Modeling and Verifying OPC UA of Aggregating Server Architecture for Vertical Integration

Zifan Liang, Wei Lin, Sini Chen, Huibiao Zhu*
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai, China

***Abstract*—With the development of Industry 4.0 (I4.0), the demand for a protocol to unify communication between different devices is dramatically increasing. OPC Unified Architecture (OPC UA) is a protocol that can be used to accomplish information interoperability in manufacturing and its usage is expanding. Some research has been conducted on the implementation approaches for aggregating server pattern OPC UA in vertical integration. However, few efforts have been made on the formal verification of OPC UA as a vertical communication protocol to facilitate manufacture.**

In this paper, we design an aircraft parts production use case where vertical communication between the management layer and shop floor devices is achieved via OPC UA aggregating server architecture. Information Model and Services are two fundamental contents of the OPC UA protocol. With the help of process algebra Communicating Sequential Processes (CSP), we build the use case OPC UA model and realize both the Information Model and Services using a verification tool, Process Algebra Toolkit (PAT). We verify the general properties of the system (Divergence Freedom and Deadlock Freedom) and functional properties of OPC UA (Service Read Success, Service Call Success, Monitor Variable Success and Monitor Event Success). The results demonstrate the feasibility of the OPC UA protocol in vertical communication.

***Index Terms*—OPC UA; Process Algebra CSP; Aggregating Server; Vertical Integration; Modeling**

I. Introduction

Developed from Classic Open Platform Communication (OPC), which is primarily designed for Windows platform, OPC UA is proposed as a platform-independent standard providing a connectivity foundation for Internet of Things (IoT) and I4.0 initiative [1]. OPC UA protocol is well designed for interoperability to unify vertical and horizontal higher production levels [2].

Contrary to horizontal integration, a consolidation of many firms that handle the same part of the production process, vertical integration is typified by one firm engaged in different parts of production [3]. Due to increasing demands for more flexibility [4] and the development of I4.0 in manufacturing systems, vertical integration of information within the production system is necessary. In order to communicate directly between different layers, for example, Enterprise Resource Planning (ERP), Manufacturing Execution System (MES) and shop-floor layer, a common communication standard [5] is in desperate need. As a communication technology offering a unified interface for information exchange [4], OPC UA is appropriate to be this standard.

When using OPC UA in vertical integration, an aggregating server pattern is often used. In book *OPC Unified Architecture* [6], Mahnke and his coworkers introduce the aggregating server concept and give an MES scenario in which the intermediate server processes production requests from MES OPC UA client and distributes subtasks to underlying aggregating servers. The pattern is shown in Fig. 1. To build a prototype solution for information integration in OPC UA, Johansson exploits the aggregating pattern and even more, he presents a chained aggregating server architecture for vertical integration in industrial information system [7].

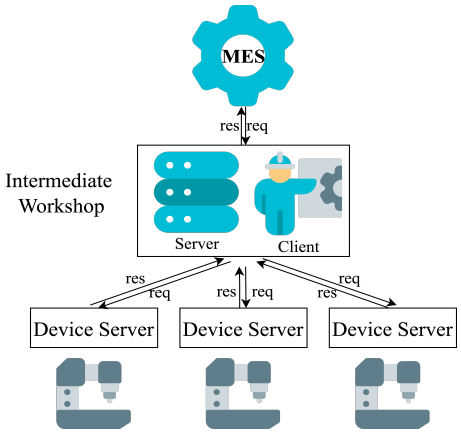

Fig. 1: Aggregating Server pattern

Some efforts have been made focusing on formally verifying the security properties of OPC UA. Puys et al. [8] and Guerra et al. [9] checked the security properties of OPC UA using verification tools ProVerif and VeriPal. They found some OPC UA vulnerabilities and provided countermeasures. However, it lacks theoretical research on verifying functional properties of OPC UA. So we use process algebra CSP to build and verify a formal model that uses OPC UA information exchange Services to complete a production process.

