Enabling Auditable Trust in Autonomous Networks with Ethereum and IPFS

Jaime Fúster de la Fuente* Álvaro Pendás Recondo* Leon Wong Paul Harvey
Rakuten Mobile *University of Oviedo* *Rakuten Mobile* *University of Glasgow*
Tokyo, Japan Gijón, Spain Tokyo, Japan Glasgow, UK
jaimefusterf@gmail.com pendasalvaro@uniovi.es leon.wong@rakuten.com paul.harvey@glasgow.ac.uk

***Abstract*—Operation and management of telecommunication networks are increasingly difficult with the demands and behaviors of users exceeding the capacity of network engineers to keep pace. This has led to increased automation of the network, enabled by various forms of intelligent software. One such proposal from the ITU-T Focus Group on Autonomous Networks (standardization group) is an architecture to achieve self-driven automation (i.e. *autonomy*) of network operation, whereby technology from different operators and third parties is *self-assembled* and deployed in production networks. This raises questions and challenges regarding transparency, auditability, and trust while maintaining interoperability.**

This work presents an initial study of a distributed and decentralized marketplace to bring transparent and auditable trust to the proposed architecture without sacrificing interoperable functionality. We demonstrated this by our proof of concept implementation of both the proposed architecture and marketplace based on the combination of Ethereum and IPFS.

***Index Terms*—autonomous networks, blockchain, decentralized marketplace, DLT, Ethereum, IPFS, smart contracts, 5G, 6G**

I. Introduction

The exponential growth of networked devices, coupled with increasing user demands for connectivity and reliability place substantial pressures on the operation and maintenance of mobile phone networks. In parallel, the ever-changing landscape of telecommunication standards and emerging non-human communication paradigms, such as Internet of Things (IoT), wearable devices or digital healthcare, require the telecommunication industry as a whole to adapt quickly [**?**], [**?**]. To address these challenges, operators are exploring the application of Artificial Intelligence (AI) and Machine Learning (ML) technologies to automate certain management and operations tasks which are traditionally performed by engineers [**?**], [**?**]. Inspired by work in computer systems [**?**], the ultimate goal of this journey is for networks to become *Autonomous Networks* (ANs).

Recently, the ITU-T Focus Group on Autonomous Networks[1] (FGAN) has proposed a new architectural approach to ANs focused on continuous *automatic* creation, validation, and application (CVA) of control logic (known as controllers) for the live operation of a dynamic, ever-changing operational environment, and fully embracing the principles of self-adaptation in multiple forms [**?**]. To remain consistent, the FGAN approach refers to CVA as *evolution* of a controller.

The FGAN approach relies on a large number of interoperable, modular software components (e.g. simulation tools, algorithms, actuation software, data aggregation logic) and associated data (e.g. production telemetry, error reporting, testing logs, performance metrics), see Section III. Additionally, as modern telephone networks are a combination of technologies from operators and third parties which are distributed over large geographical areas, the FGAN architecture is distributed and decentralized by design [**?**].

The combination of machine-generated software controllers coupled with technology contributions from different industry (and other) partners inevitably leads to questions around security, auditability, trust, and traceability. This is especially true when considering that such software will be deployed to production systems that often are described as national critical infrastructure [**?**]

In this paper, we present the design and Proof of Concept (PoC) implementation for both a) the FGAN architecture and b) a marketplace that enables the decentralized and uninterrupted evolution of a controller, while ensuring auditability of the process and technologies involved. Specifically, our marketplace allows immutable registration of software components and data required by the FGAN architecture, as well as every stage of the evolution process, including actors involved, experimental/test results, and steps taken in creating controllers. Our solution combines a private Ethereum blockchain [**?**] with the distributed and decentralized InterPlanetary Filesystem [**?**] (IPFS). Our PoC implementations of the marketplace and FGAN architecture are publicly available[2].

The remainder of the paper is organized as follows. Section II discusses related work; Section III presents our version of the FGAN architecture and the Marketplace; Section IV discussed the implementation details. Finally, Section V draws the main conclusions from this research.

II. Background

We now present a summary of relevant literature that can be used to support the audibility of the evolution process in the FGAN architecture.

