DLT Smart Contract Platforms for Software Lifecycle Management

Biser Tsvetkov[1,a)] and Hristo Kostadinov[1]

[1]*Institute of Mathamatics and Informatics, Bulgarian Academy of Sciences*

[a)]hristo@math.bas.bg

Abstract. After its introduction the smart contract platforms got practical use when they become key part of the Ethereum public blockchain and defined the concept of distributed applications (dApps). Smart contract platforms, based on distributed ledger technologies (DLT), are used in various industries such as banking, government and law, healthcare, insurance, and transportation. One area of a DLT and smart contracts could also be used is the area of the software lifecycle management (SLM). Complex SLM procedures involve many parties such as customer, software provider, technical and business consultants, auditors, hardware providers, third party software vendors, and others. In this paper we investigate the applicability of DLT-based smart contract platforms to support multi-party SLM processes for complex customer systems, components of which are running on premise, in the cloud and on edge devices.

INTRODUCTION

Software lifecycle management is the area of installation, configuration and maintenance of often complex software systems consisting of many servers, clouds or edge devices, spread on many locations that often have complex dependencies on each other. Quality requirements such as service's high availability and disaster recovery are often requested by customer thus increasing the complexity of the overall solution and the procedures used to install, configure and maintain it. In addition, in the enterprise world SLM is an area where often many different parties are involved in a single project. The SLM-relevant information about the system properties is often limited only to some of the participants, sometimes it is missing (intentionally or not) and in many cases there is wrong information as result of different parties' manipulations. The SLM history of the system as a series of procedures pinpoints specific issues that need to be targeted by any new party involved in its maintenance. Such info may not be shared by previous participants for various reasons. Due to the ever-changing nature of the software systems the related issues are rarely well known, and responsibilities of each party are not always defined in details for each project. The interest of each party is the project to be successful and each one contributes in different area for achieving the success. Sharing plans, data and commitments is important part of project's success. Also, specific deliveries in time and quality are often a milestone for passing responsibilities from one participant/partner to another.

The interactions between non-hierarchical parties that do not fully trust each other, or single trusted entity, can be handled naturally by using modern DLTs[1, 2], smart contract platforms[3, 4, 5] and, if supported, Ricardian contracts[6, 7]. The DLT family grows and becomes more diverse while trying to solve different issues and fit into different areas. There are new DLTs that are trying to stay close to original Bitcoin and are still considered a blockchain solutions (EOS[8], Multichain[9], Hyperledger Fabric, while others (Corda[10] Tangle[11], Nano[12]) explore completely different concepts to target different solutions. After the initial development of public/permissionless DLTs later were proposed several DLTs that are targeting consortium use. Naturally the access to corporate DLTs was restricted to only known and verified participants. Some of these platforms are Multichain, Hyperledger Fabric and Corda. Later another class of cryptocurrencies was introduced in the world of cryptocurrencies. These are so called pre-mined cryptocurrencies such as Ripple and Stellar which got popular primary in the financial institutions. One classification of popular DLTs is shown on Figure 1.

There are 2 major types of DLTs currently. The more popular type is the set of public/permissionless DLTs that often offer their own cryptocurrency. The other type is enterprise/permissioned DLTs. We consider several DLT