In this paper, we make the following contributions:

- We design an aircraft parts production use case with aggregating server architecture which uses OPC UA protocol to complete a production process.

*Corresponding author: hbzhu@sei.ecnu.edu.cn

DOI: 10.18293/SEKE2024-134

- We formalize a model of the use case using CSP, and realize the use case OPC UA Information Model and Services.
- We verify the properties of the system using PAT and the results certify the feasibility of the use case.

The rest of the paper is organized as follows. Section II introduces fundamental features of OPC UA and CSP. Section III presents an aircraft production use case which deploys OPC UA to integrate information from shop-floor layer to MES and implemented with aggregating server architecture. In Section IV, we give a formalized model of the use case. We present the verification results of our model in Section V. In the last section, Section VI, we make a conclusion and discuss the future possibilities of our work.

II. Background

A. OPC UA

In this part, we will give a brief introduction on the two essential parts of OPC UA: Information Model and Service. In OPC UA, Information Model formulates data formats in a domain, while Services allow the exchange of all kinds of information between different devices.

1) Information Model: Information Model is used for semantically describing the exchanged data in OPC UA, designers of which value the interoperability between systems from different vendors as the most important requirement. An OPC UA Information Model defines a standardized configuration of Nodes and References for its Servers [10]. The information exposed by an OPC UA Server is called *AddressSpace* which consists of *Nodes* connected by *References* [11]. OPC foundation has defined eight special types of Base Node Classes, including *ObjectType*, *Object*, *Variable*, *Method*, etc. Those eight node class types constitute an OPC UA Base Information Model [12]. Together with some built-in functionalities like DA (Data Access), HA (Historical Access) and so on, a standardized Information Model is outlined.

Based on the above base and standardized Information Model, OPC UA allows users to define their own Information Models [10]. Thus we design an Information Model in our aircraft parts production use case and the detail will be presented in Section III and Section IV.

TABLE I: Services grouped by use case

Use case	Service sets or services
Find servers	Discovery Services Set
Connection management between clients and servers	Secure Channel Services Set Session Service Set
Find information in the Address Space	View Service Set
Read and write data and metadata	Read and Write Service
Subscribe for data changes and Events	Subscription Service Set Monitored Item Service Set
Calling Methods defined by the server	Call Service
Access history of data and Events	HistoryRead and HistoryUpdate Service
Find information in a complex Address Space	Query Service Set
Modify the structure of the server Address Space	Node Management Service Set

2) Service: Services are methods used by an OPC UA client to access the data of the Information Model provided by an OPC UA server [6]. We reference a table from the book *OPC Unified Architecture* [6] to illustrate services provided by OPC UA in Table I.

Discovery, **Secure Channel** and **Session** are Service sets used in the communication establishment phase. The other Service Sets are related to the exchange of information. In this paper, we will focus on information exchange related Services:

- **TranslateBrowsePathsToNodeIds Service**
  This is a fundamental service which enables clients to access components of an Object based on knowledge of the Object NodeId and built-in ObjectType. This service works on the premise that BrowseNames are the same for Object components and corresponding ObjectType.
- **Read and Call Services**
  Read Service is an important feature of OPC UA, allowing clients to access Variable with NodeId of the Variable. Similarly, Call Service provides a way for clients to call Method of Object in the server's Address Space.
- **Subscription and Monitored Items Services**
  Clients subscribe data changes and events from server through Subscription and Monitored Items Services. When a client requests a server with CreateSubscription Service, a SubscriptionId is returned. Then the client could use the SubscriptionId in CreateMonitoredItems Service.

It should be noted that the other services must first utilize **TranslateBrowsePathsToNodeIds** Service to get $NodeId$ for the following usage.

