Decentralized Service Registry and Discovery in P2P Networks Using Blockchain Technology

Peter de Lange(✉), Tom Janson, and Ralf Klamma

RWTH Aachen University, Lehrstuhl Informatik 5, Ahornstr. 55, 52074 Aachen, Germany
{lange,janson,klamma}@dbis.rwth-aachen.de

Abstract. Decentralized information systems radically change the power dynamics of the Web by establishing participants as equal peers, which form a self-governing community. However, decentralized infrastructures currently do not offer a way for users to easily explore available services in the network, nor the ability to securely verify their origin and history. In this contribution, we approach these challenges by exploiting the tamper-proofness of blockchain technology to build a decentralized service registry and discovery system for an existing decentralized microservice infrastructure. With this, users are able to find services in a network and are also able to verify their integrity and origin. Our first evaluations show promising results with this kind of system in the domain of decentralized service provisioning, while also raising research questions for future research in this field.

Keywords: Service discovery · Decentralization · Microservices · Blockchain

1 Introduction

When Tim Berners-Lee proposed the Web in 1989, he envisioned a decentralized system of information repositories that facilitate organizational knowledge transfer by allowing anyone to create, reference, and access content [2]. However, Web authoring and publication required both technical expertise and hardware infrastructure. With the rise of the Web 2.0 in the early 2000s, Social Networking Sites (SNS) and Content Management Systems (CMS) enabled all users to create Web content [12]. But it simultaneously put the users at the mercy of the platform operators. Services may suddenly be shut down, erasing content and disrupting communities. As well, private data is often stored insecurely, used for commercial purposes, or even revealed in data breaches. The proprietary nature of the vast majority of these platforms leaves users little bargaining power to change those terms.

Decentralized information systems radically change this dynamic by establishing participants as equal peers, which form a self-governing community.

M. Bakaev et al. (Eds.): ICWE 2019, LNCS 11496, pp. 296–311, 2019.
https://doi.org/10.1007/978-3-030-19274-7_22

A peer-to-peer (P2P) structure can provide scalability and distribute the utilization of computing resources. In combination with public key cryptography, it allows users to sign messages and store private data securely, providing privacy without relying on trusted infrastructure. It is clear that these properties are especially appealing to online *Communities of Practice* (CoPs) [22]. These groups of people with a shared craft or profession, but not bound by a formal context, collaborate informally via the Web. In previous work [9] we presented a P2P microservice infrastructure for CoPs. This network of nodes can be hosted by the CoP itself. Microservices [11], once uploaded into the network, can be replicated through the community members' nodes according to the current need.

However, decentralized infrastructures currently don't offer a way for users to easily explore available services in the network, nor the ability to securely verify their origin and history. In distributed systems, this task is commonly solved by using service registries, providing a publish-lookup API facilitating service discovery and interoperation. Transferring this concept into the setting of open, decentralized systems is a technical challenge, since the architecture of traditional service registries relies on trusted servers, while existing P2P approaches compromise queriability and security. Beyond this technical challenge, it also raises research questions regarding end-user service discovery in the context of online communities.

In this contribution, we approach these challenges by exploiting the tamper-proofness of blockchain technology to build a decentralized service registry and discovery system for a decentralized P2P microservice infrastructure. We first briefly recap the real-world use case from previous work to then point out the challenges and potential threats this community infrastructure faces (Sect. 2). We then introduce the background and related work done in the domain of both "traditional" service discovery and blockchain technology (Sect. 3), before we present our decentralized, blockchain-based service registry as an approach to tackle the previously mentioned challenges (Sect. 4). By securely recording the release history of services, this approach provides service authors control over their services' update process and the ability to establish a reputation for quality contributions within the community. Service users, on the other hand, are able to verify the integrity, origin, and history of service releases. On this basis, the service discovery system enables searching for services both programmatically and via a user-friendly, browser-based interface, taking into account the different requirements of developers and end-users. In Sect. 4.2, we describe the technical integration of this approach into a purely P2P based architecture. Our contribution ends with a report on our evaluation of the system as well as its implications for future research (Sect. 5), before we conclude our paper (Sect. 6).

2 Use Case

In our initial use case that lead to the development of the decentralized microservice infrastructure, we supported a CoP preparing for a training course of the European Voluntary Service (EVS) program. To cope with the diverse background of the participants, the trainers used a form of question-based dialog

some days before the actual (on-premise) training course started. This application, consisting of a set of microservices and a Web frontend, enables users to participate in a sort of mind-mapping process. Our infrastructure allows members of the community to start a node and all services needed to locally run the application, or only start the node and access services of other members via the network. Another possibility is to just access the Web frontend of a community member to participate. This scenario fulfilled the need for the whole infrastructure being distributed only among the community itself without the need for any central authority. To our knowledge, this type of fully community-owned decentralized microservice infrastructure is unique. However, there are several shortcomings to it, which come to light once one takes a look at the "bigger picture". In Fig. 1 we depict this scenario. In this example, *Community A* stands for the above mentioned CoP, whilst a second *Community B* also participates in the network. Additionally, we consider a malicious actor *Eve*. This raises several problems, which we point out next.

