10-2023

Machine Tool Communication (MTComm) Method and Its Applications in a Cyber-Physical Manufacturing Cloud

S M Nahian Al Sunny
University of Arkansas-Fayetteville

Machine Tool Communication (MTComm) Method
and Its Applications in a Cyber-Physical Manufacturing Cloud

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering
with a concentration in Computer Engineering

by

S M Nahian Al Sunny
Bangladesh University of Engineering and Technology
Bachelor of Science in Electrical and Electronic Engineering, 2014

July 2020
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

_____
Xiaoqing "Frank" Liu, Ph.D.
Dissertation Director

_____
Miaoqing Huang, Ph.D.
Committee Member

_____
Qinghua Li, Ph.D.
Committee Member

_____
Alexander H. Nelson, Ph.D.
Committee Member

_____
Wenchao Zhou, Ph.D.
Committee Member

ABSTRACT

The integration of cyber-physical systems and cloud manufacturing has the potential to revolutionize existing manufacturing systems by enabling better accessibility, agility, and efficiency. To achieve this, it is necessary to establish a communication method of manufacturing services over the Internet to access and manage physical machines from cloud applications. Most of the existing industrial automation protocols utilize Ethernet based Local Area Network (LAN) and are not designed specifically for Internet enabled data transmission. Recently MTConnect has been gaining popularity as a standard for monitoring status of machine tools through RESTful web services and an XML based messaging structure, but it is only designed for data collection and interpretation and lacks remote operation capability.

This dissertation presents the design, development, optimization, and applications of a service-oriented Internet-scale communication method named Machine Tool Communication (MTComm) for exchanging manufacturing services in a Cyber-Physical Manufacturing Cloud (CPMC) to enable manufacturing with heterogeneous physically connected machine tools from geographically distributed locations over the Internet. MTComm uses an agent-adapter based architecture and a semantic ontology to provide both remote monitoring and operation capabilities through RESTful services and XML messages. MTComm was successfully used to develop and implement multi-purpose applications in in a CPMC including remote and collaborative manufacturing, active testing-based and edge-based fault diagnosis and maintenance of machine tools, cross-domain interoperability between Internet-of-things (IoT) devices and supply chain robots etc. To improve MTComm's overall performance, efficiency, and acceptability in cyber manufacturing, the concept of MTComm's edge-based middleware was introduced and three optimization strategies for data catching, transmission, and operation

execution were developed and adopted at the edge. Finally, a hardware prototype of the middleware was implemented on a System-On-Chip based FPGA device to reduce computational and transmission latency. At every stage of its development, MTComm's performance and feasibility were evaluated with experiments in a CPMC testbed with three different types of manufacturing machine tools. Experimental results demonstrated MTComm's excellent feasibility for scalable cyber-physical manufacturing and superior performance over other existing approaches.

ACKNOWLEDGMEMTS

First of all, I would like to say is all praises and thanks are due to the Almighty Allah, for blessing me with this opportunity and providing me the strength, perseverance, and knowledge to pursue this arduous journey and see it through to the end successfully.

I would like to express my humblest and sincere gratitude to my advisor and mentor, Dr. Xiaoqing “Frank” Liu, for introducing me to the emerging and fascinating world of cyber manufacturing and also for his continuous guidance, endless patience, and meticulous supervision throughout my doctoral journey. His constructive advice and invaluable critiques enabled me to progress my research in a timely manner. I would also like to thank other members of my dissertation committee, Dr. Miaoqing Huang, Dr. Qinghua Li, Dr. Alexander H. Nelson, and Dr. Wenchao Zhou, for providing me valuable feedback during my dissertation proposal presentation that assisted me to identify key areas of my research requiring improvement and advance in the right direction.

This research was supported by National Science Foundation Grant CMMI 1551448 entitled "EAGER/Cybermanufacturing: Architecture and Protocols for Scalable Cyber-Physical Manufacturing Systems". I am very thankful for this award of funding and financial support.

I would like to acknowledge my fellow lab mate and research colleague, Md Rakib Shahriar, for his cooperation in my research and for always being there to share my accomplishments and frustrations. His contributions in designing and developing our cloud system and conducting experiments as well as his critical insights, inspiration, and suggestions helped me immensely to complete my research. I am also very grateful to Asa Thacker, our undergraduate lab member from the Department of Mechanical Engineering, who provided

valuable support and knowledge during the initial phase of our testbed development. A special thanks to my friend and fellow graduate peer, Md Jubaer Hossain Pantho, for assisting me with his knowledge and perceptions in developing and evaluating the SoC FPGA hardware prototype.

Also, I welcome this opportunity to express my profound gratitude to my beloved parents, sister, and my caring wife for their love, sacrifice, and prayers. I wish to express the gratefulness beyond accountability to them whose continuous inspiration and moral boosting during every phase of this long and strenuous program kept me strong and determined. Whatever I am today, I entirely owe it to them.

Finally, I would like to thank the members of the Swiss folk metal band called “Eluvetie” for their extraordinary music which helped me immensely to maintain focus amidst the chaos of my surroundings, specially during the long hours of programming and preparing manuscripts.

DEDICATION

To my beloved parents, sister, and brother-in-law,

None of this would have been possible

without your prayers, encouragements, and strong belief in me.

The thought of your proud and smiling faces was the only thing that kept me going.

To my loving wife, Shady Afrin Jeesan,

For your love, patience, constant support, and most importantly,

for keeping me steady through the storms, over the hills, and across the valleys.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLISHED PAPERS

Chapter 2, 3, & 4: Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2018). Communication method for manufacturing services in a cyber–physical manufacturing cloud. *International Journal of Computer Integrated Manufacturing*, *31*(7), 636-652.

Chapter 3 & 4: Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

Chapter 5: Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2020, April). Development and optimization of an MTConnect based edge computing node for remote monitoring in cyber manufacturing systems. In *2020 IEEE International Conference on Fog Computing (ICFC)* (pp. 38-43). IEEE.

Chapter 6: Sunny, S. M. N. A., Liu, X., & Shahriar, M. R. (2018, July). Remote Monitoring and Online Testing of Machine Tools for Fault Diagnosis and Maintenance Using MTComm in a Cyber-Physical Manufacturing Cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 532-539). IEEE.