B. CSP

Process Algebra is an algebraic approach for analysing concurrent processes. CSP is a process algebra proposed by C. A. R. Hoare in 1978 [13]. Since then, it has been improved and refined. The syntax of CSP is specified below:

$$P, Q ::= SKIP \mid a \rightarrow P \mid c!v \rightarrow P \mid c?v \rightarrow P \mid P; Q \mid P||Q \mid P \Box Q \mid P \lhd B \rhd Q \mid P[[T]]Q$$

- $SKIP$ represents the natural exit of the process when there is no event left to execute.
- $a \rightarrow P$ represents after action a, process P will be executed.
- $c!v \rightarrow P$ outputs a value v through channel c and then execute process P.
- $c?x \rightarrow P$ receives a value stored in variable x through channel c and then execute process P.
- $P; Q$ executes process P and Q sequentially.
- $P||Q$ forms a process in which process P and Q are executed in parallel.
- $P \Box Q$ stands for a process formed by a general choice between process P and process Q and the environment can control which process will be selected.
- $P \lhd B \rhd Q$ is a conditional statement. When condition B is true, execute process P. Otherwise, execute process Q.

- $P\,[[T]]\,Q$ means process P and Q concurrent actions in channels of set T.

III. Use Case: Aircraft Parts Production

In the production field, especially the vehicle and aircraft production field where there are a great number of distinct complex devices, OPC UA could be used to provide the groundwork for IoT by breaking down the communication barriers between objects, allowing programs to exchange all the relevant information throughout a manufacturing organization [14].

We pick up a classical scene in an aircraft factory where a group of **Devices** managed by a **Workshop** is producing aircraft parts under the command of the **MES** layer. In the process of aircraft parts production, OPC UA is used as a unified protocol to represent data and exchange messages among different layers.

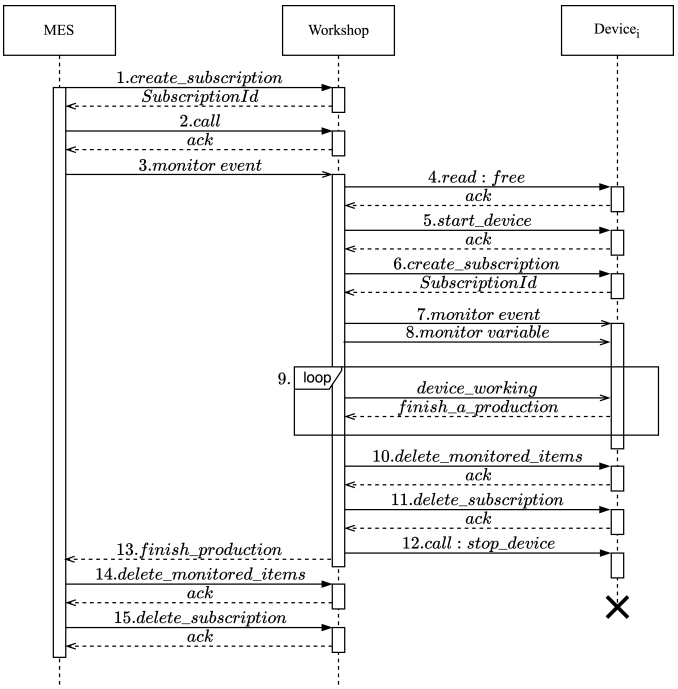

Fig. 2: Aircraft part production flow

The production flow involving a device Device_i is shown in Fig 2. The whole production flow can be divided into three phases.

Phase 1: Startup

In this phase, the production preparation work is done. First, a subscription is built up between MES and Workshop and MES gets a unique *SubscriptionId* for subsequent requests. In step 2, MES uses Call Services to set materials, employees and target part numbers to produce in Workshop. In step 3, MES uses the *SubscriptionId* to subscribe an event $finish\_production$ from Workshop. Then in step 4, Workshop invokes Read Service to get Device_i's variable status. In step 5, Workshop uses Call Service to start Device_i. The subscription between Workshop and the ith Device is invoked in step 6. In step 7 and step 8, monitored items are subscribed. Workshop subscribes event *finish_a_production* and variables *ConsumedMaterial* and *ManTime*.

