source code is available on GitHub [119].

Improvement of user experience An inherent issue of the technology is its complexity. The concept of blockchains is tough to understand for peers in the IS domain and even tougher for non-technologically minded people. Current interfaces to the blockchain (e.g., MetaMask) require the handling of cryptographic keys and lack a standard for interaction with distributed applications. The evaluation shows that the state channel of our prototype is too complex for users. Hence, future research could focus on building an intuitive user interface as layer on top of the technical concepts to reduce the complexity for users.

Further legal research The evaluation of our artifact highlighted the experts' concerns of the legal validity of smart contract-based SLAs,their actions and compliance with data protection laws. Therefore, we regard legal research as important future work. We encourage legal scholars to establish a legal assessment of the blockchain technology, especially of smart contracts and their legal status as well as the validity of their executed actions. Future studies on translating legal text to smart contract code could allow rich contractual actions effected the blockchain.

Outlook

The blockchain originated from solving problems related to cryptocurrencies [75]. On its quest for applications, it evolved to feature smart contracts, which enable the execution of distributed code [10]. The blockchain technology shows promising features to facilitate service relationships and collaborative processes. The rapid development of blockchain technology improves its maturity and finds solutions to current problems. We are eager to follow the future progress of blockchain technology and the accompanying development of innovative applications to see where the quest leads.

A. Appendix

A.1. Use Cases

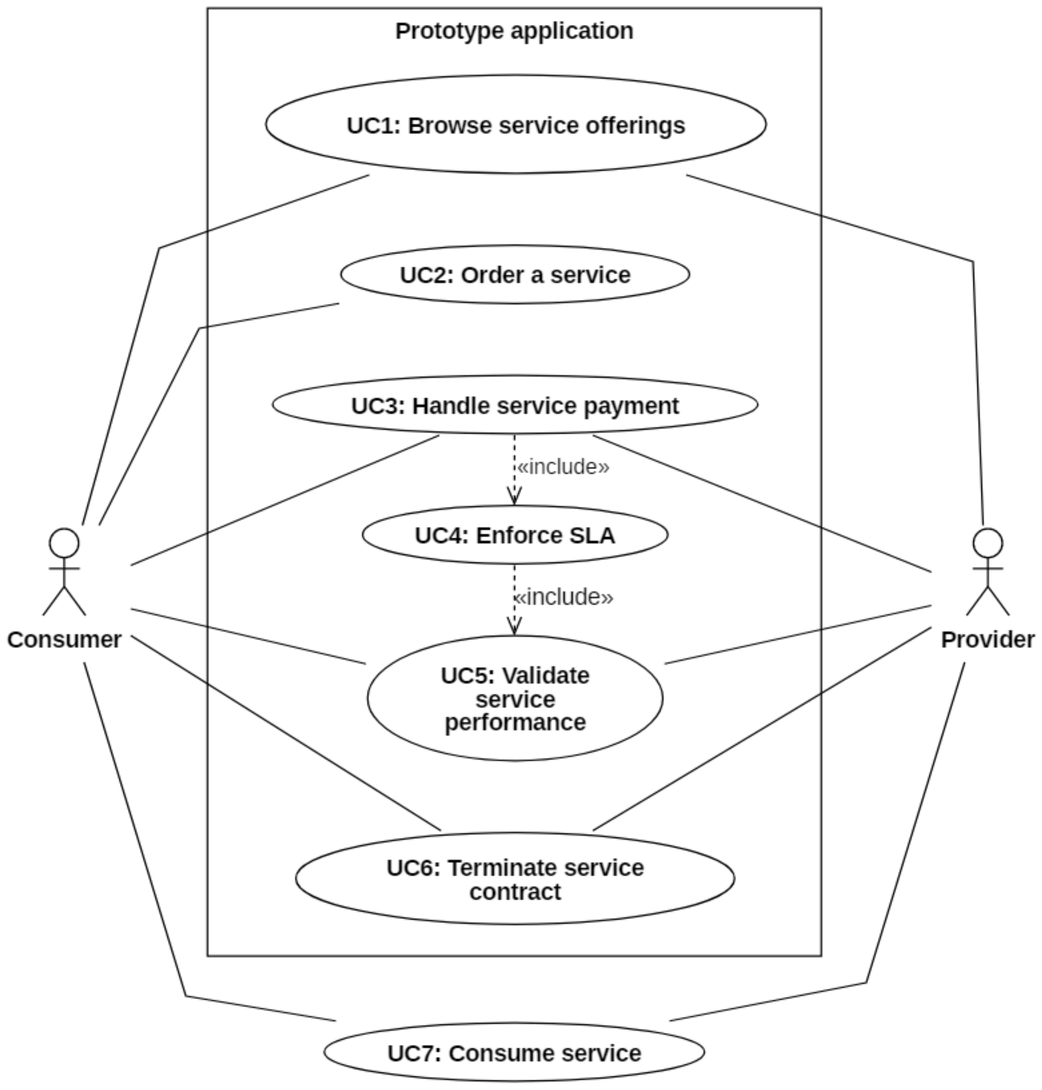

Prototype application
UC1: Browse service offerings
UC2: Order a service
UC3: Handle service payment
«include»
UC4: Enforce SLA
«include»
UC5: Validate service performance
UC6: Terminate service contract
UC7: Consume service
Consumer
Provider

Figure A.1.: Designed use cases of the prototypical application

A.2. Architecture of the Web Frontend

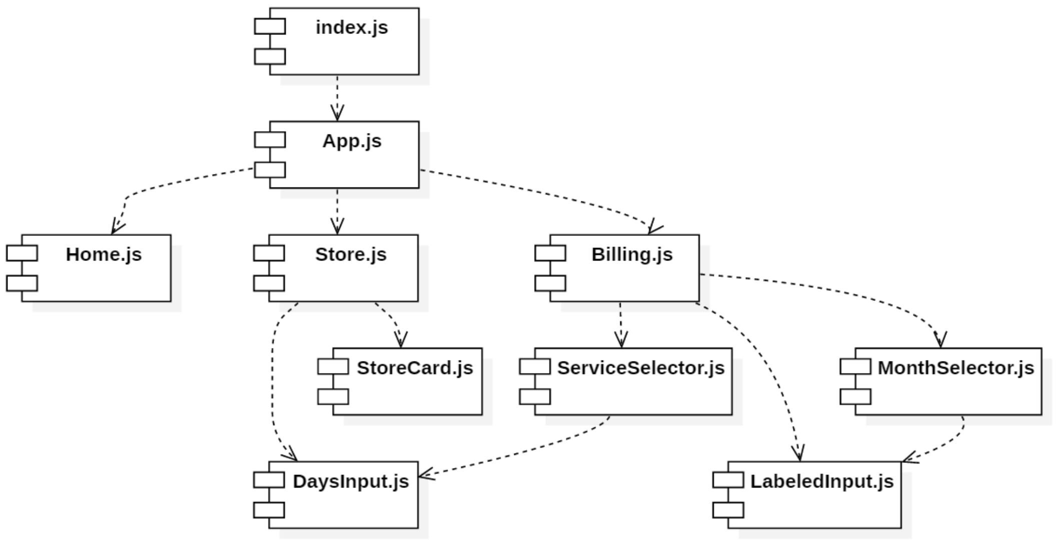
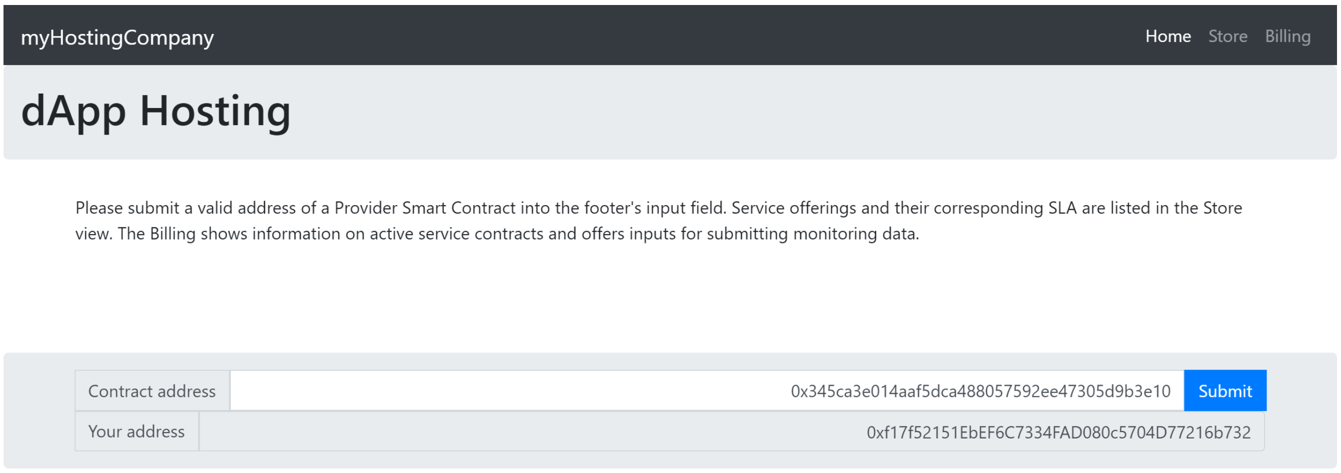


