Towards Empowering Ubiquitous Computing as a Service with Blockchain Capabilities for Sustainable Manufacturing, Agriculture, Cities, and Buildings

1st Mirco Soderi
Data Science Institute
University of Galway
Galway, Ireland
mirco.soderi@universityofgalway.ie

2nd John Gerard Breslin
Data Science Institute
University of Galway
Galway, Ireland
john.breslin@universityofgalway.ie

Abstract—**Pervasive computing, or ubiquitous computing, integrates connectivity functionalities into objects so that they can interact with one another and perform automated tasks with minimal human effort. Although concerns have been raised regarding material and energy consumption, as well as radiations, ubiquitous computing can positively impact environmental sustainability through process optimisations in manufacturing, agriculture, and city management. This work extends our software framework for Ubiquitous Computing as a Service. The framework uses containerisation, low-code platforms, modularity, parallel computing, MQTT, and API to support the creation, configuration, operation, and modification of remote software. In this work, some degree of blockchain integration is added, which makes it possible to ingest Blockchain transactions in real-time, and to submit new transactions. The benefits of integrating pervasive computing and blockchain technologies include context-aware authentication, traceability, auditing, integrity, and security. All software artefacts produced in this work are available on GitHub, including a Postman collection of API requests for demo purposes.**

Index Terms—**pervasive computing, ubiquitous computing, artificial intelligence, optimisation, sustainability, intelligent manufacturing, reconfigurable manufacturing, smart farming, smart city, smart building, software framework, distributed system, containerisation, low-code, modularity, parallel computing, MQTT, API, blockchain**

I. Introduction

Pervasive computing [1] [2] integrates connectivity functions into objects so that they can interact with each other and perform automated tasks with minimal human effort.

Concerns have been raised about the negative impact of ubiquitous computing on environmental sustainability due to power consumption, electronic waste, radiation, and social sustainability, consumer privacy, and freedom of choice [3]. Conflicts between users and non-users of the technology might also arise [4]. Smart cities are not exempt from risks and challenges, although technology has the potential to advance and accelerate environmental sustainability in urban areas [5]. An effective governance of energy and waste is crucial [6].

In recent years, efforts have been made to address the environmental and ethical implications of pervasive computing [7] and to use pervasive computing to create smarter and greener environments [8]. In some workshops aimed at exploring how to drive action and change in relation to environmental sustainability, ubiquitous computing was found to play a positive role [9]. Furthermore, pervasive computing can contribute to the challenge of tackling the acceleration of the climate crisis through new devices/services and tools for algorithmic data analysis and control [10]. This finds confirmation in some applications for sustainable forest management [11] and ambient intelligence [12].

Blockchain technology has attracted massive attention and has triggered multiple projects in different industries. Although the financial industry is its main field of application, it has also been used in food safety, verification of asset ownership, and enforcement of agreements [13]. Simply put, blockchains are distributed systems in which immutable encrypted data and automatically actionable agreements known as smart contracts are replicated in thousands of nodes, with data integrity guaranteed by links drawn between consecutive pieces of data and by consensus mechanisms [14]. It is a foundational rather than a disruptive technology, with unprecedented levels of technological, regulatory, and social complexity, whose adoption requires broad coordination and years of time [15].

The synergy between ubiquitous computing and blockchain technologies has not been investigated until recently [16] [17], when blockchain technologies and ubiquitous computing have been used to (i) streamline secure data processing and analytics in healthcare [18], (ii) help small and medium enterprises adapt to the growing demand for customised products [19], (iii) improve security by storing access control lists in public blockchains [20], and (iv) streamline access to wireless network while preventing unauthorised access [21].

This publication has emanated from research supported in part by the European Digital Innovation Hub Data2Sustain, co-funded by Ireland's National Recovery and Resilience Plan (the EU's Recovery and Resilience Facility), the Digital Europe Programme, and the Government of Ireland, and by a grant from Taighde Éireann - Research Ireland under Grant Number 12/RC/2289_P2 (Insight). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. A heartfelt thank you goes to the WiSAR Lab for logistic support, encouragement, and the lovely conversations.

II. Background

This work builds on our software framework for pervasive computing as a service. The framework was developed to be applied to reconfigurable manufacturing, to support scenarios in which geodistributed and semiautomated real-time reconfiguration was required. However, the framework is general enough to find application in a variety of fields, including environmental monitoring and preservation.

