An Application of Microservices Architecture Pattern to Create a Modular Computer Numerical Control System

Maxim Ya. Afanasev, Yuri V. Fedosov, Anastasiya A. Krylova, Sergey A. Shorokhov
ITMO University
St. Petersburg, Russian Federation
amax@niuitmo.ru, yf01@yandex.ru, {ananasn94, stratumxspb}@gmail.com

***Abstract*—Currently, the most common approach to Computer Numerical Control (CNC) system design is a monolithic architecture. However, the introduction of the concept of Cyber-Physical Production Systems (CPPS) requires a paradigm shift in the design of control systems. This paper suggests a new approach to developing modular industrial equipment using a microservices architecture pattern. Microservices architecture features are addressed, as well as advantages and disadvantages. A heterogeneous computer network, where nodes communicate via a message queue, is proposed as a basis for the computer numerical control system. Fault tolerance is provided by modules full autonomy and reliable messaging. Furthermore, NoSQL database, guaranteeing high data accessibility and increased data access speed, is applied. An apparatus for selective photopolymer laser curing of free-form surfaces is considered as an example. Common setup structure, as well as main hardware and software modules, are described. Moreover, a distributed network latency simulation was carried out to prove the viability of the proposed microservices architecture.**

I. Introduction

By analysing the evolution of modern industry it can be concluded that by far the most promising development destination is creating flexible distributed automated production lines. The novel mass production concept was established as a consequence of The Fourth Industrial Revolution (or Industry 4.0) as well as gradual CPPS spreading. “Hard” conveyor approaches are consequentially transformed into small batch custom production. Moreover, the development of small innovation enterprises and startups is still going on. On balance, the approach to modern intelligent equipment design is changing.

The first CNC machine tools appeared in the 1950’s but their development still continues. However, the development has been mostly focused on mass production. So, CNC systems have always been complex, high-performance, and extremely expensive. Furthermore, the deployment process of such systems is long-term and modern system lifetime is dozens of years. Understandably, these circumstances make the implementation of modern communication technologies more complex in that each correction requires either the restructuring of the whole production system or the creation of additional control layers in order to link outdated equipment with a modern CPPS. Such an approach is the most appropriate during a transition period and ultimately phasing out the older system.

Therefore, it is necessary to rethink the CNC equipment design paradigm itself and the ability to plug new equipment into the informational communication environment via open protocol becomes quite important. Different attempts to implement such approaches have been made. For example, Grigoriev and Martinov in their research propose the approach of designing an agile CNC kernel based on an independent library platform. The open architecture of this system includes different abstract layers related to different human-machine interfaces (HMI) along with description possibilities of system components in different programming languages. The connection of system components is established via Fieldbus [1].

Bin describe an open platform for creating CNC systems and this system consists of a set of multipurpose components that can be employed repeatedly and a set of communication modules to connect them [2]. A similar approach is presented in the paper Ma [3]. Morales-Velazques et al. propose a platform with the open architecture based on a multi-agent system of software and hardware components, named MADCON. The hardware units of their proposed system integrate control and monitoring functions providing an FPGA-based open architecture for reconfigurable applications. The software components utilize the XML structure for system description files, gathering features like a flowchart descriptive language and a graphic user interface [4].

Works of Verba [5] and Prazeres [6] deal with an interesting concept, Fog of Things. This concept is a modern interpretation of the Internet of Things (IoT) that, in fact, is a foundation of many CPPS. Fog of Things allows for the creation of a more homogeneous informational communication environment thereby improving and simplifying component communication protocol.

II. Motivation and case study

Modern control systems are complex and solid, where each element is a black box with a hard hierarchical architecture. Everything in such systems is focused on quality assurance, reliability, and trouble-proof behaviour. The inertia of such systems forces SCADA developers to use a monolithic architecture because the equipment with distributed control can not be easily added to a distributed production environment.

Consequently, the emphasis is placed on the concept of integration. In other words, developers focused on joining heterogeneous components into one production system instead of using an interoperability concept. The interoperability concept

ISSN 2305-7254

entails creating an open interface that allows components to stay in an autonomous state with communication ability.

Accordingly, we need to move on to the development of equipment with a modular architecture. Such architecture is based on two main postulates: unification and hybridization. Unification is an open software and hardware structure that allows for the assembly of new equipment and software as a "the smart children's designer." Dividing one product into several interchangeable building blocks with described input and output unification can then be achieved.

In summary, it can be noted that any manufacturing equipment consists of the following parts:

- Processing head actuator.
- Coordinate table that moves processing head actuator.
- CNC unit

It is obvious that the coordinate table is the most general-purpose unit where different processing head actuators can be mounted. Thus, the equipment type can be easily changed. For example, a milling machine tool can turn into a laser cutting machine switching the milling head to a laser head. Moreover, a 3D printer can be obtained by mounting a filament extruder.

Additionally, it is possible to create a setup that combines features of different types of equipment by applying the hybridization principle. For instance, an additive-subtractive machine can be assembled by joining some features of the 3D-printer and a three-coordinate milling machine tool. Also, a combination of milling cutter for roughing and laser for surface finishing and polishing is feasible.

However, it should not be forgotten that replacement of the processing head actuator includes changes in the control algorithm. Obviously, the CNC unit has to know about each possible processing head actuator in advance and a solid system with hierarchical control will be obtained again.

Therefore, it is necessary to determine the foundational part of the architecture and to define specifications as for communication protocol as well as for creating new software and hardware modules that can be dynamically plugged into the system. Subsequently, a foundational part will include a motor control algorithm of coordinate tables, and the rest of it will be external components.

In order to implement such architecture, a distributed network with software and hardware modules have to be developed. All modules are connected to the network, along with a connection that can be both wired and wireless. Each module gets an IP-address via DHCP and then sends a broadcast package to find a supervisor. The supervisor replies using a predetermined protocol and record the modules main physical parameters in a supervisor registry. For instance, a list of commands it can receive and data it sends to the coordinate table controller.

Such an approach provides the opportunity to change the paradigm of a production control environment completely. Unfortunately, it is not all that easy. The CNC system is not only a control algorithm, but a database, user interface, and physical environment communication component (also known as "embedded systems" and "microcontrollers"). To implement such system, a system approach is required and the authors suggest the use of a microservices architecture pattern.

