Do We Really Need Another Blockchain Framework? A Case for a Legacy-friendly Distributed Ledger Implementation based on Java EE Web Technologies

Philipp Brune[1,2]
[1]*Neu-Ulm University of Applied Sciences, Wileystraße 1, Neu-Ulm, Germany*
[2]*QWICS Enterprise Systems, Taunustor 1, 60310 Frankfurt/Main, Germany*

Keywords: Blockchain, Distributed Ledger Technology, Web Services, Web Technologies, Java EE, Enterprise Computing.

Abstract: Cryptocurrencies, blockchain technology and smart contracts could fundamentally change the way how financial products and financial services are implemented and operated. While many frameworks for implementing such blockchain applications already exist, these are usually implemented using languages either considered "fancy" today, like e.g. Go, or are traditionally used for system software, such as C++. On the other hand, the core business applications e.g. in financial services are typically implemented using enterprise platforms such as Java Enterprise Edition (EE) and/or COBOL. Therefore, to improve the integration of blockchain technology in such applications, in this paper we argue in favor of a legacy-friendly distributed ledger solution by introducing QWICSchain, an implementation build on web services using established open-source enterprise technologies such as Java EE and PostgreSQL. It supports the parallel execution of transactions on the blockchain and in existing legacy applications, thus enabling the blockchain-based modernization of existing IT infrastructures.

1 INTRODUCTION

Since the appearance of Bitcoin (Nakamoto et al., 2008), blockchain technology has become highly popular among the public as well as in research and practice (Wüst and Gervais, 2018). In particular for the banking and financial servics industries, it is widely considered as one of the pillars of digital transformation (The Financial Brand, 2018).

However, the current IT systems in banking frequently do not match the requirements of the digital age, as the typical traditional monolithic "legacy" COBOL enterprise applications are too inflexible and not open enough (Abbany, 2018; The Financial Brand, 2018). However, their underlying mainframe platform is by no means an outdated technology and probably will remain an important part of the enterprise IT landscape for a long time (Khadka et al., 2014; Nelson, 2018; Wilkes, 2018). And despite its age, together with Java, COBOL still plays an important role in enterprise application development on the mainframe (Suganuma et al., 2008; Vinaja, 2014; Farmer, 2013; Abbany, 2018) (with "mainframe" in this paper denoting IBM's S/390 platform and its descendants).

Therefore, the new blockchain-based banking technologies need to be integrated with the traditional "legacy" enterprise applications to both preserve their business value and to modernize the banking IT landscape and enable new types of digital business. To achieve this, a blockchain implemenation which seamlessly fits in traditional enterprise IT landscapes is required.

To support this claim, in this paper QWICSchain[1] is presented, an enterprise- and legacy-friendly, Java Enterprise Edition(EE)[2]-based distributed ledger implementation for financial services, enabling its users to exchange and trade all kinds of digital assets between counterparties.

Since Open Source Software (OSS) has recently been suggested as an important concept for modern banking applications (FinTech Futures, 2018), it is built using established OSS components. In partic-

[1]https://qwicschain.com

[2]Recently, Java EE has been handed over to the Eclipse Foundation to manage its future development, and therefore re-labeled as Jakarta EE (https://jakarta.ee/about/). However, for sake of simplicity in this paper still only the term Java EE is used to denote both Java EE and Jakarta EE.

Brune, P.
Do We Really Need Another Blockchain Framework? A Case for a Legacy-friendly Distributed Ledger Implementation based on Java EE Web Technologies.
DOI: 10.5220/0008344403010306
In *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pages 301-306
ISBN: 978-989-758-386-5

ular, it also integrates the recently proposed Quick Web-Based Interactive COBOL Service (QWICS)[3] (Brune, 2018) to achieve the simultaneous modernization of "legacy" COBOL applications.

The rest of this paper is organized as follows: In section 2 the related work is discussed in more detail, which leads to the concept and design of the solution described in section 3. In section 4, the actual implementation is described. We conclude with a summary of our findings.