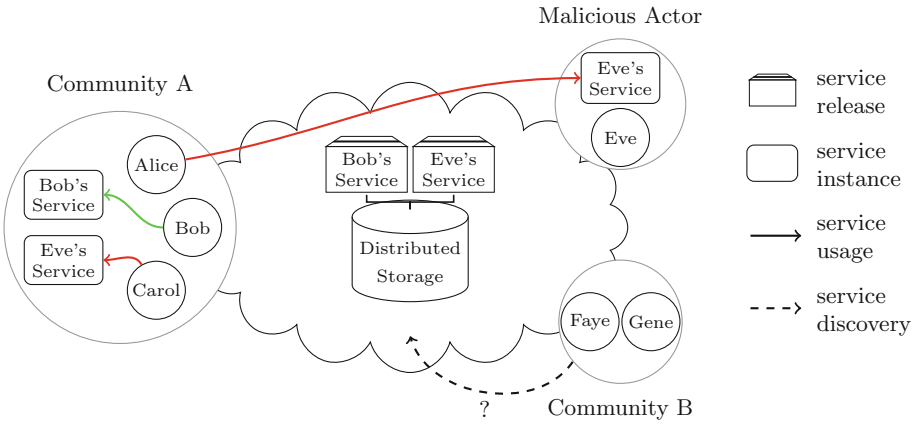

Fig. 1. Usage scenario with multiple communities

How to Explore Services Available in the Network? When *Community B* joins the network, there neither exists an overview of available service releases nor information on currently deployed service instances in the network. To new communities and community members, the network appears "empty", and information about services of interest has to come from external sources, like overview websites. Previously, our infrastructure used a "Catalog Service" for this task, that held information added by community members and displayed them on a public frontend. Since users could add any unverified service information, this approach was fundamentally insecure and also required continuous manual curation.

Where Can I Find More Information About that Service? The knowledge of services existing in the network might not always be enough to get an

impression of what usage possibilities exist. Additional information, like service descriptions, source code location, available frontends, or even usage patterns by other communities may be of relevance to new members or new communities entering the network. Also, the identity of the service developer is of relevance, since trust in a service is highly dependent on its author. In the above example, members of *Community B* might for example be interested in seeing service releases by a particular developer of *Community A*, e.g., because she is a member of both communities and forms a binding link between them.

How to Verify the Integrity and Origin of a Service? Once a community has established both the knowledge of which service might be worth exploring and where a running instance can be found, P2P networks offer no way of verifying the integrity and origin of services. Specifically, that a remotely running service instance is in fact an unmodified instance of the service release it claims to be. Even when replicating a service locally and checking its integrity via its cryptographic signature, in the absence of a registration authority, the signing key cannot be linked to the (real-life or pseudonymous) identity of the service release's author. This could result in a malicious service instance being executed on the community member's node. In the above example *Bob* has published the initial "correct" service release, the actor *Eve* publishes a malicious service release that imitates this one. Since there is no way of tracing the origin of a service instance or its release in the network, both communities could accidentally call a malicious service instance. This is depicted by *Alice* and *Carol* calling *Eve's Service Instance*, instead of the "correct" one published by *Bob*.

Derived Requirements for Decentralized Service Discovery: From the above use case, a number of requirements arise regarding service discovery in decentralized systems. It should enable both end-users and developers to easily find service releases, verify their origin and either use remote instances or replicate the release to their own node. Although most of these requirements can be solved by using some kind of central service registry (see also Sect. 3), this approach has one major drawback: It redirects the power over the infrastructure from the community to the maintainer of this centralized component and thus contradicts the whole idea of decentralization. Without the knowledge of available services and also the ability to authorize service releases, the community relies on the service registry to forward their discovery requests, which raises the same issues a decentralized infrastructure tries to tackle. To be in line with the concept and preserve its advantages, a decentralized service registry has to be governed by the whole community in terms of authorizing service releases and validating service instances. The Blockchain approach fits this idea perfectly.