Chapter 7: Sunny, S. M. N. A., Liu, X., & Shahriar, M. R. (2019, July). An Integrated IoT Enabled On-Demand Grocery Shopping and Delivery Cloud System Using MTComm at the Edge. In *2019 IEEE International Conference on Edge Computing (EDGE)* (pp. 51-55). IEEE.

CHAPTER 1

INTRODUCTION

1.1 Background

For last few decades, the physical world has become more and more integrated with the cyber world as a consequence of tremendous advancements in digitization and information technologies. The rapid growth of the Internet has significant impacts on almost every aspect of human life. Manufacturing is such a domain that is being reshaped and revolutionized by emergence of several Internet based disruptive technologies. Historically, manufacturing systems have been considered to be rather rigid, as changes usually result in increased expenses and hence are mostly limited to organization and operation of physical resources directly related to production (Lu & Ju, 2017). The ISA95[1] model classifies manufacturing systems into five logically separated layers – lower level systems (level 0-2) consist of hardware and software for monitoring and controlling physical resources, while higher level functions (level 3 and 4) are responsible for process management, enterprise resource planning, information processing etc. As applications for lower and higher level systems are typically managed by different organizations, their integration is rather difficult. This results in a rigid architecture that is slow in responding to market or technological changes. Any alteration of such manufacturing systems, such as replacing old assets with new ones, generally leads to significant increase in costs and engineering efforts. However, this trend has been changing since the emergence of digital manufacturing which transforms manufacturing systems into agile and efficient eco-systems by integration of state-of-the-art information and communication technologies to enable rapid

[1] https://isa-95.com/

response to constantly changing demands and conditions in manufacturing lifecycle (Kulvatunyou, 2016; Lu & Ju 2017). Manufacturing is now becoming more and more digital because of the convergence of disruptive technologies such as industrial automation, cyber-physical systems (CPS), Internet-of-Things (IoT), Internet-of-Services (IoS), cloud computing, cybersecurity, big data analysis, artificial intelligence etc. (Liu & Xu, 2017; Alcácer & Cruz-Machado, 2019).

In 2011, the German government introduced an initiative for developing and promoting next-generation manufacturing systems called *Industrie 4.0* (or Industry 4.0), which is current considered as the fourth industrial revolution (Weyer et al., 2015; Baena et al., 2017; Lu, 2017; Wagner, Herrmann & Thiede, 2017). The term "Industrial revolution" refers to the emergence of technological breakthroughs that have revolutionized industrialization in general. Previous industrial revolutions were triggered by the invention of mechanical manufacturing facilities (end of 18th century), the introduction of electrically-powered mass production (start of 20th century), and the intensive utilization of electronics and Information Technology (IT) (start of 1970s), respectively (Kagermann et al., 2016). The primary objective of Industry 4.0 is to enhance industrial productivity, flexibility, and efficiency by connecting physical resources to the virtual world of computation through a higher level of automation, digitization, and effective communication between human, equipment, and products (Rojko, 2017; Alcácer & Cruz-Machado, 2019). From the production approach, Leyh, Martin & Schäffer (2017) defined Industry 4.0 as the transition from centralized production towards a flexible, self-controlled, and digitized environment where all components are highly interconnected to share information both vertically and horizontally. Some researchers predicted that Industry 4.0 factory has the promise to reduce production and logistic costs by 10-30 percent and quality management costs by 10-20

percent, while providing several key advantages such as shorter time-to-market, improved customer interactions, more flexible working environments, customizable mass production, better use of natural resources and energy etc. (Rojko, 2017). With a view to assisting implementation of Industry 4.0 technologies, the Reference Architecture Model Industrie 4.0 (RAMI4.0) was introduced as a guideline for developing a shared language and a structured framework that describes the fundamental bases of Industry 4.0 and the connection between different components through a three-dimensional space (Hankel & Rexroth, 2015). Another framework called "Industry 4.0 component" was also proposed to complement RAMI 4.0 in integrating humans, products, equipment, and processes (Grangel-González et al., 2016). Industry 4.0 component consists of physical objects and virtual representations of these objects and their functionalities. The integration of RAMI 4.0 and Industry 4.0 component bridges the gap between exiting standards, enables identifying the loopholes, and also defines key technological elements of Industry 4.0. Recognizing the importance of this transition for the position of a country in a global market, several government-supported initiatives were introduced all around the world in subsequent years. Industrial Internet was proposed in North America by General Electric company in late 2012 which covered a broader application area including power generation and distribution, healthcare, manufacturing, transportation, public sector etc. (Lin et al., 2015). Other notable initiatives were 'Industrie du futur' in France[2], 'Made in China 2025' by the China Ministry of Industry and Information Technology[3], Japan's 'Robot Strategy'[4], and 'Manufacturing Innovation 3.0' (Yim, 2016) in South Korea. Despite their

[2] http://www.gov.cn/zhengce/content/2015-05/19/content_9784.htm
[3] http://www.pref.aichi.jp/uploaded/attachment/51182.pdf
[4] http://www.economie.gouv.fr/files/files/PDF/industrie-du-futur_dp.pdf

different names, the common goal of these national strategies is similar to Industry 4.0 – to achieve “intelligent” or “smart” manufacturing through integration of new technologies.