* Equal contribution from these authors

[1]https://www.itu.int/en/ITU-T/focusgroups/an/Pages/default.aspx (accessed Jan. 31, 2023)

[2]https://github.com/FGAN-Digital-Twins/BC-AN (accessed Jan. 31, 2023)

A. Blockchain-Based Trust and Auditability

Traceability and information management solutions using blockchain have been widely reported in recent years. Some key characteristics of Distributed Ledger Technologies (DLT), such as blockchain, include immutability, decentralization, traceability, security and privacy. Additionally, the increased development and support of blockchain technology presents a good solution for the decentralization of complex systems, such as the FGAN architecture.

Auditability in blockchain has been applied to healthcare, agriculture, automobiles and other fields [**?**], [**?**], [**?**], [**?**]. For instance, D. Miehle *et al.* [**?**] explored the use of traceability in the automotive industry to quickly detect faulty parts during after-sales service, implementing a decentralized supply chain using blockchain technology and creating an application using Hyperledger Fabric to trace the movement of parts through the supply chain. In another example, Salah *et al.* [**?**] proposed using blockchain and smart contracts with IPFS for traceability management in the soybean supply chain to ensure agricultural product safety.

The use of blockchain in telecommunications has also been proposed in several works within the literature and is overall regarded as an efficient solution for future decentralization of telecommunications systems beyond 5G [**?**]. Several blockchain implementations have been proposed to add trust and auditability to different solutions in 5G and 6G. For example, H. Xu *et al.* [**?**], discussed the potential use of blockchain in resource management and sharing for 6G networking, including applications such as the IoT, device-to-device communication, network slicing, and inter-domain blockchain ecosystems. Following this idea, L. Giupponi *et al.* [**?**] further delved into the concept by proposing blockchain-enabled Open-RAN (O-RAN) for the secure, traceable and trustable sharing of RAN resources. On a related note, H. Xu *et al.* [**?**] proposed a blockchain-enabled O-RAN architecture to provide decentralized identity management and privacy preserving P2P communication for 5G and beyond networks.

B. Decentralized Data Storage

Despite providing a means of ensuring decentralization, traceability and trust, blockchain implementations have well documented limitations [**?**], [**?**]. One of the primary challenges affecting them has to do with blockchain's data storage limitations, as, in order to ensure security, blockchain networks are designed such that every node maintains a copy of the ledger. While this approach does enhance security, it also leads to unnecessary data redundancy, as the same information is repeatedly stored across multiple nodes. This poses a challenge in terms of latency and scalability.

Many proposed solutions to the problem involve using off-chain storage, often in conjunction with the IPFS protocol [**?**], [**?**]. IPFS is a decentralized, peer-to-peer system for storing and sharing data, which allows users to access files from any node in a network without compromising the integrity or authenticity of the data [**?**]. Every file in IPFS is linked to a unique cryptographic hash or Content Identifier (CID), with any modification to a file resulting in the generation of a new CID. This content addressing feature allows for the integrity of blockchain artifacts to be maintained, and the use of CIDs registered on the blockchain provides auditability to file storage.

The combination of IPFS and blockchain technology has been explored in various contexts within the literature. For example, Q. Zheng *et al.* [**?**] proposed an IPFS-based data storage model for blockchain in which miners deposit the transaction data into the IPFS network and include the resulting IPFS hash of the transaction in the block, thereby reducing the amount of data stored directly on the blockchain. The proposed model was found to be highly effective in terms of reducing the amount of data stored and showed strong security and quick synchronization speed for new nodes. In [**?**], V. P. Rangantha *et al.* presented a decentralized *e-commerce* marketplace application using IPFS and smart contracts on the Ethereum blockchain. Salah *et al.* also proposed the combination of Ethereum smart contracts and IPFS for traceability in agricultural supply chains in [**?**].

In the field of telecommunications, W. Li *et al.* [**?**] presented BCTrustFrame, a framework solution for decentralized trust management in 6G based on blockchain smart contracts and IPFS. Y. Pang *et al.* [**?**] presented a blockchain and IPFS-based traceability system for telecom big data transactions which aimed to protect users' privacy, provide tamper-resistance, and facilitate rapid and accurate tracing of transaction information. While the system was implemented on both Hyperledger Fabric and Ethereum, it was found that the Ethereum implementation using the Proof of Authority (POA) consensus algorithm performed better in terms of scalability.