2 RELATED WORK

Since the cryptocurrency hype originally initiated by Bitcoin (Nakamoto et al., 2008), numerous OSS frameworks have been released to implement blockchain solutions, both for open permissionless systems (typical for crytocurrencies) as well as permissioned ones (used mostly for business blockchain networks) (Wüst and Gervais, 2018).

Most notably, these range from Ethereum[4] (Wood, 2014) for permissionless to Hyperledger[5] or Ripple[6] (Schwartz et al., 2014) for building permissioned blockchain solutions. For building business blockchain applications, further frameworks exist, like e.g. Corda[7] or NEO[8].

Typically, these frameworks are written either in languages which are considered modern or "fancy" by many, such as Go (e.g. for Hyperledger) or Kotlin (Corda), or in C and its derivates (C++, C#). Also, with the exception of Corda, they mostly use their own, non-relational data stores. On the other hand, typical large-scale online transaction processing (OLTP) enterprise applications e.g. in banks today still are mainly written in Java Enterprise Edition (EE) or even COBOL (Abbany, 2018).

So while the existing blockchain frameworks mentioned above are currently evaluated in many prototypical projects, these frequently are standalone "green field" approaches, often run separately by corporate innovation labs, and only weakly integrated with the traditional core enterprise applications. It could be expected that their usage of new and partially "exotic" languages and technologies (from an enterprise IT perspective) in this context will limit their adoption and maintainability in the long run.

For building enterprise blockchain solutions which could be seamlessly integrated with existing enterprise IT landscapes, it would therefore be beneficial if these would be built using an enterprise platform such as Java EE. Of the existing frameworks, only Corda follows a similar approach, but it uses the Kotlin language instead of Java. In addition, the integration with existing OLTP applications and in particular "legacy" COBOL applications has not been addressed by other frameworks so far.

Therefore, to address the question whether a new, legacy-friendly distributed ledger implementation using established, open-source enterprise technologies such as Java EE could enable the blockchain-based modernization of existing legacy IT infrastructures, in this paper the implementation of a Java EE-based enterprise blockchain framework is demonstrated, which uses established, enterprise-ready OSS components and integrates the previously proposed QWICS framework (Brune, 2018) for "legacy" modernization.

3 DESIGN OF THE DISTRIBUTED LEDGER FRAMEWORK

3.1 Peer-to-peer Network

As usual for blockchain solutions, the proposed solution is implemented as a peer-to-peer network of nodes (servers), which represent e.g. banks or their counterparties (Wüst and Gervais, 2018). Each node runs an instance of the software and keeps its own, private copy or replica of the distributed ledger data structure. Nodes know of and are connected to some number other nodes (but typically not all others).

Every change to the ledger is performed only by adding transactions to it. Every node therefore distributes every valid transaction it receives to all other nodes it is connected to (leading in effect to replicating every change to all nodes, but with a delay. The ledger is only eventually consistent among the nodes). Every modification is made immutable by cryptographic hashing. Every node may inspect the content of the ledger on other nodes at any time for verification (Wüst and Gervais, 2018).

3.2 Distributed Ledger Structure

The proposed distributed ledger data structure focuses on integrity and consistency of the transactions stored in it. As illustrated in figure 1, it uses the basic entities account and transaction.

[3]https://qwics.org
[4]https://www.ethereum.org/
[5]https://www.hyperledger.org/
[6]https://ripple.com/
[7]http://www.corda.net/
[8]https://neo.org/