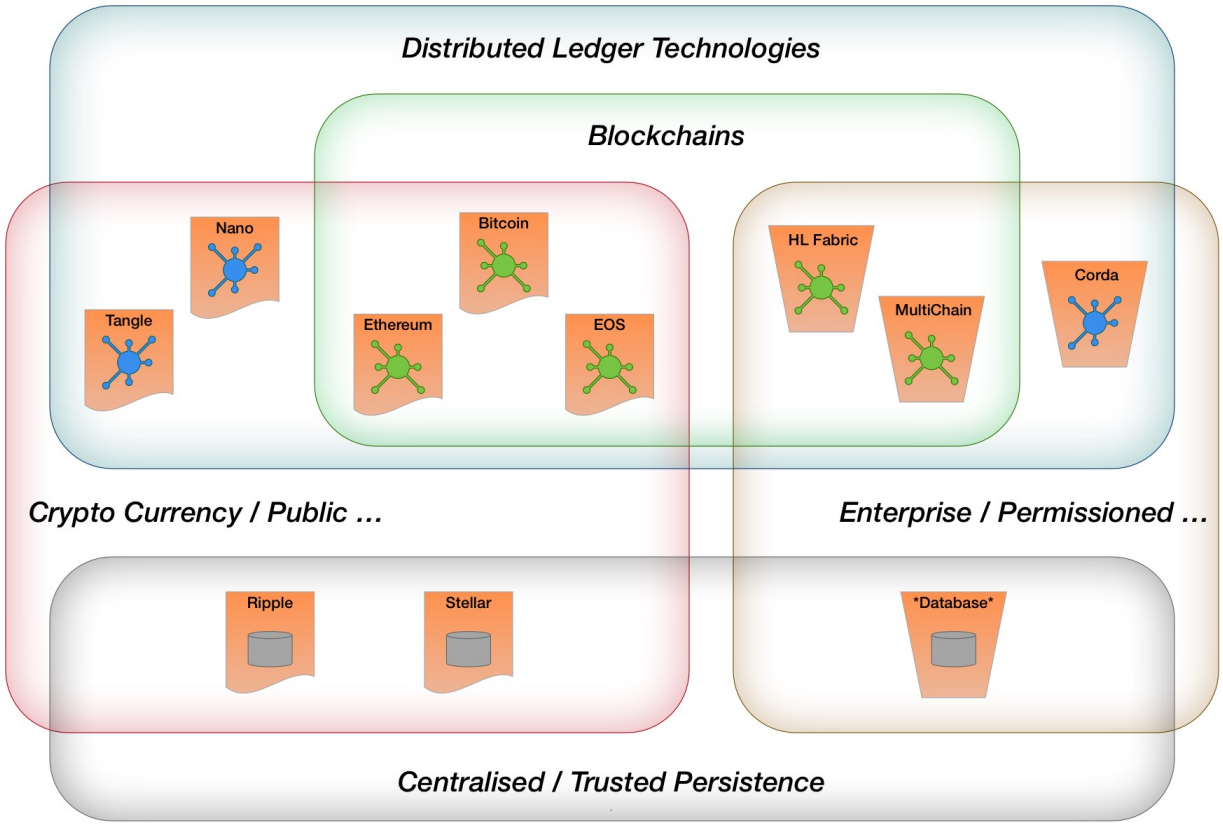

FIGURE 1. DLT Categories

options from both public and permissioned types for building a SLM solution. The area of non-DLT systems, also known as centralized trusted persistence, is not target for our research as we investigate systems that do not rely to trust intermediaries of any type. To be able to cover the most complex cases we focus our research on the DLTs from both public and consortium types.

In next section we define the issues we have in the area of complex SLM procedures. Section 3 we will describe how DLT will solve some typical SLM issues in the areas of data sharing, security and improved prediction. Comparison of several popular DLTs is done in section 4 and the final choice and prototype are described in section 5. Section 6 contains the final conclusions and points some key areas of future work.

COMPLEXITY AND ISSUES OF SLM PROCEDURES

Many parties are directly or indirectly involved in a single SLM project with each one having specific tasks and responsibilities. On Figure 2 are shown the parties, which are involved in a SLM project. Projects with external or public financing usually have some auditing authority assigned to the project. Although they do not have direct contribution to the technical characteristics of the project, they are important part of the overall project and need to have access to project resources to control their deadlines and spending. Technical consultants are key contributors for many SLM projects. They execute most of the technical steps on behalf of the customer. Their area of operation is typically customer's on-premise landscapes, components and services operating in the cloud, as well as IoT edge devices. Consultants also communicate on a technical level with all involved parties. Each complex system has strong dependencies on its software and hardware environment provided by third party software vendors and hardware providers. Operation system, databases, server and IoT hardware add to the complexity of the SLM procedures. Compatibility testing

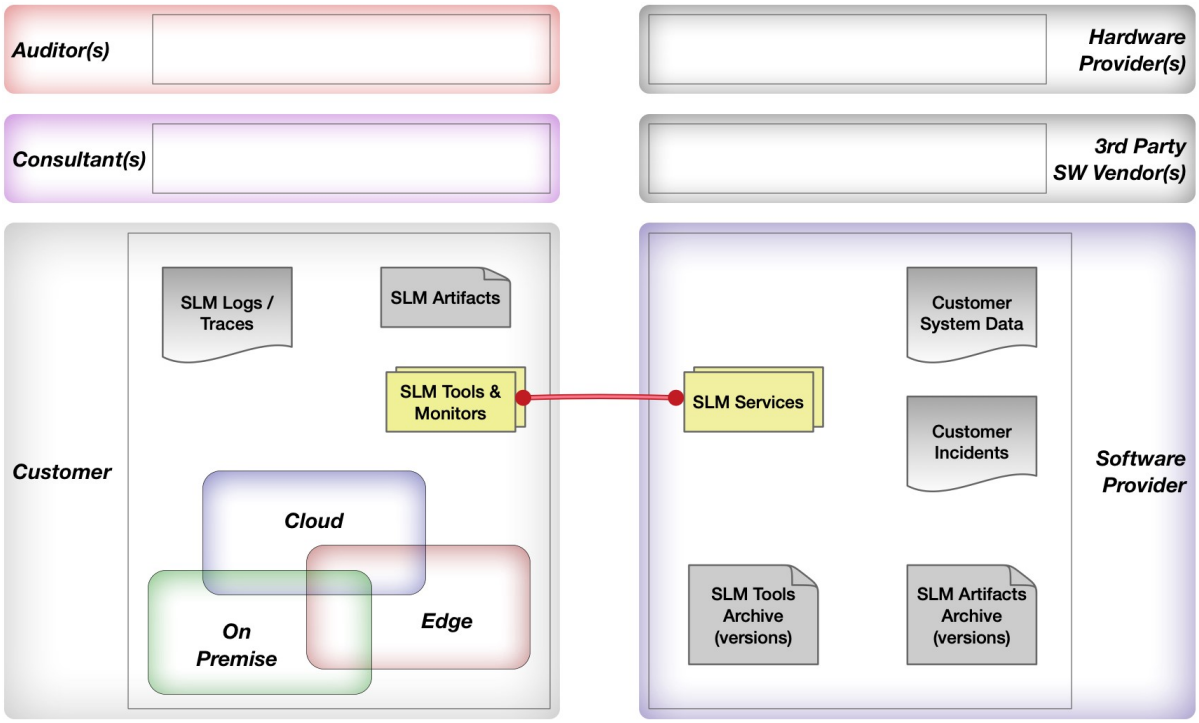

FIGURE 2. Parties involved in a complex SLM procedure.

of the third party hardware and software is essential part of any SLM procedure. The customers are participants who use the systems and as such they define the important key performance indicators (KPI) for the SLM procedure and the system maintained. The exact downtime window, availability setup (high availability and disaster recovery), the access to company resources and the final decision based on the risks and benefits for running each SLM procedure are decided by the customer. The provider of the enterprise software is providing software artifacts for the SLM procedures such as installation files, system upgrades, updates, patches, tools for configuration and software transports, etc.

There are several issues that are part of current complex SLM processes.

