Blockchain-Enabled Traceability and Verification of Customized Product Processes in Industrial Multi-Agent Control Systems

Midhun Xavier*, Sandeep Patil*, Valeriy Vyatkin* ‡
* Department of Computer Science, Computer and Space Engineering, Luleå Tekniska Universitet, Sweden
‡Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland

Email: midhun.xavier@ltu.se, sandeep.patil@ltu.se, vyatkin@ieee.org

***Abstract*—This paper introduces a method to trace products and processes in industrial multi-agent control systems using blockchain technology. The proposed solution enables one to record and verify each process that occurs while generating a customised product using Ethereum-based smart contracts. Node-RED software agents are developed with the help of Semantic Web ontologies, and these software agents interact with IEC 61499 function blocks to execute processes. The agent associated with each mechatronic component and its controller can communicate with the blockchain to record various events that occur during each process and the latter smart contract helps to verify these process orders of the customised product.**

***Index Terms*—Blockchain, Ethereum, Node-RED, IEC 61499, Multi-Agent System, Semantic Web Ontologies**

I. Introduction

Multi-agent control opens up a wide range of possibilities for industrial automation systems due to its flexible and reconfigurable architecture [**?**] [**?**]. In this modern industrial world, the increased demand for customised products imposed a change in the existing production line with the introduction of software agents. These software agents have shown promising progress in industry 4.0 [**?**] by solving complex problems in industrial control systems.

Software agents in distributed control systems can be designed using various complementary approaches. In addition to the classic FIPA (Foundation for Intelligent Physical Agents) based multi-agent APIs (Application Programming Interface) [**?**], the IEC 61499-based multi-agent system [**?**] [**?**] could be one of the approaches that can be directly taken into account. The Semantic Web technologies [**?**] help extend it with semantic annotation of actions, providing more flexibility. The W3C Web Ontology Language (OWL) [**?**] is used to describe the capabilities and skills of the agent. The multi-agent systems enhanced with the Semantic Web ontology stack provide scalable and flexible solutions for reconfigurable production lines to produce customised products on customer demand.

Customised product generation using a multi-agent system introduces verification and validation challenges [**?**], [**?**]. Each customised product will have different process sequences or scenarios on the production line. Verifying and tracking the process sequence of each product is quite tricky.

Ivan Jovovi *et al.* [**?**] explores how Industry 4.0 challenges can be addressed using emerging technologies such as 5G and Blockchain to improve the transparency, integrity and security of data. The use of smart contracts [**?**] in industrial control systems could provide efficient, automated and tamper-proof processes execution, reducing the need for human intervention and increasing the reliability of the systems.

In this paper, we propose a smart contract-enabled solution to record process sequences on the blockchain, using the latter to verify whether the generated product follows the right process order sequence. The recorded information about processes on the blockchain cannot be altered, which helps to identify and remove defective products in real-time. It also provides a provision for the customer to verify the production process information by checking the block information.

The paper is organised as follows. Section II provides an overview of the advantages and challenges associated with the integration of blockchain technology into industrial control systems. Section IV explains in detail the methodology for the design and verification of multi-agent systems using the concept of smart contracts. Section V describes the illustrative example of a developed multi-agent system and presents the result of the work. Finally, Section VI concludes the article and outlines future goals.

II. Blockchain in Industrial Control Systems

Research in the field of applications of blockchain technology in industrial control systems (ICS) has been increasingly gaining attention in recent years [**?**], [**?**]. The blockchain can be used to record and verify the execution of processes in ICS, ensuring that the system is operating correctly and securely. This can be achieved by using smart contracts, which are self-executing contracts with the terms of the agreement directly written into code. Ethereum [**?**] is one of the most popular platforms for smart contract development and can be used to record and verify the execution of processes in ICS, ensuring that the system is operating correctly and securely.

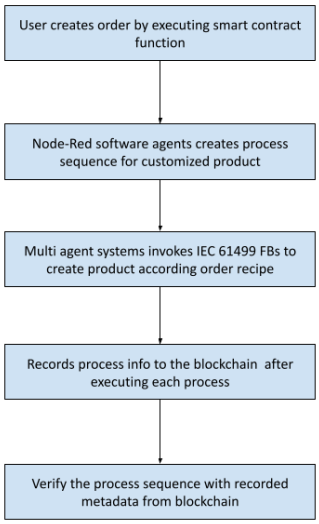

Fig. 1. Workflow diagram for verifying product process sequences.