The vision and necessity of fusing the cyber world with the physical world led to the formulation of a new paradigm – cyber-physical system (CPS) (Lee 2006 Lee, Bagheri & Kao, 2015). CPS offers the ability to connect physical devices to web-based applications via the Internet by integrating cybernetics, mechatronics, design, and process science (Hancu et al., 2007; NSF 2011; Khaitan & McCalley 2015). It also enables high degree of automation, reconfigurable dynamics, and multi-level networking at multiple scales (Miclea & Sanislav, 2011). CPS includes complex combination and coordination between physical and computational elements (Rad et al., 2015). Because of its transformative potential, CPS has drawn enormous attention from academia, industry, and governments, and has already found applications in a wide range of domains including power, healthcare, manufacturing, aerospace, city management, environmental control, infrastructure, defense, robotics etc. (Tham & Luo, 2013; Herterich, Uebernickel & Brenner, 2015; Wang, Törngren & Onori, 2015; Liu et al., 2017). CPS deployment is centered on using distributed intelligence (e.g. realized by multi-agent systems), enabling distribution of a complex problem into a network of modular, intelligent, adaptive, and pluggable components (Leitão, Colombo & Karnouskos, 2016). According to several researchers, CPS is the core enabling technology of Industry 4.0 (Kagermann et al., 2016; Leyh, Martin & Schäffer, 2017; Wagner, Herrmann & Thiede, 2017; Alcácer & Cruz-Machado, 2019). According to Zhang et al. (2017), CPS provides a theoretical framework for mapping everything related to manufacturing processes to the computing space and thus enables easier and more efficient modeling of manufacturing systems. Monostori (2014) coined the term of cyber-physical production system (CPPS) which consisted of multiple interconnected manufacturing

CPSs and discussed the major challenges to realizing CPPS, including context-adaptive and autonomous systems, cooperative production systems, identification, and prediction of dynamical systems, communication etc. Although numerous manufacturing machines are network compatible nowadays, very few of them are operated in a networked CPS environment due to lack of scalability and interoperability of physical resources. Cloud Manufacturing (CMfg), another emerging manufacturing paradigm, can address this issue by providing manufacturing services over cyber space based on integration of service-oriented manufacturing with cloud computing (Li et al., 2010; Tao et al., 2011; Xu, 2012). CMfg addresses issues of virtualization and perception of physical manufacturing resources and providing manufacturing cloud services over the Internet. It also enables sharing of manufacturing machines from multiple facilities and multiple manufacturers for fulfilling collaborative manufacturing demands (Liu, Li & Wang, 2011; Tao et al., 2014). In recent years, several researchers proposed different architectures, methodologies, and frameworks to amalgamate CPS and CMfg capabilities with a view to developing complex industrial systems with both vertical and horizontal integration (Colombo et al., 2014; Majstorovic et al., 2016; Alam & Saddik, 2017; Wang, 2017; Mourtzis & Vlachou, 2018; Wang & Wang, 2018; Qi et al., 2019). This fusion can not only improve production rate, optimization and efficiency of manufacturing processes and reduce production cost, but also connect consumers directly with manufacturers and their resources.

In our previous studies, we introduced the development of a new paradigm called cyber-physical manufacturing cloud (CPMC) by combining CPS and CMfg technologies for remote monitoring and direct operation of heterogeneous machine tools (Liu et al., 2016, 2017). CPMC is a scalable service-oriented framework for offering and managing manufacturing services from cloud by directly operating and sharing machine tools from many locations and monitoring

manufacturing processes over the Internet. Unlike most existing cloud-based manufacturing platforms, CPMC allows direct operations of heterogeneous connected manufacturing machine tools from the cloud. As mentioned above, one of the key barriers of constructing such a system is the lack of a communication method that enables both monitoring and operation of heterogeneous machine tools remotely by cloud applications over the Internet; as existing industrial communication protocols do not provide such capabilities. However, there are several research challenges to fully realize an integrated scalable CMfg and CPS manufacturing platform. Most manufacturing machine tools operate using their own proprietary languages and protocols; hence their usages are limited to manufacturer-specific applications. Recent burst of IoT devices and sensors, which use their own sets of communication protocols and data formats, increases this issue multi-fold (Sunny, Liu & Shahriar, 2019). Therefore, one of the major challenges of connecting machine tools physically over the Internet to create CPMC is lack of standardized communication protocols allowing direct operation of heterogeneous manufacturing resources remotely alongside continuous and rapid data transmission in a service-oriented approach. To address this, the research presented in this dissertation aimed at developing an Internet-scale communication mechanism to support aforementioned capabilities for different machine tools in a cloud-based cyber-physical manufacturing system.

1.2 Limitations of existing systems

Since the digitation of manufacturing started, a myriad of communication protocols and mechanisms was developed and implemented. At the beginning, the primary focus was at the low-level, e.g. transferring machining data and command in bits over bus-based networks (Zurawski, 2014). In the last decade or so, the focus has shifted towards application level protocol development, specially for easy integration with Internet-based services for seamless

data transmission. This is particularly important for CPS and CMfg systems, as those rely heavily on web-based applications and services. Although there exists a number of publications which broadly discussed various CMfg and manufacturing CPS architectures and enabling technologies at conceptual level (Li et al., 2010; Tao et al., 2011; Xu, 2012, Liu et al., 2017), very few of those focused on establishing a scalable communication mechanism for enabling service-oriented exchange of manufacturing services, even fewer that conducted experiments in an actual testbed to test and optimize such a communication method specifically for manufacturing CPSs. In fact, we found no research that presented successful implementation of an integrated cloud based cyber-physical manufacturing system with both monitoring and direct operation capabilities of heterogeneous machine tools provided over the Internet.

In recent years, MTConnect has emerged as potential communication standard for cyber manufacturing and is being a widely accepted communication standard for collecting monitoring data from machine tools (Vijayaraghavan et al., 2008). MTConnect is designed as an open protocol for improving interoperability of heterogeneous manufacturing machines by providing a uniform XML (eXtensible Markup Laguage) based data reporting structure via RESTful (Representational State Transfer) services. Figure 1 shows the architecture and methodology of MTConnect. The data flow between machines and MTConnect components is uni-directional – from machine to MTConnect adapter/agent. It collects data from physical resources in their own language and then converts the data into a common XML format following a uniform ontology. Several researchers adopted MTConnect as a viable communication mechanism for CMfg and manufacturing CPSs (Sunny, Liu & Shahriar, 2018). But MTConnect only offers monitoring capabilities; it cannot be used to operate machines remotely, which is very crucial for a CPS. Some researchers adopted low-level transmission mechanisms like UDP (User Datagram