- **System Consistency and Security.** There is always a risk that some of the artifacts such as important executables, system configurations, logs and traces are manipulated at customer site. Proper version detection of component executables is also important aspect of overall system security. One possible attack is on SLM data could be hacking the indication that a certain critical patch or fix has already been applied to the productive systems while in the reality the system is still unpatched and exposed to known malicious attacks. It is essential to have a simple and reliable method to verify system integrity.
- **Data Visibility.** Each involved party has access to pieces of the overall data related to its contribution. Often the big picture is not fully transparent for all decision maker. For example, software provider not always knows what the exact component and product versions are used by specific customer. Current usage data is visible to customer and eventually to the technical consultants, but it could not be used by software provider to precisely predict the important parameters of an upcoming update procedure. In case there is a critical fix for specific version of an installed component software provider initially contacts only directly affected customers to handle the risks before going public and getting unwanted attention.
- **Downtime and resource prediction, risk analysis.** Combined and anonymized data is used to make precise prediction on the resources and time for each future procedure. It is also used to evaluate the risks for overtime downtime and other issues related to resource planning. Having a big pool of data for previous procedures allows prediction to be done by various algorithms including artificial intelligence. Data anonymization and processing is something that DLTs handle naturally by properly setup smart contract platform and it benefits from processing and storing data using zero knowledge proof elements.

The three aspects mentioned above will be addressed by using a SLM systems based on a DLT.

DLT-MANAGED SLM PROCEDURES

In this section we investigate the applicability of DLT as solution to the three types of issues described in the previous section. An SLM solution based on DLT benefits from several key characteristics. First, it should be based on a smart contract platform as this would allow strict and flexible rules to be set and enforced. Working with external to the DLT data is essential part of the execution of such system. This requirement will be solved by establishing proper DLT oracles. For example, the software provider maintains a list of supported software versions and artifacts in a central location visible to all participants in the process. A DLT oracle is used to keep the artifact version list on-chain updated and visible to all interested parties. This way of operation guarantees the data of each downloaded SLM artifact is genuine as it contains on-chain a hash of the file stored. So, to validate authenticity of an artifact only a hash has to be compared to stored file. Another usage of DLT oracles is for the installed system to report regularly its KPIs to a central DLT. In case of public DLT this info is properly protected via encryption or other method. Smart contracts could be used to monitor the values and either report unusual values, recommend actions to be taken or even directly to take action for getting the system back in normal condition. Data from all existing processes will be collected. One immediate benefit of using such system is the availability of the collected data from real customer SLM procedures for future analysis by various algorithms. Customer data protection is guaranteed by properly established ZKP system where only relevant data is persisted, aggregated and analyzed by prediction algorithms. As a result, anonymized and aggregated data from many customer installations is used to improve predictions for key parameters of any planned SLM processes. Some of these key SLM procedure parameters are technical and business downtimes, estimated resource usage and risk analysis. Distributed ledger technologies are designed to operate in a no-trust environment and establish rules and fair-play incentives between parties that are not hierarchically organized. The processes are usually transparent although the data could be encrypted or even inaccessible by unauthorized parties. Usage of DLT guarantees that the data is not modified, and some key steps of the procedures could be automated by using smart contracts. There are many factors that have to be taken into account in order to decide on the suitability of specific DLT for automation of SLT processes. Let's explore some of the key abilities of the modern DLT-based development platforms:

- **Smart Contracts(SC).** Smart contracts are defined as customer digital protocols intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract between two or more parties. SCs operate in credible transactions that are trackable and irreversible without relying on third parties for validate. Although the concept of smart contract precedes practical use of DLTs its rise starts with Ethereum. The original concept Ethereum smart contracts were immutable - once deployed they can no longer be removed or modified. The implicit assumption is that their coding is perfect from the very beginning and will never need a fix or extension. This concept took a big hit with "The DAO" case. "The DAO" dApp had a known coding error that was not fixed due to the immutable nature of Ethereum contracts. An exploit was discovered later that allowed hackers to drain most of the funds in the contracts and led to the split of Ethereum network via hard fork.
- **Ricardian Contracts.** Ricardian contracts were invented by Ian Gregg in 1996, some years before first Blockchain was introduced. Their purpose is to provide a method of recording a document as a contract at law and linking it securely to operations of digital systems. Some of the modern DLT systems (EOS, R3 Corda) offer robust support for Ricardian contracts bound to their smart contracts interfaces. They will be used in case of law suits to guarantee that the intent of the smart contract's creators and other SC users. As contrast the popular "The DAO" dApp based on smart contracts on Ethereum platform has only guaranteed that it will only do what its coding will do denying any wrong doing if the contract does not work as its users expect. This approach moves the responsibility of any unexpected behavior to the users of the system claiming that the source code of the contract is the ultimate judge and every user is expected to read the code and figure out all technical aspects. As result even if "The DAO" was hacked and its funds stolen there could be no legal consequences. The hack used part of "The DAO" coding so it was legally acceptable for hackers to exploit it according to the contract. Binding Ricardian contracts to the smart contracts makes sure the intention and responsibilities of the contract and its users are transparent and legally binding.
- **Oracles.** DLTs are intentionally blind to the outside world. This important feature guarantees that each transaction can be validated reliably by everyone. If a smart contract queries an external service, receive result and act upon it, there is no guarantee that the same data will be received (if at all), and execution result can be

validated by other validating parties. This is the reason all smart contract calculations have to be deterministic and predictable. There are some implications from this specific DLT feature. All data has to be made available for processing by the blockchain code by external services a.k.a. Oracles. One of the implications of the requirement of the predictability and determinism for blockchain processing is the generation of true random values that are often required for some algorithms. Another example - a SLM solution in which the Software Provider makes new software versions available in the DLT an Oracle should be established to keep data in the DLT updated. This way the DLT guarantees the authenticity of the SLM artifacts provided.