3 Related Work

Service Discovery and Registration Architectures. The term *service discovery* encompasses varying degrees of functionality, depending on the context: In its most basic form, it refers to the publication and lookup of the network

location of a service which is already known by name in a registry (*service location discovery*). This registry may also allow the retrieval of services matching a formal description (*semantic service discovery* or *matchmaking*), and thus requires that services publish a machine-interpretable description of their capabilities. This meaning is central to the vision of the Semantic Web [3], in which data stored in potentially disparate sources (e.g., published in different formats, by different communities) can be automatically discovered, processed, and to some degree understood by machines [21]. Finally, *end-user service discovery* goes beyond programmatic discovery and aims to help users find Web services relevant to their interests, e.g., by employing recommender systems or the user's physical and logical context [8]. A great variety of architectures for service discovery has been proposed. They are often classified according to the degree of centralization of the registry [7,15]. However, the most simple scheme is to not use a registry at all, but to propagate service queries or advertisements via flooding. It is clear that the communication overhead of this approach prevents it from scaling to large networks, instead it was suggested for home networks or even in cars, where the number of participating devices was presumed to be very small [4,6]. For medium-sized networks, a single central registration server may be used. The API of such a registry consists of *service publication* and *service lookup*. The registry simply caches the service description published by the *service provider* until some time-out is met and answers the *service requesters'* queries accordingly. While this approach can work well in controlled environments, several issues arise when attempting to serve large, geographically distributed, or heterogeneous networks: First, having a single registry server is neither fault-tolerant nor scalable. Further, if services should be accessible from across the globe, latency may be an issue. Finally, the registry is under the control of and must be maintained by a single entity. Out of these considerations emerged distributed service registry architectures, which can be classified into three domains according to the way they store service descriptions and state information:

1. *Replicating*, where registries attempt to have the same, complete state [19]
2. *Distributed* or *federated*, where registries only store information about local services, but forward queries about other services to a cooperating registry
3. *Peer-to-peer*, where information is also stored decentrally, but all participating registries use a common P2P protocol, negating the need for manually configuring and setting up sharing agreements between them [7,20]

Each of these is appropriate for certain use cases. Current commercial systems such as Netflix's Eureka[1] and HashiCorp's Consul[2] fall into the first two categories (or some hybrid combining both), with local registries assigned to each data center or region. P2P service registries have to our knowledge been primarily the subject of academic inquiry rather than deployed in practice. Most of them utilize a distributed hash table (DHT), where service descriptions are

[1] https://github.com/Netflix/eureka.
[2] https://www.consul.io/intro/.

addressed by the hash of their contents. However, unstructured P2P registries have also been proposed [7]. In both cases queriability is a crucial issue, specifically the search capabilities of the P2P storage beyond exact match lookup and even *completeness*, i.e., the guarantee that an existing entry can be located. Much effort has been put into extending structured P2P overlays to allow attribute, wildcard-based and other advanced queries (e.g., [13,16–18,24]), but these limitations remain a major obstacle.

Trust and Consensus in Decentralized Systems. When we discussed distributed service registries in the previous section, we implicitly assumed that all nodes comprising the system are trustworthy, i.e., operating correctly without either accidental or malicious misbehavior. For corporate networks and many other use cases this is a reasonable assumption. But for a decentralized system that is open for anyone to participate in, as a peer among equal peers, a different approach is required. We use the term *open decentralized system* to denote exactly that: A system of autonomous peers, who may join or leave at any time, and whose goals may not align with one another. Further, there is no single centralized authority, which could coordinate or serve as a universally trusted entity in the system. A fundamental problem of such decentralized systems is how to ensure that received information is up-to-date and authentic, despite being unable to trust any particular node [10]. This is an instance of a *consensus problem*, which has been in the focus of distributed systems research since the early 1980s [14]. More recently the topic has come into the spotlight due to the apparent success of cryptocurrencies, first and foremost *Bitcoin*, which purport to solve the problem on a global scale. In essence, a functioning decentralized system needs to agree on a common state. The nodes of a distributed database need to agree on the contents and order of the applied operations, while cryptocurrencies deal with the specific case of tracking the participants' account balances. Thus a secure, scalable consensus algorithm lies at the heart of decentralized systems. We argue that consensus algorithms, with their ability to keep a shared state across network nodes, are promising candidates for the storage backend of a decentralized service registry.

Smart Contracts. *Ethereum* [23], as a so-called second generation cryptocurrency, utilizes a proof-of-work and blockchain based consensus scheme. But instead of being only used as a cryptocurrency (like for example Bitcoin), Ethereum sees itself as a general purpose platform for decentralized applications. This is reflected technically in the syntax of its transactions. These can include code in a Turing-complete, stack-based bytecode language, whereas the transactions in Bitcoin's blocks are deliberately less expressive. This allows Ethereum users to write and upload scripts to the network, whose functions can be invoked by sending special transactions. Such scripts are called *Smart Contracts*. Each deployed smart contract consists of its program code, a data store, as well as an account containing *Ether*, Ethereum's currency. When a transaction triggers a smart contract function, the miner that includes the transaction executes the code and includes the updated state in the new block. All other nodes must also execute the code in order to determine whether the new block is valid (includes

the correct result of the computation). Given certain restrictions on the computing power of an attacker in comparison to the nodes behaving correctly, this approach provides an immutable history of transactions.