Figure A.2.: Architecture of the web frontend

A.3. Screenshots of the Web Frontend

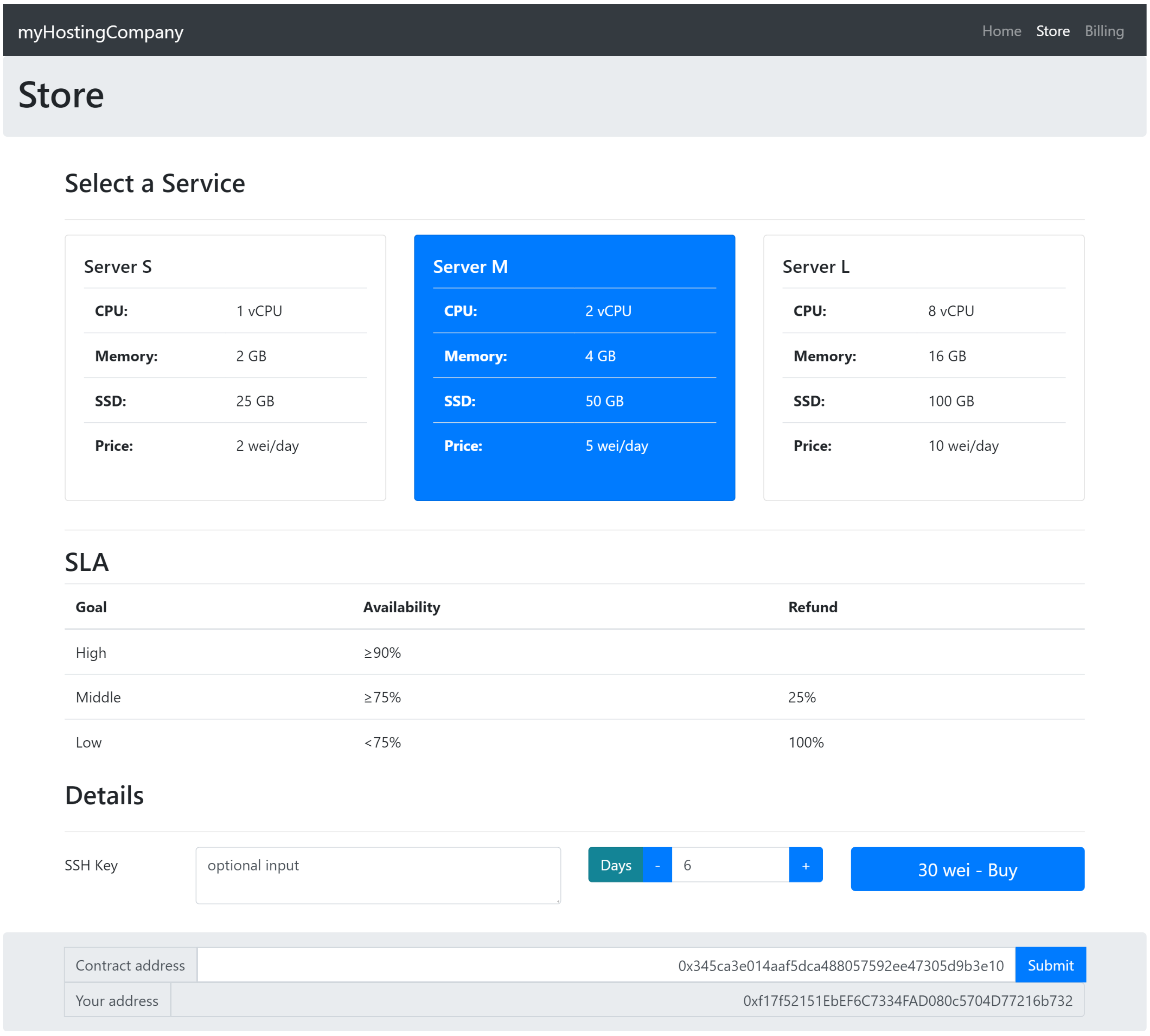


dApp Hosting

Please submit a valid address of a Provider Smart Contract into the footer's input field. Service offerings and their corresponding SLA are listed in the Store view. The Billing shows information on active service contracts and offers inputs for submitting monitoring data.

Contract address	0x345ca3e014aaf5dca488057592ee47305d9b3e10
Your address	0xf17f52151EbEF6C7334FAD080c5704D77216b732

Figure A.3.: Screenshot of the Home view



myHostingCompany

Home Store Billing

Store

Select a Service

Server S	
CPU:	1 vCPU
Memory:	2 GB
SSD:	25 GB
Price:	2 wei/day

Server M	
CPU:	2 vCPU
Memory:	4 GB
SSD:	50 GB
Price:	5 wei/day

Server L	
CPU:	8 vCPU
Memory:	16 GB
SSD:	100 GB
Price:	10 wei/day

SLA

Goal	**Availability**	**Refund**
High	$\geq$90%	
Middle	$\geq$75%	25%
Low	<75%	100%