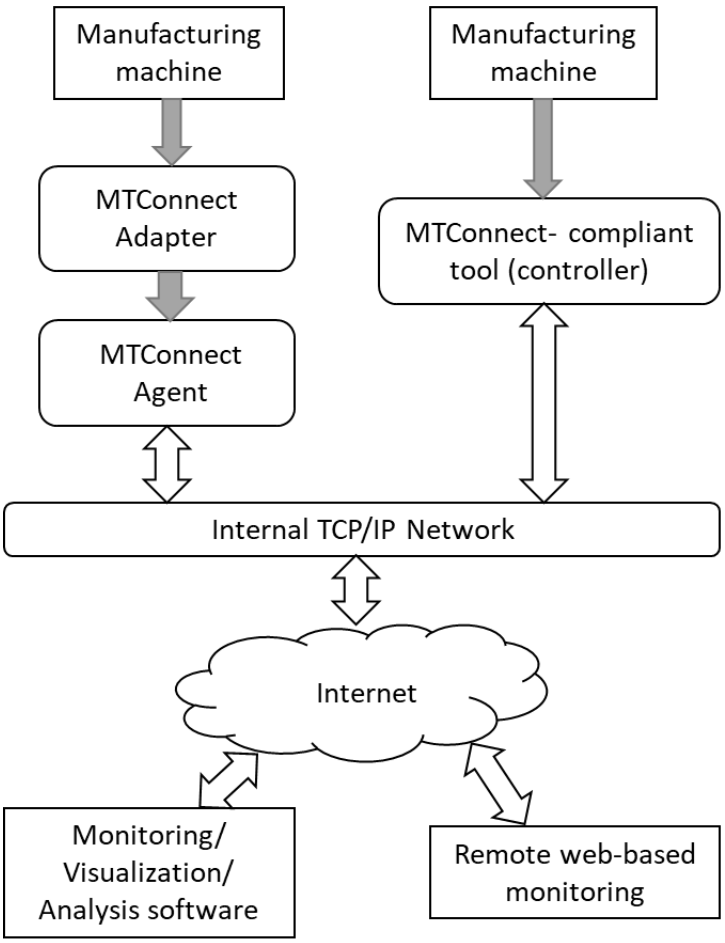


Figure 1. Architecture and methodology of MTConnect (Vijayaraghavan et al., 2008)

Protocol) or TCP (Transmission Control Protocol) for transmitting raw control commands (Wang, Gao & Ragai, 2014; Liu et al., 2017), while some used mechanisms that only works with a specific type of physical resources (Lin, Lin & Chiu, 2015; Parto, 2017; Okwudire et al., 2018). None of these works can be utilized for developing a CPMC system of aforementioned capabilities.

Another significant limitation of existing works is the lack of optimization of communication methods. Nowadays manufacturing machine tools generate tremendous amount of data every second which are required to be transmitted at high-speed. The rapid increase in the

number of devices and quantity of data generated, fueled by heterogeneity of data types, often results in complex scenarios where continuous data storing and processing become inefficient and expensive (McKee et al., 2018). This may also lead to network traffic bottleneck and scalability issues, specially for large-scale CPMSs managing thousands of machines at the same time. Moreover, rapid data transmission is often required in manufacturing CPSs, specially for applications like fault diagnosis and mitigation, collaborative manufacturing etc. Since the amount of the real-time data is enormous, the data transmission rate from the MTConnect agent to the application in the current prototype is limited to 100 milliseconds (Liu et al., 2018). This can be improved by optimizing the communication mechanism and application programs. However, we found no literature focused on developing such optimization techniques for cyber manufacturing.

1.3 Objectives

The primary objective of the research presented in this dissertation was to design, develop, and evaluate an Internet-scale communication method that could facilitate manufacturing services offering both remote monitoring and direct operation of heterogeneous machine tools in a scalable CPMC environment.

The secondary objective was to find unique applications of the developed communication method in the cyber manufacturing domain that were not available in other existing researches.

The final objective of this research was to identify possible areas for improvements of the developed method through experimentations and devise appropriate alteration strategies to improve its overall performance, efficiency, and feasibility.

1.4 Methodology

To facilitate aforementioned capabilities in a CPMC environment, an Internet-scale application level communication method named Machine Tool Communication (MTComm) was designed and developed using a semantic ontology, RESTful services, and XML based messaging structures. MTComm provides a common methodology to support both remote monitoring and direct operations of heterogeneous machine tools over the Internet. It was used to implement a CPMC testbed and develop several manufacturing applications including data acquisition from physical resources, performing manufacturing operations remotely, collaborative manufacturing, remote active testing based fault diagnosis etc. We also developed novel optimization strategies to improve MTComm's robustness, performance, and efficiency. Later, MTComm was used to introduce edge computing capabilities in the CPMC for enabling rapid localized data processing and enhancing overall system scalability and a System-on-Chip (SoC) based middleware prototype was developed. Lastly, to evaluate its feasibility to achieve cross-domain interoperability, MTComm was extended to support IoT devices located in consumer's home and an on-demand grocery shopping and delivery cloud system was proposed and developed. At every stage of its development, the performance of MTComm was evaluated through practical implementation and experimentations and finally compared with other existing cyber manufacturing communication methodologies.

1.5 Organization of this dissertation

Chapter 2 discusses key concepts related to this research including existing industrial communication methods, CPS, CMfg, and edge/fog computing in manufacturing, as well as a literature review of relevant works.

Chapter 3 describes the MTComm method including its architecture, semantic ontology, different services, and methodology.

Chapter 4 presents the development, implementation, and evaluation of remote and collaborative manufacturing in a CPMC testbed using MTComm.

Chapter 5 explains the design and development of optimization strategies for improving MTComm's performance. It also elaborates development of an edge based MTComm middleware, as well as the implementation and evaluation of a hardware prototype.

Chapter 6 describes the application of MTComm for facilitating active testing-based fault diagnosis and maintenance both in the cloud and at the edge.

Chapter 7 presents an IoT enabled cloud-based grocery shopping and delivery system as an example of MTComm's feasibility to be applied in other domains and achieve cross-domain interoperability.

Chapter 8 summarizes the overall methodology and contributions, and concludes with a comparison of MTComm's capabilities with state-of-the-art.

1.6 References

Aazam, M., Zeadally, S., & Harras, K. A. (2018). Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics, 14*(10), 4674-4682.