In the framework, the Network Factory [22] is a containerised Node-RED application available on the Docker Hub[1], which exposes APIs to create and organise containerised software locally or remotely.

Among the software components that can be created, the Service Nodes [23] deserve a special mention. They are configurable Node-RED applications. At creation time, they all look the same. However, they expose APIs that are used for configuring a variety of aspects, including the function that the node must execute, as well as the data source(s), and the destination(s) of the output(s). The implementation of the function is loaded into the Service Node from the Transformation Library, which is a containerised Node-RED application that contains a collection of functions that are loaded into the Service Nodes on request.

Such functions are implemented as Node-RED subflows; they can be as simple as calculating a configured mathematical expression, or as complex as (i) interfacing with an Artificial Intelligence Server (AIS) [24], (ii) displaying long sequences of data points on a line graph served through a Web page in real time [25], (iii) turning the device into a Bluetooth Low Energy (BLE) server [26], or (iv) running a parametric Postman collection of API requests [22].

Artificial intelligence servers are another notable containerised application that we have developed and that a Network Factory can instantiate. They are modular and extensible applications based on Scala and Spark. They expose APIs so that Service Nodes can interface with them, configure the job to be executed, then start and stop the execution. This can be done by multiple service nodes at the same time, independently of each other.

There is a unique feature of Node-RED that makes it especially suitable for implementing resilient applications that are capable to automatically or semi-automatically evolve in response to either predictable or unpredictable events. In Node-RED, it is possible to implement APIs that make arbitrary changes to the implementation of the application itself with immediate effect. We provide such APIs in specialised Service Nodes, named Crazy Nodes [27]. The name represents the fact that although it is undoubtedly a powerful feature, it must be used cautiously because of its security implications, as well as to avoid introducing errors that cause the application to fail or not behave as intended.

In terms of tools, technologies, protocols, it can be said that Docker, Node-RED, MQTT, HTTPS, Scala, and Spark are the foundations of the framework. GitHub, npm, websocket, sbt, and HTML also play a pivotal role. Kafka, Hadoop, Bluetooth Low Energy (BLE), the web3 npm module, and Postman have instead been used for the purposes of the applications that have been built, configured, and run, using the framework.

[1]https://hub.docker.com/r/msoderi/network-factory

III. Blockchain Integration

Three new functions (Node-RED subflows) have been introduced in the Transformation Library to support blockchain integration. Thanks to them, it is possible to (i) get the newly submitted pending transactions in real-time through Blockchain RPC made via websocket; (ii) display transactions in tabular format on the Web; (iii) submit transactions.

A. *The holeskyskt subflow*

The holeskyskt subflow (Fig. 1) is used to make RPC calls to a blockchain via websocket to subscribe/unsubscribe.

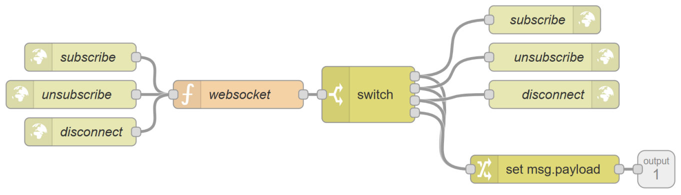

Fig. 1. The holeskyskt subflow

It exposes configuration APIs to set the URL of the websocket and to retrieve the URL that is currently configured; authorisation is enforced through a request to an external application, which is configured for each Service Node at the time of its creation. Configurations are stored in Node-RED memory and are globally available in the Service Node.

It exposes *subscribe* and *unsubscribe* APIs. The request payload is expected to be a valid JSON-RPC payload and is sent to the blockchain as is. A valid payload for the Ethereum Holesky blockchain, for example, must contain the following properties: (i) *id*, a numeric identifier, which is used to link responses to requests transmitted through the websocket; (ii) *jsonrpc*, the JSON-RPC version; (iii) *method*, either *eth_subscribe* or *eth_unsubscribe*; (iv) *params*, which specify what the subscription is for, or which subscription should be canceled. The methods and parameters may vary for different types of blockchain.

It also exposes a *disconnect* API, which can be called to terminate the websocket connection from the client side.