III. State-of-the-art in service-oriented architecture

Service-oriented architecture (SOA) is a rapidly developing concept, which has increasingly been used in modern distribution systems. The basis of this architecture is a set of replaceable, low coupling components that possess unified interfaces for standard protocol communication. SOA was first mentioned in 2009 and used as a framework for complex web applications, aggregating data from multiple sources and representing it in a single view. Google maps, Amazon, and eBay are the most well-known examples. Consequently, SOA became widely used in cloud-based services such as SaaS, PaaS, IaaS, etc [7].

Microservice architecture (MSA) is a modern SOA interpretation that is applied to distribution systems. The service is a process executed by an operation system and communicates with another process via a network interface to reach a common goal. Microservices can be built using any programming language and framework but should interact with each other via a single protocol. Moreover, each microservice works with a limited set of tasks and performs a minimum number of operations.Currently, there is no clear definition of MSA. However, we attempt to pick out the key features of the approach that are most important for the development of the distribution CNC systems [8].

MSA provides the opportunity to easily replace the modules included in the system. For example, HMI for industrial equipment can be considered where a conventional HMI is a set of the physical elements, such as indicators, buttons, switches, etc. This interface is really convenient for mass production when equipment is used as a part of the conveyor line. Although, for a fabrication laboratory (FabLab) such a way of control is overhead. Using the MSA makes it easy to replace the physical controller with a virtual one that will enable the user to operate the equipment through the web-interface from any device connected to the network.

All modules are organized around functions. MSA allows you to split the functionality of each block. For example, the milling head consists of a physical part—motor, collet, controller, sensors, and a logical part—control unit. Physically, both parts are on the same block because the control unit has its inner memory (containing low-level and high-level logic) and can be connected to the consolidated network [9].

In terms of MSA, it is two different services. One of them is in charge of low-level motor control commands, data acquisition, etc. The other is a high-level HMI by which the user can control the settings of a production process or develop programs in ISO-7bit programming language. Each of these services is autonomous and has its own algorithm.

On the other hand, each of them is in charge of an exact function and know nothing about other microservices implementation. Certainly, duplication is possible as the same function can be implemented in different microservices and in different ways. However, it does not violates the low coupling

principle and is mostly considered as code writing discipline. MSA provides the opportunity to connect modules written in different languages seamlessly, but it doesnt forbid the unification of programming tools.

Each microservice is elastic and easy to modify, and along with that, it is a complete program product. This means that developers need to stick to cohesion and coupling principles during CNC system development. Conversely, it helps to focus on development and debugging of a particular block, which allows the modification of the entire system behaviour.

Increasingly, publications describing microservices architecture application for reconfigurable production systems are appearing. For instance, da Silva [10] represent architecture for reconfigurable production systems based on MSA and an agent-holonic approach. The suggested system supports knowledge exchange in a heterogeneous production system and provides the opportunity to control equipment placed on geographically remote production areas. The paper demonstrates that the architecture allows such a system to be fault-tolerant and reliable even in difficult production conditions.

Furthermore, Vresk [11] and Butzin [12] describe MSA that they use to increase interoperability and scalability of IoT. It is commonly known that this conception is associated with the notion of Industry 4.0. Therefore, a represented approach can be used to CNC systems development. Based on these examples, we can conclude that MSA is a modern, rapidly developing trend of CPPS, which can be successfully employed for creating a CNC system.

IV. Communication protocol

As previously stated, simple and open MSA has to apply lightweight communication protocol. Because distributed network, that is a basis of the approach, is heterogeneous, it is necessary to use “smart” message receivers and “silly” communication channels. Such a decision gives the opportunity to make system modules more independent and focused on the exact task. Also, it is important to remember that network performance can be obtained using a lightweight message bus and reducing the amount of communications between system components.

It appears that the most obvious way of data exchange in TCP/IP networks is “raw” sockets. Bytes data flow can be transmitted over this interface, which is pretty convenient for many applications. However, all modules in a developed system deal with structured data so packages will be more appropriate. Each package is a message from one module to another. A receiver module, that receives a message, has to confirm it (if content is comprehensible and can be handled) or reject it.

Though such messages are quite similar to those that are used in XML-RPC, SOAP, BPEL and WSDL, an important difference is showing up. All these protocols are used in enterprise applications and are inapplicable for embedded systems. To create a lightweight open and extensible control system necessary to minimize the abstraction layer and combine low-level sockets with routing ability. Thus, the most appropriate communication way between CNC system modules is message queues [13].

A message queue is an asynchronous communication protocol, so the receiver and sender dont communicate directly and use a message queue instead. Generally, message queues have a message size constraint so that data is transmitted via one message or more. Following message queue features allow us to conclude that this way to transmit data is the most efficient for designing a CNC system for several reasons [14]:

- Message queues allow system components to stay most independent from each other and avoid possible deadlocks when one of the participants has to wait for resources to release that are necessary for data transmission.
- Message queues give the opportunity to save system resources because there is no need for network buffers where untransmitted and unhandled data is usually stored. In fact, the message queue is a multipurpose network buffer on its own, which does not depend on any node or process.
- Message queues possess of feasible volume and throughput scalability.
- Message queues smooth network load peaks. It is considerably important for the designed system due to a possibility of different traffic types mixing together. In particular, traffic can be divided into high priority (sensors data, control commands) and low priority (G-code loading, a video stream from processing area) traffic according to suggestions in *Section VI* CNC system. The vehicle control system uses two independent CAN buses (high-speed for sensors and control commands and low-speed for comfort functions) to solve this problem, which doesnt simplify its architecture.
- Message queues improve system fault-tolerance as messages stay in the queue and can be handled even if the transmitter node fails. A g-code program can be considered as an example. An operator creates a g-code program using interface block and then transmits it to the execution block as text command flow. Obviously, the program will be divided into several messages placed in the queue. If the interface block freezes, a watchdog commits that fact and initiates operation system reboot. However, during the reboot execution the block keeps handling messages from the queue and data transmitted process is not interrupted.
- Message queues guarantee message delivery, at least until one node is active and can handle it.
- Message queues do not violate messages order. As a rule, messages can be received in the order they were sent.