The integration of these technologies provides clear advantages in addressing the auditability of the CVA process in the FGAN architecture. A decentralized, transparent, and auditable marketplace based on blockchain and IPFS can facilitate interoperability in ANs and enable the decentralized and uninterrupted CVA of AN controllers while ensuring traceability and auditability of the process and technologies involved.

III. System Architecture

We now present our implementation design of the FGAN architecture which was used to test our main contribution: *the Marketplace*. The Marketplace architecture and its integration into the PoC are also described.

A. PoC FGAN Architecture

Our implementation for the evolution of controllers is based on the FGAN architecture [**?**] and is shown in Fig. 1. Blue nodes represent *actors* and green nodes represent *elements*.

Actors are controllers responsible for the evolution process of other controllers[3]. The element category identifies artifact's information which is registered by actors in the Marketplace

[3]While our implementation supports the self-reflective application of evolution to actors - so-called *self-evolution* - such discussion is outwith the scope of this work.

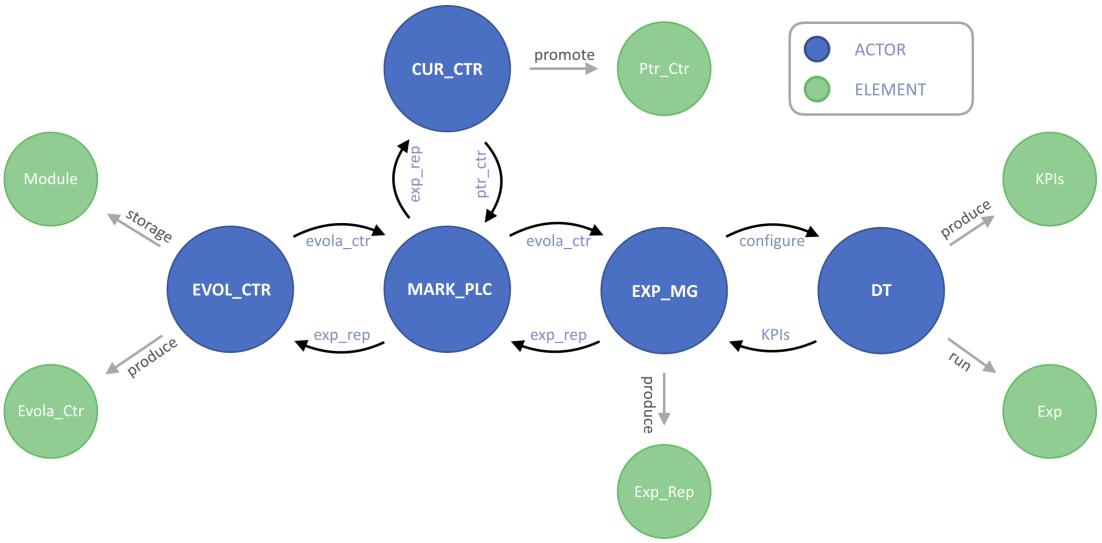

Fig. 1. System Architecture.

TABLE I
DESCRIPTION OF SYSTEM COMPONENTS

Component	Description
Evolution Controller (Evol_Ctr)	Controller responsible for evolution process
Marketplace (Mark_Plc)	Decentralized and auditable knowledge base
Experimentation Manager (Exp_Mg)	Controller responsible for configuring the Digital Twin and collecting experiment results
Digital Twin (DT)	Tool for the validation of a controller
Curation Controller (Cur_Ctr)	Controller responsible to decide which Evola_Ctr are deployable
Module	Composable logic "building block" from which controllers are formed
Evolvable Controller (Evola_Ctr)	Controller which may be evolved
Experiment (Exp)	Experiment/s designed for testing a type of controller
Key Performance Indicators (KPIs)	Figure/s of merit of controller performance
Experimentation Report (Exp_Rep)	Description of experimental results
Protected Controller (Ptr_Ctr)	Controller considered eligible for deployment

(Section III-B). This includes the controller designs and the software modules they are composed of. Experimental tests for controllers and the generated results are also stored in the Marketplace.