Phase 2: Device Working

In this phase, Device_i cooperates with other devices to produce the required number of parts which is set in step 2. Workshop maintains a *ProducedNumber* variable. Each time a part is produced, Device_i publishes *finish_a_production* to notify Workshop and Workshop will increase *ProducedNumber* by one. When the required number is reached, our production moves into the next phase.

Phase 3: Wrap-Up

In this stage, resources are released and reset. When reaches the production target, Workshop invokes DeleteMonitoredItem Service, DeleteSubscription Service and uses Call Service to stop Device_i. Then, Workshop publishes $finish\_production$ notification to MES. After that, MES deletes monitored items and subscriptions to Workshop.

IV. Modeling

A. Information Model

We define three ObjectTypes: *ResourceType*, *ProductType* and *DeviceType* to provide type definitions of Objects shown in Fig. 3. These ObjectType knowledge is universally programmed in both OPC UA clients and servers.

Besides, there are three Object nodes: *Resource*, *Product* and *Device*, each representing an entity in the server Information Model structuring variables and methods. *Resource* and *Product* are nodes in the Workshop Server Information Model representing storage information. While *Device* Object in Device layer Server Information Model would describe the Device status and provide methods.

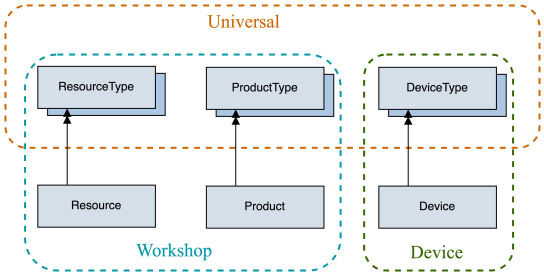

Fig. 3: Use Case Information Model

B. Sets and Channels

First, we give the definition of the sets.

- **Node** involves nodes we defined in address space.

$$Node =_{df} \{ResourceType, ProductType, DeviceType, Resource, Product, Device\}$$

- **Constant** contains constants that are set to initiate Node variables. Specifically, *D* is an important constant which specifies the number of Devices in the use case. *D*

will be mentioned quite a lot in the following modeling processes.

- **SubscriptionId** defines Ids of the subscription between MES and Workshop, between Workshop and Devices.

$SubscriptionId =_{df} \{MSSubId, SESubId[1..D]\}$

- **MonitoredItemId** defines Ids for monitored items assigned by Workshop server or Device server.

$MonitoredItemId =_{df}$
$\{ReachTarMEId, ReachAProMEId,$
$ConMatMVId, WorNumMVId\}$

- **NodeId** is the set of NodeId for ObjectType, Object, Variables and Methods in our use case address space.

$NodeId =_{df}$
$\{resourceNodeId, productNodeId,$
$DeviceNodeId[1..D], resourceTypeNodeId,$
$productTypeNodeId, DeviceTypeNodeId\}$

- **Function** contains functions of nodes for accessing variables and invoking methods.

Next, we give the definition of the channels.

The channel *Time* is used to tell the time for synchronizing devices' working processes. Other channels are Service-oriented. As shown in Fig 4, we define two groups of channels. *ComMS* fomalizes channels between MES and Workshop and $ComSE_i$ fomalizes channels between Workshop and $\mathrm{Device_i}$.

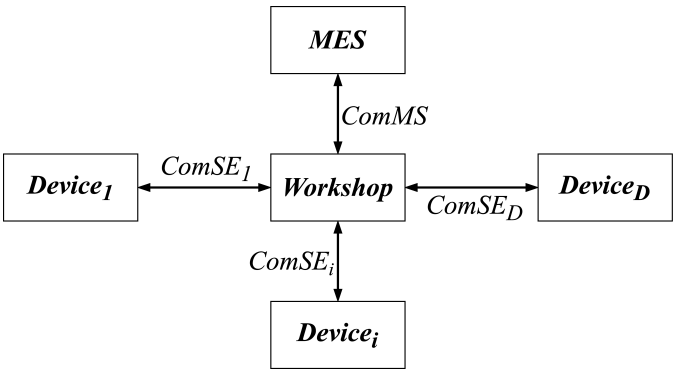

Fig. 4: Aircraft part production flow

Each group contains following channels for each Service:

- ***Com_TransNodeId*** set are channels used in TranslateBrowsePathstoNodeIds Service. It includes:
  ComMSTN, ComSETN[i]
- ***Com_Sub*** set contains channels used to create and delete subscriptions. Including:
  CreMSSub, DelMSSub, CreSESub[i], DelSESub[i]
- ***Com_MV*** set are channels used to monitor variables which includes:
  CreMSMV, ComMSMV, DelMSMV,
  CreSEMVWorker[i], CreSEMVMaterial[i],
  ComSEMVWorker[i], ComSEMVMaterial[i],
  DelSEMVWorker[i], DelSEMVMaterial[i]
- ***Com_ME*** set are channels used to create and delete monitored events. Including:
  CreMSME, ComMSME, DelMSME,
  CreSEME[i], ComSEME[i], DelSEME[i]
- ***Com_Call*** set are channels used to implement call method Service. Including:
  ComMSCall, ComSECall[i]
- ***Com_Read*** set are channels used to implement OPC UA read Service. Including:
  ComMSRead, ComSERead[i]

C. Overall modeling

The aircraft part use case focuses on the vertical communication from MES to Workshop, down to Devices. Correspondingly, our model consists of three major processes: MES, Workshop and Device.

$\boldsymbol{System} =_{df}$
$\boldsymbol{Init}; (\boldsymbol{MES} \parallel \boldsymbol{Workshop} \parallel \boldsymbol{Device} \parallel \boldsymbol{Clock}(0))$

Init mainly deals with the initialization of Nodes. After the sequential running of **Init** process, Nodes in our use case Information Model are defined and the ith Device's NodeIds (including its components) and values are generated randomly.

D. Clock Modeling

In our model, we adopt a design of time from [15] to record the passage of time. Clock in our model is used to synchronize different Device production processes and to calculate production man-hours for measuring individual production progress.

$\boldsymbol{Clock}(t) =_{df}$
$tick \rightarrow \boldsymbol{Clock}(t+1) \,\square\, Time!t \rightarrow \boldsymbol{Clock}(t)$

E. MES Modeling

MES mainly acts as an OPC UA client. In ***MESCreSub***, the client of MES initiates the process with CreateSubscription Service to subscribe Workshop server and get a SubscriptionId in return. ***MESController*** is composed of some Call Services requested by the MES client from the Workshop server.

$\boldsymbol{MES} =_{df}$
$\boldsymbol{MESCreSub}; \boldsymbol{MESController}; \boldsymbol{MESMonitorEvent};$

In ***MESController*** process, we first use *ComMSTN* channel to request TranslateBrowsePathsToNodeIds Service to get Method's NodeId which will be used for Call Service through *ComMSCall* channel. And the service will be verified in Section V.

$\boldsymbol{MESController}(nodeId, browseName, arg) =_{df}$
$ComMSTN!nodeId.browseName \rightarrow$
$ComMSTN?id \rightarrow$
$ComMSCall!id.arg \rightarrow SKIP;$

We formalize ***MESMonitorEvent*** process to represent Monitor Event Service in MES client. The action *finish_production* represents the monitored event triggered by Workshop Server to inform MES that the production is finished.