It is clear that this massively redundant code execution is expensive in terms of resources. Ethereum charges a dynamic fee based on the number of executed instructions, which must be paid for by the transaction's sender. If its funds are insufficient, the execution is stopped. It is thus not economically feasible to deploy computationally expensive code or to directly store large amounts of data. It should be noted that the cost of executing smart contracts limits Ethereum's scalability (in terms of throughput), and there are numerous proposals to alter the Ethereum protocol in order to improve performance, including radical changes to the consensus system [5]. Examples of smart contract applications currently in use include financial contracts, games of chance, and notary applications [1], which can largely be implemented with very simple program logic, while documents can sometimes be stored elsewhere (e.g., via IPFS[3], a peer-to-peer storage system) and securely referenced.

4 A Decentralized Service Registry

4.1 Conceptual Overview

We propose a decentralized service registry based on a private blockchain that enables the discovery of services and the secure verification of their release metadata. Combining the completeness and time-preserving properties of a blockchain with space-efficient distributed storage allows us to utilize the strengths of each technology and compensate their respective weaknesses. Specifically, the registry consists of two smart contracts for both services and users. The data written to the blockchain belongs to four types, which are shown along with their respective fields in Fig. 2. Because storing data "on-chain" is inefficient and expensive, only essential fields are stored directly on the blockchain, while supplemental fields (marked in italics) are stored "off-chain" in the distributed storage and securely referenced by their hash.

The *user contract* serves as a decentralized identity management system that ties usernames to their (online) identities via public key cryptography. In contrast to a centralized public key infrastructure, the user has direct control (i.e., ownership) of their entry, including the decision to reveal personal data. Thus some users may reveal their real life identities in order to facilitate trust, while others may choose to remain pseudonymous. Registered users can then use the *service contract* to publish *service releases*. This encompasses reserving a service name and linking specific releases to additional metadata. Just like the usernames, these entries are owned by their author by linking them to the author's public key. Finally, we allow the announcement of *service instances*, indicating that a user is running a publicly usable instance of the service on their node. Storing this data in a blockchain provides an immutable, auditable historic record of

[3] InterPlanetary File System, https://ipfs.io/.

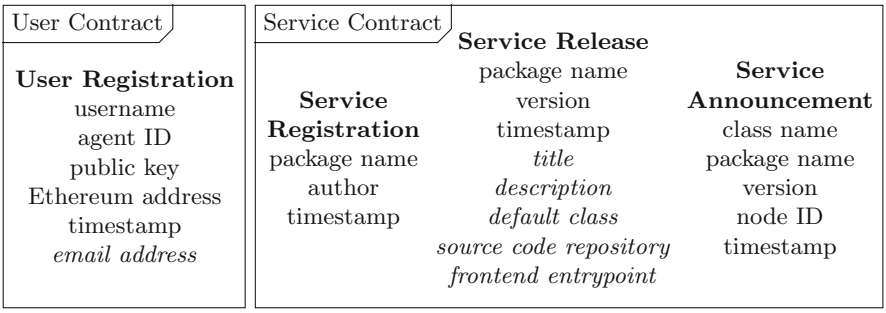

Fig. 2. The registry smart contracts' data

the registry entries and ensures that they can only be updated by their owners, while also making the data readily available and queriable to all peers.

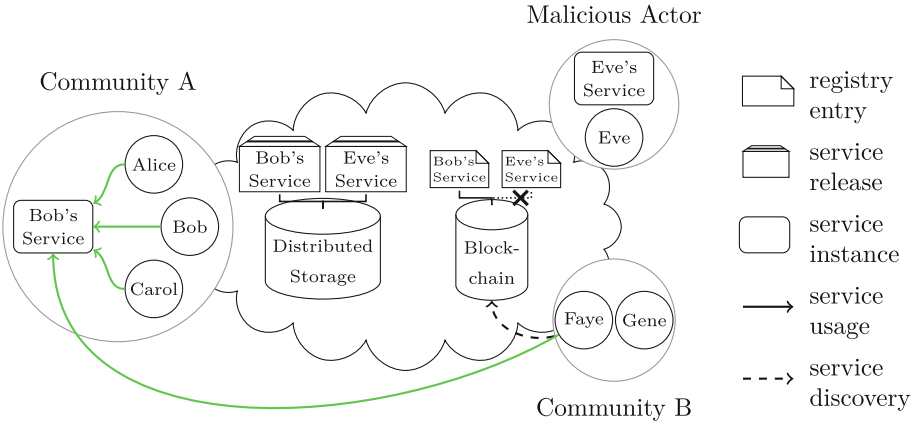

Fig. 3. Usage scenario with decentralized service registry

Returning to our example (Fig. 3), *Bob* registered both a username and the name of his service in the network's decentralized registry. The registry entry for *Bob's Service* is linked to his username and key pair, which Bob uses to sign his service releases. *Eve* can still store her malicious, modified release of Bob's Service in the distributed storage. However, she is unable to register it under the same name, nor can she attach Bob's name to it. All network participants can access the blockchain to see published services and their running instances, and can perform arbitrary queries (e.g., a keyword search over the service metadata). Thus *Faye* can easily discover Bob's Service even if they are part of disjoint communities. Just like *Alice* and the other members of *Community A*, Faye also sees the running instance of Bob's Service. If she feels she can trust the user who sent the

service announcement and operates the instance, she can access it directly. Otherwise, she can replicate the service locally. By fetching the service release from the distributed storage and comparing its signature against the registry entry, she can verify that the service she starts was in fact authored by *Bob*.

