Distributed Ledger Technology for Decentralization of Manufacturing Processes

Mauro Isaja
Research & Development Unit
Engineering Ingegneria Informatica (ENG)
Rome, Italy
mauro.isaja@eng.it

John Soldatos
IoT Group
Athens Information Technology (AIT)
Maroussi, Greece
jsol@ait.gr

Abstract — **Distributed Ledger Technology (DLT) is probably going, in the near future, to disrupt B2B and B2C interactions even more than the advent of the World Wide Web, thanks to the transfer of trust from personal and commercial relationships to computing algorithms. However, a less commonly perceived property of DLT is that of enabler of decentralized computing. In this paper, we explore the use of DLT to innovate Industrial Cyber-Physical Systems.**

Keywords — DLT; Blockchain; Manufacturing; ICPS; Decentralization.

I. Introduction

Today, Distributed Ledger Technology (DLT) – popularly known as Blockchain – is often perceived as the catalyst of an IT revolution to come, likened by some to the advent of the World Wide Web in the nineteen-nineties. The Bitcoin network, which can be considered as the "killer application" of DLT, has been dubbed "Internet of Money" [1]. Hype notwithstanding, the implications of DLT on the growing connected world can indeed be huge, provided some technical (e.g., lack of scalability) and non-technical barriers (e.g., lack of regulation) are removed or lowered. Its better-known feature is indeed ground-breaking: peer-to-peer trustworthy and secure transactions on a public network without requiring trusted intermediaries, or even any form of trust between parties. Moreover, enabling *zero-trust* business ecosystems is not the only arrow in the DLT quiver, as will be discussed further on.

In recent years, analysts all over the world have identified a great number of applications that would benefit from – or even made possible by – the introduction of DLT, the most common use cases being in the realms of finance, energy, supply chains and e-government. In this paper, though, we choose to tread an unconventional path: exploring DLT as means of *decentralization* of control over manufacturing processes. In other words, determine how to exploit the unique characteristics of this technology to virtualize, flatten out and open up the centralized/hierarchical *automation pyramid* [3], with the final goal of enabling more flexible, secure and robust Industrial Cyber-Physical Systems (ICPS). This is not an easy task, as many of the non-functional requirements of ICPS are pushing the envelope of DLT as we currently know it.

The challenge we are taking is then two-sided: on the one hand, we need to assess the capabilities of DLT and of its state-of-art (SotA) implementations, not only in terms of functionality but also from the perspective of performance and scalability in ICPS scenarios; on the other, we have to identify those use cases that are both tractable and promising.

II. The Distributed Ledger Concept

Generally speaking, a distributed ledger is a "system of record" that is replicated and kept in-sync across multiple nodes of a network, where all nodes are *peers* and no "master" copy of the ledger exists. More specifically, in concrete DLT systems the ledger is implemented as a linear sequence of records that are individually *immutable* and timestamped. The sequence itself can only be modified by appending new records that have been validated by *consensus* among peers, so that records are guaranteed to be legitimate and cannot be censored or retracted after commitment. The integrity of both records and sequence is protected by means of strong cryptographic algorithms [12]. Given that both the ledger and the consensus mechanism are decentralized, the only way of compromising or putting down a "pure DLT system" (i.e., one that does not have any single-point-of-failure in its architecture) is by taking control of a significant number of its nodes – ranging from N/3 + 1 to N/2 + 1, depending on the consensus mechanism used, where "N" is the total number of nodes.

While the potential advantages and pitfalls of such an architecture are quite clear, it was not until the publishing of the seminal Bitcoin white paper [2], shortly followed by the public release of the first Blockchain implementation in 2009, that DLT has been recognized as a viable approach for building real-world applications. It should be noted that DLT and Blockchain are not synonymous: the former is a more general definition than the latter, which is tied to a specific technical design where the records on the ledger are *blocks*, each containing a group of *transactions* – i.e., atomic changes applied to system state. Most DLT implementations of today are Blockchain-based, two well-known exceptions being FAR-EDGE project [15]. Whatever the implementation, though, the individual transaction is always the finest-grained element of the ledger and, as we will shortly see, the most critical one with respect to our objectives.