Alam, K. M., & El Saddik, A. (2017). C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. *IEEE access*, *5*, 2050-2062.

Alcácer, V., & Cruz-Machado, V. (2019). Scanning the industry 4.0: A literature review on technologies for manufacturing systems. *Engineering Science and Technology, an International Journal*.

Baena, F., Guarin, A., Mora, J., Sauza, J., & Retat, S. (2017). Learning factory: The path to industry 4.0. *Procedia Manufacturing*, *9*, 73-80.

Colombo, A. W., Bangemann, T., Karnouskos, S., Delsing, J., Stluka, P., Harrison, R., ... & Lastra, J. L. (2014). Industrial cloud-based cyber-physical systems. *The Imc-aesop Approach*, *22*, 4-5.

Grangel-González, I., Halilaj, L., Coskun, G., Auer, S., Collarana, D., & Hoffmeister, M. (2016, February). Towards a semantic administrative shell for industry 4.0 components. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)* (pp. 230-237). IEEE.

Hancu, O., Maties, V., Balan, R., & Stan, S. (2007). Mechatronic approach for design and control of a hydraulic 3-dof parallel robot. *Annals of DAAAM & Proceedings*, 321-323.

Hankel, M., & Rexroth, B. (2015). Industrie 4.0: The reference architectural model industrie 4.0 (rami 4.0). *ZVEI: Die Elektroindustrie*.

Herterich, M. M., Uebernickel, F., & Brenner, W. (2015). The impact of cyber-physical systems on industrial services in manufacturing. *Procedia Cirp*, *30*, 323-328.

Ivezic, N., & Srinivasan, V. (2016). On architecting and composing engineering information services to enable smart manufacturing. *Journal of computing and information science in engineering*, *16*(3).

Kagermann, H., Anderl, R., Gausemeier, J., Schuh, G., & Wahlster, W. (Eds.). (2016). *Industrie 4.0 in a Global Context: strategies for cooperating with international partners*. Herbert Utz Verlag.

Khaitan, S. K., & McCalley, J. D. (2014). Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, *9*(2), 350-365.

Lee, E. A. (2006, October). Cyber-physical systems-are computing foundations adequate. In *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap* (Vol. 2, pp. 1-9). Citeseer.

Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, *3*, 18-23.

Leitão, P., Colombo, A. W., & Karnouskos, S. (2016). Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry*, *81*, 11-25.

Leyh, C., Martin, S., & Schäffer, T. (2017, September). Industry 4.0 and Lean Production—A matching relationship? An analysis of selected Industry 4.0 models. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 989-993). IEEE.

Li, B. H., Zhang, L., Wang, S. L., Tao, F., Cao, J. W., Jiang, X. D., ... & Chai, X. D. (2010). Cloud manufacturing: a new service-oriented networked manufacturing model. *Computer integrated manufacturing systems*, *16*(1), 1-7.

Lin, S. W., Miller, B., Durand, J., Joshi, R., Didier, P., Chigani, A., ... & King, A. (2015). Industrial internet reference architecture. *Industrial Internet Consortium (IIC), Tech. Rep*.

Lin, Y. L., Lin, C. C., & Chiu, H. S. (2015, March). The development of intelligent service system for machine tool industry. In *2015 1st International Conference on Industrial Networks and Intelligent Systems (Iniscom)* (pp. 100-106). IEEE.

Liu, C., & Xu, X. (2017). Cyber-physical machine tool-the era of machine tool 4.0. *Procedia Cirp*, *63*, 70-75.

Liu, C., Vengayil, H., Zhong, R. Y., & Xu, X. (2018). A systematic development method for cyber-physical machine tools. *Journal of manufacturing systems*, *48*, 13-24.

Liu, N., Li, X., & Wang, Q. (2011, October). A resource & capability virtualization method for cloud manufacturing systems. In *2011 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 1003-1008). IEEE.

Liu, Y., Peng, Y., Wang, B., Yao, S., & Liu, Z. (2017). Review on cyber-physical systems. *IEEE/CAA Journal of Automatica Sinica*, *4*(1), 27-40.

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, *6*, 1-10.

Lu, Y., & Ju, F. (2017). Smart manufacturing systems based on cyber-physical manufacturing services (CPMS). *IFAC-PapersOnLine*, *50*(1), 15883-15889.

Majstorovic, V. D., Durakbasa, N. M., Mourtzis, D., & Vlachou, E. (2016). Cloud-based cyber-physical systems and quality of services. *The TQM Journal*.

McKee, D. W., Clement, S. J., Almutairi, J., & Xu, J. (2018). Survey of advances and challenges in intelligent autonomy for distributed cyber-physical systems. *CAAI Transactions on Intelligence Technology*, *3*(2), 75-82.

Miclea, L., & Sanislav, T. (2011, September). About dependability in cyber-physical systems. In *2011 9th East-West Design & Test Symposium (EWDTS)* (pp. 17-21). IEEE.

Monostori, L. (2014). Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia Cirp*, *17*, 9-13.

Mourtzis, D., & Vlachou, E. (2018). A cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance. *Journal of manufacturing systems*, *47*, 179-198.

NSF (US National Science Foundation). (2011, July). Cyber-Physical Systems (CPS). *Online.* Accessed April 22, 2020. https://www.nsf.gov/pubs/2011/nsf11516/nsf11516.pdf

Okwudire, C. E., Huggi, S., Supe, S., Huang, C., & Zeng, B. (2018). Low-level control of 3d printers from the cloud: A step toward 3d printer control as a service. *Inventions*, *3*(3), 56.

Parto Dezfouli, M. (2017). *A secure MTConnect compatible IoT platform for machine monitoring through integration of fog computing, cloud computing, and communication protocols* (Doctoral dissertation, Georgia Institute of Technology).

Qi, L., Chen, Y., Yuan, Y., Fu, S., Zhang, X., & Xu, X. (2019). A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems. *World Wide Web*, 1-23.

Rad, C. R., Hancu, O., Takacs, I. A., & Olteanu, G. (2015). Smart monitoring of potato crop: a cyber-physical system architecture model in the field of precision agriculture. *Agriculture and Agricultural Science Procedia*, *6*, 73-79.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2018). Communication method for manufacturing services in a cyber–physical manufacturing cloud. International Journal of Computer Integrated Manufacturing, 31(7), 636-652.