Currently, there is a sufficient amount of various implementations of described approach [15–18]. All of them can be divided into commercial (proprietary) and open source (free license). Among commercial implementations the following frameworks and libraries should be noticed: IBM WebSphere MQ, Oracle Advanced Queuing, Amazon Simple Queue Service (SQS), StormMQ, IronMQ. Along with these, message queues are implemented in popular commercial real-time operating systems such as (RTOS) QNX and VxWorks.

The most interesting open source libraries are Apache ActiveMQ, Apache Kafka, Apache Qpid, Beanstalkd, HTTPSQS, JBoss Messaging, JORAM, RabbitMQ, Tarantool, Celery, HornetQ, StormMQ, NATS Messaging, ZeroMQ, nanomsg, Apollo, Darner, Gearman, RestMQ.

Only open source tools were analysed as the developed system is an open one. The following requirements based on analysis were established:

- The library has to be written in C/C++ because the designed system contents components are based on microcontrollers where programs can be written only in these languages.
- The library has to provide access to the message queue without a broker—a special node for addressing, routing, and queuing. This component reduces system fault-tolerance because a broker is a single point of failure, which is unacceptable for CNC systems. Certainly, redundancy can be provided by having additional reserve brokers. This, however, completely negates the main advantage of the system with the broker—reducing the complexity.
- There have to be an open API to add new transport protocols. Most of the message queues are based on TCP and websockets. In developed system connection between interface block and execution block established via UART. Therefore, a message queue has to be adapted to that transport protocol.

Given the above requirements, the most appropriate message queue library is *nanomsg*. This library is written in pure-C without any dependencies. Nanomsg is quite compact and allows developers to use it in embedded systems as well as in POSIX-compatible systems.

At the moment, the library implements following communication patterns:

- PAIR—Simple one-to-one communication.
- BUS—Simple many-to-many communication.
- REQREP—Allows building clusters of stateless services to process user requests.
- PUBSUB—Distributes messages to large sets of interested subscribers.
- PIPELINE—Aggregates messages from multiple sources and load balances them among many destinations.
- SURVEY—Allows querying the state of multiple applications in a single go.

The developed system uses PUBSUB for broadcast messages and BUS for decentralized exchange between system blocks (Fig. 1). Data in these queues, which are based on nanomsg, is transmitted as BLOB. Therefore, *Messagepack* (a binary format that is compatible with JSON and packs data up to 15–20 % more efficiently) is applied to transmit database entries [19], [20].

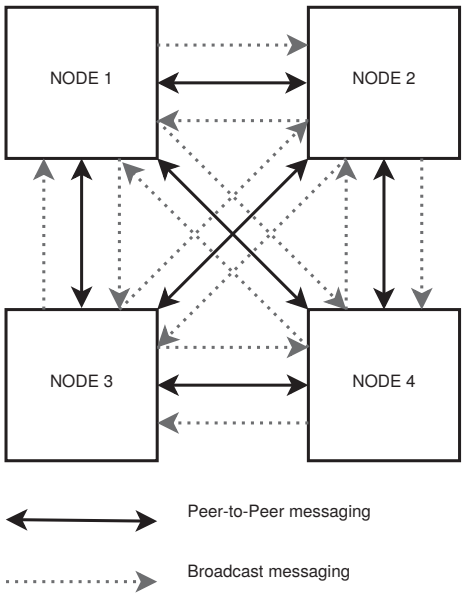

Fig. 1. PUBSUB and BUS communication patterns

V. Distributed database

For the effective realization of storing data in a decentralized network, which is the core CNC-equipment control system, it is necessary to solve the scaling problem. Obviously, the described open architecture does not allow using traditional relational databases. Despite the fact that the current relational databases support replication and scaling, similar solutions still do not provide the proper data accessibility and consistency.

That is why using the so-called NoSQL databases is most appropriate. The main features of such databases are:

- Availability of data, which means that each call to database is guaranteed to return the requested data or reports that it is impossible to obtain the data.
- The system can change its state even without new input data because state changing can be necessary to reach the consistency of data. NoSQL databases do not have the database schema that contains descriptions of the content, structure, and data integrity constraints, so at some moment the data integrity may be compromised and then automatically restored.
- All data remains consistent, a system does not check all the data, i.e. it does not follow the integrity of each transaction, which improves performance and availability.

Currently, there are three basic NoSQL paradigm realizations. The first, *"key-value" data storages*, which are in fact ordinary hash-tables. They are usually used as an alternative to the distributed file systems and can be used for creating various file caches as well as problems with "Big Data". They often keep data in temporary storage distributed in random access memory that gives a serious performance boost. The following systems have the same solution: Dynamo, Riak, Berkeley DB, FoundationDB, HyperDex, InfinityDB, LMDB, MemcacheDB, and Redis [21–23].

The second, *document-oriented databases*, which represent hierarchically structured data storages. In fact, these are databases based on the JSON standard which makes them especially suitable for use in web applications. The main representatives of this category are CouchDB, MongoDB and eXist.

The third, *graph databases* are used for presenting data with a large number of connections. It is the most suitable for social network organization. Examples of this type of database are Neo4j, OrientDB, AllegroGraph and InfiniteGraph.

Also, it should be noted that other NoSQL databases design approaches exist but they are a logical development of the three basic branches or a combination of them. Obviously, micro service architecture of CNC-equipment control system realization needs a database with advantages combined from document-oriented databases and "key-value" storages. That is why the following requirements were formulated:

- Serverless realization. Many of the considered NoSQL databases have a dedicated server, which organizes interaction and replication. The presence of a server complicates the inclusion of new modules of the management system, as it requires certain actions related to server administration.
- Availability of atomicity during data transfer and transaction support that is necessary to create a system that works in "soft real-time" conditions.
- The possibility to keep all data in one file without having to create temporary files. The designed system suggests the possibility of connecting modules with different capacities and internal storage capacity so the database should use as little resources as possible.
- Cross-platforming and unification on the API level. Systems realized on C/C++ languages are preferred.
- Open-source software project.