Details

SSH Key: optional input

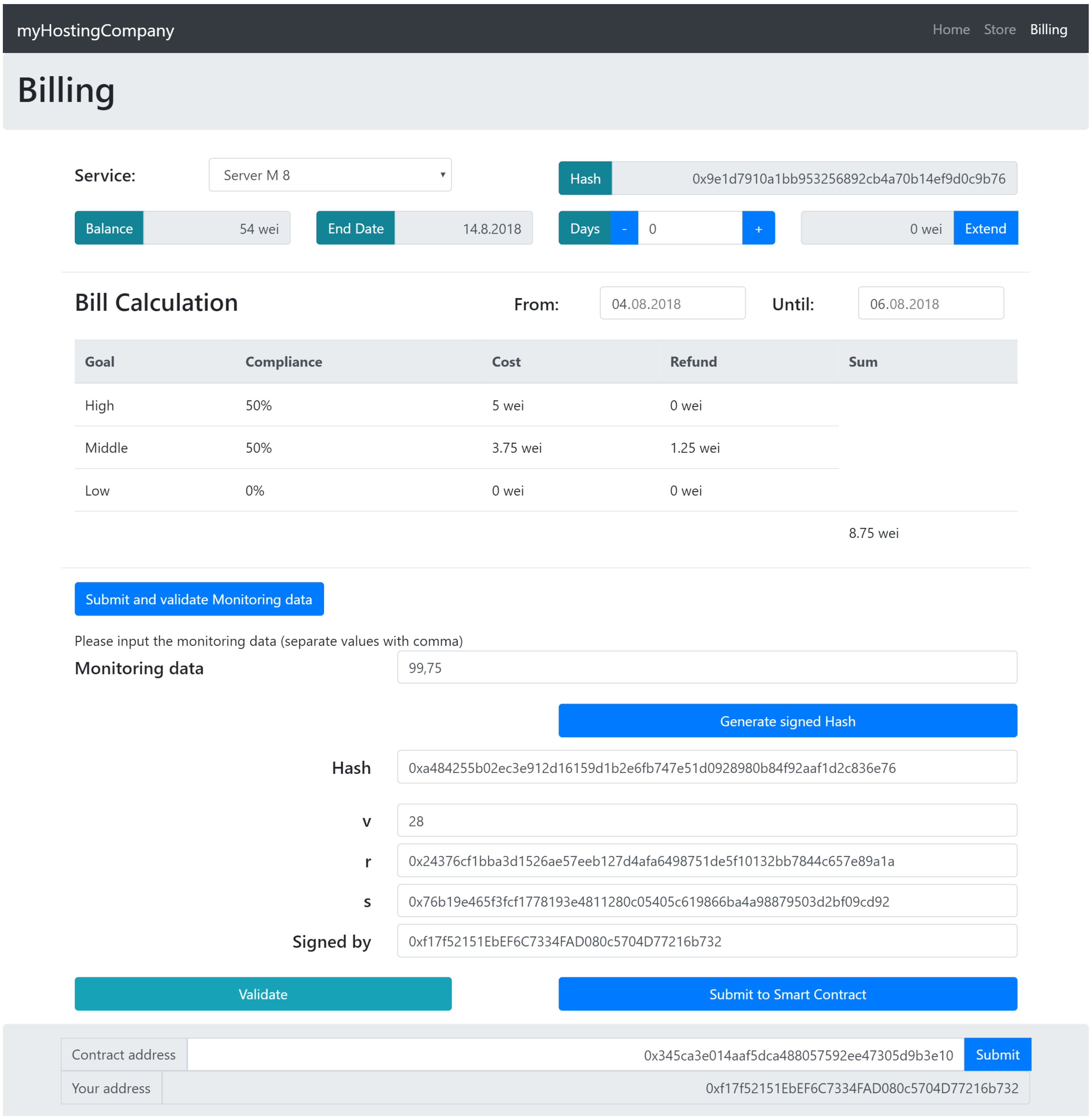
Days - 6 +

30 wei - Buy

Contract address: 0x345ca3e014aaf5dca488057592ee47305d9b3e10 Submit

Your address: 0xf17f52151EbEF6C7334FAD080c5704D77216b732

Figure A.4.: Screenshot of the Store view



Billing

Service: Server M 8

Hash 0x9e1d7910a1bb953256892cb4a70b14ef9d0c9b76

Balance 54 wei

End Date 14.8.2018

Days - 0 +

0 wei Extend

Bill Calculation

From: 04.08.2018

Until: 06.08.2018

Goal	Compliance	Cost	Refund	Sum
High	50%	5 wei	0 wei	
Middle	50%	3.75 wei	1.25 wei	
Low	0%	0 wei	0 wei	
				8.75 wei

Please input the monitoring data (separate values with comma)

Monitoring data 99,75

Hash 0xa484255b02ec3e912d16159d1b2e6fb747e51d0928980b84f92aaf1d2c836e76

v 28

r 0x24376cf1bba3d1526ae57eeb127d4afa6498751de5f10132bb7844c657e89a1a

s 0x76b19e465f3fcf1778193e4811280c05405c619866ba4a98879503d2bf09cd92

Signed by 0xf17f52151EbEF6C7334FAD080c5704D77216b732

Contract address 0x345ca3e014aaf5dca488057592ee47305d9b3e10 Submit

Your address 0xf17f52151EbEF6C7334FAD080c5704D77216b732

Figure A.5.: Screenshot of the Billing view

A.4. Architecture of the Smart Contracts

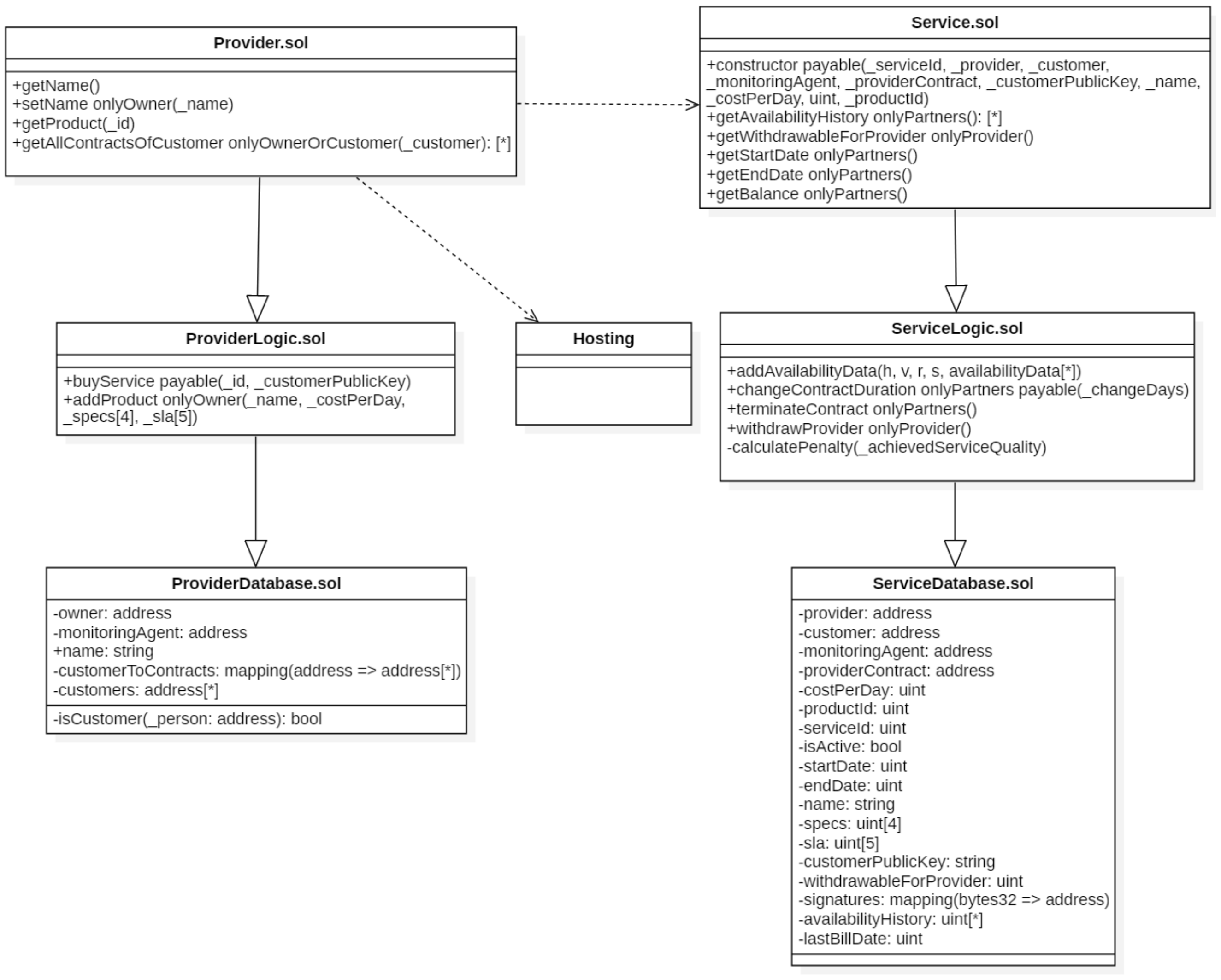

Provider.sol
+getName()
+setName onlyOwner(_name)
+getProduct(_id)
+getAllContractsOfCustomer onlyOwnerOrCustomer(_customer): [*]
Service.sol
+constructor payable(_serviceId, _provider, _customer, _monitoringAgent, _providerContract, _customerPublicKey, _name, _costPerDay, uint, _productId)
+getAvailabilityHistory onlyPartners(): [*]
+getWithdrawableForProvider onlyProvider()
+getStartDate onlyPartners()
+getEndDate onlyPartners()
+getBalance onlyPartners()
ProviderLogic.sol
+buyService payable(_id, _customerPublicKey)
+addProduct onlyOwner(_name, _costPerDay, _specs[4], _sla[5])
Hosting
ServiceLogic.sol
+addAvailabilityData(h, v, r, s, availabilityData[*])
+changeContractDuration onlyPartners payable(_changeDays)
+terminateContract onlyPartners()
+withdrawProvider onlyProvider()
-calculatePenalty(_achievedServiceQuality)
ProviderDatabase.sol
-owner: address
-monitoringAgent: address
+name: string
-customerToContracts: mapping(address => address[*])
-customers: address[*]
-isCustomer(_person: address): bool
ServiceDatabase.sol
-provider: address
-customer: address
-monitoringAgent: address
-providerContract: address
-costPerDay: uint
-productId: uint
-serviceId: uint
-isActive: bool
-startDate: uint
-endDate: uint
-name: string
-specs: uint[4]
-sla: uint[5]
-customerPublicKey: string
-withdrawableForProvider: uint
-signatures: mapping(bytes32 => address)
-availabilityHistory: uint[*]
-lastBillDate: uint

Figure A.6.: Diagram of solidity Smart Contracts

A.5. Sequence Diagrams

The following presents the sequence of interactions between components of the application for each use case in the style of UML sequence diagrams.