In legacy Online Transaction Processing (OLTP) systems, like modern relational databases or even old-school mainframe applications, committed transactions are *final*: the only way to void the effect of a committed transaction is to commit a second transaction which counters the effects of the first. In the DLT world, however, this is not always the case, due to the need of going through a complex and sometimes time-consuming consensus protocol that may involve the entire network. Most Blockchains will allow committed transactions – i.e., transactions included in a block that has been already appended to the chain – to be discarded because the ledger, as a whole, was *rolled back* to some past checkpoint. Events like this will leave internal consistency intact but make the whole system *nondeterministic* from the point of view of external watchers, as explained in more detail in the next section. At first sight, this may appear as a fundamental flaw, but actually many real-world applications can easily live with it. For example, the transfer of ownership of digital tokens (e.g., a cryptocurrency like Bitcoin) between two digital wallets affects the token balance of the involved parties: if for some reasons the transfer is first approved and then made void, the balance of both wallets is automatically restored to its previous value (more correctly, the change never actually happened). Probably, the same transaction – if legitimate – will then be repeated successfully. In this case, the use case works fine enough because it is self-contained: there are no external side effects that cannot be cancelled. Clearly, this is not the case when a DLT-based application is in control of a physical process: a hole drilled in a metal part cannot be undone because the triggering command was cancelled after the event.

So this is the first, mandatory requirement for any DLT platform looking for adoption in smart factory solutions: *finality of transactions*. To understand how DLT can meet this requirement and, most importantly, what is the impact of this most desired feature on other system properties, we first must understand in detail what goes on under the hood. Let's then have a brief look at the inner workings of a typical Blockchain.

III. INSIDE THE BLOCKCHAIN

As the name implies, a Blockchain is a sequence of blocks which are cryptographically sealed and chained together in a way that ensures that any breach in their integrity can be easily revealed. Each block contains a number of committed transactions. Transactions issued by clients but still waiting for validation are left pending in a system-wide virtual memory area (often called "mempool"), until a new block is created and appended to the Blockchain. This happens on a regular basis (10 minutes average interval in Bitcoin, 15 seconds in Ethereum) to allow an optimal balance between the needs of consistency (transactions being properly validated) and of responsiveness (transactions not taking too long before being confirmed). The big problem, however, is that there is no central coordination in place, so which node is going to validate pending transactions and append the new block? How the system as a whole will express its agreement, or disagreement, with the block's contents? Most importantly, the entire process should still be possible and reliable in case some nodes are unresponsive and/or misbehaving.

Decentralized validation of messages is an issue known in game theory as the Byzantine Generals Problem [4]. Any decentralized system that implements a concrete solution to this problem is said to have the property of *Byzantine Fault Tolerance (BFT)* [5]. Usually, a concrete BFT solution has three distinct *phases*: P1) the system elects a validator node, P2) the validator node validates the transaction(s), P3) the system reaches a consensus on the correctness of such validation, possibly rejecting it if there is no positive agreement. The critical steps are P1 and P3, because they have no easy solution in a context where mutual trust is limited or even not an option at all, like when system nodes are operated by anonymous entities – which was, we should remember, the reference scenario in the early days of DLT.

The greatest innovation introduced by Bitcoin – and still used by most second-generation DLT architectures – is the *Proof of Work (PoW)* scheme in conjunction with a system of economic incentives, to the effect that node owners are turned into business stakeholders. In practice, there's a random cryptographic problem that a node has to solve, by brute computing force, before all the others in order to gain the right to validate a new block of transactions. The winner of this race is the elected validation node (BFT P1, see above). The validation node will then:

1. Validate all pending transactions according to system rules and to its knowledge of the current state of the ledger (BFT P2).
2. Create a new block containing the *allegedly valid* transactions, the correct solution of the cryptographic problem (the proof of the work done) and a link to the latest block on the chain that is deemed valid (more on this later, when we discuss how BFT P3 works).
3. Broadcast the new block on the network, so that all online nodes can update their copy of the ledger (offline nodes will sync up when they get back online by retrieving any new blocks from their peers).
4. After the new block is accepted by the system (BFT P3), *receive a price*; in cryptocurrency systems, this is an amount of currency tokens that are created anew for the purpose, which is why this process is commonly known as "mining".

Fig. 1 below shows how a Blockchain ideally looks like, with all its fundamental elements in place.