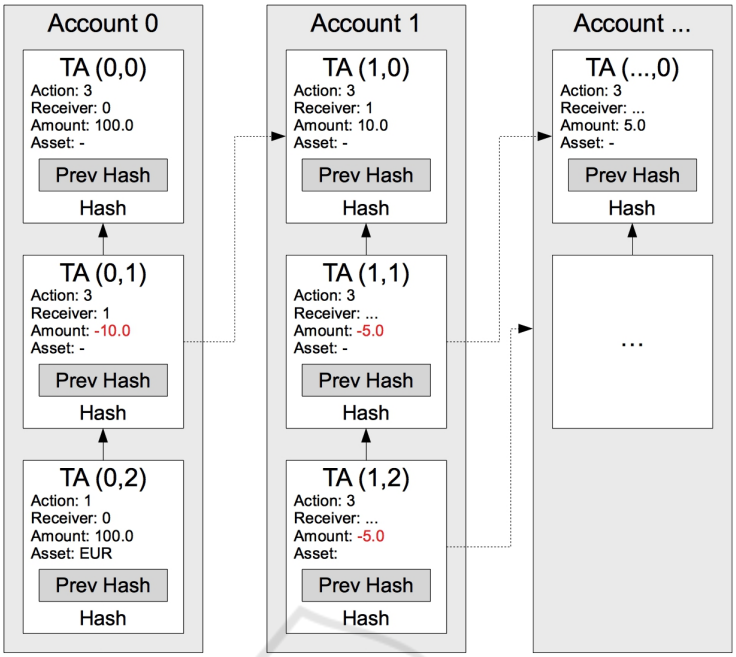

Figure 1: Concept of the proposed distributed ledger data structure consisting of accounts and their respective transactions, representing changes to the accounts.

An account contains an amount of coins and a set of digital assets. It is represented by a numerical account id. Ownership of an account is represented by a private-public cryptographic key pair, of which the public key is stored in the account, while the private key remains kept secret by the account owner. The account id is a hash value of this public key.

Every transaction belongs to an account and represents one of the three possible elementary operations on it: The creation of assets or the transfer of coins or assets to another account. A valid transaction needs always to be signed using the private key of the account owner, before it is added to the ledger. This ensures that "transferring away" coins or assets from an account may only be initiated by the account owner.

New accounts are created by transferring an initial amount of coins to a so far non-existent account id from an existing account (the "spender"). This requires the ab-initio existence of a genesis account (id=0), which serves as a "central bank", and in the beginning owns all available coins. This account is controlled by the manager of the blockchain. This approach is similar to e.g. Ripple (Schwartz et al., 2014).

Therefore, different from other distributed ledger implementations, the transactions here in fact do not form an actual "chain", but a tree-like structure. While it has some similarities with the "tangle" used by IOTA[9] (Popov, 2016), there are also fundamental differences to IOTA, e.g. regarding the order and approval of the transactions. In QWICSchain, every transaction needs to have exactly one predecessor, and contains the hash value of it. The first transaction of each account contains the hash value of the account-creating transaction in the "spender" account. Thus, still a manipulation of an earlier transaction could be easily detected as it affects most (but not all) subsequent transactions.

In addition, the tree-like structure at any time allows to delete all transactions of an account (except the account-creating transaction) without violating the integrity of the rest of the ledger. This enables the deletion of person-related information from the ledger e.g. upon request, therefore improving compliance of the solution with data privacy regulations such as EU GDPR[10].

Each transaction is validated separately before being added to the ledger, they are not grouped into blocks. Thus, a block is identical to a transaction in this approach.

A set of rules guarantees that it is as difficult as possible to inject fraudulent transactions to the network: A valid transaction needs to be signed by the account owner, needs to transfer coins or assets which the transaction's account contains before, needs to