- **Big file storage.** One of the characteristics of most secure DLTs is the expensive storage for big amount of data on-chain. All the data is replicated among many nodes and is stored for long periods (theoretically forever), so the usage of centralized storage would be a flaw in the distributed nature of the overall solution. It could also become single point of failure for some dApps. Another option is to use distributed storage systems such as the popular Inter-Planetary File System (IPFS). Unlike DLTs data in IPFS could be deleted when needed. Keeping a hash of a file in DLT guarantees that it contains the intended information guaranteed by the blockchain-identifiable source.
- **Data privacy, Security, Zero Knowledge Proofs.** There are many legal regulations that limit collection, use, and retention of personal data (such as GDPR). DLTs normally lack the means to remove on-chain data from their structures once it is added. In addition, public DLTs (Bitcoin, Ethereum, EOS) typically have all their unencrypted data visible to everyone. One way to keep data secret on a public DLT is by keeping it properly encrypted. Another popular option is storing only hash of the data on-chain and the data itself stored off-chain - either in a private/centralized location or in distributed storage. When retention of data is required the safe options are either to delete the decryption/private key for encrypting the specific entry or store the sensitive data off-chain. Another important feature of protecting sensitive data is the mechanism of zero knowledge proofs (ZKP). ZKPs are used when regulations limit the access to parts of the information available which is required by smart contracts for making decision. Using ZKP mechanisms a smart contract could produce and store a proof that certain conditions are met for a given object without persisting personal data. For example, ZKP could be used to make sure the person ordered an alcohol beverage is of legal age for the location he is and where the goods will be delivered. It does not share his name, age, address or other info that is not relevant for the process of ordering.
- **Decentralization, Transaction Finalization, Service Outages.** One aspect of the modern DLTs is the level of their ”decentralization” compared to other DLTs. While being decentralized is perceived as a good thing it brings little value directly to the customers. Proof of work (PoW) consensus method is considered the safest approach toward guaranteed decentralization as each miner is incentivized to validate and create valid blocks competing with the other miners. Major drawback of PoW is its power consumption and scalability. There are many alternative consensus methods such as Delegated Proof of Stake that are solving these issues. One advantage of the public blockchains is that they are always available. Even if large part of the responsible nodes goes down for any reason the rest of the network will keep the DLT running. Another important feature is the transaction finalization - a feature that guarantees to all involved parties that once decision is taken and executed on DLT it could never be reverted. This feature is considered given for public blockchains and have to be considered carefully for consortium/permissioned DLTs. Another useful feature of public DLTs is that they are always highly available and have natural disaster recovery feature.

SUITABLE DLTs FOR SLM SCENARIOS

In this Section we focus our research on four different DLTs and will explore their suitability to support a SLM system. Two of them are public blockchains: Ethereum and EOS, and two are corporate/permissioned ones: Hyperledger Fabric and R3 Corda. One of the major differentiators for choosing which DLT to use is public vs. permissioned DLTs. Having solution based on a public DLT enforces a set of rules that cannot be changed by developer. For example, a system based on Ethereum blockchain will enforce the use of immutable smart contracts and varying block producing time. This decision will have huge impact on the further lifecycle of the overall solution. Costs related to the use of public DLT and distributed storage also differ a lot. Permissioned DLTs are more flexible for development as their deployed smart contracts as a rule are mutable so they are easily upgradeable and extendable. When usage of a permissioned DLT is considered for specific project one key question is the maintenance of the DLTs infrastructure. Here the choice is between relying on a more centralized infrastructure maintained by a smaller group of participants or approach where each participant has to maintain its own piece of infrastructure to participate in the

distributed process. First option negates the benefits of using distributed solution and the second one is more costly for all participants that could make it unattractive. Here is a short overview of the four DLTs we investigated:

- **Etherium.** Ethereum is the first blockchain-based smart contract platform. It uses proof of work (PoW) as a consensus mechanism. There are ongoing activities to switch Ethereum to use proof of stake (PoS) consensus. Currently the top 3 Ethereum mining pools are controlling about 64% of its hash rate. This is seen as decentralization issue since only 3 organizations could anytime agree and execute a"51% attack" on the network. Its block producing time is varying with average about 15 seconds and the total throughput is less than 20 transactions per second globally. The smart contract language is Solidity - a Turing-complete Java script clone designed specifically for running Ethereum smart contracts. A distinctive feature of Ethereum is that its smart contracts are immutable. This approach is designed to guarantee that all data processing is fixed and fare from the very beginning and cannot be tweaked in someone's favor later. . One effect of this design is that even the most devastating bug could not be directly fixed. The consequences of this design decision were seen in "The DAO" case. The 'fix' for that specific dApp issue enforced a 'hard fork' with network revert to older state for the whole Ethereum network and left the original network as Ethereum Classic. User account creation on Ethereum is free but each assets transfers and smart contract deployment and execution are charged respectively in ether and gas. Users of the dApp are charged gas price for each call they make. Paying additional gas for a call helps to get it executed faster. There is good integration between Ethereum and IPFS already.
- **EOS.** EOS is a third generation blockchain platform based on delegated proof of stake (DPoS). It is open source and available for duplication. The main network is the EOS Mainnet but there are several sister-chains such as Worbli, BOS, Telos and others. All sister-chains share the base EOS coding and each dApp normally works on each chain. The blocks on EOS Mainnet are produced by registered EOS block producers (BPs) selected once for each 63 second interval. Each EOS account is allowed to vote for several BPs with its vote weighted with the amount of EOS owned by the account. Only top 21 of the BPs are creating block for each 63 second interval. New blocks are produced each 0.5 seconds. This speed makes EOS one of the fastest public blockchains. The smart contracts on EOS are typically written on C++, compiled locally to Web Assembly and uploaded. The smart contracts on EOS are typically written in C++, compiled locally to Web Assembly, and uploaded to their target network. By default, EOS smart contracts are mutable fixes and extension are done easy by developers with or without touching the existing data and data structures of the dApp. In case developer decide to turn existing smart contract immutable there is an option to remove its accounts public keys. This makes the contract practically immutable since no private key could be used to deploy changes to the contract. There is even a third approach in discussion when the key is removed but it may be returned by consensus of 15 of the 21 BPs. If implemented this feature will be a good compromise between full freedom for dApp developers and the reliability that immutable contracts offer. One important feature of EOS is its support of Ricardian contracts. Using this feature of the EOS smart contract clearly defines purpose and boundaries for using the contract and bound its use legally for both provider and user. The pricing model for EOS is very different to Ethereum and is based on the allocation and usage of EOS's three resources - CPU, NET and RAM. Calling smart contracts and transfer of funds on EOS networks is free. Its internal costs in CPU and NET are automatically replenished in time for each caller account. Creation of accounts require a minimum amount of resources to be allocated. Usage of CPU and NET require some EOS funds to be staked. Stake of CPU grants access to EOS computing time. Stake of NET gives access to its networking features. Used CPU and NET for the account replenishes in time. Staking more gives higher usage limits for each resource and increases the speed of replenishing the resource. CPU and NET can also be un-staked so the allocated EOS funds could be used for other purposes. The account's RAM is measuring maximum storage available to the account. RAM has volatile price and need to be bought before using. When RAM is no longer needed it can be sold back to the EOS RAM service. The cost of running a dApp on EOS Mainnet is mainly the RAM that is used for storing the Web Assembly and data used by the dApp. There are mechanisms in EOS to store dApp data on behalf of the user RAM or on smart contract's RAM pool. In similar manner the CPU and NET used by the contract for consecutive calls could either be charged to the original caller of the service or they could be 'paid' by the smart contract itself. This flexible model allows different models to be implemented. Ricardian contracts guarantee that these mechanisms will not be abused.
- **Hyperledger Fabric.** Hyperledger Fabric is part of the Hyperledger open source community. It was contributed by IBM and was built specifically targeting enterprise use. It does not have its own cryptocurrency. The smart contracts on Fabric are called chaincode and are typically developed on Golang. Fabric chaincode is the only way to access or modify data in Hyperledger Fabric's. It has concepts for "Ledger", "State DB" and "Side

DB". Accessing the Hyperledger Fabric's 'ledger' for example gives access to all transactions that happened in the past regardless of the current state of the objects involved (such as 'deleted' ones). The State DB is rebuilt for each node based on the Ledger info and it contains only the current state of the objects and does not include objects that are set as 'deleted'. "Side DB" is a storage for off-chain data that is still transmitted and validated by Hyperledger Fabric, but it is not part of the its blockchain structures. Technically each Fabric channel is a separate blockchain with own chaincode and users. Parties that have access to the channel see all the information that is part of this channel. In order to restrict the visibility of certain data additional channels are used. The consensus protocols for Hyperledger fabric are highly customizable by using the Endorsement, Ordering and Validation phases for adding data in Fabric. The flexibility of Hyperledger Fabric is an advantage for complex SLM scenarios. Its ability to add whole organizations as participant and its LDAP support allows easy integration in different companies. As Hyperledger Fabric is a blockchain each node/user has a copy of and access to all the information that is relevant for it. This makes it a good fit for guaranteeing system consistency and security. It provides a good medium for sharing information between involved parties and also common store of information that could be used for collecting historical data for past SLM procedures. Often the performance of the system will be improved considerably by properly configuring its smart contracts and policies. Its main advantages are its flexibility and adaptability to different and changing processes. In addition, it is designed to easily integrate with already established enterprise systems. One of its weaknesses is the multi-blockchain approach for data privacy. Each event that require unique set of viewers require separate channel/blockchain to be created. It is not always practical to set separate blockchains depending on intended viewers of a data piece.

- **R3 Corda.** Corda is a non-blockchain DLT built by R3 Consortium for use primary in financial institutions. It has improved scalability compared to any classical blockchain due to its innovative architecture. Unlike the Hyperledger Fabric the consensus for Corda is reached on transaction level (there are no blocks) and always involve interaction with one or more nodes of the so-called Notary cluster to guarantee transaction uniqueness. Corda transactions are by default visible only to the involved parties, and there is possibility on request the full history of funds to be tracked. The transaction info is explicitly and verifiably shared with entities that are not directly involved in the transaction. This consensus approach is perceived as more secure as it does not 'leak' any extra info and allows for very fine-grain tweaks for each process. The Corda smart contracts are typically written in Kotlin language. By design the Corda smart contracts (as well as those of Hyperledger Fabric) are not guaranteed to be deterministic. As such the responsibility of the smart contract developer is to write non-probabilistic algorithms to avoid complicated attacks on the system. To improve SLM processes Corda offers the best scalability under stress, and configurable consensus based on Notary Cluster. Its security model on data visibility is the most complicated so it is much harder for testing, analysing and proving to be flowless. Considering the performance aspect, when properly configured Corda is the best choice for permissioned DLT. Corda could support complex multi-party system for SLM processes operating under heavy load.

One of the assumptions for decision on using DLTs for improving SLM processes is that the benefits of its use are tangible and not motivated by the existing Blockchain and DLT hype. End users will judge the final solution based on the value it brings, its usability, initial cost and cost to keep it running every day. In the ideal case the end user of the system should not know if the system is internally implemented on DLT or not. The comparison between the given 4 DLTs shown on Fig.3 highlights the differences that exist between different technologies. Corporate or consortium usage of the permissioned DLTs (HL Fabric or R3 Corda) requires distributed technical infrastructure to be established and maintained by the involved parties. This includes resource allocation for both hardware and software as well as allocating resources for ongoing updates and configuration changes of the system. Certain qualities of service such as high availability and disaster recovery, following best security practices, etc. has to be planned and executed by all participants. For an organizationally distributed system these costs are considerable if done in truly trustless manner. In addition, if we already have a good level of trust between parties than we will greatly simplify the overall setup by using classical databases instead of DLTs. Using public DLTs on the other hand is restricted by its internal rules. Infrastructure of the public DLTs is already established and maintained by other parties. The dApps running on public DLT are directly consumable even from mobile devices.