API requests go into input to the *websocket* node, where most of the logic takes place. Requests are stored in the memory of the node with the numeric identifier used as a key, so when responses come asynchronously from the blockchain via websocket, it is possible to use the identifier in the response to link it to the correct request. At the first API request, the websocket connection is initialised and stored for later use. Then, if the request is to subscribe or unsubscribe, the payload is sent as is to the blockchain via websocket. When a response comes still via websocket, a Node-RED message is created and sent to the *switch* node, and eventually to (i) the *http out* nodes, to provide a response to the API request, and (ii) the *output*

1, to notify transaction consumers. Otherwise, if the request is to disconnect, the websocket connection is closed, and when the *close* event is triggered (asynchronously), a Node-RED message is created and sent forward to provide a response to the API request and notify transaction consumers. When a transaction is received through the blockchain websocket as a result of an active subscription, a Node-RED message is created with the transaction as payload and sent forward to the *switch* node, and eventually to the *output 1* node, but not to the *http out* nodes, as there is no API request that requires a response in this scenario. Remarkably, websocket communication is entirely wrapped in the *websocket* node and completely transparent to any other Node-RED node.

B. The holeskytbl subflow

The holeskytbl subflow (Fig. 2) is responsible for displaying blockchain transactions in a table on a web page. The table is built incrementally; for each incoming message, a row is added to the table. Incoming messages are expected to be JSON objects that represent blockchain transactions, and they are expected to have the following properties: (i) From, (ii) To, (iii) Value, (iv) Gas, (v) Gas Price, (vi) Max Fee Per Gas, and (vii) Max Priority Fee Per Gas. Some formatting is performed on the *tabulator* node to produce Node-RED messages that have a payload that complies with the *tabulator* JavaScript framework for dynamic tables and data grids, which is what the *table* node is based on. The *table* node comes with the npm module *node-red-node-ui-table*, and displays incoming Node-RED messages in an HTML table. The *node-red-dashboard* npm module is also required for this subflow.

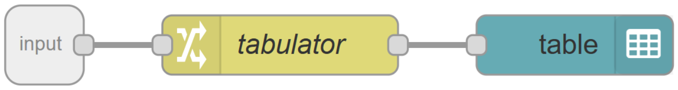

Fig. 2. The holeskytbl subflow

C. The holeskywrt subflow

The *holeskywrt* subflow submits new transactions to a blockchain. It exposes configuration APIs to set the HTTP URL of the blockchain and the private key of the sender's wallet, and to retrieve the configured values; authorisation is enforced as described in Section III-A. Most importantly, it exposes the *transaction* API, to be called in POST with a JSON payload that carries two properties: *to* and *value*.

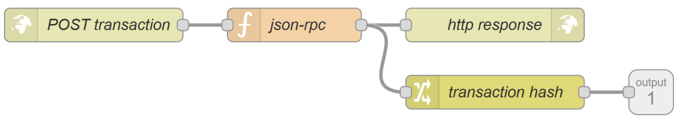

Fig. 3. The holeskywrt subflow

Most of the logic is wrapped in the *json-rpc* Node-RED function node. In it, the *web3* npm module is used to instantiate the sender's web3 Ethereum account from the sender's wallet private key, and then the sender's wallet address is retrieved from the account and used to fill the *from* property of the transaction. The *getGasPrice* and *estimateGas* methods of the Web3 Ethereum account instance are used to fill the *gasPrice* and *gas* properties of the new transaction, respectively. The *to* and *value* properties are filled with the values specified in the payload of the *transaction* API request. The transaction is then signed using the *signTransaction* method of the sender's web3 Ethereum account instance and sent using the Web3 Ethereum package. The *transactionHash* is then extracted from the send receipt and used as a payload in a Node-RED message that the *json-rpc* function node sends in output, and that goes to *output 1*, so that an MQTT client or other software component can be optionally connected to *output 1* and be notified every time a new transaction is sent. The same Node-RED message also reaches a *http out* node, so a response is returned to the request made to the *transaction* API.

IV. Evaluation

A Postman collection of API requests has been prepared to evaluate the proposed approach; it is available on GitHub[2]. The collection has a variable *DockerHost*, which defaults to *localhost*, which is okay if requests are made from the same host where a Docker Engine is running, and the Docker daemon is exposed on port 2375 over HTTP. For the purposes of the demo, the Google Cloud Blockchain RPC service can be used; the service provides the socket and HTTP URL for a variety of blockchains, including a Holesky blockchain (recommended). A wallet is necessary; free wallets can be obtained from a variety of providers, including MetaMask. The wallet must be non-emopty; 1 Holesky ETH can be obtained through the Google Cloud Ethereum Holesky Faucet. The socket and HTTP URLs of the blockchain, and the private key of the wallet, must be provided as payload in requests no. 10, 42.1, and 42.2 of the Postman collection.