We now describe the components of our implementation, as shown in Table I and the Fig. 1 flowgraph. For further details please see the FGAN architecture [**?**]. Controllers which can be evolved are know as *evolvable*. The Evolution Controller (Evol_Ctr) stores the list of available modules and uses them to compose an Evolvable Controller (Evola_Ctr) that is uploaded to the Marketplace (Mark_Plc). Although the Marketplace is represented as a single node, it is a distributed network composed of every actor. The Marketplace will share the Evolvable Controller with the Experimentation Manager (Exp_Mg). This actor is responsible for configuring the Digital Twin (DT). That means setting the correct parameters to reproduce the desired experimental environment and sending a list of experiments to be run. The implementation of the Digital Twin is out of the scope of this work, although we introduce it for consistency with the FGAN architecture. The Digital Twin runs the Experiments (Exp) and collects Key Performance Indicators (KPIs) for the controller under test. Results are sent back to the Experimentation Manager which composes an Experimentation Report (Exp_Rep) including all the executed experiments with their results. The Experimentation Report is then shared in the Marketplace and sent to the Curation Controller (Cur_Ctr), which decides if the Evolvable Controller is promoted to a Protected Controller (Ptr_Ctr). This information is also shared in the Marketplace. The category of Protected Controller denotes a selected population that will continue through the evolution process and is considered eligible for deployment. The Experimentation Report is also forwarded to the Evolution Controller as feedback for the next evolution cycle. For simplicity, in this PoC the Marketplace directly forwards the received messages that contain an update to the other nodes. Future work would see this changed to a subscription-based approach.

B. The Marketplace

The Marketplace is a decentralized and auditable data knowledge base that enables the traceable, transparent and certifiable evolution of controllers. Rather than running on a single node of the network, the Marketplace is a distributed network of several interconnected nodes. To ensure a traceable and secure evolution process the Marketplace is built using IPFS and Ethereum. Ethereum was chosen due to its versatility in accommodating the needs of the FGAN architecture, however, this work could be extended to other platforms, such as Hyper Ledge fabric.

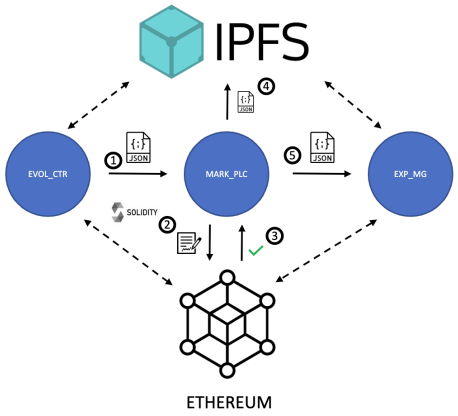

Fig. 2. Example of Marketplace operation.

To ensure the privacy, trustworthiness, and security of the software and data used in the evolution process, a private permissioned blockchain was chosen over a public permissionless blockchain. Unlike a public blockchain, a permissioned blockchain requires nodes to obtain permission before joining and operating on the network [**?**]. This means that access to the network is restricted to certain nodes, and the rights of these nodes may also be restricted. The use of a private blockchain allows for the protection of sensitive information and the prevention of tampering by actors, ensuring the reliability and trustworthiness of the data stored on the blockchain. This also fits well with the idea of trusted partners, as is currently the case in the telco industry.

The Marketplace is distributed and accessible from instances of the Evolution Controller, Experimentation Manager, Digital Twin and Curation Controller, connecting every actor to the traceable ledger of evolution artifacts created throughout the evolution process. These are stored in IPFS and accessible through the private Ethereum blockchain. An example of the operation of the Marketplace is illustrated in Figure 2. During the evolution process, whenever a node of the network creates or updates an artifact, the transaction and artifact CID are registered in Ethereum by means of a *smart contract*. After being confirmed, the file is stored in IPFS and distributed to the corresponding nodes.

Smart contracts are programmable applications that are stored and run on the blockchain network and automatically enforce the terms of an agreement when certain conditions are met, without the need for intermediaries to facilitate the process [**?**]. The Marketplace smart contract is tasked with recording and storing artifacts created during the evolution and experimentation process onto the blockchain, which ensures that they are traceable, secure, and easily accessible. Further details are provided in Section IV-C.

IV. System Implementation

This section presents the details for our implementation of the FGAN architecture and Marketplace. The goal of the implementation is to understand how to achieve a trustworthy and auditable store for the FGAN architecture and data. Implementation of the tools and algorithms to drive the architecture on real-world use cases is beyond the scope of this work.