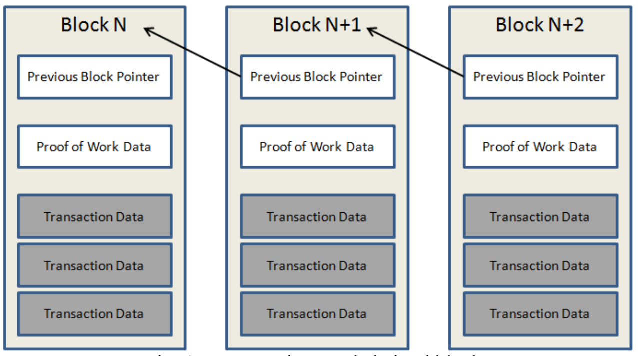

Fig. 1 - Transactions and chained blocks

PoW-based BFT has two interesting properties: it is expensive for nodes to solve the cryptographic problem, and the chance of winning the race is linearly proportional to the computing resources invested. This is what being a business stakeholder actually means: there's a significant cost in participating, and profit will come only by strictly adhering to system rules. However, PoW also has a big downside: it is extremely resource-intensive and wasteful, as all nodes, possibly thousands, have to repeat the same heavy work endlessly, most of the times to no avail. Other mechanisms have been devised in recent years (e.g., Proof-of-Stake [14]) to achieve similar results with less adverse effects, but our analysis is not going to follow this thread as it is not relevant to the key point of our research: finality of transactions is only impacted by how the consensus protocol – i.e., BFT P3 – is designed. To understand why, let's start by looking at how this is phase is resolved in our typical Blockchain.

Interestingly enough, BFT P3 starts *after* a new block of transactions is written to the ledger (step #4, see above): consensus takes an indefinite amount of time to consolidate and the ledger itself is used as a persistent "data bus" during the process. Initially, no consensus can be assumed: the block at the head of the chain contains just what the latest validation node *alleges* are valid transactions. However, every new block that is appended is implicitly stating an endorsement: all transactions in all preceding blocks are valid (step #2, see above). So what happens when the current validation node disagrees with the current state of the ledger? In that case, it will ignore the head of the chain and will instead append the new block after the latest block that it deems valid. This action will create a temporary *fork* in the Blockchain (called "soft fork"), as represented in Fig. 2 below. The two resulting branches are, to all effects, alternate and incompatible versions of reality that must now compete against each other in order for the "right" one to prevail.

In the upper half of the above picture, two short-lived

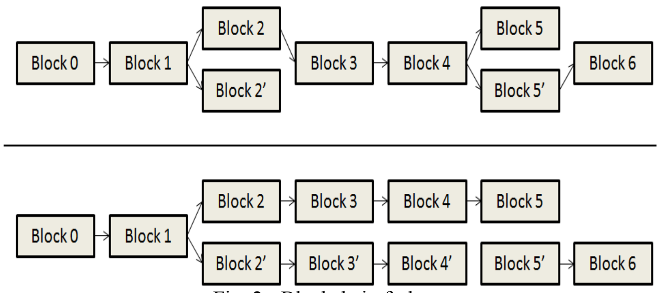

Fig. 2 - Blockchain forks

forks have happened. The first time, nodeA appended Block2 after Block1, endorsing both Block 1 and Block 0. Then, when nodeB became the elected validator, it decided that one or more transactions in Block2 were not valid. To express its dissent, it created an alternate version without the offending data – Block2' – and appended it after Block1 as well, to signify endorsement of Block1 and Block0 but not of Block2: this created a fork. Later on, when nodeC created Block3, it decided that Block2 was more correct than Block2', so it choose to endorse the upper branch of the fork. Over time, this choice was also endorsed by other nodes, so that the lower became a dead branch. A similar situation occurred after Block4, this time with Block5' emerging as the winner over Block5. This example is typical, in that the occasional divergence in how different nodes "see" the common system state is reconciled quickly with little or no disruption at all. In some cases, less frequent in practice, the consensus process may take a longer time to elect the winner branch, as depicted by the lower half of the picture. Note that there is no guarantee of correctness implied in this algorithm: there is just the reasonable assumption that the majority of nodes will, in the long run, make the best choices.