4.2 Architecture

We implemented our approach on top of the decentralized microservice infrastructure introduced in Sect. 2, called *las2peer*[4]. Figure 4 provides an overview of its extended architecture and information flow.

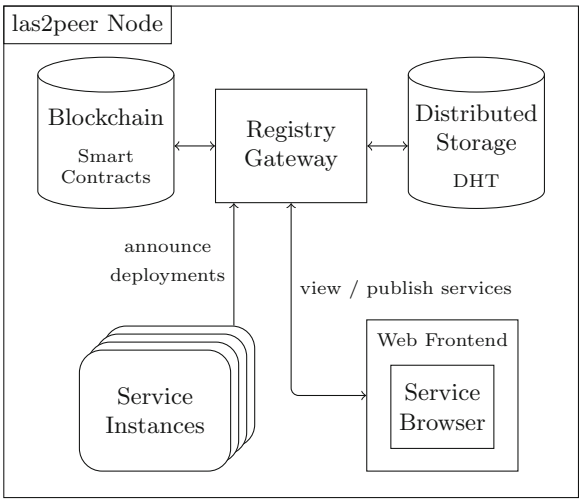

Fig. 4. Architecture and information flow during common operations

There are three main components realizing the decentralized service registry:

1. Smart Contracts: The foundation of the registry is implemented as Ethereum smart contracts that store and retrieve data from the blockchain. The contracts are written in Ethereum's high-level scripting language *Solidity*[5]. In addition to the user and service smart contracts described above, a small library contract is employed to handle the verification of signatures for delegated function calls.

2. Registry Gateway: Every node contains a registry gateway for accessing the Ethereum blockchain. It transparently stores and fetches the supplemental data fields in the distributed storage, realized as a DHT, and combines them with the

[4] https://las2peer.org/.
[5] https://solidity.readthedocs.io/.

data retrieved from the blockchain to utilize the benefits of both storage types. The registry gateway also caches service information to provide efficient lookup.

3. Service Browser: The node's Web frontend contains a service browser that allows viewing and uploading service releases, as well as managing local service instances and accessing their frontends. Figure 5 gives an impression of it.