According to the criterion, a comparative analysis was made, which resulted in the onset of merits and demerits and the UnQLite database was selected. This system is a NoSQL serverless database. In contrast to the well-known NoSQL engines like MongoDB, Redis, and CouchDB, UnQLite requires no installation or configuration and does not start a separate process to access the data. All data is stored in a single file with standard JSON serialized data. Otherwise, UnQLite is a storage for key-value pairs with cursor support and keeping data in HDD as well as RAM possibility.

In addition to this, UnQLite can work as document storage and support the transactions. The UnQLite engine is fully programmed in ANSI C language which allows using this database even on embedded systems used in a project. In the designed project, UnQLite will be used for storing structured data (modules, related to the implementation of the user interface of the controller and complex logic), as well as rapid storage of data obtained from the sensors (storage type "key-value"). In the last mode, UnQLite supports transactions, and the data stored in this way are best suited for the rapid transmission through a message queue

VI. Application example

Let us consider the implementation of the proposed CNC controller microservice architecture as an example of an apparatus designed for selective photopolymer curing. This equipment uses laser processing technology the same as LDF (Laser Direct Structuring) process. The experimental installation represents a device for arbitrary surfaces processing by laser emition. The equipment contains a fixed coordinate table (Fig. 2) where processing the object takes place, a laser head (Fig. 3) is placed over the work table and is movable in a horizontal plane along two coordinates with a laser emitting source (laser module) connected optically with the laser head.

The laser head has a gyro-stabilized suspension system based on the modified Stewart platform. Linear piezo motors are used for platform moving. The suspension system allows the laser beam deflection on two self-perpendicular surfaces and focus distance changing by the head objective moving along Z-coordinate. This allows performing the following tasks. The first task is the laser ray is focused at the processing point. The second task is complex surfaces

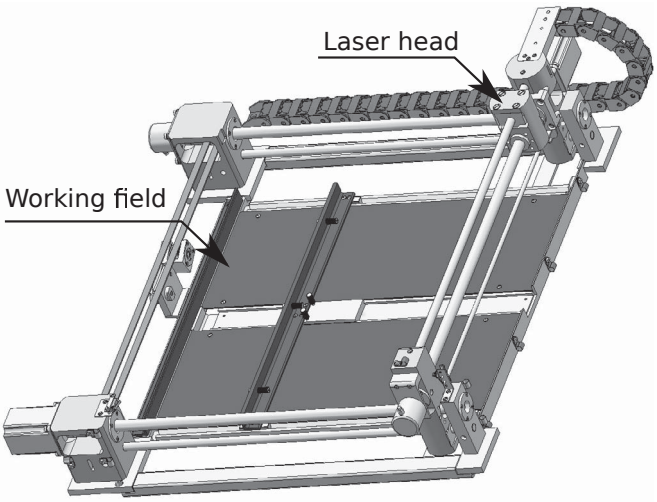

Fig. 2. Coordinate table

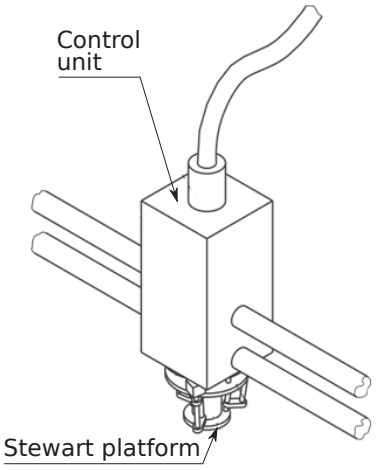

Fig. 3. Laser head

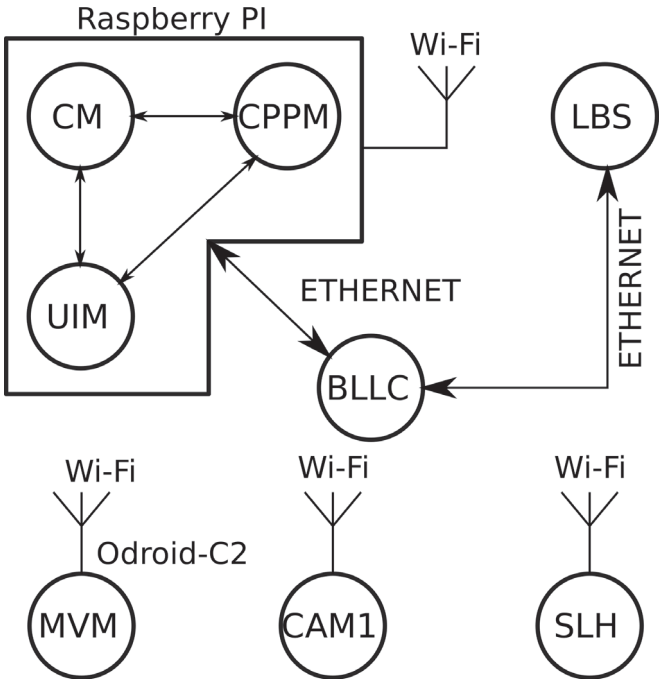

Fig. 4. Equipment control system

processing availability by optical axis deflecting. The slope allows the laser beam to always remain perpendicular to the processed surface and respectively contact patch is not distorted. The third task is active vibrations compensation. High-precision single-axis solid state accelerometers and a camera with a telescopic objective, for real-time spot form checking, are used to record vibrations. In addition, this laser head is equipped with the second camera for automatic setting of a reference point and watching the processing area.

A. Equipment control system

The equipment control system contains (Fig. 4):

Communication Module (CM). Physically it looks like the microcomputer Raspberry Pi with the operating system, Debian Linux, installed on it. This modules functions are tasked with the recording and dispatching of the other modules (fault tolerance of system at all providing).