That said, finality of transactions is not compatible with Blockchain forks. What we need is a BFT P3 process such that all stakeholders are able to reach a quick and final agreement over the network *before* a transaction is actually saved to the ledger. At the time of writing and to our knowledge, this objective cannot be achieved without placing some degree of trust on some "special" nodes that play a privileged role. Different architectures have been devised along these lines, but they all share two common traits: a) nodes have a strong digital identity, which implies a trusted, centralized identity provider/manager, and b) there is a neat separation of roles between transaction validation (still decentralized) and actual commit, which requires a trusted, centralized queue manager to serialize validated transactions in a consistent order. Such DLT architectures are commonly referred to as *permissioned* because nodes must receive permission by a central authority in order to operate according to their assigned role. Permissioned platforms are typically used to support private business networks, and have a big advantage over permissionless, or public, systems: they do not have to rely on complex and inefficient mechanisms, like PoW, to elect validation nodes. Such radical simplification may have a dramatic positive effect on scalability, if not countered by other bottlenecks. However, we must also clarify that the "permissioned" quality by itself does not bring any guarantee of transaction finality: permissioned Blockchains do exist (e.g., private Ethereum networks) that still rely on forks to resolve conflicts.

Before delving into SotA analysis, though, in search of implementations that meet this requirement, we must briefly discuss another key DLT capability: the support for complex business logic as an integral part of transaction validation.

IV. DLT AS A COMPUTING PLATFORM

In the previous chapter we saw how, in DLT, each valid transaction must abide by common rules. More concretely, these rules are implemented as executable code: a program that gets its input from the payload of the transaction (i.e., data provided by the issuing client) and from the current state of the ledger.

One of the key differences between DLT platforms is how business logic implemented and managed. First-generation ones, like Bitcoin and its derivatives, have hardcoded rules and a fixed data model, on account of being dedicated cryptocurrency systems. Second-generation platforms as Ethereum are much more flexible and powerful: custom business logic and data models can be defined by users in the form of *smart contracts* Smart contracts can do what hardcoded business logic does and also much more – e.g., running computations, calling external network services (with care, as will be explained later) and, most importantly, produce some output that is saved on the ledger. Their deployment process is very simple: a transaction that writes the smart contract's code to the ledger. This implies that the code is "sealed" and can be therefore safely executed by any node of the system. In the Ethereum public network, a well-proven infrastructure is also in place that prevents misuse of resources (basically, invoking a smart contract requires the caller to pay to the network an amount of "local currency" – the Ether – that is calculated from the actual use of CPU cycles and ledger storage space). In permissioned platforms, where a governance entity is in charge, these restrictions do not apply but smart contract authors must be authorized to deploy their code.

The strong point of smart contracts is that they turn the Blockchain – a very solid, secure but "passive" database – into a *distributed service platform*. Smart contract-powered services are akin to *serverless* cloud architectures (e.g., AWS Lambda [16]) but inherit some of the best qualities of DLT: decentralization and robustness. But then, they also get some bottlenecks from DLT, affecting performance and scalability. The three main limiting factors are *storage inefficiency*, *serialization of transactions* and *confirmation latency*.

Storage inefficiency is the obvious drawback of having the full ledger locally replicated by each and every validation node in the network; this is a big problem in permissionless systems with thousands of nodes (e.g., >27000 in the Ethereum public network), but a lesser issue in permissioned ones with a few privileged nodes doing all the work.

Unfortunately, serialization of transactions is a bottleneck that applies equally to any kind of Blockchain. For this reason, it deserves a few more words of explanation. In OLTP systems, concurrent data access is managed at the record level: multiple transactions are executed in parallel if the affected record sets do not overlap. Conversely, DLT transactions affect the ledger as a whole, so data updates must be serialized to prevent conflicts. This feature is also what makes DLT systems less scalable: adding more computing nodes will not help coping with increased workload, given that transactions must still be queued in a single line. New DLT architectures are emerging that will address the serialization problem in the future. One of them is the *Sharding* strategy proposed by Ethereum, according to which the global ledger can be partitioned into smaller ones having a narrower scope. Another example is the *Tangle* architecture from IOTA, where individual transactions are linked in a direct acyclic graph [6]. At the time of writing, though, even the most evolved of these systems are still at the proof-of-concept stage.