The total costs of running a SLM dApp on Ethereum and EOS are considerably different and have different pricing model. If we think of all cost for all participants in the procedure, we see that all costs in Ethereum are final. Developer of the dApp make a single payment when deployment of the dApp. DApp updates are not supported so no further investment is required by developer. Users on the other hand will keep paying for using the dApp since each call to the dApp costs Gas to the caller. EOS model is based different. The running cost associated with using

	Ethereum	EOS (Mainnet)	HL Fabric	R3 Corda
Type of DLT	Public	Public	Permisioned	Permisioned
Consensus Type	Proof of Work	Delegated Proof of Stake	Configurable per channel	Notary service + UTXO
Smart Contract Type	Immutable	Mutable/Immutable/BP	Mutable	Mutable
Smart Contract Programming Language	Solidity	C++	Golang	Kotlin
Legal Binding Agreement	- - -	Ricardian Contracts	- - -	Ricardian Contracts
Deployment Cost for dApp Publisher/Operator	Gas price for deployment	Cost: RAM (code + data)	(Distributed) Infrastructure	(Distributed) Infrastructure
Maintenance Cost for dApp Publisher/Operator	- - -	Cost: RAM delta (+/-)	Maintain (Distributed) Infrastructure	Maintain (Distributed) Infrastructure
Motivation to run DLT	Miner's reward per block (inflation)	Block producers' reward (inflation)	Infrastructure maintained by involved parties	Infrastructure maintained by involved parties
End user joining	Free	~ Free minimum RAM, CPU, NET	Restricted	Restricted
End user costs	Gas cost per dApp call	Practically free / Replenishable resources	- - -	- - -
On-chain data visibility	All persistent dApp data is visible globally	All persistent dApp data is visible globally	Full visibility per Fabric channel	Only participants see transactions. Transaction proof may be shared.
Scalability	Limited due to PoW used	Improved due to DPoS and limited number of BPs	Each channel is separate blockchain	Excelent, Native sharding support
Consensuis	Proof of Work	Delegated Proof of Stake	Configurable per channel	Verify: Required signers Unique: Notary service
DLT Security	Top 3 Mining pools control 64% of hashrate	Top 21 Block Producers votable each 63 sec.	Private (Distributed) Infrastructure	Private (Distributed) Infrastructure
Transactions per Second	Up to 20 transactions per second globally	Limited to 4000 TPS (Mainnet only)	Configurable per channel	N/A
Block producing time	~ 15 seconds	0.5 seconds	Configurable per channel	N/A

FIGURE 3. Comparison between selected DLTs.

the dApps are predominantly replenishable. After CPU and NET are 'consumed' during EOS smart contract calls are made, they start to replenish with a speed proportional to the staked amounts for these resources. In addition, even the staked resources later are unstaked, when they are no longer used, the released EOS funds could be used for other purposes. Price of the RAM resource is payed for initial deployment of a dApp and is used to store any collected data by the dApp and its user. The EOS pricing model is more flexible compared to Ethereum and allow different pricing options. Another benefit block producing time is significantly short allowing more responsive applications and close to real-time data processing. An additional benefit of EOS for running prototypes is the ease to do any modifications in already deployed smart contracts.

For the aim of the current investigation we choose EOS as a platform for our prototype development. In next Section we shall describe this prototype.

PROTOTYPE FOR DLT-BASED SLM SYSTEM

Based on the discussions in the previous section we choose to build our prototype on EOS Mainnet and use IPFS as distributed storage for large or sensitive files. The overall solution will have the two distributed systems interacting with each other in a straight forward manner. Each relevant file stored in IPFS will be a referenced by smart contracts of one or more dApps running on EOS Mainnet containing its hash. As usual the reference itself is the hash of the