[9]https://www.iota.org/
[10]https://eur-lex.europa.eu/eli/reg/2016/679/oj

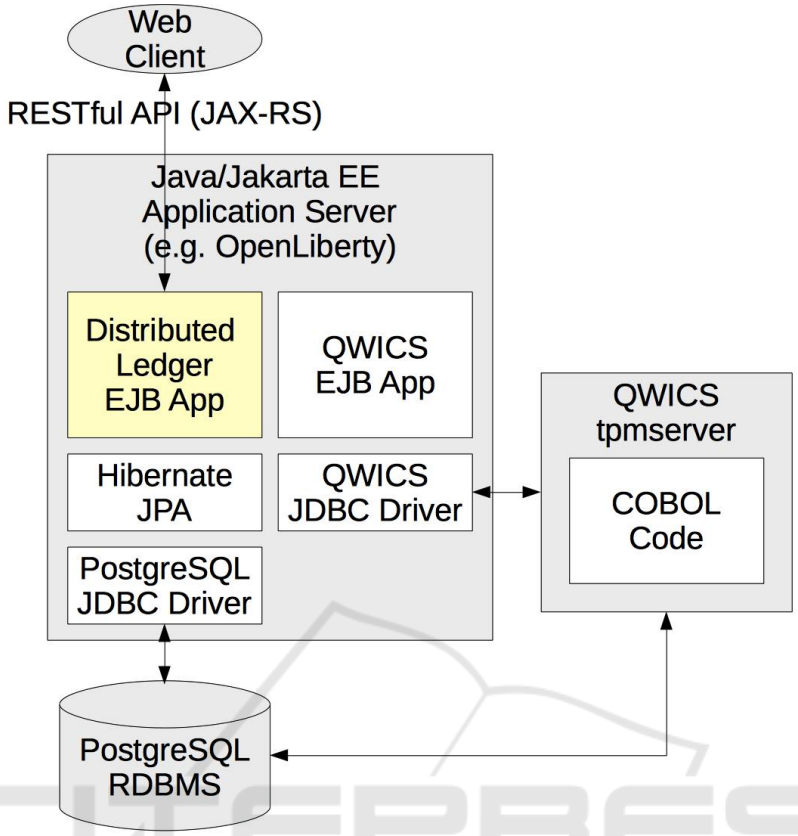

Figure 2: Architecture overview of the proposed Java EE-based distributed ledger implementation including the integration with legacy COBOL applications using the QWICS framework (Brune, 2018).

have a valid transaction as its predecessor, and so on.

3.3 Consensus Scheme

Consensus about the validity of every new transaction added to the ledger needs to be obtained among all nodes, at least the non-fraudulent ones. In Bitcoin (Nakamoto et al., 2008) and other popular blockchain frameworks like e.g. Ethereum (Wood, 2014), this consensus about new transactions is obtained by the so-called Proof-of-Work (PoW) algorithm, which is highly criticised for wasting a lot of energy and therefore not being sustainable.

In case of the proposed solution, PoW is not necessary, since it is designed as a permissioned blockchain (Wüst and Gervais, 2018). In the latter, it is not easily possible for an adversary to add a large number of fraudulent nodes, which could be used then to manipulate the ledger. Therefore, the majority of the nodes can be assumed to be trustful. In this case, the problem of determining if a new transaction is valid is equivalent to the Byzantine Generals problem (Lamport et al., 1982).

Here, the proposed approach uses the algorithm with signed messages as described in section 4 in the work of Lamport et al. (Lamport et al., 1982). The choice function in that algorithm here is implemented in such a way that a transaction is considered valid by a node if more then 80% of a node's neighbours consider it valid as well. This is similar to the approach taken by Ripple (Schwartz et al., 2014).

3.4 Software Architecture

The software architecture of the platform is designed to fit well in a traditional enterprise IT landscape, e.g. of financial institutions, by using established, mature, enterprise-ready components and frameworks. On the other hand, it still should be as lightweight, open and flexible as possible.