The collection creates, configures, and runs three Service Nodes and an MQTT broker (Fig. 4). The Service Nodes are (i) HoleskysktSN, which runs the holeskyskt subflow; (ii) HoleskytblSN, which runs the holeskytbl subflow; and (iii) HoleskywrtSN, which runs the holeskywrt subflow. The former two are linked through the MQTT broker. The HoleskywrtSN also publishes the transaction hash of every new transaction that is sent to the blockchain to a dedicated topic on the same MQTT broker, though it is not listened to.

The numbers on the arrows indicate that in the hypothesis a new transaction is sent to the blockchain through the HoleskywrtSN, what happens next is that (i) as soon as the HoleskywrtSN gets a receipt from the blockchain, it extracts the transaction hash and publishes it to the *newtransactionhash* topic of the MQTT broker; (ii) a few instants later, the blockchain sends the new transaction via websocket to HoleskysktSN, which had previously subscribed; (iii) HoleskysktSN publishes the new transaction to the *pendingtransactions* topic of the

[2]https://github.com/mircosoderi/State-of-the-art-Artifacts-for-Big-Data-Engineering-and-Analytics-as-a-Service/blob/162cd5b53fe67f42beff2499a338277af2be4bce/BDEAaaS-Blockchain.postman_collection

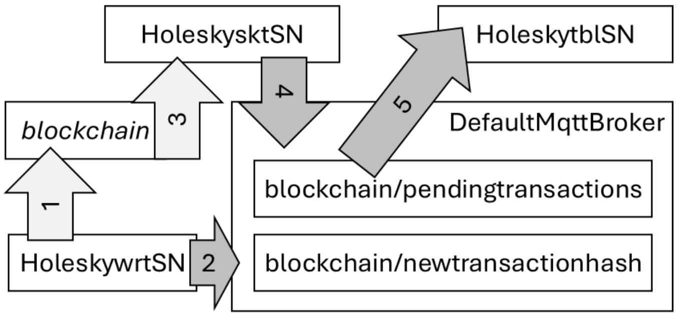

Fig. 4. The demo architecture

MQTT broker; (iv) HoleskytblSN is listening to the *pending-transactions* topic, so it receives the transaction, and adds it to the table on the Web page.

In terms of execution times of the requests in the collection, generally speaking, those requests that are aimed at configuring, starting, and using the Service Nodes execute in less than a second, whereas the requests that are aimed at creating the Service Nodes, the broker, the library of the subflows that are then loaded into the Service Nodes tend to be slower because they encompass the download of software artefacts from the GitHub repository and possibly the download of Docker images from the Docker Hub.

The main challenges that this work presented list as follows: (i) design of the interfaces; (ii) state tracking; (iii) familiarisation with the RPC method definitions of the specific blockchain, and with the JavaScript libraries used to implement blockchain clients.

V. Conclusion

In this work, it has been demonstrated how ubiquitous computing as a service and blockchain technologies can be used together to build maximally resilient, secure, explainable, and auditable applications. This finds application in a variety of fields, including manufacturing and environmental monitoring.

Future directions include (i) integration with a wider range of platforms for data storage, processing, visualisation; (ii) support of a wider range of communication protocols; (iii) performance optimisation of Node-RED applications; (iv) support of additional low-code platforms; (v) exploration of the benefits of semantic technologies; (vi) collaborations with industry partners for on-the-field evaluation.

References

[1] D. Saha, A. Mukherjee, S. Bandyopadhyay, D. Saha, A. Mukherjee, and S. Bandyopadhyay, “Pervasive computing,” *Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems*, pp. 1–37, 2003.

[2] M. Satyanarayanan, “Pervasive computing: Vision and challenges,” *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[3] A. Koehler and C. Som, “Effects of pervasive computing on sustainable,” *IEEE Technology and Society Magazine*, vol. 24, no. 1, pp. 15–23, 2005.

[4] L. M. Hilty, C. Som, and A. Köhler, “Assessing the human, social, and environmental risks of pervasive computing,” *Human and Ecological Risk Assessment*, vol. 10, no. 5, pp. 853–874, 2004.