$\boldsymbol{MESMonitorEvent} =_{df}$
$CreMSME!MSSubId.productNodeId.REQ_{MSME} \rightarrow$

$CreMSME?monitorId \rightarrow finish\_production \rightarrow$
$DelMSME!MSSubId.ReachTarMEId.DEL\_MSME \rightarrow$
$DelMSME?rep \rightarrow$
$(fail \lhd (rep == Del\_Fail) \rhd \boldsymbol{MESDelSub})$

F. Workshop Modeling

The whole Workshop process consists of two parts: client and server. The Workshop server would provide Services to reply to MES client's requests.

$$\boldsymbol{Workshop} =_{df} \boldsymbol{WorkshopService}; \boldsymbol{WorkshopClient}$$

Workshop server provides with many services, so we only show part of the process.

$$\begin{aligned}
&\boldsymbol{WorkshopService}_{(part.)} =_{df} \\
&\quad ComMSTN?nodeId.bn \rightarrow \\
&\quad \begin{pmatrix} ComMSTN!Resource.GetNodeId(bn) \rightarrow \\ \boldsymbol{WorkshopService} \\ \lhd (nodeId == resourceNodeId) \rhd \\ \begin{pmatrix} ComMSTN!product.GetNodeId(bn) \rightarrow \\ \boldsymbol{WorkshopService} \\ \lhd (nodeId == productNodeId) \rhd \\ fail \rightarrow \boldsymbol{WorkshopService} \end{pmatrix} \end{pmatrix} \\
\Box \quad &CreMSME?subId.nodeId \rightarrow \\
&\quad CreMSME!ReachTarMEId \rightarrow \\
&\quad start_product \rightarrow \\
&\quad finish_production \rightarrow \\
&\quad reset_Workshop_production \rightarrow \\
&\quad DelMSME?subId.monitorId.req \rightarrow \\
&\quad \begin{pmatrix} reset_MSMornitorId \rightarrow \\ DelMSME!Del_Suc \rightarrow SKIP \\ \lhd \begin{pmatrix} req == DEL_{MSME} \wedge \\ monitorId == ReachTarMEId \end{pmatrix} \rhd \\ DelMSME!Del_Fail \rightarrow SKIP \end{pmatrix}
\end{aligned}$$

The first part formalizes TranslateBrowsePathstoNodeIds Service. The Workshop server judges if the received NodeId corresponds to the product instance NodeId or the resource one. The second part shows how Workshop deals with the MES's monitor event request. When Workshop receives the SubscriptionId, it returns a MonitoredItemId and begins to use *start_product* action to activate the aggregating Devices.

After the Workshop client and server concurrently run the event *start_product*, the client continues the exchange of information with underlying Devices.

$$\begin{aligned}
&\boldsymbol{WorkshopClient} =_{df} \\
&\quad start_product \rightarrow \\
&\quad \boldsymbol{WorkshopReadFreeReq}(1); \\
&\quad \boldsymbol{WorkshopStartDevice}; \\
&\quad \boldsymbol{TurnDeviceTick}(1); \\
&\quad (\Box\, x : \{1..D\}@\boldsymbol{WorkshopWorkingWithADevice}(x));
\end{aligned}$$

$$\begin{aligned}
&\boldsymbol{WorkshopReadFreeReq}(o) =_{df} \\
&\begin{pmatrix} ComSETN[o-1]!DeviceNodeId[o-1]. \\ \quad DeviceType.GetFreeBN() \rightarrow \\ ComSETN[o-1]?nodeId \rightarrow \\ ComSERead[o-1]!nodeId \rightarrow \\ ComSERead[o-1]?free \rightarrow SKIP; \\ \boldsymbol{WorkshopReadFreeReq}(o+1); \\ \lhd (o \leq D) \rhd \\ finish_read \rightarrow SKIP; \end{pmatrix}
\end{aligned}$$

G. Device Modeling

In ***FullADeviceService*** each Device acts as a server to provide OPC UA services for Workshop client. Also, in ***ADeviceRunning*** they synchronize with ***Clock*** to get time for calculating if their production has reached the expected time under its employee number.

$$\begin{aligned}
&\boldsymbol{Device} =_{df} \\
&\quad (\Box x : \{1..D\}@\boldsymbol{FullADeviceService}(x)) \\
&\| \ (\| x : \{1..D\}@\boldsymbol{ADeviceRunning}(x))
\end{aligned}$$

V. Verification and Results

We simulate our model in model checker PAT, using which we analyzed two general system properties and four functional properties specifically in OPC UA to evaluate the feasibility of the protocol in vertical communication.