Therefore, as illustrated in figure 2, each node of the network is implemented using Java EE, based on the lightweight, open-source OpenLiberty[11] application server (currently version 18.0.0.4). The latter was selected due to its small memory footprint, high

[11] https://openliberty.io

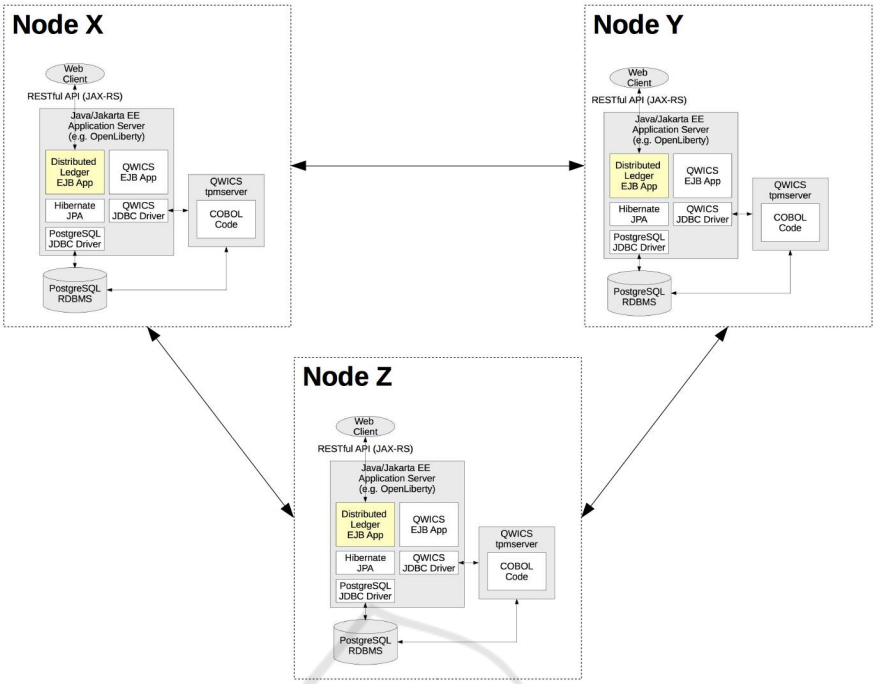

Figure 3: Examplified network consisting of three nodes X, Y and Z, each running a separate node.

speed, easy configuration and cloud-readiness, while being well integrateable in enterprise environments.

To persistently store the distributed ledger, the Hibernate[12] Java Persistence API (JPA) provider implementation is used in conjunction with the PostgreSQL[13] relational database. PostgreSQL has been selected since it is the most mature, enterprise-quality open source database available, e.g. due to features such as full transaction support (Karremans, 2018).

In figure 2, also the optional integration with a "legacy" COBOL application modernized by integrating it into Java EE is illustrated in an examplified way. Every transaction booked on the distributed ledger is forwarded optionally to another Enterprise Java Bean (EJB) application for handling it there as well. In this case, an EJB application is used, which embeds COBOL code in Java EE applications.

Each node in the network runs its own instance of the described software. The nodes communicate with each other using a RESTful web service API implemented using JAX-RS. This is illustrated in figure 3 for an examplified network of three nodes X, Y and Z. Of course, all counterparties (nodes) joining the network may run their copy of the distributed ledger on premises, in their own cloud environment or as a Software-as-a-Service solution (Mell et al., 2011) as provided by QWICS Enterprise Systems at https://qwicschain.com.

A new transaction is initially sent to one node by an account owner, and after validation by this node distributed to its neighbouring nodes, which distribute it further. Every node adds its cryptographic signature to the transaction before distributing it. Thus, the path on which a transaction propagates through the network is always known and traceable. Fraudulent nodes may be detected more easily due to this scheme. This corresponds to the algorithm using signed messages as described in section 4 in the work of Lamport et al. (Lamport et al., 1982).

4 IMPLEMENTATION

In the architecture described in section 3, the actual distributed ledger functionality on one hand is implemented using Java JPA entity classes to map the basic data structures transaction, account and asset as illustrated in figure 1.

On the other hand, EJB session beans are used to implement the actual functionality of adding and validating transactions, obtaining consensus between nodes and managing the local node. Also the web service API for inter-node communication and adding transactions is implemented in that way.