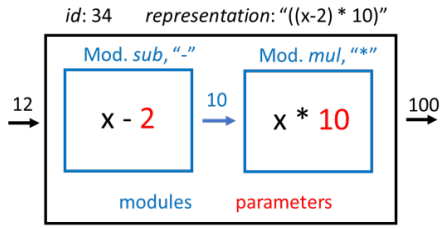

Fig. 3. Representation of a *controller* class object.

A. Modules and Controllers

In our implementation, modules and controllers are Python classes that represent mathematical operations. The *module* class has four attributes: a (unique) module id; a mathematical operator implemented using a Python function; a parameter that tunes that function; and a string representation of the module's operator. The attributes of the *controller* class are: a (unique) controller id; a Python list of modules that compose that controller; and a Python list of the parameters tuning those modules. Fig. 3 shows a *controller* class object. The represented controller is composed of the modules *sub* and *mul* (subtraction and multiplication) with arbitrary parameters 2 and 10, respectively. Although the id of modules is a string, the id of a controller is an integer, in this case, 34. The representation attribute provides insight into how the controller operates. The *controller* class has a method *execute* that takes one input and returns the output that results from its operation. Fig. 3 shows the example with the input 12, which returns 100. This level of complexity is sufficient to create a diverse set of modules and controllers generated by different stakeholders.

B. Controller Evolution and Experimentation

The evolution process is emulated by the random combination of modules and parameters to compose new controllers from a limited set of parameterized modules. Given the goal of this work, this simple approach is sufficient. Experimental validation of controllers is done against two scenarios:

- **Average**: Find the controller that yields the highest value for a fixed input.
- **Value**: Find the controller that yields the closest value to a desired output.

Each generated controller is tested against both scenarios, producing different controllers as being most appropriate for each scenario. Therefore, a controller which becomes *protected* for one use case scenario does not necessarily do so for the other.

Information about controllers and experimentation is shared between nodes using the JSON format [**?**]. JSON was chosen as it is self-describing and readable by humans. The *controller* class has a method that returns a Python dictionary with its current attributes, which can be easily converted to

```
{type: "controller",
 id: 34,
 modules :  ["sub", "mul"],
 parameters : [2, 10],
 representation: "((x-2)*10)"}
```

Listing 1: JSON representation of a *controller* class object.

JSON format. Listing 1 shows the JSON representation of the *controller* object shown in Fig. 3. The configuration and results of experiments are also represented as JSON files.

Finally, in this PoC we assume that all nodes have the list of available modules and their source code, and thus, the ability to instantiate a *controller* object from its JSON representation. In future work, we will support the runtime addition of new modules to the system. The Marketplace already contains the functionality necessary to facilitate the consistent dissemination of these modules, although the Python implementation does not.

C. The Marketplace Smart Contract

The Marketplace smart contract is tasked with registering artifacts generated during the evolution and experimentation process. The private IPFS network acts as the storage solution for artifacts, providing scalability while maintaining availability. The Ethereum blockchain tracks all transactions and records metadata that is included in the JSON representation of controllers (illustrated in Listing 1) and the File structure of the *Marketplace.sol* smart contract.

To guarantee full transparency and accountability, the blockchain records three types of file registration transactions - controller, protected controller and experimentation report. The management of file registration is carried out by the *Marketplace.sol* smart contract, which is written in the Solidity programming language and designed to capture meaningful information for ease of file identification and register browsing.

Figure 4 presents a class diagram of the smart contract, showcasing its data structures and functions. Data artifacts generated during the evolution and experimentation process are registered by the Marketplace smart contract through the File structure. This data structure captures several details about the artifacts, as illustrated in Figure 4.

Firstly, a mapping is assigned to the File structure and the total number of files uploaded to the Marketplace, which is tracked using the *fileNumber* attribute. Secondly, the IPFS CID of the artifact is saved as a string. It is important to note that while the controller id (see Section IV-A) represents a specific instance of a controller, the CID is a unique cryptographic hash that identifies a file and remains unchanged as long as the contents of the JSON stay the same (e.g. when the evolution is run multiple times). Finally, the name of the artifact and its object type are also registered in the File structure. The *objectType* element can be of type controller, protected controller, or experimental report.