One of the main applications of blockchain technology in ICS is data security [**?**]. A study [**?**] presents a novel architecture, ICS-BlockOpS, that aims to improve industrial control system (ICS) data security by using blockchain technology. ICS-BlockOpS is designed to enhance data security by using an integrity check mechanism and data redundancy by applying an efficient replication mechanism. The prototype implementation of ICS-BlockOpS uses the Ethereum blockchain in a local network as part of the tamper-proofing mechanism and was tested in an operational six-stage water treatment plant. Another study by Tsai *et al.* [**?**] proposes a blockchain-based network architecture to secure the integrity of data in Industrial Control Systems (ICS) in Industry 4.0. The proposed architecture is implemented on physical industrial equipment, and it verifies and records the transmission of industrial control signals based on authority through a specialised client-server network model. The architecture aims to improve the security of ICS against adversarial attacks.

Previous research has explored the blockchain in ICS for the security, transparency and integrity of data [**?**], [**?**], [**?**]. However, there has been limited research on its use in the verification and validation of execution processes in an industrial automation system. This paper proposes a novel method that uses smart contracts to record and verify process sequences in industrial control systems while creating customised products. However, more research is needed to evaluate the practicality and scalability of this proposed method in real-world industrial control systems.

III. Background

A. IEC 61499

B. Ethereum

C. IPFS

IV. Methodology

A. Workflow Description

The workflow diagram for the verification of the process sequences of the customised product is shown in Figure. 1. Initially, the user sends a request to create a customised product by executing a transaction on the blockchain network. Node-RED software agents accept the order request and create a process sequence for the product according to the user's need. The multi-agent system invokes IEC 61499 function blocks (FBs) and executes the task based on the order recipe. Process description, the status of the process, the timestamp, and other information (i.e., video of execution or picture of the product) are added as metadata while performing each step in an order recipe. Finally, the agent verifies the process sequences of the order recipe by comparing the metadata associated with the product from the blockchain.

B. Proposed Solution

The proposed architecture for blockchain-enabled verification of a multi-agent system is shown in Figure. 2. In this manufacturing marketplace, users can request to generate customised products, and the generation of the product is done with the help of Node Red-based software agents. These software agents help make the production line flexible, configurable, and scalable. Software agents interact directly with the IEC 61499 function blocks and execute the respective task at a low level. Software agents analyse the order request and create an order recipe consisting of process sequences. One of the Node-RED software agents called the processing agent will create proposals according to the order recipe and send them to all other executing agents. Executing agents check the proposal and accept or reject it on the basis of each agent's abilities and constraints. The processing agent receives the accepted proposals and determines the best proposals from the executing agents. In a distributed control system, each mechatronic component is associated with a different controller, and these controller function blocks are connected to their respective executing software agents.

A front-end user interface runs as a web server on Node-RED and it performs the computation at the edge. Whenever a user creates an order, then the Node-RED user agent calls a smart contract function to record the creation of a product with the help of the APIs. The smart contract runs on an Ethereum Virtual Machine (EVM) and stores the product ID and its order recipe. According to the order recipe, the execution agents interact with the IEC 61499 function blocks to perform a set of tasks or processes. IEC 61499 function blocks run on different mechatronic components and perform the processes according to the order recipe. After executing each process, the FBs interact with Node-RED agents via MQTT, and this