Currently, the demo and trial implementation available online at https://qwicschain.com/

[12]http://hibernate.org/
[13]https://www.postgresql.org/

QwicsChain/login.html runs inside Docker[14] containers and using OpenJDK8 in conjunction with the Eclipse OpenJ9[15] Java Virtual Machine (JVM). The latter has been selected to run smoothly not only on x86 architecture, but also on enterprise platforms such as the mainframe (S/390 architecture).

QWICSchain is publicly available for a free trial on https://qwicschain.com to evaluate and further promote its concepts. While this already demonstrates the feasibility and strengths of the approach, ongoing real-world evaluation will led to further insights and improvements.

5 CONCLUSION

In conclusion, in this paper a legacy-friendly distributed ledger implementation has been presented, to address the question about the possibility of a blockchain-based modernization of existing legacy IT infrastructures.

The respective software architecture and implementation have been described, which are based on mature web and enterprise technologies used in traditional transaction processing applications, such as Java EE and the PostgreSQL RDBMS. The usage of these established, open-source technologies promises to simplify the integration of distributed ledger technology with traditional enterprise applications.

In particular, by integrating the previously proposed QWICS framework. it also supports the modernization of "legacy" online transaction processing (OLTP) applications like they are typical for financial service providers.

By means of a free online trial implementation, the proposed approach is currently evaluated in a public beta test. While the first results are promising and indicate that indeed the described approach might be beneficial for future enterprise applications, further research is needed to evaluate it in greater depth in lab and field tests.

REFERENCES

Abbany, Z. (2018). Fail by design: Banking's legacy of dark code. https://m.dw.com/en/fail-by-design-bankings-legacy-of-dark-code/a-43645522. Accessed: 2019-01-05.

Brune, P. (2018). A hybrid approach to re-host and mix transactional cobol and java code in java ee web applications using open source software. In *Proceedings of the 14th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST,*, pages 239–246. INSTICC, SciTePress.

Farmer, E. (2013). The reality of rehosting: Understanding the value of your mainframe.

FinTech Futures (2018). How open will your bank become? https://www.bankingtech.com/2018/11/how-open-will-your-bank-become/. Accessed: 2019-01-05.

Karremans, J. (2018). Postgres in the enterprise: Real world reasons for adoption. https://www.enterprisedb.com/blog/postgres-enterprise-real-world-reasons-adoption. Accessed: 2019-01-05.

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., and Hage, J. (2014). How do professionals perceive legacy systems and software modernization? In *Proceedings of the 36th International Conference on Software Engineering*, pages 36–47. ACM.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401.

Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Nakamoto, S. et al. (2008). Bitcoin: A peer-to-peer electronic cash system.

Nelson, J. (2018). Why banks didn't 'rip and replace' their mainframes. https://www.networkworld.com/article/3305745/hardware/why-banks-didnt-rip-and-replace-their-mainframes.html. Accessed: 2019-01-05.

Popov, S. (2016). The tangle. http://tanglereport.com/wp-content/uploads/2018/01/IOTA_Whitepaper.pdf. Accessed: 2019-07-26.

Schwartz, D., Youngs, N., Britto, A., et al. (2014). The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5.

Suganuma, T., Yasue, T., Onodera, T., and Nakatani, T. (2008). Performance pitfalls in large-scale java applications translated from COBOL. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 685–696. ACM.

The Financial Brand (2018). The four pillars of digital transformation in banking. https://thefinancialbrand.com/71733/four-pillars-of-digital-transformation-banking-strategy/. Accessed: 2019-01-05.

Vinaja, R. (2014). 50 th aniversary of the mainframe computer: a reflective analysis. *Journal of Computing Sciences in Colleges*, 30(2):116–124.

Wilkes, A. (2018). The mainframe evolution: Banking still needs workhorse tech. https://www.finextra.com/blogposting/16067/the-mainframe-evolution-banking-still-needs-workhorse-tech. Accessed: 2019-01-05.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

Wüst, K. and Gervais, A. (2018). Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE.

[14]https://www.docker.com/

[15]https://www.eclipse.org/openj9/