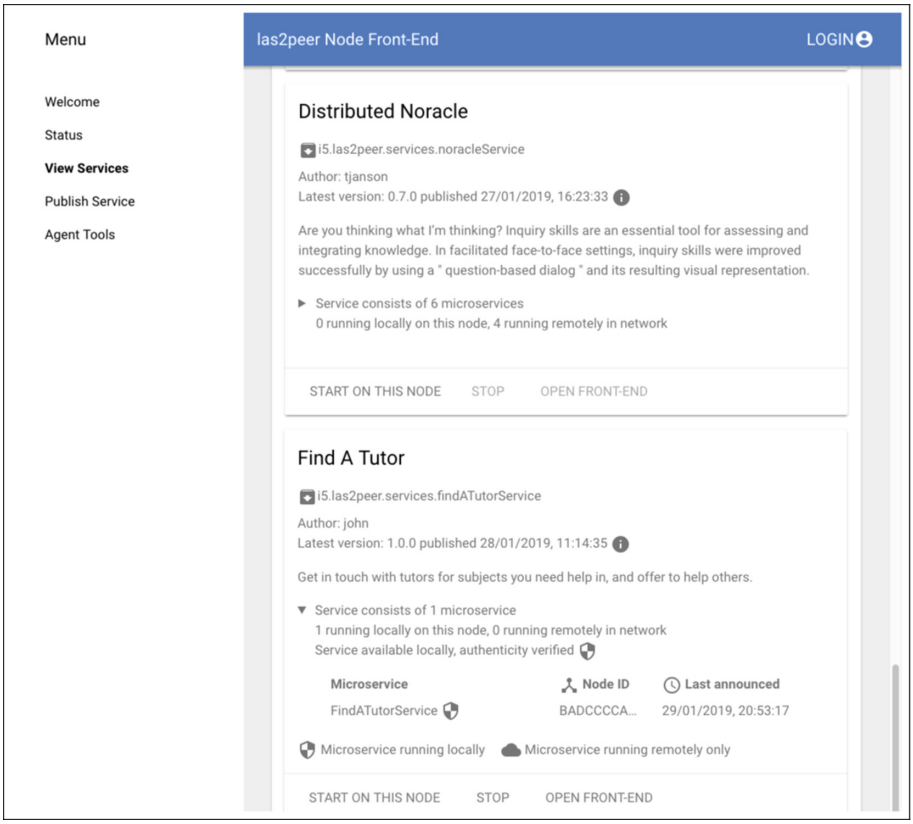

Fig. 5. The service browser

4.3 Interacting with the Registry

User Registration. The user and service smart contracts are essentially *name registries* that assign human-friendly names on a first come, first served basis. As such, the user contract provides functions to check the availability, register, and look up the data associated with a username. An important concept of smart contracts is the distinction between *state-changing* functions and those that are "read-only". The former are processed as transactions appended in a new block (and thus transmitted to and executed by all nodes), while the latter are

executed locally and immediately return a value. When a user wishes to register a username, a read-only function is used to check whether the desired name is still available. If so, we call the registration function, which is state-changing: The call data, consisting of the function name and the arguments, is encoded and broadcast in the form of a transaction signed by the user. When a miner processes the transaction to include it in a block, the arguments are extracted and the call is executed. The smart contract code again checks whether the name is already assigned to someone (e.g., in the case that someone else attempted to register the name at nearly the same time). If not, the user entry is created.

We also allow a pattern called delegated function call, in which the user does not sign the transaction herself, but rather prepares a signed certificate of authority (Fig. 6). This also contains the call data and is signed by the user. Now any user can prepare a transaction that passes this certificate to a special function of the user contract, which unpacks the arguments, verifies the signature,

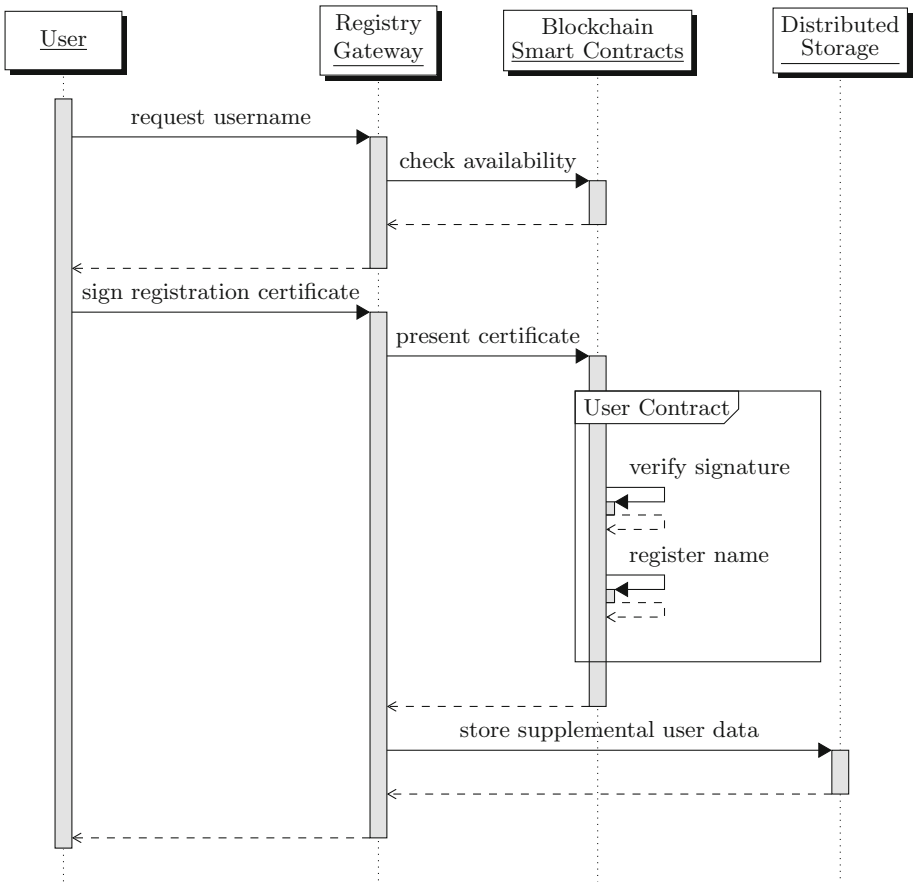

Fig. 6. Delegated username registration with signed smart contract call

and registers the username on behalf of the original user. The advantage of this pattern is that it transfers the burden of paying any transaction fees from the user wishing to register to the user who actually sends the transaction. In our use case, we expect that established community members who operate a node may offer to cover the registration fees for users who are unable to run their own node. If the registration was successful, a file containing the supplemental data fields is uploaded to the distributed storage.