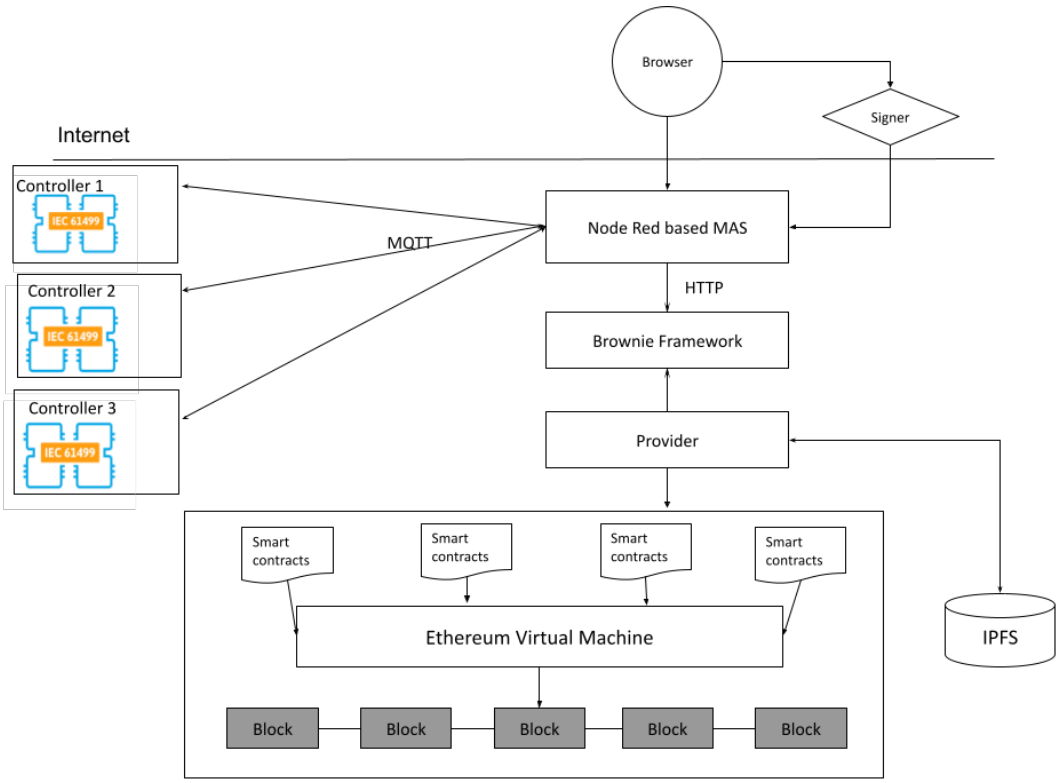

Fig. 2. Architecture for blockchain-enabled verification of a multi-agent system.

information is stored as the product metadata by executing another smart contract call. Once it finishes executing all tasks, we can compare the metadata with the order recipe for validation purposes.

V. CASE STUDY

A. Colour customisation using software agents

1) User agent: Users request the customised colours of the products with the required amount, and the user agent handles the order. Once the user agent receives the order, it accepts or rejects the order by comparing it with the agent's capability and information about the skills from GraphDB. Whenever the user agent accepts the order, it records the order information in Graph DB by executing SPARQL queries. The product id, name, colour, and amount are the product details updated in the GraphDB, and the same information is also recorded on the blockchain via the execution of a smart contract function call.

```
INSERT DATA {
agents:Product_${msg.payload.id} a agents:Product;
agents:hasProductId "${msg.payload.id}";
agents:hasName "${msg.payload.name}";
agents:hascolour "${msg.payload.colour}";
agents:hasAmount "${msg.payload.amount}".
}
```

This piece of code is to insert data into a database in the RDF (Resource Description Framework) format using the SPARQL query language. The code creates a new resource, identified by the URI "Product_$msg.payload.id", that is of type "agents:Product". It is then setting properties on that

Colour	Type	Combination
red	Primary	red
yellow	Primary	yellow
blue	Primary	blue
orange	Secondary	red -yellow
purple	Secondary	red-blue
green	Secondary	yellow-blue
teal	Tertiary	yellow-blue-blue
chartreuse	Tertiary	yellow-blue-yellow
vermilion	Tertiary	red-yellow-red
magenta	Tertiary	red-blue-red
violet	Tertiary	red-blue-blue
amber	Tertiary	red-yellow-yellow

Fig. 3. Colour information chart.

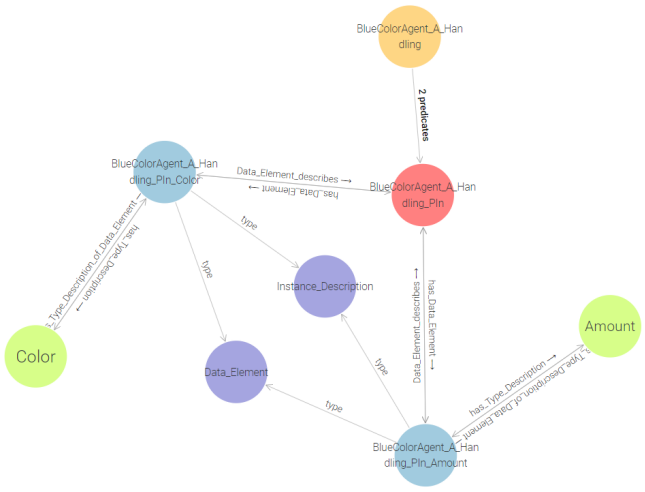

Fig. 4. An example of a colour agent's visual graph representation in GraphDB.

resource, such as "agents:hasProductId", "agents:hasName", "agents:hascolour", and "agents:hasAmount", and assigning values to those properties, which appear to be coming from a JavaScript object called "msg.payload".