Base Low-Level Controller (BLLC). Physically it is a control module based on the Cortex-M4 microcontroller with the real-time operating system, FreeRTOS. The main BLLC appointment is to generate control signals for coordinating table actuators and to process sensors signals. The controller is fully self-contained and able to work even without connection to the main control network. This module obtains programs in ISO 7-bit (G-code) programming language as an input data. G-code is loaded by nanomsg protocol, and Ethernet is used as physical level protocol. Also by this protocol, BLLC sends to the net the data from sensors, which can be taken and interpreted by every node of the control network. The rest of the unit is completely independent and after G-code loading and processing, it works in standalone mode. The the work algorithm of BLLC processes all contingencies by itself even without connection with the control network. In addition, it registers information about all the events that BLLC transmitted to the network and is recorded in the event log.

In addition to the control program, BLLC can receive single control signals, for example, “move the carriage to point (X; Y)”, “move along X-axis with the velocity V”, and “start the self-calibration process.” These commands are needed to interact with the operator of the apparatus, and for automatic zero point (reference point) search procedure initiated by the machine vision module (MVM). Additionally, the low-level controller has a debug port, USART, which allows the operator to interact with the controller by command line interface. This function is necessary in the control system debug stage and in case of contingency appearances because of faults in the network. BLLC also transmits to the network the control signal for the laser beam source (LBS). At the point of view of G-code, a laser beam is just a tool like a cutter or a chisel. The controller can send the command to turn the laser on or off or to set the emitting capacity (tool rotation speed analog). There is bilateral data exchange between BLLC and LBS. BLLC sends one of the control signals, LBS gets it and sends to BLLC readiness confirmation and changes the laser state afterwards. In case of failure or another contingency, BLLC takes an appropriate answer from LBS. At the same time, this signal is obtained by other control network components, and all systems turn to the recovery mode. The same happens if the connection with the LBS is lost. The LBS has internal storage to accommodate a database that keeps an event log, CNC programs and user interface components.

Laser Beam Source (LBS). It has the same architecture as the BLLC part, i.e. it is based on a Cortex-M4 microcontroller with the operating system, FreeRTOS, on it and it is also connected to the network. As already mentioned, LBS functions are turning the laser on and off (it is carried out by changing the position of the movable mirror), changing its power (by PWM) and receiving and processing the data from the sensors. LBS has an internal logic and can work off contingencies in standalone mode and transmit sensor data to the network. In the case of failure, LBS transmits an emergency stop and recovery mode signal. Despite the fact that the LBS has the slave role towards BLLC, it constantly interrogates it. In the case of an absence of an answer for some time, it also distributes an emergency stop signal. It turns out that these two network nodes produce a constant mutual monitoring that increases the resiliency of the whole control system. LBS has internal storage to accommodate a database that keeps an event log, CNC programs and user interface components.

Smart Laser Head (SLH). SLH is a software and hardware module based on a Cortex-M4 microcontroller with the operating system, FreeRTOS. The main functions of SLH are: mechanical vibrations compensation, the inclination of the optical axis adjustments for complex surfaces processing and laser beam self-focusing. As the others, the module is autonomous and independently handles all emergencies with the transmission of the corresponding signals in the network. It also has internal storage to accommodate a database that keeps an event log, CNC programs and user interface components.

Machine Vision Module (MVM). It is a software and hardware module based on the Odroid-C2 microcontroller with operating system, Debian Linux. MVM functions are: search for a reference point on the workpiece and transfer the video

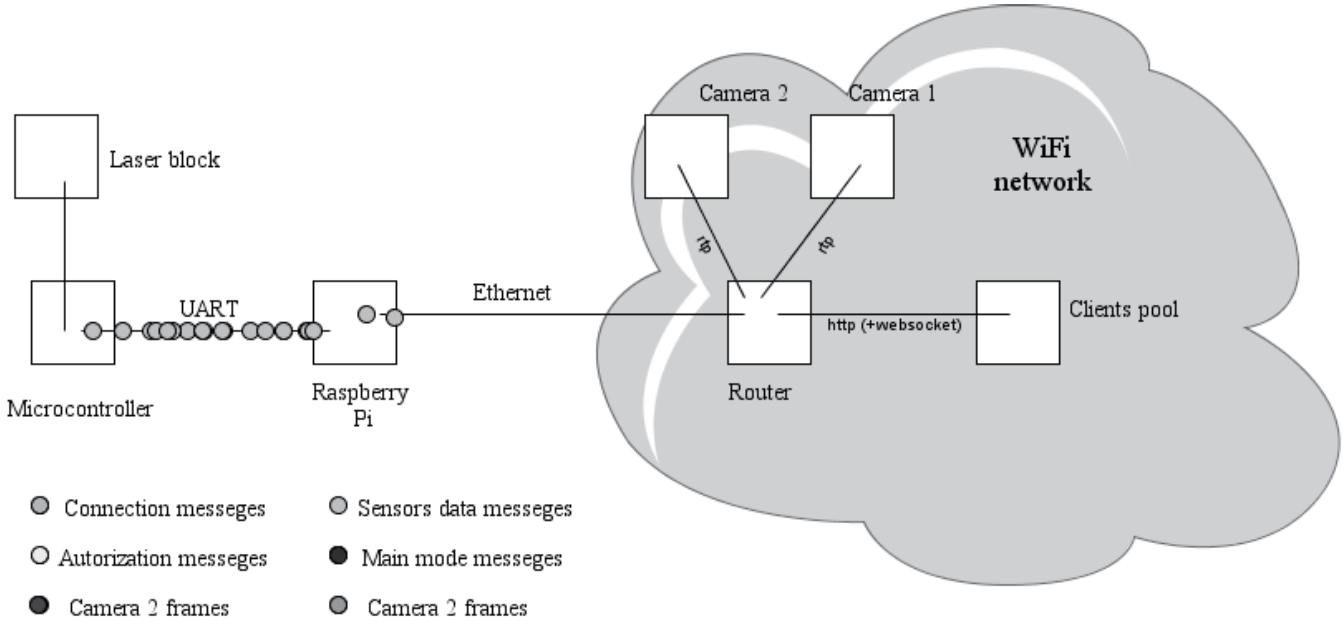

Fig. 5. AnyLogic model graphic representation

stream to display on the operator interface. BLLC and MVM interact by this algorithm: in front of the CNC program running on BLLC execution it asks for MVM coordinates of the reference point, MVM starts to scan the workspace (by sending control commands to BLLC) and, when the reference point is found, MVM turns into command waiting mode. MVM failure is critical only in case of searching a workpiece zero point process, in other cases, CM just alerts the operator that MVM is not available, but processing does not stop. The MVM also has internal storage to accommodate a database that keeps an event log, CNC programs and user interface components.