Property 1: Deadlock Freedom

We verify the absence of deadlock in our model to avoid the situation when no action could be taken even though some processes are capable of further action [16]. We examined our model deadlock freedom in PAT as follows.

$$\#\boldsymbol{assert\ System}\ deadlockfree;$$

Property 2: Divergence Freedom

Divergence Freedom is a basic property for the system to ensure that it won't be trapped in an infinite loop consuming resources and never end. We check the divergence freedom of our model with the primitive in PAT.

$$\#\boldsymbol{assert\ System}\ divergencefree;$$

Property 3: Service Read Success

We use Linear Temporal Logic (LTL) assertions in PAT to verify whether the Workshop could get the Device number value through OPC UA Service read. The $\diamond$ operator means the Service Read Success formula could be achieved eventually.

$$\begin{aligned}
&\#\boldsymbol{define}\ Service_Read_Success \\
&\quad after_read_freeDevice == D; \\
&\#\boldsymbol{assert\ System} \models \diamond\ Service_Read_Success;
\end{aligned}$$

Property 4: Service Call Success

There are four object methods in our model: AddMaterial and AddEmployee in Resource Object, method Produce in Product, method Start and End in Device Object. Similarly, we use a LTL assertion to verify these Service calls are performed successfully.

$$\begin{aligned}
&\#\boldsymbol{define}\ Service_Call_Success \\
&\quad after_call_addmaterial == MESAssignMaterial \\
&\quad \&\& \ after_call_addemployee == MESAssignEmployee \\
&\quad \&\& \ after_call_produce == MESProduce \\
&\quad call_start_end_success;
\end{aligned}$$

$\#\boldsymbol{assert\ System} \models \diamond\ Service\_Call\_Success;$

Property 5: Monitor Variable Success

Workshop would monitor two variables in Devices: ConsumedMaterial and WorkerNumber. To show that the monitor variable Service works well in vertical communication with aggregating server architecture, we examine whether sum of the resource number in Devices and in Workshop would always equal to the corresponding initial system parameter in any state of the system.

$$\begin{aligned}&\#\boldsymbol{define}\ Monitor\_Variable\_Success\\&\quad MESAssginEmployee ==\\&\qquad Device\_all\_workernum + resource\_avbemployee\\&\quad \&\&MESAssginMaterial ==\\&\qquad Device\_all\_consmaterial + resource\_avbmaterial;\\&\#\boldsymbol{assert\ System} \models \Box\ Monitor\_Variable\_Success;\end{aligned}$$

Property 6: Monitor Event Success

Monitor event is also a monitor Service based on subscription. The occurrence of finish_production means the required number of productions has been produced and Workshop Server notifies MES Client through their subscription.

$\#\boldsymbol{assert\ System} \models \diamond\ finish\_production;$

Verification Results

The verification results of our model are shown in Fig. 5. *divergencefree* and *deadlockfree* means the system won't be stuck or endlessly loops. Validity of *Service_Read_Success* and *Service_Call_Success* imply Read Service and Call Service of OPC UA protocol are applicable in vertical communication through different layers in manufacturing. Correctness of *Monitor_Variable_Success* and *Monitor_Event_Success* mean Monitored Item Service and its prerequisite Subscription Service are reliable in massive applications between varied instruments.