2) Colour agents: The colour generation process is handled by the colour software agents. The colours red, yellow, and blue are the primary colours and these colours are used to generate secondary and tertiary colours. The secondary colours are Green, Orange, and Purple, and these colours are generated by mixing two primary colours. The tertiary colours are Teal, Chartreuse, Vermilion, Magenta, Violet, and Amber, and these colours can be generated by mixing primary and secondary colours. The colour information chart shown in Figure 3 explains which colour must be mixed with others.

The types of three colour agents red, yellow, and blue are enough to make other colours, so whenever the user requests to generate a particular coloured product, these agents work together and build a product with the requested colour. It is possible to have several primary colour agents of the same type in the system. Each colour agent has data elements of 'colour' and 'amount', a colour tells which colour can be processed by that agent, and the amount determines the number of products that can be produced. The ontology schema developed by Aljohsa Köcher *et al.* [**?**] is used for this case study and each type of agent has the same structure in GraphDB to store the information. One of the colour agents, that is, BluecolourAgent, used in this case study is shown in Figure 4. It has two elements, that is, colour and amount, and is associated with handling capability.

In this example, we use three different types of colour agents, red, yellow, and blue. Agents can be registered and unregistered on the fly, and this feature makes the system flexible and configurable. Agents are registered by executing SPARQL queries and each agent's information is stored in GraphDB. An example SPARQL query snippet of the registration of an agent (YellowcolourAgent) is shown below.

```
INSERT DATA {
agents:YellowcolourAgent_A rdf:type agents:
    colourAgent ;
agents:hasCapability agents:
    YellowcolourAgent_A_Handling.
agents:YellowcolourAgent_A_Handling VDI3682:hasInput
     agents:YellowcolourAgent_A_Handling_PIn;
VDI3682:hasOutput agents:
    YellowcolourAgent_A_Handling_POut;
  rdf:type agents:Handling;
agents:hasUrl "/YellowcolourAgent_A".
agents:YellowcolourAgent_A_Handling_PIn rdf:type
    agents:Product ;
DINEN61360:has_Data_Element agents:
    YellowcolourAgent_A_Handling_PIn_Amount ,
agents:YellowcolourAgent_A_Handling_PIn_colour.
agents:YellowcolourAgent_A_Handling_PIn_Amount rdf:
    type DINEN61360:Data_Element,
DINEN61360:Instance_Description ;
DINEN61360:has_Instance_Description agents:
    YellowcolourAgent_A_Handling_PIn_Amount ;
DINEN61360:has_Type_Description agents:Amount ;
DINEN61360:Expression_Goal "RequirementInteger" ;
DINEN61360:Logic_Interpretation "<=" ;
DINEN61360:Value 50 .
agents:YellowcolourAgent_A_Handling_PIn_colour rdf:
    type DINEN61360:Data_Element,
DINEN61360:Instance_Description ;
DINEN61360:has_Instance_Description agents:
    YellowcolourAgent_A_Handling_PIn_colour ;
DINEN61360:has_Type_Description agents:colour ;
DINEN61360:Expression_Goal "RequirementString" ;
DINEN61360:Logic_Interpretation "==" ;
DINEN61360:Value "Yellow" .
agents:YellowcolourAgent_A_Handling_POut rdf:type
    agents:Product ;
DINEN61360:has_Data_Element agents:
    YellowcolourAgent_A_Handling_POut_Amount ,
agents:YellowcolourAgent_A_Handling_POut_colour.
agents:YellowcolourAgent_A_Handling_POut_Amount rdf:
    type DINEN61360:Data_Element ,
DINEN61360:Instance_Description ;
DINEN61360:has_Instance_Description agents:
    YellowcolourAgent_A_Handling_POut_Amount ;
DINEN61360:has_Type_Description agents:Amount ;
DINEN61360:Expression_Goal "Assurance" ;
DINEN61360:Logic_Interpretation "<=" ;
DINEN61360:Value 50 .
agents:YellowcolourAgent_A_Handling_POut_colour rdf:
    type DINEN61360:Data_Element ,
DINEN61360:Instance_Description ;
DINEN61360:has_Instance_Description agents:
    YellowcolourAgent_A_Handling_POut_colour ;
DINEN61360:has_Type_Description agents:colour ;
DINEN61360:Expression_Goal "Assurance" ;
DINEN61360:Logic_Interpretation "==" ;
DINEN61360:Value "Yellow" .
}
```