A.5.1. UC1: Browse Service Offerings

When a user opens the web frontend, the React component App.js first queries the count of service offerings and then requests each product. When all products have been retrieved, App.js forwards these to its child component Store.js, which handles the display of these service offerings in the web frontend.

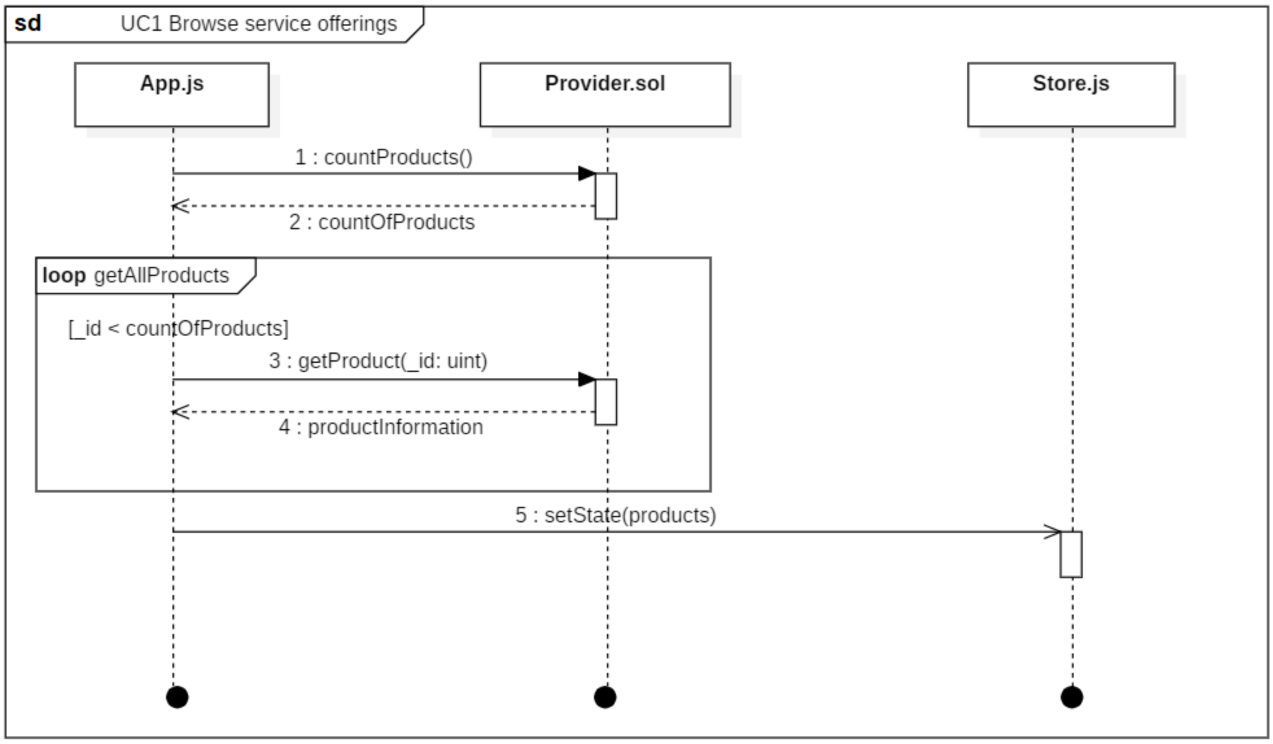


Figure A.7.: Sequence diagram of use case 1: Browse service offerings

A.5.2. UC2: Order a Service

Ordering a service is initiated by a user in the Store.js view, which triggers the deployed *Provider* contract with the function *buyService*. The *Provider* contract initializes a new *Service* contract for that order and continues with setting the necessary service and SLA details. By calling the function *changeContractDuration()* with an attached value, the *Provider* contract transfers funds to the *Service* contract which calculates the contract duration based on the transmitted value and daily cost of the service.

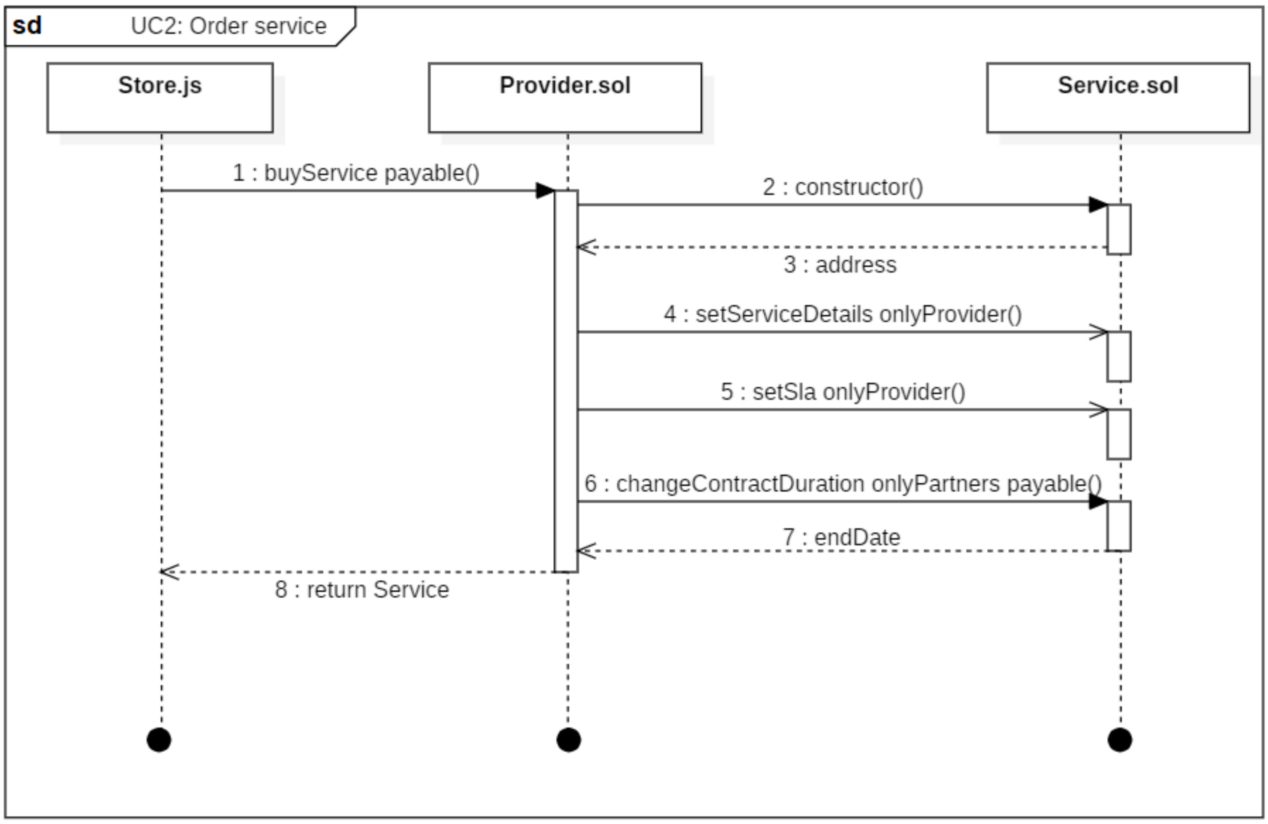


Figure A.8.: Sequence diagram of use case 2: Order a service

A.5.3. UC3: Handle Service Payment

The service payment rests upon the monitoring data. Provider and consumer sign the monitoring data in the Billing.js view of the web frontend and provide it to the state channel via the *addAvailabilityData* function, which verifies that both parties agreed to the data. The *Service* contract then initiates the payment to the provider by calculating possible penalties based on the provided monitoring data. If the service performed beneath a service level goal, the associated reimbursement penalty is deducted from the service fee. This function implements the use cases of "Validate service performance (UC5)" and "Enforce SLA" (UC4). After the calculation, the *Service* contract applies the Withdrawal-Pattern and internally holds the payment until the provider requests a withdrawal.