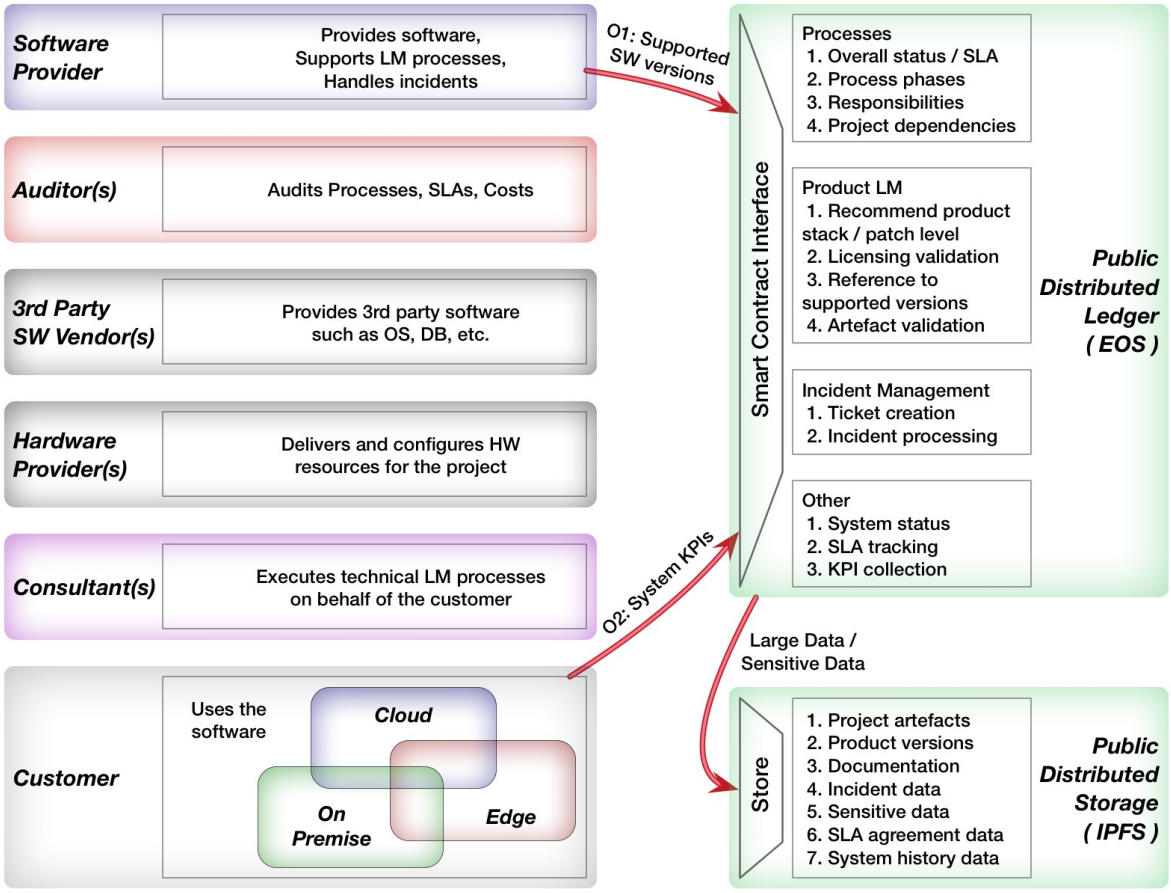

FIGURE 4. DLT-based SLM Prototype (EOS Mainnet + IPSF).

file used to locate it in IPFS. The smart contracts have several parts responsible for different SLM functionalities of the prototype. As it is shown on Fig.4 some functions are Processes Management, Product Lifecycle Management, Incident Management and others. All non-encrypted data of the dApp is accessible by all participants of the processes, as EOS is a public blockchain. For the purpose of the prototype a couple of oracles are established to test the abilities and costs of the selected DLT. First oracle (O1) is responsible to update the catalog of supported software artefacts uploaded by software provider directly in IPFS and referenced by EOS Mainnet. Once new artifact version is offered publicly the oracle uploads the relevant file(s) to IPFS and stores its hash on EOS Mainnet. Deprecated artifacts are 'unpinned' from IPFS and their hash removed from their on-chain storage. Data provided by this oracle is not updated often only when new software artifact version is released or deprecated. For this oracle the critical resource is the EOS RAM as the list of product versions and update artifacts may grow in time and each entry will require some RAM to be available. The amount of RAM for each entry is small as it contains only hash/reference of the IPFS file and some relevant metadata such as type of the artifact and associated product versions. Fortunately, once a software artifact is no longer supported its RAM is released. This RAM could either be kept for using for other catalog entries later or immediately sold back to EOS network. As part of the process the file stored in IPFS is unpinned and ready for deletion. The usage of CPU and NET for this oracle are negligibly small as entries to EOS are small and updates of the catalog are done rarely. Access to the software catalog is open to public so no further security measures are required to protect or encrypt it.

The second oracle (O2) will have an agent(s) running in the customer landscape - on premise, in cloud, or edge. It will regularly update customer systems status on the DLT by uploading its recent KPI-relevant data to the dApp/smart contract. If KPI data is not in its normal boundaries the systems status will signal that the system require attention directly to customer, its consultants, software provider, or other relevant parties. In the further stages of the prototype as

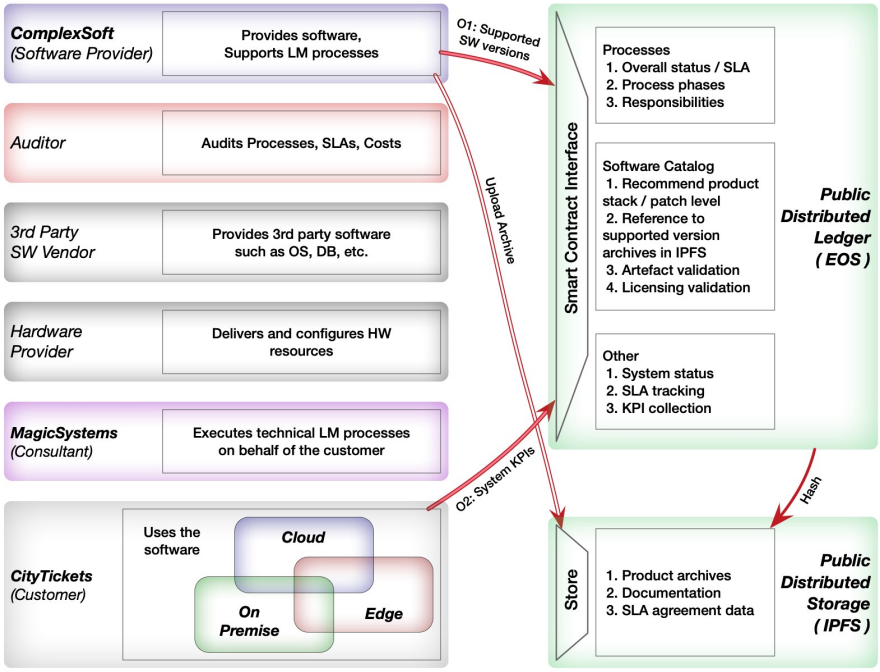

FIGURE 5. Case study for DLT-based Prototype.

reaction to such events the DLT system will trigger actions to mitigate the exceptional case or even enforce automatic payments to compensate for the breach of the SLA terms. The critical EOS resource for O2 oracle is the CPU used as we need to have enough to constantly supply real-time data to the dApp. Our initial tests show that a simple data supplier to EOS Mainnet is called once per minute indefinitely by staking about 1 EOS for CPU. Usage of RAM and NET for O2 are negligible as the data on the service overwrites its older values and keep only a fixed set of data on EOS Mainnet. The information of about business systems status is sensitive information, so the real data collected by such process will always be protected or encrypted. Data encryption is not established initially for our prototype, but it could be added at next stages of its development.