This data describes an agent called "YellowcolourAgent_A" of type "colourAgent" which has the capability of "YellowcolourAgent_A_Handling". This "YellowcolourAgent_A_Handling" is of type "Handling" and has an input product "YellowcolourAgent_A_Handling_PIn" and an output product "YellowcolourAgent_A_Handling_POut". The input product "YellowcolourAgent_A_Handling_PIn" has two data elements: "YellowcolourAgent_A_Handling_PIn_Amount" and "YellowcolourAgent_A_Handling_PIn_colour". These data elements have type descriptions of "Amount" and

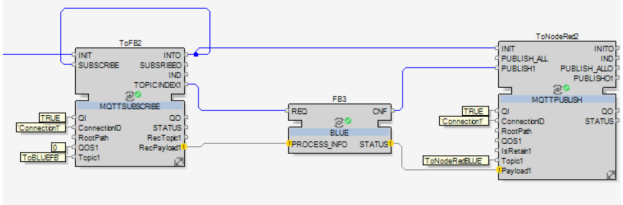

Fig. 5. Controller FB in IEC 61499 standard

"colour" respectively and have specific logical interpretations and values.

When the colour agent is registered in the system, it begins to listen to the MQTT topic in order to receive proposals from the processing agent. The received proposals are analyzed by the colour agents, and they accept a proposal only if all constraints of that agent are met. Upon accepting a proposal, the colour agent responds by providing the details of the execution price for the task. Once approval is obtained from the processing agent, the agent proceeds to create the product using the specified colour. These colour agents are the execution agents associated with each controller function block in accordance with the IEC 61499 standard. These blocks perform the task at the hardware level. Communication between function blocks and agents is facilitated by employing the MQTT protocol. Subsequently, after the execution of each process, the agent records the process information by invoking a smart contract function on the blockchain.

3) Processing agent: The processing agent is responsible for creating process sequences for mixing colours. The colour recipe generated by the processing agent is then used to generate secondary and tertiary coloured products.

The processing agent looks at the order recipe and creates proposals, and these proposals are shared with registered agents with the same capability. The colour agents check the proposal, and if all constraints are satisfied, then it accepts the proposal with the price and sends it back to the processing agent. Colour agents and processing agents are communicated through MQTT topics, and these topics are saved while registering each colour agent. The processing agent listens to its MQTT topic to collect accepted proposals from the colour agents. Once the processing agent collects enough accepted proposals, it then identifies the best proposal and later gives permission to a colour agent to execute the task on the hardware level.

B. Interaction of IEC 61499 FBs with colour agents

The colour agents are associated with its IEC 61499 controller function blocks of the mechatronic component, i.e. the red colour agent triggers the execution of the function block corresponding to its red colour mechatronic components via the MQTT protocol. The software agent and IEC 61499 function blocks interact with the help of specified MQTT topics. After the execution of the given task by the controller, it informs their respective colour agent. It is also possible to write the process information directly on the blockchain from the same Programmable logic controller (PLC) via APIs. The IEC 61499 application in the Schneider Electric EcoStruxure Automation Expert (EAE) tool and one of the composite function blocks of the colour agent are shown in Figure 5.

C. Customized product order recipe verification using blockchain

After executing each process by an agent, the description of the process, the status of the process, the timestamp and other information (that is, the video of the execution or the picture of the product) are added as metadata while performing each step in an order recipe. This is done with the help of the execution of the smart contract function call on EVM. In this paper, the smart contract is created using the Solidity programming language and is deployed on the Ethereum testnet and private chain. A Python framework brownie is used for the development and deployment of the smart contract. In this case study, a few functions are implemented to record and retrieve the process information from the blockchain, which is shown below.

```
pragma solidity ^0.8.0;

contract ProductStorage {
    struct Product {
        uint256 productId;
        string productName;
        string colour;
        mapping(uint256 => Metadata[]) productToMeta
    ;
    }
    struct Metadata {
        uint256 processId;
        string processInfo;
        string uri;
    }
    mapping(uint256 => Product) public idToProduct;
    function mintProduct(uint256 _id, string memory
    _name, string memory _colour)
        public
        returns (uint256 productId)
    {
        productId = _id;
        Product storage newProduct = idToProduct[
    productId];
        newProduct.productName = _name;
        newProduct.colour = _colour;
    }
    function addMetaData(
        uint256 _productId,
        uint256 _processId,
        string memory _processInfo,
        string memory _uri
    ) public {
        idToProduct[_productId].productToMeta[
    _productId].push(
            Metadata(_processId, _processInfo, _uri)
        );
    }
    function retrieveProductById(uint256 _id)
        public
        view
        returns (string memory)
    {
        Product storage derivedProduct = idToProduct
    [_id];
        return (derivedProduct.productName);
    }
```

```
    function retrieveProductMetaById(uint256 _id)
        public
        view
        returns (Metadata[] memory)
    {
        Metadata[] memory derivedMeta = idToProduct[
    _id].productToMeta[_id];
        return (derivedMeta);
    }
.........................................
}
```