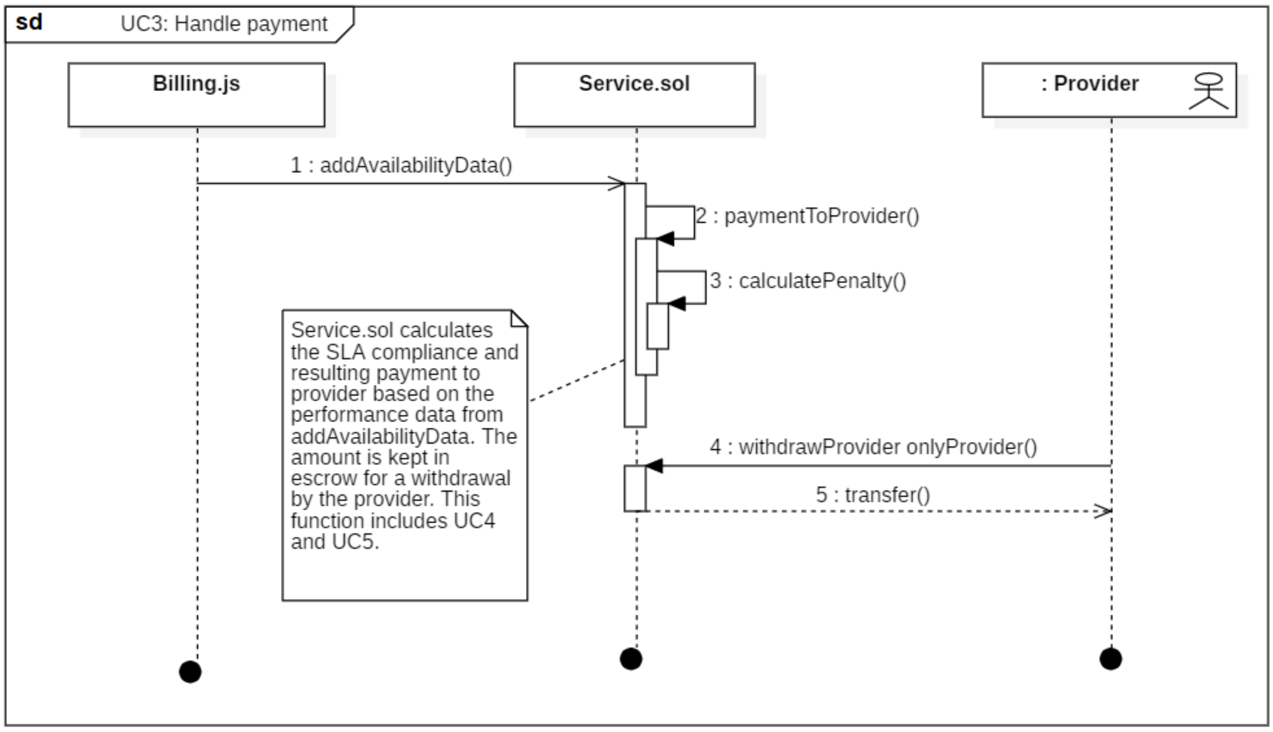

sd
UC3: Handle payment
Billing.js
Service.sol
: Provider
1 : addAvailabilityData()
2 : paymentToProvider()
3 : calculatePenalty()
Service.sol calculates the SLA compliance and resulting payment to provider based on the performance data from addAvailabilityData. The amount is kept in escrow for a withdrawal by the provider. This function includes UC4 and UC5.
4 : withdrawProvider onlyProvider()
5 : transfer()

Figure A.9.: Sequence diagram of use case 3: Handle service payment

A.5.4. UC6: Terminate Service Contract

The *changeContractDuration* function implements the termination of a service contract by setting the end date to tomorrow and transferring the remaining funds back to the consumer. The Billing.js view in the web frontend provides access to this function.

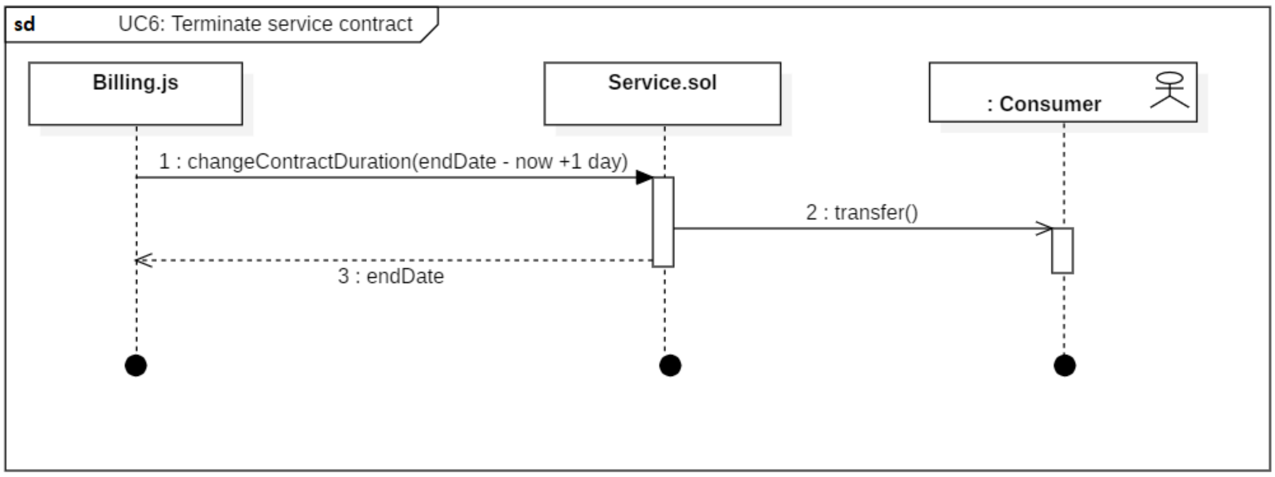

sd
UC6: Terminate service contract
Billing.js
Service.sol
: Consumer
1 : changeContractDuration(endDate - now +1 day)
2 : transfer()
3 : endDate

Figure A.10.: Sequence diagram of use case 6: Terminate service contract

Acronyms

DSR Design Science Research

DSRM Design Science Research Methodology

EVM Ethereum virtual machine

IPFS InterPlanetary File System

IS Information Systems

IT Information Technology

QoS Quality of Service

SLA Service Level Agreement

SLI Service Level Indicator

SLO Service Level Objective

UML Unified Modeling Language

USD United States Dollar

List of Figures

1.1. DSRM process by Peffers et al. [81] 4

2.1. Content of Service Level Agreements [5, 79, 80, 91, 102, 103] 8

A.1. Designed use cases of the prototypical application 68
A.2. Architecture of the web frontend 69
A.3. Screenshot of the Home view 70
A.4. Screenshot of the Store view 71
A.5. Screenshot of the Billing view 72
A.6. Diagram of solidity Smart Contracts 73
A.7. Sequence diagram of use case 1: Browse service offerings 74
A.8. Sequence diagram of use case 2: Order a service 75
A.9. Sequence diagram of use case 3: Handle service payment 76
A.10.Sequence diagram of use case 6: Terminate service contract 77

List of Tables

2.1. Overview and characteristics of oracle services 19

4.1. Challenges in the domain of digital service SLA 33
4.2. Design objectives for the IT artifact 38

5.8. Status of the implementation of use cases 51

6.1. Cost and latency measurement results of the prototype 58

Bibliography

[1] Amazon. *CloudFront SLA*. 2018. URL: `https://aws.amazon.com/de/cloudfront/sla/` (visited on 08/09/2018).

[2] M. Bartoletti and L. Pompianu. *An empirical analysis of smart contracts: platforms, applications and design patterns*. Cagliari, Italy, Mar. 18, 2017.

[3] R. G. Berbée, P. Gemmel, B. Droesbeke, H. Casteleyn, and D. Vandaele. "Evaluation of hospital service level agreements." In: *International journal of health care quality assurance* 22.5 (2009), pp. 483–497. ISSN: 0952-6862. DOI: `10.1108/09526860910975599`.