Sunny, S. M. N. A., Liu, X., & Shahriar, M. R. (2019, July). An Integrated IoT Enabled On-Demand Grocery Shopping and Delivery Cloud System Using MTComm at the Edge. In 2019 IEEE International Conference on Edge Computing (EDGE) (pp. 51-55). IEEE.

Tao, F., Zhang, L., Venkatesh, V. C., Luo, Y., & Cheng, Y. (2011). Cloud manufacturing: a computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *225*(10), 1969-1976.

Tao, F., Cheng, Y., Da Xu, L., Zhang, L., & Li, B. H. (2014). CCIoT-CMfg: cloud computing and internet of things-based cloud manufacturing service system. *IEEE Transactions on Industrial Informatics*, *10*(2), 1435-1442.

Tham, C. K., & Luo, T. (2013). Sensing-driven energy purchasing in smart grid cyber-physical system. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *43*(4), 773-784.

Wagner, T., Herrmann, C., & Thiede, S. (2017). Industry 4.0 impacts on lean production systems. *Procedia Cirp*, *63*, 125-131.

Wang, L. (2017). An overview of internet-enabled cloud-based cyber manufacturing. *Transactions of the Institute of Measurement and Control*, *39*(4), 388-397.

Wang, L., Törngren, M., & Onori, M. (2015). Current status and advancement of cyber-physical systems in manufacturing. *Journal of Manufacturing Systems*, *37*, 517-527.

Wang, L., Gao, R., & Ragai, I. (2014, June). An integrated cyber-physical system for cloud manufacturing. In *ASME 2014 International Manufacturing Science and Engineering Conference collocated with the JSME 2014 International Conference on Materials and*

Processing and the 42nd North American Manufacturing Research Conference. American Society of Mechanical Engineers Digital Collection.

Wang, L., & Wang, X. V. (2018). *Cloud-Based Cyber-Physical Systems in Manufacturing* (pp. 163-192). Cham: Springer International Publishing.

Weyer, S., Schmitt, M., Ohmer, M., Gorecky, D., Weyer, S., Schmitt, M., ... & Gorecky, D. (2015). Standardization as as the the crucial crucial challenge challenge Towards Standardization as the crucial challenge for highly production systems for highly modular, multi-vendor production systems for highly modular, multi-vendor productio. *IFAC-PapersOnLine*, *48*, 579-584.

Vijayaraghavan, A., Sobel, W., Fox, A., Dornfeld, D., & Warndorf, P. (2008). Improving machine tool interoperability using standardized interface protocols: MT connect.

Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and computer-integrated manufacturing*, *28*(1), 75-86.

Yim, M. S. (2016). The convergence between manufacturing and ict: The exploring strategies for manufacturing version 3.0 in korea. *Journal of Digital Convergence*, *14*(3), 219-226.

Zhang, Y., Qian, C., Lv, J., & Liu, Y. (2016). Agent and cyber-physical system based self-organizing and self-adaptive intelligent shopfloor. *IEEE Transactions on Industrial Informatics*, *13*(2), 737-747.

Zurawski, R. (2014). *Industrial communication technology handbook*. CRC Press.

CHAPTER 2

LITERATURE REVIEW

2.1 Industrial Communication Methods and Standards

2.1.1 Fieldbus and Ethernet based communication

Interoperability and automation of machine tools have always been a big concern for manufacturers. In the early 20th centuries, the mechanical technology and analog devices were the primary components of the process control systems and manufacturing systems. In 1970s, Programmable Logic Controller (PLC) with limited control functions was introduced which replaced the conventional relay based control systems (Erickson, 1996). With the development of digital computers, the scenario changed radically. Numerically Controlled (NC) machines came into play and labyrinths of mechanical linkages were substituted by point-to-point wiring. But this created a new difficulty. Optimized communication networking among different machines in the factory floor became a necessity. In 1985, Fieldbus systems emerged to reduce the complexity of conventional point-to-point wiring systems by connecting digital and analog devices to central controllers (Thomesse, 2005; Zurawski, 2014). Because of being an open protocol, many Fieldbus systems were developed in parallel and today there exist a number of variations. Over the past two decades, Fieldbus systems have gone through a lot of modifications and become standardized, although not unified. PROFIBUS is considered to be the most successful fieldbus technology and is widely used in industrial automation systems including factory and process automation (Zurawski, 2014) . PROFIBUS & PROFINET International (PI) group indicates on its website that PROFINET offers digital communication for data processing and transmission with speeds up to 12 Mbps and supports up to 126 addresses. Control Area Network (CAN) bus is a high-integrity serial bus system which was fundamentally designed to

be an automotive vehicle bus (Tindell, Hansson, & Wellings, 1994). CANopen and DeviceNet are higher level protocols standardized on top of CAN bus to allow interoperability with devices on the same industrial network (McFarlane, 1997). Modbus is a simple, robust and openly published, royalty free serial bus protocol that connects up to 247 nodes (Modbus Organization, 2006). Modbus is easy to implement and operate on RS-232 or RS-485 physical links with speeds up to 115K baud. CC-Link was originally developed by Mitsubishi and is a popular open-architecture, industrial network protocol in Japan and Asia (Wikipedia, 2019).