User Interface Module (UIM). It is a software module and is an aggregator for unit control elements (displays and widgets) to display them in the browser. Control elements are automatically loaded into it during the interaction with the modules. The basic idea is that the entire interface logic is written using a technology stack of HTML5 + ECMAScript + CSS, and stored directly in the database of each of the modules. During initialization of apparatus, all these static files are loaded to UIM, and then transmitted over the network for display in the browser. From the viewpoint of realization, UIM is a high-performance multi-threaded web-server written in Go programming language, and a display library of control units (buttons, switches, sliders, text fields, etc), widgets and displays. This server is physically placed on the same microcontroller with CM. However, it should be noted that due to microservice architecture, it could be located on a single computer or server.

Control Programs Preprocessing Module (CPPM). It is used for the preparation of G-codes directly in the browser. Like hardware modules interfaces, it is a web-application dynamically included in common equipment interface by UIM. It is physically placed on the same microcontroller as the CM and the UIM, and as the latter can be put on a separate server.

VII. Simulation and numerical results

It is well known that every distributed system has one significant disadvantage—its productivity depends on data throughput. In the case of complicated topology, bottlenecks can emerge, which in turn may limit the overall bandwidth of the network or at the very least will create load imbalance as a result of increased contention across the slow link. Obviously, the reviewed distributed microservice architecture is completely adequate for the demand of self-sufficiency for all modules. All operations connected with the manufacturing process are executed only in real-time mode and are not dependent on network delays. Nevertheless, all modules are connected to each other, hence when delays occur, the total time of operations such as initialization, jump from one control program to another or interacting with the operator can increase. Consequently, the network simulation model was designed and main modules load analysis was carried out.

The analysis considers that the reviewed network is heterogeneous, being composed of segments, that uses different data transfer mediums (Wi-Fi, or cable connection) and different transport protocols (TCP, RVDS). To calculate the network load metrics queueing theory was used. HTTP protocol also CPU-intensive load by most critical modules were used instead of absolute values of communication channel capacity and network delays.

This model is simplified, and not accounting for the CPPM and LBS parameters calculation, as soon these modules are working effectively in stand-alone mode and consume no network capacity. It should be noted that a Wi-Fi router is included in the model. This component is not a part of reviewed microservice architecture, but it has an influence on the network capacity. For the network architecture modelling simulation environment AnyLogic was used. The model graphic representation is pictured in Fig. 5. As the simulation of network latency in a decentralized network was the main

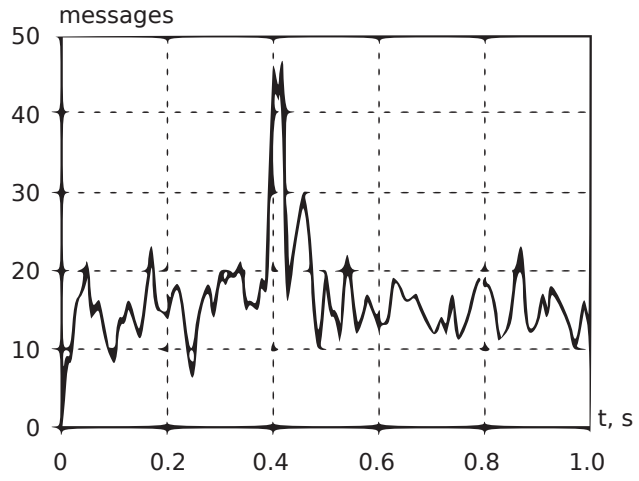

Fig. 6. Starting the system

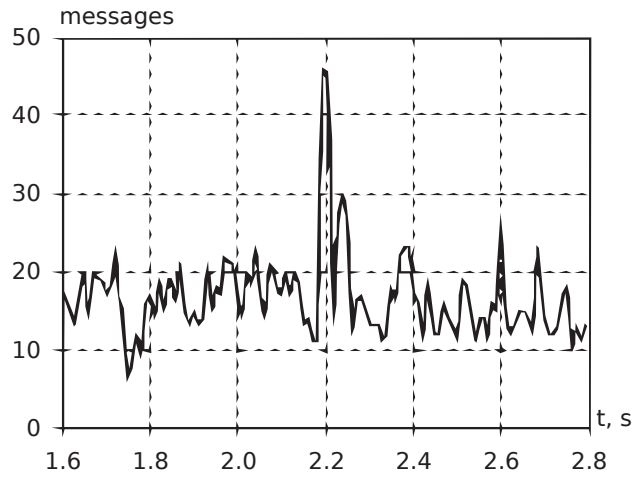

Fig. 7. Connecting the first client

purpose it is represented as entity flow with delays in the nodes. Therefore, discrete event simulation was the most appropriate. What is more, Anylogic allows to combine several approaches in one simulation, but in this case, discrete event approach was sufficient.

A number of experiments with various network parameters was carried out. Not surprisingly, the biggest network load takes place during control system initialization (turning the machine on). Initially, all modules statuses are polled and sensors data is transmitted. Experiments showed that main traffic is generated by two cameras (MVM and SLH), and a BLLC as well, which transmits all its current parameters (state and coordinates) all the time during the processing.

The network load can be increased when a number of users interacts with the system. Dealing with classical industrial management systems, it is hard to produce a case when more than one operator works with the apparatus. However, when equipment included in the integrated CPPS, this pattern changes. An operator as well as a dispatcher or technologist can work with the apparatus. It should be kept in mind, that the SCADA system can work as a client with a number of connections.

In Fig. 6 the diagram of the relation between time and network message amount is pictured. In this diagram, the first second of the control system activity is presented. From the diagram it is apparent that for 400 ms the number of messages in the network is smallest, then a short-duration peak occurs. This peak demonstrates that at this moment all modules have completed the self-checking procedures and the negotiation process is started. Upon completion, the network turned to the standby mode waiting for the client connections.