Access to dApp public data by all involved participants is done directly by using any public EOS block explorer. EOS block explorer API is used for integration of applications and processes. The most prominent approach for data protection is having it encrypted. For the initial prototype version data encryption is not a part of the evaluation.

Lets consider a simple case study where we could trace in concrete steps the interaction between specific participants. The company CityTickets is selling tickets for city attractions via several channel directly online or via special hardware located in the city. Their productive landscape consists of some cloud-based services some critical systems running on premise and their ticket selling machines as IoT edge devices. The generic software for the ticket selling is provided by ComplexSoft. The implementation of ComplexSoft for CityTickets is responsibility of the consulting company MagicSystems. The proposed setup is used by the CityTickets to monitor and support their system for automatic ticket. As part of the system the ComplexSoft, has established agents reporting the systems KPIs via O2 interface. Based on the collected real use data from the systems used by CityTickets (and other clients using the same product versions) ComplexSoft pinpoints performance bottlenecks and identify in which of the delivered components they are located. Their newly developed version of the affected component addresses the issue and it was successfully tested internally. Now they have to deliver the new component version to its clients so they could benefit from the improved performance. ComplexSoft prepares an archive of all relevant artifacts and the new version metadata and upload the archive to IPFS. Once the archive is there it is publicly accessible by its unique hash value. Next step for making the new version available is to make it visible by adding it to the on-chain software catalog via O1. Note that only the IPFS hash and component version metadata is uploaded to the DLT, no binaries, documentation or configuration need to be kept on-chain. Once the software version catalog is updated in the public distributed ledger all involved parties are notified about the existence of the new version of the software they are using. The Auditing party is notified

about the new version. CityTickets and their implementation consultant MagicSystems get familiar with the benefits that the new software version brings. Consultant could download the relevant archive from IPFS and run test of the update procedure to evaluate the impact of the procedure eventual downtime(s), investment of resources and risks of the procedure. In some cases, they need to get involved also the providers of the hardware and the third-party software providers in the evaluation process. Based on the provided data from MagicSystems evaluation CityTickets decides between the cost for recent software update against the benefits of the new version being used. Now CityTickets can make informed decision if they need to update or not. If they decide to go for the new version, they choose downtime window for the procedure and assign running the procedure to their consultants MagicSystems. Once the procedure is executed the software provider is automatically notified by O2, so they also know the parameters of the update procedure: if it was executed on time, on budget and if the expected benefits of the new version are in place after the update. The overall setup and interactions are shown on Fig.5.

In summary - our prototype consists of several components. The core is a set of EOS smart contracts combined into single dApp deployed to EOS Mainnet. It also has a set of sample files stored in IPFS that are used as a model for versioned SLM artifacts, two oracles for maintaining software artifact catalog and for supplying real-time data. The prototype also includes client applications for data retrieval and (in the future) action based on the dApp information available on the EOS Mainnet. The purpose of the prototype is to validate the overall assumptions that a solution for SLM system based on EOS is a practical solution to the existing SLM issues.

CONCLUSION

In this paper we presented a prototype of a distributed system based on DLT used to automate and optimize the execution of multi-party SLM processes. Even if all the DLTs that are analyzed are fulfilling the requirements for a complex SLM procedures we have chosen EOS as a base of our prototype. EOS-based solution require relatively low level of ongoing investment and it best fits the natural performance and scalability requirements, it supports free transaction, uses simple but powerful development model and flexible pricing models. One issue that is not covered in this paper is the protection of confidential data stored in the distributed ledger and distributed storage. Some SLM processed are covered successfully without investing in data encryption and complex retention policy. A flexible and generic SLM system needs to be compatible to various data protection regulations such as GDPR. In the future work we will focus on exploring options for providing proper data protection and support the data retention policy required by various regulations.

ACKNOWLEDGMENTS

This research was partially supported by the National Scienti c Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)", financed by the Ministry of Education and Science, Bulgaria.

REFERENCES

[1] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.
[2] V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform*, 2013.
[3] X. Xu, I. Weber, M. Staples *Archtecture of Blockchain Application*,Springer, 2019, ISBN-978-3-030-03034-6.
[4] I. Bashir, *Mastering Blockchain*, Packt Publishing, 2018, ISBN:978-1788839044.
[5] R. Etwaru, *Blockchain: Trust Companies*, Dog Ear Publishing, Indianopolis, 2017, ISBN:978-1457556626
[6] J. Bambara, P. Allen, K. Iyer, R. Madsen, S. Lederer, M. Wuehler *Blockchain: A Practical Guide to Develop Business, Law and Technology Solutions*, McGraw Hill Professional, 2018, .
[7] M. Corrales, M. Fenwick, H. Haapio *Legal Tech, Smart Contracts and Blockchain*, Springer Nature Singapore, 2019, ISBN: 978-981-13-6085-5.
[8] D. Larimer, *EOS.IO Technical White Paper*,2017.
[9] G. Greenspan, *MultiChain Private Blockchain?White Paper*, 2015.
[10] R. Brown, *The Corda Platform: An Introduction*, 2018.
[11] S. Popov, *The Tangle*, 2015.
[12] C. LeMahieu, *Nano: A Feeless Distributed Cryptocurrency Network*, 2018.