Although the Local Area Networks (LAN) based on Ethernet, as part of the TCP/IP and User Datagram Protocol (UDP) stack, rapidly gained much popularity for home and office use, it initially did not gain much acceptance in the industrial automation domain. However, advances in Ethernet technology made the LANs more suitable to industrial use. The increased data rates of newer Ethernet standards (for example 802.3u Fast Ethernet) made it easier to create real-time Ethernet protocols (Decotignie, 2009). The implementation of full-duplex Ethernet lines allows data transmission and reception to occur simultaneously, simplifying bus arbitration difficulties. Introduction of switched networks allowed Ethernet to be more acceptable for industrial use as opposed to the older hub-based networks. The introduction of Ethernet into the field of industrial networking also presented some new challenges (Zurawski, 2014). The existing Ethernet standards needed to be extended or modified to meet the rigorous requirements of industrial networks. This was achieved at various levels of the IP stack, and using various approaches. Backwards compatibility with existing fieldbus protocols was also an issue. Many of the newer Ethernet-based fieldbus protocols are extensions of existing protocols and various compatibility philosophies have been implemented. These are classified into four categories by Sauter (Sauter, 2010). The first is full compatibility at higher layer protocols, such as exists with High-Speed

Ethernet (HSE): Emerson, Foundation Fieldbus (Fieldbus Foundation, 2001); Modbus/TCP: Schneider (Swales, 1999), and Ethernet/IP: Rockwell (Brooks, 2001). This approach is especially prevalent in building automation fieldbuses. HSE is implemented on top of the TCP/IP stack, with additional use of standard IP interfaces such as dynamic host configuration protocol and simple network management protocol (Vincent, 2001). Modbus is an application layer messaging protocol for Client/Server based communication between devices connected via different types of buses or networks. The Application Layer Protocol Data Unit (APDU) of Modbus (Function Code and Data) has been encapsulated into an Ethernet frame. Ethernet/IP uses Common Industrial Protocol (CIP) which represents a common application layer for all physical networks of Ethernet/IP, ControlNet and DeviceNet (ControlNet International and Open DeviceNet Vendor association, 2001). Ethernet/IP uses the standard Ethernet and switches; thus it can have an unlimited number of nodes in a system. Another approach is compatibility of data objects and models, such as is the case with PROFINET: Siemens, PNO. This method uses proxy hardware to allow communication between the fieldbus media. PROFINET has three different classes - PROFINET Class A, PROFINET Class B, also referred as PROFINET Real-Time (PROFINET RT), and PROFINET Class C or PROFINET IRT (Isochronous and Real-Time). A lesser amount of compatibility is offered through the use of application layer profiles from existing protocols, as is implemented in Ethernet Powerlink: Bernecker & Rainer (IEC, 2004c; Zurawski, 2014) [IEC 2004f; Zurawski 2014], EtherCAT: Beckhoff, and SERCOS III (IEC, 2004b; IEC, 2004d; Neumann, 2007). Ethernet POWERLINK is implemented on top of IEEE 802.3 and, therefore, permits a free selection of network topology, cross connect, and hot plug. It utilizes a polling and time slicing mechanism for real-time data exchange. Such a system is appropriate for all kinds of automation systems ranging from PLC-to-PLC communication down

to motion and I/O control. EtherCAT is a protocol that is optimized for data processing using standard IEEE 802.3 Ethernet Frames. Each slave node processes its datagram and inserts the new data into the frame while each frame is passing through. EtherCAT is the MAC layer protocol and is transparent to any higher level Ethernet protocols such as TCP/IP, UDP, Web server etc. SERCOS III is the third generation of Serial Real-time Communication System (SERCOS). It accumulates on-the-fly packet processing for delivering real-time Ethernet and standard TCP/IP communication to deliver low latency industrial Ethernet. Lastly, completely new protocols have been developed for Ethernet that have no relationship with any existing protocols and have forgone any compatibility. Examples of such protocols are Ethernet for Plant Automation and Time-Critical Control Network (TCNet) (IEC, 2004a). Modbus/TCP (Swales, 1999), PROFINET IO (IEC, 2004e), and Ethernet/IP with Time synchronization (IEC, 2004f) are also noteworthy Ethernet based Fieldbus protocols.

2.1.2 Open Platform Communication (OPC)

Open Platform Communication (OPC) is a significant of many manufacturing networks at higher levels by offering a standardized interface for communication of industrial data. Maintained by the OPC Foundation, The OPC Specification has combined OLE (Object Linking and Embedding), COM (Component Object Model), and DCOM (Distributed Component Object Model) technologies developed by Microsoft (Leitner & Mahnke, 2006). The OPC specification outlined a standard set of objects, interfaces, and methods for use in process control and manufacturing automation applications to facilitate interoperability. OPC Classic is based on Microsoft's Component Object Model (COM) and Distributed Component Object Model (DCOM) technology and has three major specifications for data integration between process level and the management level (Mahnke, Leitner, & Damm, 2009). These specifications are

Data Access (DA), Alarm and Events (A&E) and Historical Data Access (HDA). OPC Data Access (OPC DA) is the most commonly used OPC specification, which is used to read and write real-time data. It allows real-time communication of process values over Ethernet with a client-server model. Several other variants of OPC have also been developed, including OPC Historical Data Access which permits for acquiring stored values, OPC Data Exchange for two-way communication using a server-server model and OPC XML Data Access which uses XML for communication. These specifications were mainly used in Human-Machine Interfaces (HMI) and Supervisory Control and Data Acquisition (SCADA) systems. OPC Foundation states that OPC Classic does not meet today's industry requirements because of the Windows platform dependent COM and the networking issues of DCOM, which effectively prevent Internet communication (Mahnke et al., 2009). OPC Classic also has no information security or scalability. OPC XML Data Access (DA) tried to make OPC platform-independent by replacing COM and DCOM with HTTP, Simple Object Access Protocol (SOAP) and Web Service technologies, but eventually failed due to inferior performance (Hirvonen, 2017).

Later in 2006, the OPC Unified Architecture (OPC UA) was specified and was being tested and implemented through its Early Adopters program (IEC, 2012). OPC UA combines the functionality of the existing OPC interfaces with new technologies such as XML and Web Services to deliver higher level Manufacturing Execution System (MES) and Enterprise Resource Planning (ERP) support. The primary goal of OPC UA is to replace previously defined COM-based specifications without degrading performance (Hirvonen, 2017) . OPC UA is based on common technologies such as TCP/IP, HTTP, Ethernet, and XML. It allows both client-server and publish-subscribe architectures. OPC UA provided the opportunity of accessing machine tool not only from factory floor but also from outside the factory. The OPC Foundation

aims to establish a common framework for industries with the platform independent, secure, and extensible OPC UA and its support for object-oriented information modelling capabilities (Mahnke et al., 2009).