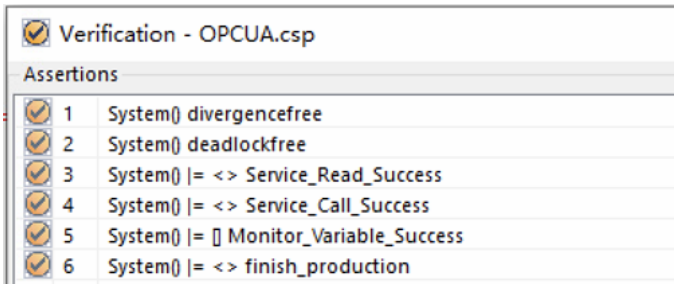

Fig. 5: Verification Results

These verification results show our model is reliable and OPC UA protocol can satisfy production use case through its unified Information Model and series of Service sets.

VI. Conclusion and Future Work

In this paper, we applied the OPC UA protocol in an aircraft production use case using aggregating server architecture. We modeled the system with process algebra CSP to verify the practicability of OPC UA in vertical communication. We designed the Information Model for each layer and realized OPC UA Services in our model to finish the production process. We verified general system properties and functional properties related to OPC UA: *Deadlock Freedom*, *Divergence Freedom*, *Service Read and Call Success*, *Monitor Variable Success* and *Monitor Event Success*. The validity of these properties shows that OPC UA's potential in the industry communication field is far beyond expectations.

In the future, we hope to explore applications of OPC UA in other impactful scenarios and contribute to formally proving the usability of OPC UA as a standardized communication protocol in industry.

VII. Acknowledgements

This work was partially supported by the National Key Research and Development Program of China (No. 2022YFB3305102), the National Natural Science Foundation of China (No. 62032024), the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software (No. 22510750100), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

[1] opcfoundation.org, "Update for iec 62541 (opc ua) published," 2015, Accessed: 2024-05-23.

[2] Peter Drahoš, Erik Kučera, Oto Haffner, and Ivan Klimo, "Trends in industrial communication and opc ua," in *2018 cybernetics & informatics (K&I)*. IEEE, 2018, pp. 1–5.

[3] wikipedia, "Vertical integration," 2022, Accessed: 2024-05-25.

[4] Sebastian Schmied, Daniel Großmann, Selvine G Mathias, and Suprateek Banerjee, "Vertical integration via dynamic aggregation of information in opc ua," in *Intelligent Information and Database Systems: 12th Asian Conference, ACIIDS 2020, Phuket, Thailand, March 23–26, 2020, Proceedings 12*. Springer, 2020, pp. 204–215.

[5] Max Hoffmann, Tobias Meisen, Daniel Schilberg, and Sabina Jeschke, "Multi-dimensional production planning using a vertical data integration approach: A contribution to modular factory design," in *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*. IEEE, 2013, pp. 1–6.

[6] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm, *OPC unified architecture*, Springer Science & Business Media, 2009.

[7] Markus Johansson, "Aggregating opc ua server for generic information integration," 2017.

[8] Maxime Puys, Marie-Laure Potet, and Pascal Lafourcade, "Formal analysis of security properties on the opc-ua scada protocol," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 67–75.

[9] Enrico Guerra, Mariano Ceccato, and Marco Rocchetto, "Formal verification and risk assessment of an implementation of the opc-ua protocol," 2022.

[10] Matti Siponen, Ilkka Seilonen, Samuel Brodie, and Timo Oksanen, "Next generation task controller for agricultural machinery using opc unified architecture," *Computers and Electronics in Agriculture*, vol. 203, pp. 107475, 2022.

[11] opcfoundation.org, "Opc unified architecture specification part 5: Information model.," 2023, Accessed: 2024-05-27.

[12] Kudzai Manditereza, "Opc ua base information model and companion specifications [3 of 11]," 2021, Accessed: 2024-05-30.

[13] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, 1978.

[14] dataPARC, "Opc ua: A framework for the industrial internet of things," Accessed: 2024-05-25.

[15] Ran Li, Huibiao Zhu, Lili Xiao, Jiaqi Yin, Yuan Fei, and Gang Lu, "Formalization and verification of vanet.," in *SEKE*, 2020, pp. 1–6.

[16] C. A. R. Hoare, *Communicating sequential processes*, vol. 178, Prentice-hall Englewood Cliffs, 1985.