[4] D. Berberova and B. Bontchev. "Design of Service Level Agreements for Software Services." In: *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*. Ed. by B. Rachev. New York, NY: ACM, 2009. ISBN: 9781605589862.

[5] T. Berger. "Konzeption und Management von Service-Level-Agreements für IT-Dienstleistungen." Dissertation. Darmstadt: TU Darmstadt, 2005.

[6] bitcoinfeesinfo. *Bitcoin Transaction Fees*. 2018. URL: `https://bitcoinfees.info/` (visited on 07/17/2018).

[7] BlockchainHub. *Blockchain Oracles*. 2018. URL: `https://blockchainhub.net/blockchain-oracles/` (visited on 07/03/2018).

[8] M. Braun. *Ergebnisorientierte Leistungsvereinbarungen im Technischen Gebäudemanagement: Ergebnisorientierte Leistungsvereinbarungen im Technischen Gebäudemanagement*. Saarbrücken: VDM Müller, 2008. ISBN: 9783836456753.

[9] H. Brocke, F. Uebernickel, and W. Brenner. "Customizing IT Service Agreements as a Self Service by means of Productized Service Propositions." In: *2011 44th Hawaii international conference on system sciences*. Piscataway, N. J.: IEEE, 2011, pp. 1–10. ISBN: 978-1-4244-9618-1. DOI: `10.1109/HICSS.2011.134`.

[10] V. Buterin. *A next-generation smart contract and decentralized application platform*. 2014.

[11] V. Buterin. *SchellingCoin: A Minimal-Trust Universal Data Feed*. 2014. URL: `https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/` (visited on 07/12/2018).

[12] V. Buterin. *The Meaning of Decentralization – Vitalik Buterin – Medium*. 2017. URL: `https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274` (visited on 07/05/2018).

[13] R. Buyya. "Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility." In: *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009*. Ed. by F. Cappello. Piscataway, NJ: IEEE, 2009, p. 1. ISBN: 978-1-4244-3935-5. DOI: `10.1109/CCGRID.2009.97`.

[14] C. Cachin. "Architecture of the hyperledger blockchain fabric." In: *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. Vol. 310. 2016.

[15] C. D. Clack, V. A. Bakshi, and L. Braine. *Smart Contract Templates: essential requirements and design options*. Dec. 15, 2016.

[16] C. D. Clack, V. A. Bakshi, and L. Braine. *Smart Contract Templates: Foundations, design landscape and research directions*. 2016.

[17] ConsenSys. *Subscription Services on the Blockchain: ERC-948 – ConsenSys Media*. 2018. URL: `https://media.consensys.net/subscription-services-on-the-blockchain-erc-948-6ef64b083a36` (visited on 07/13/2018).

[18] ConsenSys. *Truffle Suite - Your Ethereum Swiss Army Knife*. 2018. URL: `https://truffleframework.com/` (visited on 07/28/2018).

[19] P. Cuccuru. "Beyond bitcoin: An early overview on smart contracts." In: *International Journal of Law and Information Technology* 25.3 (2017), pp. 179–195. ISSN: 0967-0769. DOI: `10.1093/ijlit/eax003`.

[20] M. Di Ferrante. *Ethereum Payment Channel in 50 Lines of Code – Matthew Di Ferrante – Medium*. 2017. URL: `https://medium.com/@matthewdif/ethereum-payment-channel-in-50-lines-of-code-a94fad2704bc` (visited on 06/17/2018).

[21] E. Di Pascale. *SLA-solidity*. 2017. URL: `https://bitbucket.org/edipascale/sla-solidity/overview` (visited on 07/13/2018).

[22] E. Di Pascale, J. McMenamy, I. Macaluso, and L. Doyle. *Smart Contract SLAs for Dense Small-Cell-as-a-Service*. 2017.

[23] Didil. *Off-Chain Data Storage: Ethereum & IPFS – Didil – Medium*. 2017. URL: `https://medium.com/@didil/off-chain-data-storage-ethereum-ipfs-570e030432cf` (visited on 07/17/2018).

[24] DigitalOcean. *Droplet Policies and Procedures | DigitalOcean Product Documentation*. 2018. URL: `https://www.digitalocean.com/docs/accounts/policies/droplet-policies/` (visited on 08/09/2018).

[25] A. Dillinger. *Interview with a provider*. 2018.

[26] C. Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. 8. Aufl. München: De Gruyter, 2013. ISBN: 9783486721386. DOI: `10.1524/9783486735871`.

[27] B. Egelund-Müller, M. Elsman, F. Henglein, and O. Ross. "Automated Execution of Financial Contracts on Blockchains." In: *Business & Information Systems Engineering* 59.6 (2017), pp. 457–467. ISSN: 2363-7005. DOI: `10.1007/s12599-017-0507-z`.

[28] S. Ellis, A. Juels, and S. Nazarov. *Chainlink: A Decentralized Oracle Network*. Sept. 4, 2017.

[29] S. Eskandari, J. Clark, V. Sundaresan, and M. Adham. *On the feasibility of decentralized derivatives markets*. 2017.

[30] ETH Gas Station. *ETH Gas Station*. 2018. URL: `https://ethgasstation.info/index.php` (visited on 07/17/2018).

[31] Ethereum Foundation. *ethereum/go-ethereum*. 2018. URL: `https://github.com/ethereum/go-ethereum` (visited on 08/07/2018).

[32] Ethereum Foundation. *ethereum/web3.js*. 2018. URL: `https://github.com/ethereum/web3.js/` (visited on 07/28/2018).

[33] Ethereum Foundation. *ethereum/wiki*. 2018. URL: `https://github.com/ethereum/wiki/wiki/Programming-languages-intro` (visited on 08/14/2018).

[34] Etherscan. *Ethereum Charts and Statistics*. 2018. URL: `https://etherscan.io/charts` (visited on 08/01/2018).

[35] Ethersphere. *Swarm: distributed storage platform*. 2016. URL: `https://github.com/ethersphere/swarm` (visited on 06/03/2018).

[36] Facebook Inc. *React - A JavaScript library for building user interfaces*. 2018. URL: `https://reactjs.org/` (visited on 07/28/2018).

[37] S. Farrell, H. Machin, and R. Hinchliffe. *Lost and found in smart contract translation – considerations in transitioning to automation in legal architecture*. 2016.

[38] Flightright. *Flugverspätung? Jetzt Entschädigung sichern | Flightright*. 2018. URL: `https://www.flightright.de/ihre-rechte/flugverspaetung-entschaedigung` (visited on 07/09/2018).

[39] N. Foundation. *Node.js*. 2018. URL: `https://nodejs.org/en/` (visited on 07/28/2018).

[40] C. K. Frantz and M. Nowostawski. "From Institutions to Code: Towards Automated Generation of Smart Contracts." In: *IEEE 1st International Workshops on Foundations and Applications of Self-* Systems*. Ed. by S. Elnikety, P. R. Lewis, and C. Müller-Schloer. Los Alamitos, California, Washington, and Tokyo: Conference Publishing Services, IEEE Computer Society, 2016, pp. 210–215. ISBN: 978-1-5090-3651-6. DOI: `10.1109/FAS-W.2016.53`.

[41] G. Greenspan. *Why Many Smart Contract Use Cases Are Simply Impossible - CoinDesk*. 2016. URL: `https://www.coindesk.com/three-smart-contract-misconceptions/` (visited on 12/18/2017).

[42] S. Gregor and A. Hevner. "Positioning and Presenting Design Science Research for Maximum Impact." In: *Management Information Systems Quarterly* 37.2 (2013), pp. 337–355.

[43] Z. Hess, Y. Malahov, and J. Pettersson. *Aeternity Blockchain: The trustless, decentralized and purely functional oracle machine*. 2017. URL: `https://aeternity.com/` (visited on 02/15/2018).

[44] A. Hevner. "A Three Cycle View of Design Science Research." In: *Scandinavian Journal of Information Systems* 19.2 (2007).

[45] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research." In: *MIS Quarterly* 28.1 (2004), pp. 75–105. ISSN: 02767783.