Finally, confirmation latency refers to a problem that stems from BFT: from the point of view of the issuer, a transaction is only confirmed when approved by the whole system – i.e., confirmed by consensus, the third phase of BFT. Now, this may take a short or a long – sometimes very long – time, depending on the DLT implementation. If our system does not support "final" transactions, the only way to be on the safe side is to wait until a number of blocks are appended after the block that contains our transaction. How many blocks? It depends on the system and on the desired level of confidence, but it is not something that can be determined in objective terms. In the Bitcoin network, the standard practice is 6 blocks, which amounts to 1 hour time (and this is another good example of the issues to face when dealing with a nondeterministic accounting machine). However, as explained in the previous section, final transactions are those that are subject to an online consensus protocol and reach the ledger only after approval. In this context, we can expect a significantly reduced latency – in the order of magnitude of milliseconds – as this kind of interaction is little more than a peer-to-peer poll. Once again, permissioned architectures (a prerequisite of transaction finality) come to our rescue. That said, a word of caution: even if we can remove BFT-induced latency from the picture, DLT will still perform worse than legacy OLTP systems in terms of raw transaction throughput, due to the transaction serialization bottleneck.

Having identified all the weak points of DLT, it's time to put this knowledge at work. DLT performance is a critical issue when faced with real-time nature of ICPS. How the two worlds can cooperate? This question can be decomposed into:

- What is the *comfort zone of DLT* performance? I.e., what kind of workloads DLT platforms can digest without throttling dependent processes?
- What are the *typical ICPS scenarios / use cases* that can benefit the most from DLT, but are also compatible with its limitations?

V. THE DLT COMFORT ZONE

The objective of this study was to set the boundaries of the so-called DLT comfort zone in terms of a few simple, objective (i.e., measurable) and high-impact *key performance indicators (KPI)*, targeting the weak points of DLT:

- Transaction Average Latency (TrxAL) – The average waiting time for a client to get confirmation of a transaction, expressed in seconds. The lower the value, the better.
- Transaction Maximum Sustained Throughput (TrxMST) – The maximum number of basic transactions (i.e., no business logic) that can be processed in a second, on average. The higher the value, the better.
- Data Replication Factory (DRF) – The average number of times data items are physically duplicated on the distributed ledger. The lower the value, the better.

Of course, this kind of benchmark can be only defined by putting *concrete* DLT implementations at test, in a controlled environment. Due to time and resource constraints, we knew from the beginning that we could only focus on a very small number of platforms. We then started with some basic SotA analysis in order to select the most promising candidates.

First, we drastically reduced the number of platforms to be assessed by applying a qualitative filter: only major open source projects, backed by a large and active community of developers. This criterion cuts down the DLT landscape to just 9 platforms: Bitcon and derivatives, Ethereum, EOS, NEO, Graphene, IOTA, Hyperledger Fabric, Hyperledger Sawtooth, R3 Corda. Then, we further pruned our list by excluding those platforms that do not support transaction finality; this left us with just two candidates: NEO [17] and Hyperledger Fabric (HLF) [18].

For brevity, we omit here the detailed analysis of the specs of these two platforms, which are very different in all respects except for their common vocation: supporting enterprise and cross-enterprise applications. We just want to mention the consensus protocol implemented, as it's the key enabler of transaction finality. NEO is based on an original Delegated BFT algorithm [7]. HLF also uses its own algorithm, called SIEVE, based on "classic" Practical BFT [8] with the addition of speculative execution of validation logic [9]. The really interesting part of the story is how these deterministic BFT implementations are performing. Here below is a summary of our findings – for some of which we in debt with the BLOCKBENCH team, which developed an open source stress-test software suite specifically targeted at Blockchain platforms [10].

TABLE I. KPIS: NEO VS. HLF

			NEO	***HLF***
TrxAL (Transaction Average Latency)		sec.	7-10	0.1-51
TrxMST (Transaction Maximum Sustained Throughput)	gross	trx/sec.	1000	1250
	fast	trx/sec.	0	160
DRF (Data Replication Factor)			8	8

It is important to understand the difference between the two distinct measurements of the TrxMST indicator reported above. The first, "gross", is the maximum number of transactions that can be processed in the time unit *regardless of latency*. This value is useful to assess the capabilities of a platform in general, but what we really need to know is the maximum throughput *within the limits of an ICPS-compatible latency*. In order to determine the maximum acceptable value of TrxAL in our context, we are forced to adopt some degree of arbitrariness: we first have to identify a subset of ICPS use cases that are tractable by DLT, and from there derive generic TrxAL requirements. Ruling out real-time automation, which is clearly beyond DLT capabilities, we argue that *indirect control* and *notarization* are the best reference scenarios. Two concrete examples are factory-wide coordination/orchestration of local real-time processes and security-related logging, respectively. In this context, based on professional experience, we consider a one-second latency to be the upper limit. This explains the second measurement of the TrxMST indicator, "fast", on which the TrxAL <= 1 condition applies.