In Fig. 7 the process of connecting the first client is pictured. Obviously, a number of messages related to the webserver, which transmits to client static files, containing the user interface description. In Fig. 8 the process of reference point search is pictured. It can be noted that the MVM camera generates traffic during the whole workpiece reference point search. Fig. 9 demonstrates a change of the network loading during client connection to the camera, which translates the video stream from the working zone. Video traffic is transmitted from the camera and through the UIM module is delivered to users. The load on the network is not reduced as long as at least one user receives the signal from the camera.

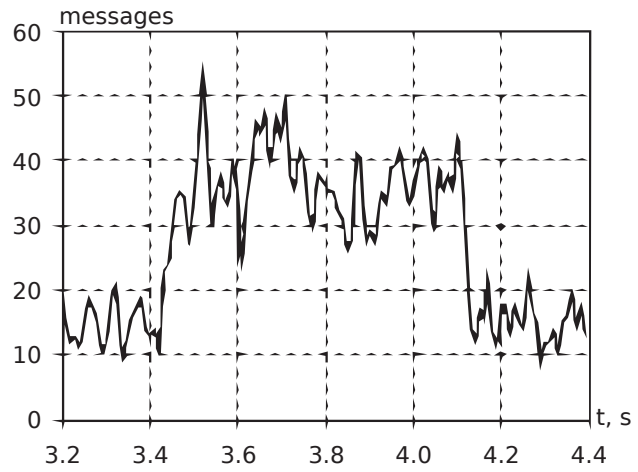

Fig. 8. Searching of the reference point

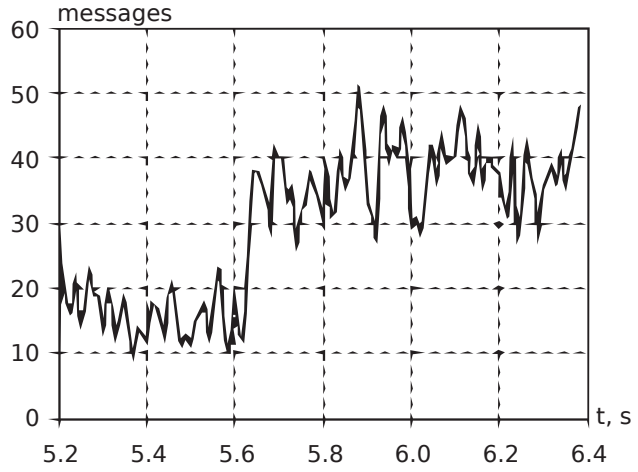

Fig. 9. Video stream from the working zone

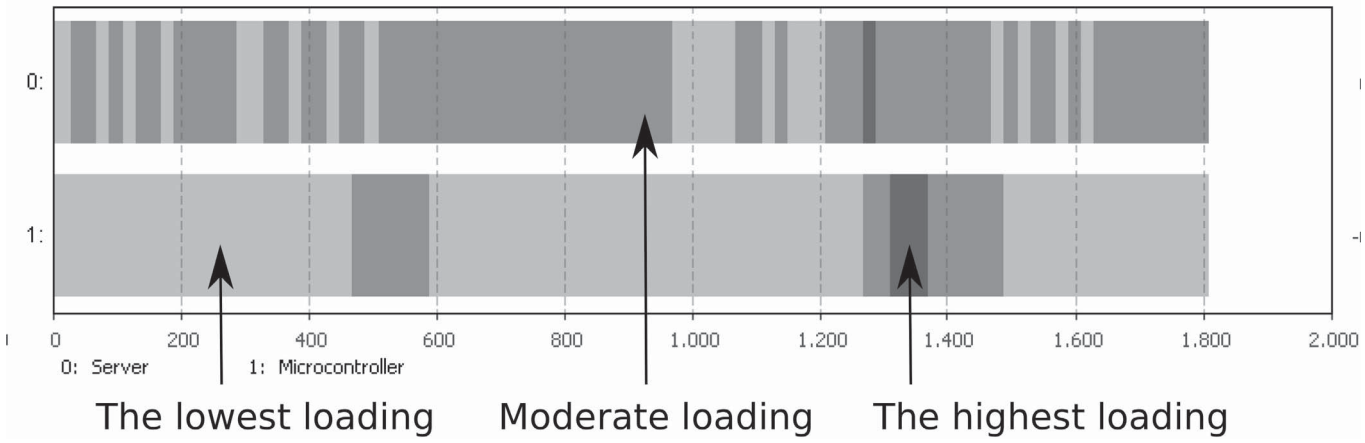

Fig. 10. Server and microcontroller loading

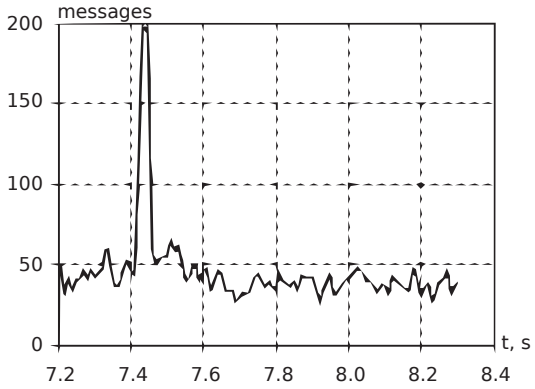

Fig. 11. Network maximum loading

TABLE I. CPU LOAD AS A FUNCTION OF A NUMBER OF CLIENTS

Number of clients	CPU load, %	
	Server node	Microcontroller node
1	64.5	13.6
2	68.5	15.2
3	71.6	30.8
4	69.1	39.8
5	74.9	43.7
6	73.0	50.3
7	74.9	50.8
8	75.7	56.5
9	80.2	57.3
10	83.3	59.8

Fig. 10 shows a time colour chart. There are two bars that represent server and microcontroller nodes and their states per time unit. The chart describes the BLLC CPU load and also Raspberry Pi microcomputer load, on which CM, UIM, and CPPM are installed.A network load test was carried out as well. We increased the number of clients to determine the maximum loads for the network, microcomputers CPU, and BLLC. The testing proved, that despite the increasing the number of clients, only a short peak of the network messages occurs (Fig. 11), and afterwards it turns to standby mode, while the microcomputer load increases up to $\sim 85\,\%$ and decreases no more (Table I). Thus, the number of clients simultaneously connected to the network, should not exceed ten.