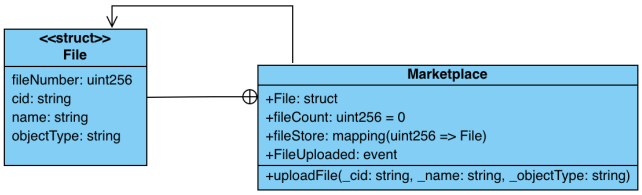

Fig. 4. Class diagram of the Marketplace smart contract.

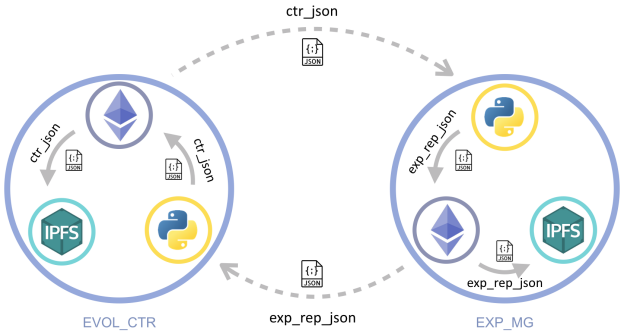

Fig. 5. Example of the functional level architecture of the PoC.

D. Network Implementation

Figure 5 shows the functional level architecture of the PoC, where two nodes (Evol_Ctr and Exp_Mg) are shown to illustrate the different technologies implemented. Each one of the actors presented in Figure 1 is composed of a subset of services that represent a specific function within the PoC. These functions range from creating and updating modules and controllers to managing the registration and storage of artifacts in Ethereum and IPFS. Different technologies are used depending on the type of functionality implemented by each service, and all independent services have been implemented through Docker containers, orchestrated and configured via a Docker Compose (*docker-compose.yml*) file. Each container/service of the system has an IP address and a specific set of ports exposed for communication with other nodes within the network.

E. Controller Deployment

Our implementation supports the automatic deployment of a controller as a Representational State Transfer (REST) service by the xOpera[4] orchestrator. Given the CID of a controller as input, the orchestrator performs the following tasks (Fig. 6):

- Fetch the JSON representation of the selected controller from the Marketplace.
- Create an instance of the class *controller* based on the JSON representation.
- Deploy a REST service in a Docker container.

For the development of the REST service, we used the Python framework Flask combined with Gunicorn for concurrency. Once the service is deployed, it will reply to HTTP GET requests. The produced HTTP response is a JSON file

[4]https://github.com/xlab-si/xopera-opera (accessed Jan. 31, 2023)

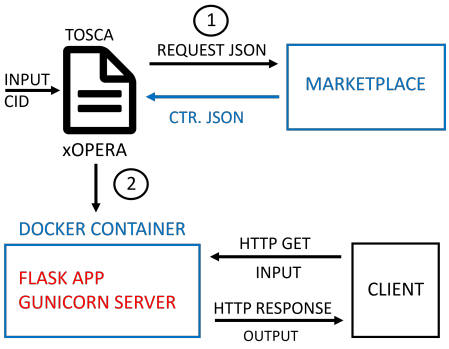

Fig. 6. Controller deployment representation.

containing the information about the deployed controller and the output of the requested input.

V. Conclusion

In this paper, we have presented an initial feasibility study on the use of auditable trust in the ITU-T proposed AN architecture. We presented the design and PoC implementation of a decentralized marketplace that enables software from different stakeholders to be self-assembled and deployed via the ITU-T network architecture, which was also implemented. Both implementations are publicly available. Leveraging a private Ethereum blockchain for immutability and auditability with IPFS for distributed storage, we demonstrate how trust can be achieved in this context using working examples of the various software components and data involved. Through this work, we address the limitations in the current proposed architecture in terms of auditability and transparency, removing a barrier to progress on the road to fully ANs. Future work will increase the complexity of the software components in the implementation and the use cases to which they are applied to explore scalability and performance of the approach.

Acknowledgments

This work was supported by the EU-Japan Centre for Industrial Cooperation under the Vulcanus in Japan 2021/2022 program. The work of A. Pendás was partly supported by the Principality of Asturias under “Severo Ochoa” Program Grant no. BP21-116.

References