The DRF indicator is also worth commenting, because it tells us about the viability of a DLT system in data-intensive scenarios. Typically, this value reflects a performance characteristic of a specific deployment, not of a software implementation in general: in most DLT platforms – NEO and HLF are no exception – a full copy of the ledger is maintained by each validation node, so that the DRF value is simply the number of validation nodes actually deployed.

Having described the reference framework, we can now analyze our KPI results. These have been obtained on two equivalent systems: 8 "virtualized" (Docker images) validation nodes running on low-end server class commodity hardware.

In the first place, these results tell us that the two platforms under test, while having similar "gross" processing power, have very different behaviour with respect to latency. NEO's TrxAL is nearly constant, amounting to 7-10 seconds. This is due to a process that queues up confirmed transactions until a new block containing them is written to the ledger, which happens at quite regular intervals regardless of the actual workload. Instead, HLF's TrxAL is workload-dependent, ranging from 0.1 seconds or less under light load but rising to over 50 seconds when the maximum throughput limit is reached. If we look at the "fast" TrxMST indicator, though, we see that HLF is capable of processing 160 transactions per second with ICPS-compatible latency, while NEO – for the reason explained above – cannot comply at all.

When it comes to DRF, both platforms are on par. HLF is a native permissioned platform, while NEO is a public platform that can be run in permissioned mode as a private network; in both cases, networks will always have a limited number of validation nodes. Using NEO, we can create more decentralized networks than with HLF, which has a practical limit of 16 validation nodes before BFT performance starts degrading (this issue might be solved or mitigated in future versions of the software). That said, adding more nodes does *not* affect scalability: it only improves continuity and resistance against attacks. The 8-node configuration used in the test is a good compromise between these qualities and data storage efficiency.

To conclude, given these KPI results, we can set the *benchmark for ICPS-compatible DLT performance* – or, from the opposite point of view, the boundaries of the DLT comfort zone – as follows:

- **0.1 <= TrxAL <= 1.0**
- **0 <= TrxMST <= 160**
- **No restrictions on data storage**

This benchmark actually corresponds to the performance envelope of HLF, which in our study emerged as the only viable DLT platform (at the time of writing) for use in ICPS solutions.

VI. DLT AND ICPS: AN EDGE COMPUTING APPROACH

In this last section we introduce some guidelines for the use of DLT as a key enabling technology of decentralization in ICPS. These guidelines represent a novel use of the blockchain in the industrial space, as they are focused on large-scale, reliable, plant wide state synchronization rather than on manufacturing chain traceability and optimization, which is a common use of DLT in supply chains [13].

As mentioned before, one of the reference scenarios is the coordination/orchestration of local real-time processes. This approach is currently being researched by the FAR-EDGE project [15], which is exploring the *edge computing* paradigm applied to factory automation. In particular, the FAR-EDGE Reference Architecture (RA), depicted in Fig. 3 below, is based on the concept of a logical layer of Edge Gateways (EG) that are in charge of Edge Processes – i.e., time-critical and/or data-intensive computations that are integrated with physical process on the Field [11].

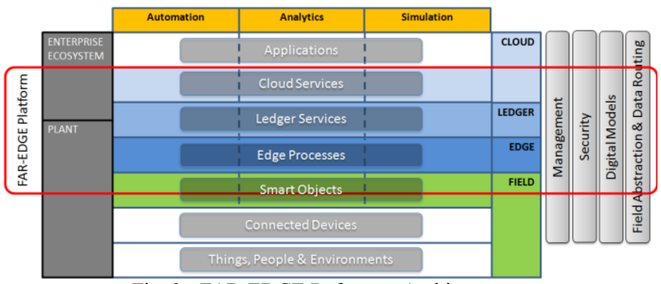

Fig. 3 - FAR-EDGE Reference Architecture

In FAR-EDGE, EGs are computing devices that are deployed in close proximity to Field objects, so that any network bottlenecks are reduced to the bare minimum. This strategy, however, is also introducing more friction because *local business logic* – which is distributed across multiple EGs – must obey to the *global business logic* of the factory/enterprise – which is typically centralized in ERP/MES systems. Given that one of the objectives of the project is to virtualize and flatten the hierarchy of the legacy automation pyramid, FAR-EDGE developed a radically innovative approach: moving (some of) the global business logic assets to a new logical layer called Ledger, where they are implemented as smart contracts and executed on a DLT platform. These smart contracts are then exposed as Ledger Services to EG consumers. By means of Ledger Services, distributed computing processes running on EG devices can synchronize their state, publish locally-scoped information that needs to be aggregated on a global scope, etc.