Solidity contract for a ”ProductStorage” smart contract that allows for the creation and management of product information on the Ethereum blockchain. It has several functions that can be called by users.

```
The "mintProduct" function allows for the creation
    of a new product, with a given id, name, and
    colour. This information is stored in the "
    Product" struct, and is mapped to the id in the
    "idToProduct" mapping.
The "addMetaData" function allows for the addition
    of metadata to a specific product, identified by
     its id. The metadata includes a process id,
    process info, and uri, and is stored in the "
    Metadata" struct.
The "retrieveProductById" function allows for the
    retrieval of product information by its id and
    returns the product name.
The "retrieveProductMetaById" function allows for
    the retrieval of metadata of a specific product
    by its id and returns an array of metadata.
```

APIs are created using the flask framework to invoke the smart contract, and these APIs are deployed on the network so that agents and IEC 61499 FBs can directly invoke these functions to record and retrieve the information from the blockchain. The recorded metadata of the product can be obtained via the ‘retrieveProductMetaById’ smart contract function. The metadata of the product consists of process description, process status, timestamp, and other information like the video of execution or picture of the product. Eventually, this information is compared with the actual order recipe to see if any deviation has occurred. Once data are recorded on the blockchain, it is not possible to tamper with or alter the information. The user requested to customise the product can also retrieve the same information and can be validated from the user side. This functionality enhances trust and reliability, mitigating the risk of defective products entering the market even if individuals within the factories attempt to manipulate any process. The user interface, depicted in Figure 6, facilitates agent registration, order placement, retrieval of product details, and verification of process sequences.

VI. Conclusion and future work

The customised product’s order recipe and process sequence data on the blockchain are compared and successfully verified. The proposed solution provides flexible and reconfigurable production lines with the introduction of software agents. The registration and removal of software agents on the fly enable the implemented framework’s scalability. The metadata information on the blockchain cannot be altered or tampered with by anyone, and this adds security and trust. The user can also retrieve the same information about the product from the blockchain, which makes the system more transparent and reliable.

The proposed framework needs to be tested on the industrial production line model. Implementing secure and reusable smart contract solutions on Ethereum could be considered in future work. The development of the Non-Fungible Token (NFT) feature for each product helps to track, identify, and prevent counterfeit products on the market. Integration of an NFT-based manufacturing marketplace with multi-agent system capabilities and verification possibilities could be the next step in the future.

VII. Acknowledgements

This research was supported, in part, by the Horizon Europe Zero-SWARM project funded by the European Commission (grant agreement: 101057083).

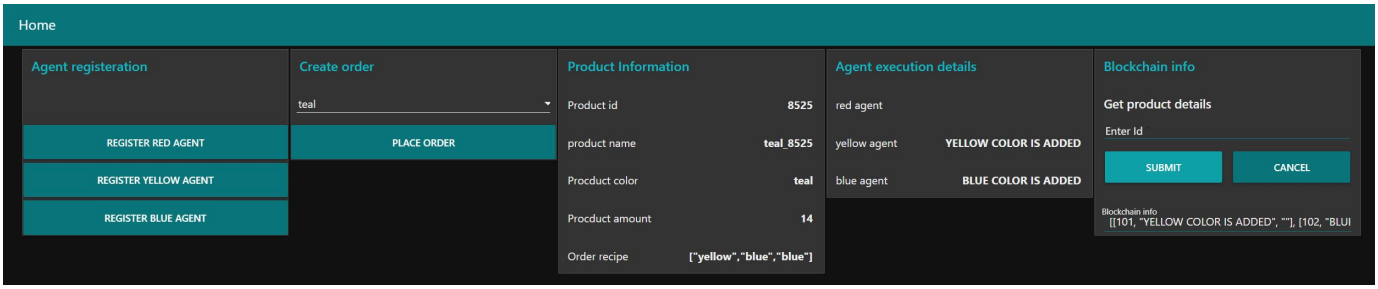

Fig. 6. Agent registration, order placement, and product verification via user interface