[46] R. Hitchens. *contract development - Are there well-solved and simple storage patterns for Solidity? - Ethereum Stack Exchange*. 2017. URL: `https://ethereum.stackexchange.com/questions/13167/are-there-well-solved-and-simple-storage-patterns-for-solidity` (visited on 07/17/2018).

[47] R. Hull. "Blockchain: Distributed Event-based Processing in a Data-Centric World." In: *DEBS'17*. Ed. by Unknown. New York, New York: The Association for Computing Machinery, 2017, pp. 2–4. ISBN: 9781450350655. DOI: `10.1145/3093742.3097982`.

[48] R. Hull, V. S. Batra, Y.-M. Chen, A. Deutsch, F. F. T. Heath III, and V. Vianu. "Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes." In: *Service-Oriented Computing*. Ed. by Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri. Cham: Springer International Publishing, 2016, pp. 18–36. ISBN: 978-3-319-46295-0.

[49] Intel Corporation. *Intel SGX for Dummies (Intel SGX Design Objectives) | Intel Software*. 2013. URL: `https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx` (visited on 07/12/2018).

[50] N. Karten. “With Service Level Agreements, Less is More.” In: *Information Systems Management* 21.4 (2004), pp. 43–44. ISSN: 1058-0530. DOI: `10.1201/1078/44705.21.4.20040901/84186.5`.

[51] M. Kauk. *Interview with a customer*. 2018.

[52] A. Keller and H. Ludwig. “The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services.” In: *Journal of Network and Systems Management* 11.1 (2003), pp. 57–81. ISSN: 10647570. DOI: `10.1023/A:1022445108617`.

[53] M. Klems. *DESEMA: Prototype of a blockchain-based decentralized service marketplace.* 2017. URL: `https://github.com/markusklems/desema` (visited on 07/13/2018).

[54] M. Klems, J. Eberhardt, S. Tai, S. Härtlein, S. Buchholz, and A. Tidjani. “Trustless Intermediation in Blockchain-Based Decentralized Service Marketplaces.” In: *Service-oriented computing*. Ed. by E. M. Maximilien. Vol. 10601. LNCS sublibrary. SL 2, Programming and software engineering. Cham, Switzerland: Springer, 2017, pp. 731–739. ISBN: 978-3-319-69034-6. DOI: `10.1007/978-3-319-69035-3_53`.

[55] A. König. *Technical review of the artifact*. 2018.

[56] A. Kothapalli and C. Cordi. *A Bribery Framework using Smart Contrats*. 2017.

[57] R. Koulu. “Blockchains and Online Dispute Resolution: Smart Contracts as an Alternative to Enforcement.” In: *SCRIPTed* 13.1 (2016), pp. 40–69. DOI: `10.2966/scrip.130116.40`.

[58] J. de Kruijff and H. Weigand. “Ontologies for Commitment-Based Smart Contracts.” In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Ed. by H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, and R. Meersman. Cham: Springer International Publishing, 2017, pp. 383–398. ISBN: 978-3-319-69459-7.

[59] J. de Kruijff and H. Weigand. *Towards a Blockchain Ontology*. Tilburg, 2017.

[60] P. Labs. *IPFS is the Distributed Web*. 2014. URL: `https://ipfs.io/` (visited on 07/17/2018).

[61] S. Leimeister, M. Böhm, C. Riedl, and H. Krcmar. “The Business Perspective of Cloud Computing: Actors, Roles and Value Networks.” In: *ECIS 2010 Proceedings* (2010).

[62] C. Lim, T. Saw, and C. Sargeant. *Smart Contracts: Bridging the Gap Between Expectation and Reality*. Oxford Business Law Blog, 2016.

[63] Y. Liu, Q. Lu, X. Xu, L. Zhu, and H. Yao. "Applying Design Patterns in Smart Contracts." In: *BLOCKCHAIN - ICBC 2018*. Ed. by S. Chen, H. Wang, and L.-J. Zhang. Vol. 10974. Lecture Notes in Computer Science. [S.l.]: Springer, 2018, pp. 92–106. ISBN: 978-3-319-94477-7. DOI: 10.1007/978-3-319-94478-4_7.

[64] T. Locher, S. Obermeier, and Y.-A. Pignolet. *When Can a Distributed Ledger Replace a Trusted Third Party?*

[65] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. *Web Service Level Agreement (WSLA) Language Specification*. 2003.

[66] Machina Research. *Service Level Agreements in M2M and IoT*. 2014. URL: https://machinaresearch.com/report/service-level-agreements-in-m2m-and-iot/ (visited on 07/09/2018).

[67] E. Marilly, O. Martinot, S. Betge-Brezetz, and G. Delegue. "Requirements for service level agreement management." In: *2002 IEEE workshoop on IP operations and management*. IEEE, 2002, pp. 57–62. ISBN: 0-7803-7658-7. DOI: 10.1109/IPOM.2002.1045756.

[68] E. Marilly, O. Martinot, H. Papini, and D. Goderis. "Service level agreements: a main challenge for next generation networks." In: *ECUMN 2002*. IEEE, 2002, pp. 297–304. ISBN: 0-7803-7422-3. DOI: 10.1109/ECUMN.2002.1002118.

[69] J. Mendling, I. Weber, W. van der Aalst, J. Vom Brocke, C. Cabanillas, F. Daniel, S. Debois, C. Di Ciccio, M. Dumas, S. Dustdar, et al. "Blockchains for business process management-challenges and opportunities." In: *arXiv preprint arXiv:1704.03610* (2017).

[70] M. Mikeln and L. Perović. *Eventum: Platform for Decentralized Real-World Data Feeds*. 2018. URL: https://eventum.network (visited on 02/16/2018).

[71] D. Miorandi, S. Sicari, F. de Pellegrini, and I. Chlamtac. "Internet of things: Vision, applications and research challenges." In: *Ad Hoc Networks* 10.7 (2012), pp. 1497–1516. ISSN: 15708705. DOI: 10.1016/j.adhoc.2012.02.016.

[72] Monax. *The Five Types Model*. 2016. URL: https://github.com/monax/monax/blob/master/docs/solidity/solidity_1_the_five_types_model.md (visited on 05/29/2018).

[73] J. Murkin, R. Chitchyan, and A. Byrne. "Enabling peer-to-peer electricity trading." In: *4th International Conference on ICT for Sustainability*. 2016, pp. 234–235.

[74] Nagios. *Nagios - The Industry Standard In IT Infrastructure Monitoring*. 2018. URL: https://www.nagios.org/ (visited on 08/09/2018).

[75] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[76] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies*. 2016.

[77] Oraclize. *Oraclize - blockchain oracle service, enabling data-rich smart contracts*. 2016. URL: `http://www.oraclize.it/` (visited on 07/11/2018).

[78] G. Oswald, D. Soto Setzke, T. Riasanow, and H. Krcmar. "Technologietrends in der digitalen Transformation." In: *DIGITALE TRANSFORMATION*. Ed. by G. Oswald and H. Krcmar. Informationsmanagement und digitale Transformation. [S.l.]: Gabler, 2018, pp. 11–34. ISBN: 978-3-658-22623-7. DOI: `10.1007/978-3-658-22624-4_3`.

[79] A. Paschke and M. Bichler. "Knowledge representation concepts for automated SLA management." In: *Decision Support Systems* 46.1 (2008), pp. 187–205. ISSN: 01679236. DOI: `10.1016/j.dss.2008.06.008`.

[80] A. Paschke and E. Schnappinger-Gerull. "A Categorization Scheme for SLA Metrics." In: *Service Oriented Electronic Commerce* 80.25-40 (2006), p. 14.

[81] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. "A Design Science Research Methodology for Information Systems Research." In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77. DOI: `10.2753/MIS0742-1222240302`.

[82] J. Poon and T. Dryja. *Lightning Network: Scalable, Instant Bitcoin/Blockchain Transactions*. 2016. URL: `https://lightning.network/` (visited on 07/17/2018).

[83] N. Prat, I. Comyn-Wattiau, and J. Akoka. "ARTIFACT EVALUATION IN INFORMATION SYSTEMS DESIGN-SCIENCE RESEARCH – A HOLISTIC VIEW." In: *PACIS 2014 Proceedings* (2014).

[84] J. Pries-Heje, R. Baskerville, and J. Venable. "Strategies for Design Science Research Evaluation." In: *ECIS 2008 Proceedings* (2008).