Service Registration, Discovery, and Replication. The service contract operates using the same principles as the user contract. Once again we employ the delegated function call pattern to allow users to cover the transaction fees for a service author. The registration of a service name and publication of a service release is analogous to the user registration procedure. When the deployment of a service instance is requested (e.g., through a node's Web frontend), the service release is fetched from the distributed storage. Through two subsequent smart contract calls, first the username of the author of the service release is looked up in the service contract, then her public key is looked up in the user contract, enabling the system to verify the signature of the service release. Once its authenticity is established, the service is started. If the instance is intended for public use, its deployment is announced to the registry.

5 Preliminary Evaluation

Setup. In order to gather feedback from users, we carried out five evaluation sessions with two to three participants each. The network setup consisted of five permanent nodes and up to three additional nodes started during the session by the participants. Technically, the nodes were started as Docker containers on a single server in order to simulate ideal network conditions. Further, we used a modified version of the Go Ethereum client, `geth`. Since we did not focus on the technical parameters of the blockchain network in this evaluation, we started the client with a very low mining difficulty, leading to short block creation intervals (the "block time"). While many of the participants had experience with software development, the majority was unfamiliar with las2peer, our decentralized community service infrastructure. After a brief introduction to las2peer and its service registry, the participants were given written tasks that included finding existing services as well as registering a user and publishing their own service. During these tasks, the users first accessed the Web frontend of one of the permanent nodes. Later they accessed a newly started node, that joined the existing network. This hands-on experience lasted about 30 min. Afterwards, participants filled out a questionnaire.

Results. Figure 7 shows the results of our questionnaire. As one can see, most of the participants were able to understand the basic concepts of the approach and were also able to fulfill the given tasks. We received lowest scores for the question regarding node ownership. This is due to the fact that the majority

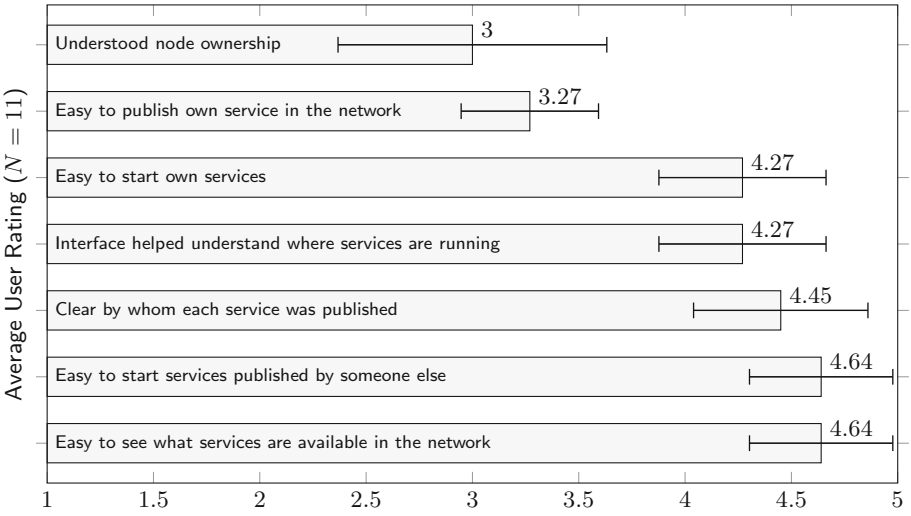

Fig. 7. Evaluation results

of participants were not very familiar with P2P infrastructures and mixed up concepts like running services and nodes. Another quite low rating was received for the question about the ease of publishing own services in the network. This was mostly colored by the fact that it was necessary to post exact class names and frontend URLs, that were not checked for correctness in our initial prototype. This will be improved in future versions of the frontend. The remaining questions received quite high scores, which confirms the usability of our developed service registry and its user interface in form of the service browser. We also asked free-text questions about the relevance of (verified) service authorship. Most of the participants stated that this depended highly on the context, and was only relevant for sensitive data. This is in line with our expectations that decentralized infrastructures are valued particularity in the context of sensitive data exchange. We also received interesting statements regarding the question of contribution to the infrastructure, e.g., in terms of computing power. The majority of responses stated that they would support "their" community if resources were sparse and highly in need. Finally, we asked the participants about the trade-off between response time and required storage space. On average, participants were willing to wait up to ten minutes (although a large minority voted for ten seconds or less) for their actions, e.g., service releases and announcements, to be visible on the blockchain.

Implications for Future Work. One aspect that received little attention in this contribution is transaction fees and incentives to participate in the mining process. Our private blockchain's currency is specific to its corresponding network and does not have any economic value. In future work we want to explore whether this currency can represent status in the community. One direction

worth investigating is allowing users to provide bounties for certain actions in the network, like the development or deployment of needed services.