[5] S. E. Bibri and S. E. Bibri, “Transitioning from smart cities to smarter cities: the future potential of ict of pervasive computing for advancing environmental sustainability,” *Smart Sustainable Cities of the Future: The Untapped Potential of Big Data Analytics and Context–Aware Computing for Advancing Sustainability*, pp. 535–599, 2018.

[6] A. Köhler and L. Erdmann, “Expected environmental impacts of pervasive computing,” *Human and Ecological Risk Assessment*, vol. 10, no. 5, pp. 831–852, 2004.

[7] P. McCullagh and S. Moore, “Addressing ethics and sustainability in ubiquitous computing and ambient intelligence,” in *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer, 2024, pp. 859–864.

[8] J.-H. Park, Y. Pan, H.-C. Chao, and N. Y. Yen, “Ubiquitous systems towards green, sustainable, and secured smart environment,” *The Scientific World Journal*, vol. 2015, p. 281921, 2015.

[9] M. Foth, E. Paulos, C. Satchell, and P. Dourish, “Pervasive computing and environmental sustainability: Two conference workshops,” *IEEE Pervasive Computing*, vol. 8, no. 1, pp. 78–81, 2008.

[10] M. L. Mauriello and M. Hazas, “Pervasive sustainability,” *IEEE Pervasive Computing*, vol. 23, no. 2, pp. 4–6, 2024.

[11] M. F. Khan, “Pervasive computing for sustainable forestry management: Developing an iot-connected timber harvesting and forest conservation platform.”

[12] A. Bimpas, J. Violos, A. Leivadeas, and I. Varlamis, “Leveraging pervasive computing for ambient intelligence: A survey on recent advancements, applications and open challenges,” *Computer Networks*, vol. 239, p. 110156, 2024.

[13] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, “Blockchain,” *Business & information systems engineering*, vol. 59, pp. 183–187, 2017.

[14] I. Bashir, *Mastering blockchain*. Packt Publishing Ltd, 2017.

[15] M. Iansiti, K. R. Lakhani *et al.*, “The truth about blockchain,” *Harvard business review*, vol. 95, no. 1, pp. 118–127, 2017.

[16] R. Singh, R. Kumar Tyagi, A. Kumar Mishra, and U. Choudhury, “Review on applicability and utilization of blockchain technology in ubiquitous computing,” *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 16, no. 7, pp. 51–65, 2023.

[17] R. Singh, A. Pandey, and L. K. Dixit, “Blockchain in ubiquitous computing,” in *Web 3.0*. CRC Press, pp. 190–207.

[18] S. Ayyasamy, “Metadata securing approach on ubiquitous computing devices with an optimized blockchain model,” *Journal of Ubiquitous Computing and Communication Technologies*, vol. 4, no. 2, pp. 57–67, 2022.

[19] A. Vatankhah Barenji, Z. Li, W. M. Wang, G. Q. Huang, and D. A. Guerra-Zubiaga, “Blockchain-based ubiquitous manufacturing: A secure and reliable cyber-physical system,” *International Journal of Production Research*, vol. 58, no. 7, pp. 2200–2221, 2020.

[20] S. Rani, D. Gupta, N. Herencsar, and G. Srivastava, “Blockchain-enabled cooperative computing strategy for resource sharing in fog networks,” *Internet of Things*, vol. 21, p. 100672, 2023.

[21] M. X. Ng, “A context-aware authentication method using blockchain for pervasive computing,” Ph.D. dissertation, UTAR, 2021.

[22] M. Soderi and J. G. Breslin, “A service for resilient manufacturing,” in *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2023, pp. 195–197.

[23] M. Soderi, V. Kamath, J. Morgan, and J. G. Breslin, “Ubiquitous system integration as a service in smart factories,” in *2021 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*. IEEE, 2021, pp. 261–267.

[24] M. Soderi, V. Kamath, and J. G. Breslin, “A demo of a software platform for ubiquitous big data engineering, visualization, and analytics, via reconfigurable micro-services, in smart factories,” in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2022, pp. 1–3.

[25] ——, “Toward an api-driven infinite cyber-screen for custom real-time display of big data streams,” in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2022, pp. 153–155.

[26] M. Soderi and J. Gerard, “Ble servers and ubiquitous analytics aas,” *AICS 2022 Digital Book of Abstracts*, 2022.

[27] M. Soderi and J. G. Breslin, “Crazy nodes: towards ultimate flexibility in ubiquitous big data stream engineering, visualisation, and analytics, in smart factories,” in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2022, pp. 235–240.