VIII. CONCLUSION

In this paper a new approach to design architecture of modular-type CNC-driven industrial equipment was reviewed. As the basis of the developing system use of a distributed communication network where each node is a microservice is proposed. Due to its interoperability, such a system can be easily integrated into a cyber-physical manufacturing environment. Analysis of the distributed software with microservice architecture as well as system components communication and databases was carried out. As the main communication protocol, the system providing data transmission by using the message queue was chosen.

Such an approach makes the modules as autonomous as possible and simplifies the scaling, also providing an opportunity of network load leveling and provides a fail-safe feature by guaranteed message delivery. Protocol software implementation is based on the nanomsg library; broadcast messages and peer-to-peer messaging are used.

Data is stored in the system by using the distributed NoSQL database, UnQLite. This database system combines the storage advantages of the type “key-value” and document-oriented databases. The system is created as a useful library and using the similar data storage interface that easily integrates by the nanomsg protocol.

Realization of the proposed protocol is exemplified by an apparatus for selective polymers curing on a free-form surface by laser emission. All system modules and their interoperation algorithms are meticulously described. The advantages of utilizing a distributed microservice architecture in comparison to ordinary monolithic solutions are shown.

Simulation and numerical modelling of the distributed network supplies communication of all apparatus modules were done. The experiment showed that the highest network load occurs during system initialization as well as searching the reference point, but that these peaks are of temporary nature and do not significantly influence the modules during the polymer curing process. General modules load-testing was also carried out and shows that the number of clients connected to the system via HTTP protocol should not exceed ten.

REFERENCES

[1] S. N. Grigoriev and G. M. Martinov, "Research and development of a cross-platform CNC kernel for multi-axis machine tool," *Procedia CIRP*, vol. 14, pp. 517–522, 2014, 6th CIRP International Conference on High Performance Cutting, HPC2014.

[2] "A research on open CNC system based on architecture/component software reuse technology," *Computers in Industry*, vol. 55, no. 1, pp. 73–85, 2004.

[3] X. Ma, Z. Han, Y. Wang, and H. Fu, "Development of a PC-based open architecture software-CNC system," *Chinese Journal of Aeronautics*, vol. 20, no. 3, pp. 272–281, 2007.

[4] "Open-architecture system based on a reconfigurable hardware/software multi-agent platform for CNC machines," *Journal of Systems Architecture*, vol. 56, no. 9, pp. 407–418, 2010.

[5] N. Verba, K.-M. Chao, A. James, D. Goldsmith, X. Fei, and S.-D. Stan, "Platform as a service gateway for the Fog of Things," *Advanced Engineering Informatics*, 2016, to be published.

[6] C. Prazeres and M. Serrano, "SOFT-IoT: Self-organizing Fog of Things," in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, March 2016, pp. 803–808.

[7] "Chapter 2—service-oriented architecture and web services," in *Service Computing*, Z. Wu, S. Deng, and J. Wu, Eds. Boston: Academic Press, 2015, pp. 17–42.

[8] M. Rafighi, Y. Farjami, and N. Modiri, "Studying the deficiencies and problems of different architecture in developing distributed systems and analyze the existing solution," in *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Nov 2015, pp. 826–834.

[9] A. W. Colombo, S. Karnouskos, J. M. Mendes, and P. Leitao, "Chapter 4—industrial agents in the era of service-oriented architectures and cloud-based industrial infrastructures," in *Industrial Agents*, P. Leitão and S. Karnouskos, Eds. Boston: Morgan Kaufmann, 2015, pp. 67–87.

[10] R. M. da Silva, F. Junqueira, D. J. S. Filho, and P. E. Miyagi, "Control architecture and design method of reconfigurable manufacturing systems," *Control Engineering Practice*, vol. 49, pp. 87–100, 2016.

[11] T. Vresk and I. Čavrak, "Architecture of an interoperable IoT platform based on microservices," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2016, pp. 1196–1201.

[12] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the Internet of Things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–6.

[13] A. E. Rheddane, N. D. Palma, A. Tchana, and D. Hagimont, "Elastic message queues," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 17–23.

[14] F. Reid, "Chapter 15—message queues," in *Network programming in .NET*, F. Reid, Ed. Burlington: Digital Press, 2004, pp. 419 – 452.

[15] N. Estrada and H. Astudillo, "Comparing scalability of message queue system: ZeroMQ vs RabbitMQ," in *2015 Latin American Computing Conference (CLEI)*, Oct 2015, pp. 1–6.

[16] V. C. Barroso, U. Fuchs, and A. Wegrzynek, "Benchmarking message queue libraries and network technologies to transport large data volume in the ALICE O system," in *2016 IEEE-NPSS Real Time Conference (RT)*, June 2016, pp. 1–5.

[17] V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, Sept 2015, pp. 132–137.

[18] K. Vandikas and V. Tsiatsis, "Performance evaluation of an IoT platform," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, Sept 2014, pp. 141–146.

[19] K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, May 2012, pp. 177–182.

[20] S. Popić, D. Pezer, B. Mrazovac, and N. Teslić, "Performance evaluation of using protocol buffers in the Internet of Things communication," in *2016 International Conference on Smart Systems and Technologies (SST)*, Oct 2016, pp. 261–265.

[21] G. W. Poerwawinata and A. I. Kistijantoro, "Memcachedb persistent implementation using leveldb," in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, Aug 2015, pp. 283–287.

[22] D. Stjepanovic, M. Savic, J. Jokić, and S. Marić, "Performance measurements of some aspects of multi-threaded access to key-value stores," in *2015 23rd Telecommunications Forum Telfor (TELFOR)*, Nov 2015, pp. 831–834.

[23] S. Wen, H. Zhang, and Y. Zhang, "A construction and implementation of high performance embedded database," in *2011 International Conference on Electrical and Control Engineering*, Sept 2011, pp. 4238–4241.