Another technical aspect that requires further evaluation is the trade-off between required storage space and block time, as well as between CPU usage and security, i.e., the integrity of the blockchain data. The block time used in our evaluation was set to an extremely short value (1 s with a difficulty of 1) that is not feasible for real-world settings, since it requires about 500 MB of space per day per node, regardless of the amount of information stored in the blocks. This amount scales inversely with the block time. To study this further, we started modified versions of the Ethereum client with more "realistic" difficulties of 4×10^5 and 128×10^5. These numbers are to be interpreted as the average number of hashes required before a new, valid block is found. We ran these clients over a period of a week, letting each mine with a single CPU core of an Intel® Xeon® Processor E5-4627 v4, resulting in an average block time of 9.3 s (for the lower difficulty) and 271 s (for the higher difficulty), respectively. Both of these block times are within the acceptable range for the majority of our evaluation participants. Over six days, the blockchain size on disk was 61 MB and 2.1 MB, respectively, which certainly a is a more manageable and scalable requirement.

6 Conclusion

In this paper, we presented a service registry and discovery system for a decentralized community information system. With its foundation in blockchain technology, it keeps the advantages of decentralization, such as empowering communities to take control over their infrastructure. Simultaneously, it enables many properties "traditional", centralized services registries provide, like searching for service releases and instances, and verification of these. While in the future further evaluations will provide deeper insights into the long-time effects of this approach, we are confident that blockchain-based decentralized service registries can provide a valuable addition in the domain of P2P systems.

Acknowledgments. The authors would like to thank the German Federal Ministry of Education and Research (BMBF) for their kind support within the project "Personalisierte Kompetenzentwicklung durch skalierbare Mentoringprozesse" (tech4comp) under the project id 16DHB2110.

References

1. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) FC 2017. LNCS, vol. 10323, pp. 494–509. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_31
2. Berners-Lee, T.: Information management: a proposal (1989)

3. Berners-Lee, T., Hendler, J.A., Lassila, O.: The semantic web. Sci. Am. **284**(5), 28–37 (2001)
4. Bettstetter, C., Renner, C.: A comparison of service discovery protocols and implementation of the service location protocol. In: EUNICE 2000, 6th Open European Summer School (2001)
5. Buterin, V.: A Modest Proposal for Ethereum 2.0 (2017)
6. Guttman, E.: Service location protocol: automatic discovery of IP network services. IEEE Internet Comput. **3**(4), 71–80 (1999)
7. Klusch, M.: Semantic web service coordination. In: CASCOM: Intelligent Service Coordination in the Semantic Web, pp. 59–104. Birkhäuser Basel (2008)
8. La Torre, G., Monteleone, S., Cavallo, M., D'Amico, V., Catania, V.: A context-aware solution to improve web service discovery and user-service interaction. In: 2016 International IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, pp. 180–187 (2016)
9. de Lange, P., Göschlberger, B., Farrell, T., Klamma, R.: A microservice infrastructure for distributed communities of practice. In: Lifelong Technology-Enhanced Learning, pp. 172–186 (2018)
10. Mattila, J.: The Blockchain Phenomenon - The Disruptive Potential of Distributed Consensus Architectures (2016)
11. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly, Sebastopol (2015)
12. O'Reilly, T.: What is web 20: design patterns and business models for the next generation of software. Commun. Strat. **65**, 17–37 (2007)
13. Paolucci, M., Sycara, K.P., Nishimura, T., Srinivasan, N.: Using DAML-S for P2P discovery. In: Proceedings of the International Conference on Web Services, pp. 203–207 (2003)
14. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
15. Rambold, M., Kasinger, H., Lautenbacher, F., Bauer, B.: Towards autonomic service discovery a survey and comparison. In: IEEE International Conference on Services Computing, pp. 192–201 (2009)
16. Sahin, O.D., Gerede, C.E., Agrawal, D., El Abbadi, A., Ibarra, O., Su, J.: SPiDeR: P2P-based web service discovery. In: Service-Oriented Computing, pp. 157–169 (2005)
17. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services. In: Peer-to-Peer Computing, pp. 104–111 (2002)
18. Schmidt, C., Parashar, M.: A peer-to-peer approach to web service discovery. World Wide Web **7**(2), 211–229 (2004)
19. Sun, C., Lin, Y., Kemme, B.: Comparison of UDDI registry replication strategies. In: IEEE International Conference on Web Services, pp. 218–225 (2004)
20. Thaden, U., Siberski, W., Nejdl, W.: A Semantic Web based Peer-to-Peer Service Registry Network (2003)
21. W3C: W3C Data Activity: Building the Web of Data (2013). https://www.w3.org/2013/data/

22. Wenger, E.: Communities of Practice: Learning, Meaning, and Identity. Cambridge University Press, Cambridge (1998)
23. Wood, G.: Ethereum: A Secure Decentralized Transaction Ledger (2014)
24. Yan, F., Zhan, S.: A peer-to-peer approach with semantic locality to service discovery. In: Jin, H., Pan, Y., Xiao, N., Sun, J. (eds.) GCC 2004. LNCS, vol. 3251, pp. 831–834. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30208-7_116