[85] r3 cooperation. *corda: The open source blockchain for business.* 2017. URL: `https://www.corda.net/` (visited on 07/17/2018).

[86] A. Rai and V. Sambamurthy. "Editorial Notes—The Growth of Interest in Services Management: Opportunities for Information Systems Scholars." In: *Information Systems Research* 17.4 (2006), pp. 327–331. DOI: `10.1287/isre.1060.0108`.

[87] raidenNetwork. *Raiden network: Fast, cheap, scalable token transfers for Ethereum*. 2018. URL: `https://raiden.network/` (visited on 07/17/2018).

[88] Reality Keys. *Reality Keys: Facts about the future, cryptographic proof when they come true.* 2016. URL: `https://www.realitykeys.com/` (visited on 02/15/2018).

[89] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz. “On blockchain and its integration with IoT. Challenges and opportunities.” In: *Future Generation Computer Systems* 88 (2018), pp. 173–190. ISSN: 0167739X. DOI: 10.1016/j.future.2018.05.046.

[90] P. Rimba, A. B. Tran, I. Weber, M. Staples, A. Ponomarev, and X. Xu. “Comparing Blockchain and Cloud Services for Business Process Execution.” In: *ICSA 2017*. Ed. by I. I. C. o. S. Architecture. Piscataway, NJ: IEEE, 2017, pp. 257–260. ISBN: 978-1-5090-5729-0. DOI: 10.1109/ICSA.2017.44.

[91] M. Schmidt. *Zufriedenheitsorientierte Steuerung des Customer Care: Management von Customer Care Partnern mittels Zufriedenheits-Service Level Standards*. Wiesbaden: Gabler, 2008. ISBN: 978-3-8350-0917-2. DOI: 10.1007/978-3-8349-9641-1.

[92] R. Scholderer. *Management von Service-Level-Agreements: Methodische Grundlagen und Praxislösungen mit COBIT, ISO 20000 und ITIL*. 2., aktualisierte und erweiterte Auflage. Heidelberg: Dpunkt.verlag, 2016. ISBN: 978-3-86490-397-7.

[93] S. Seebacher and R. Schüritz. “Blockchain Technology as an Enabler of Service Systems: A Structured Literature Review.” In: *Exploring services science*. Ed. by S. Za, M. Drăgoicea, and M. Cavallari. Vol. 279. Lecture Notes in Business Information Processing. New York NY: Springer Berlin Heidelberg, 2017, pp. 12–23. ISBN: 978-3-319-56924-6. DOI: 10.1007/978-3-319-56925-3_2.

[94] M. A. Sieke, R. W. Seifert, and U. W. Thonemann. “Designing Service Level Contracts for Supply Chain Coordination.” In: *Production and Operations Management* 21.4 (2012), pp. 698–714. ISSN: 10591478. DOI: 10.1111/j.1937-5956.2011.01301.x.

[95] C. Sillaber and B. Waltl. “Life Cycle of Smart Contracts in Blockchain Ecosystems.” In: *Datenschutz und Datensicherheit - DuD* 41.8 (2017), pp. 497–500. ISSN: 1862-2607. DOI: 10.1007/s11623-017-0819-7.

[96] E. Solaiman, I. Sfyrakis, and C. Molina-Jimenez. “A State Aware Model and Architecture for the Monitoring and Enforcement of Electronic Contracts.” In: *18th IEEE Conference on Business Informatics*. Ed. by E. Kornyshova, G. Poels, and C. Huemer. Piscataway, NJ and Piscataway, NJ: IEEE, 2016, pp. 55–63. ISBN: 978-1-5090-3231-0. DOI: 10.1109/CBI.2016.15.

[97] H. Surden. *Computable Contracts*. 2012.

[98] N. Szabo. *Smart Contracts: Building Blocks for Digital Markets*. 1996. URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (visited on 11/17/2017).

[99] P. Szalachowski. *Blockchain-based TLS Notary Service*. 2018.

[100] S. Tikhomirov. "Ethereum: State of Knowledge and Research Perspectives." In: *Foundations and practice of security*. Ed. by A. Imine. Vol. 10723. LNCS sublibrary: SL4 - Security and cryptology. Cham, Switzerland: Springer, 2018, pp. 206–221. ISBN: 978-3-319-75649-3. DOI: 10.1007/978-3-319-75650-9_14.

[101] TLSNotary. *TLSNotary - prove an https page was in your browser*. 2014. URL: https://tlsnotary.org/ (visited on 07/12/2018).

[102] TM Forum. *SLA Management Handbook: Volume 4 Enterprise Perspective*. 2004.

[103] P. Tonks and H. Flanagan. "Positioning the Human Resource Business Using Service Level Agreements." In: *Health Manpower Management* 20.1 (1994), pp. 13–17. ISSN: 0955-2065. DOI: 10.1108/09552069410053777.

[104] J. J. Trienekens, J. J. Bouman, and M. van der Zwan. "Specification of Service Level Agreements: Problems, Principles and Practices." In: *Software Quality Journal* 12.1 (2004), pp. 43–57. ISSN: 0963-9314. DOI: 10.1023/B:SQJO.0000013358.61395.96.

[105] F. Tschorsch and B. Scheuermann. "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies." In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 2084–2123. DOI: 10.1109/COMST.2016.2535718.

[106] P. Unterharnscheidt and A. Kieninger. "Service Level Management-Challenges and their Relevance from the Customers' Point of View." In: *AMCIS*. 2010, p. 540.

[107] S. L. Vargo and R. F. Lusch. "Service-dominant logic: continuing the evolution." In: *Journal of the Academy of Marketing Science* 36.1 (2008), pp. 1–10. ISSN: 0092-0703. DOI: 10.1007/s11747-007-0069-6.

[108] W. Viryasitavat, L. Da Xu, Z. Bi, and A. Sapsomboon. "Blockchain-based business process management (BPM) framework for service composition in industry 4.0." In: *Journal of Intelligent Manufacturing* 12.2 (2018), p. 133. ISSN: 0956-5515. DOI: 10.1007/s10845-018-1422-y.

[109] F. Volland. "Identification of Programming Patterns in Solidity." Master's thesis. Munich: TUM Faculty of Informatics, 2018.

[110] F. Volland. *Solidity Patterns*. 2018. URL: https://github.com/fravoll/solidity-patterns (visited on 07/16/2018).

[111] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. "Untrusted Business Process Monitoring and Execution Using Blockchain." In: *Business Process Management*. Ed. by M. La Rosa, P. Loos, and Ó. Pastor. Vol. 9850. Switzerland: Springer, 2016, pp. 329–347. ISBN: 978-3-319-45347-7. DOI: 10.1007/978-3-319-45348-4_19.

[112] C. Welzel, K.-P. Eckert, F. Kirstein, and e. al et. *Mythos Blockchain: Herausforderungen für den Öffentlichen Sektor*. Berlin, 2017.

[113] K. D. Werbach and N. Cornell. *Contracts Ex Machina*. 2017.

[114] M. Wöhrer. *Solidity Patterns*. 2018. URL: `https://github.com/maxwoe/solidity_patterns` (visited on 07/16/2018).

[115] M. Wöhrer and U. Zdun. "Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity." In: *1st International Workshop on Blockchain Oriented Software Engineering @ SANER 2018*. 2018.

[116] A. Wright and P. de Filippi. *Decentralized Blockchain Technology and the Rise of Lex Cryptographia*. 2015.

[117] X. Xu, I. Weber, L. Zhu, M. Staples, J. Bosch, L. Bass, C. Pautasso, and P. Rimba. "A Taxonomy of Blockchain-based Systems for Architecture Design." In: *1st IEEE International Conference on Software Architecture (ICSA 2017)* (2017).

[118] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. "Town Crier." In: *CCS'16*. Ed. by E. Weippl, S. Katzenbeisser, C. Kruegel, A. Myers, and S. Halevi. New York, New York: The Association for Computing Machinery, 2016, pp. 270–282. ISBN: 9781450341394. DOI: `10.1145/2976749.2978326`.

[119] S. Zumkeller. *Distributed application for hosting service providers*. 2018. URL: `https://github.com/stzu/hosting-dapp` (visited on 08/12/2018).