This design is a synergy between edge and serverless computing, the latter obtained by means of DLT. It achieves the objectives of decentralization, in particular those related to flexibility and resilience of production lines, without introducing performance bottlenecks, as DLT is only used within the limits of its comfort zone. Moreover, moving factory-level business logic to the Ledger layer enables even more advanced scenarios where decentralization is pushed to the extreme: semi-autonomous Smart Objects on the Field (tools, machinery, workstations with embedded intelligence) that are coordinated directly by the Ledger, making the shopfloor a modular and reconfigurable facility.

While this use of DLT is far from what is commonly perceived as its main role of “enabler of trust”, it is worth noting that it allows manufacturing enterprises to deploy their *virtual private cloud* on factory premises. This means they can benefit, to some extent, from a cloud computing architecture for running critical processes, but at the same time they don’t have to invest into a cloud computing infrastructure or outsource to an external provider. DLT will enable more agile value chains, faster product innovations, closer customer relationships, and quicker integration with the IoT and cloud technology.

Acknowledgment

Most of the work reported in this paper was done in the scope of the FAR-EDGE research and innovation project, funded by the European Commission’s Horizon 2020 program under grant agreement n. 723094.

References

[1] A. Antonopoulos, “The Internet of Money”, Merkle Bloom LLC, 2016.

[2] S. Nakamoto (pseudonym), “Bitcoin: a peer-to-peer electronic cash system”, unpublished, 2008. Retrived from: https://bitcoin.org/bitcoin.pdf

[3] T. Sauter, S. Soucek, W. Kastner, D. Dietrich, “The evolution of factory and building automation”, IEEE Industrial Electronics Magazine, September 2011.

[4] L. Lamport, R. Shostak, M. Pease, “The Byzantine Generals problem”, ACM Transactions on Programming Languages and Systems, volume 4 n. 3, p. 382-401, 1982.

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, “An overview of Blockchain technology: architecture, consensus, and future trends”, proceedings of IEEE 6th International Congress on Big Data, 2017.

[6] S. Popov, “The Tangle”, unpublished, 2017. Retrieved from: https://iota.org/IOTA_Whitepaper.pdf

[7] NEO Team, "NEO white paper", unpublished, 2017. Retrieved from: https://github.com/neo-project/docs/blob/master/en-us/index.md

[8] M. Castro, B. Liskov, "Practical Byzantine fault tolerance and proactive recovery", ACM Transactions on Computer Systems, Vol. 20, No. 4, 2002.

[9] M. Kapristos, Y. Wang, V. Quema, A. Clement, L. Alvisi, M. Dahlin, “All about Eve: Execute-Verify replication for multi-core servers”, proceedings of 10th USENIX Symposium on Operating Systems Design and Implementation, 2012.

[10] T. Dinh, J. Wang, G. Chen, R. Liu, C. Ooi, K. L. Tan, “BLOCKBENCH: a framework for analyzing private Blockchains”, unpublished, 2017. Retrieved from: https://arxiv.org/pdf/1703.04057.pdf

[11] M. Isaja, “FAR-EDGE achitecture and components specifications”, unpublished, 2017. Retrieved from: https://goo.gl/5SnG36

[12] H. Halpin, M. Piekarska, "Introduction to Security and Privacy on the Blockchain," IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Paris, 2017, pp. 1-3.

[13] T. Ahram, A. Sargolzaei, S. Sargolzaei, J. Daniels, B. Amaba, "Blockchain technology innovations," IEEE Technology & Engineering Management Conference (TEMSCON), Santa Clara, CA, 2017, pp. 137-141.

[14] Proof-of-Stake, wiki article: https://en.bitcoin.it/wiki/Proof_of_Stake

[15] FAR-EDGE project web site: http://www.faredge.eu/

[16] AWS Lambda web site: https://aws.amazon.com/lambda/

[17] NEO project web site: https://neo.org/

[18] Hyperledger Fabric project web site: https://www.hyperledger.org/projects/fabric