Over the years, OPC UA has been implemented in various types of process monitoring and control systems in different industries such as Smart Grid (Claassen, Rohjans, & Member, 2011), Oil and Gas production (El Zawawi & El-Sayed, 2012), and Public transportation systems (Maka, Cupek, & Rosner, 2011). In recent years, many researchers considered OPC UA as the key to Industry 4.0 (Hannelius, Salmenpera, & Kuikka, 2008; Hoefling et al., 2015; Hoffmann et al., 2016; Iatrou & Urbas, 2016a; 2016b; Kožár & Kadera, 2016; Lindström, 2015; Liu et al. 2019; Luo et al., 2017; Palm et al., 2015; Rentschler, Trsek, & Dürkop, 2016; Seilonen et al. 2016; Wu et al., 2017). Schlechtendahl et al. (2015) proposed to use OPC UA as a critical enabler for discovering resources, enabling data communication through cloud-based gateways, and eventually transforming current production systems to cyber-physical production systems (CPPS) for Industry 4.0. Pauker et al. (2015) presented a service orchestration method for flexible manufacturing systems using sequential functional charts and OPC UA to enhance flexibility. They (2016) also proposed a systematic approach using OPC UA to develop an information model to represent a manufacturing system. Garcia et al. (2016) proposed to use OPC UA based information model to access factory floor data in a low-cost CPPS. Müller, Wings, and Bergmann (2017) developed open-source cyber-physical systems for service-oriented architectures using OPC UA. Luo et al. (2017) proposed a three-layered architecture based smart manufacturing process using OPC to integrate different industrial resources with factory energy management systems. Ayatollahi et al. (2018) developed a prototype OPC UA server enabling remote control of machine tools. Liu et al. (2019) proposed an OPC-UA based

cyber-physical machine tools platform for CNC machines using a generic OPC UA information model.

2.1.3 Machine Tool Connect (MTConnect)

In recent years, Machine Tool Connect (MTConnect) has acquired much acknowledgements after the release of its version 1.0 in 2008 (Vijayaraghavan et al., 2008). MTConnect is designed to enhance interoperability of manufacturing machines by providing a uniform XML-based data reporting structure. It is fundamentally a read-only framework, i.e., its principal focus is data monitoring and analysis. MTConnect enables manufacturing machines to be monitored over the Internet. The primary objective of MTConnect is to create a universal machine language that is understandable to all machines and also to the users. MTConnect provides a RESTful interface – there is no need of establishing any session or logon/logoff sequence to acquire data. As MTConnect is not designed for any specific type of machines, it has been used with different types of manufacturing resources such as CNC machine, industrial robot, milling machine, 3D printer (Liu et al., 2016) etc. In 2010, The OPC Foundation and the MTConnect Institute declared a cooperation to ensure interoperability and consistency between the two standards (ThomasNet, 2010).

Several projects and applications have emerged where MTConnect based data are collected from manufacturing machines and used for data analytics. Xu (2012) presented the potential of MTConnect for cloud based manufacturing systems. A collaborative research group including Boeing, NIST, and other members of the Open Modular Architecture Controllers User Group recognized MTConnect as a viable communication method by evaluating its performance on a distributed “Dual Ethernet” factory testbed (Michaloski et al., 2009) . Edrington et al. (2014) presented a web-based machine monitoring system that enabled data collection, analysis,

and machine event notification for any MTConnect compatible machines. Wang et al. (2014) used Transmission Control Protocol (TCP) alongside MTConnect to control and monitor machine tools respectively. Lin, Lin, and Chiu (2015) adopted the principles of MTConnect to develop standards for intelligent service system, named Taiwanese Machine Tool Connect (TMTC), and "ServBox", a device containing multiple adapters for CNC machines from different manufacturers. The authors claimed to achieve faster and lower traffic on multi-trip data transmission than MTConnect. System Insights and ROS-Industrial used MTConnect in a peer-to-peer communication architecture for connecting CNC machines and robotic arms (Liu et al., 2017) . STEP Tools Inc. and UI-LABS collaborated to develop a web environment enabling the orchestration of machining and measurement processes using mobile computing devices, and MTConnect was recommended as the communication standard to achieve that (UI Labs, 2017). Lei et al. (2017) extended MTConnect data models to implement a web-based monitoring system. Wu et al. (2017a) developed a fog computing-based framework for process monitoring and prognosis in cyber-manufacturing by integrating OPC UA and MTConnect with milling machines. Liu et al. (Liu et al., 2017) proposed an MTConnect-based cyber-physical machine tool platform. Mazak Cooperation showcased a scalable, end-to-end Industrial Internet of Things platform called Mazak SmartBox which connects manufacturing equipment to a factory's network and management systems via MTConnect (Mazak Corporation, 2017).

2.1.4 Others

Universal Plug and Play (UPnP) is a high-level protocol led by the Open Connectivity Foundation that aims to expand the simplicity and autoconfiguration of device Plug and Play to entire networks of intelligent appliances, wireless devices and PCs of all sizes (Miller et al., 2001) . Using common internet components such as standard internet protocol (IP), hypertext

transfer protocol (HTTP) and XML, TCP and UDP, it allows a device to join a network, obtain an IP address, and communicate with other devices without any need for device drivers

STEP-NC Application Protocol (AP) 238 is an alternative CNC programming language to G-codes which is developed on the base of a STEP data format under ISO 14649 standard (ISO, 2002). STEP-NC provides significantly richer information to both CNC controller and its operators, enabling better understanding of the product being manufactured. Wosnik et al. (2006) proposed a STEP-NC enabled feedback loop for manufacturing processes in which a mathematical observer calculated the cutting forces from motor current data acquired using STEP-NC. Zhao et al. (2008) proposed a closed-loop machining mechanism for on-line inspection process using STEP-NC data. Ridwan, Xu, and Liu (2012) developed a machining parameter optimization framework using the STEP-NC data obtained during machining process. Danjou et al. (2016) used STEP-NC in a manufacturing knowledge management system to assist product designers to optimize machining conditions during design. STEP Tools Inc used STEP-NC with MTConnect to develop a digital thread solution which keeps the design, manufacturing, and inspection data of a product connected around a digital twin (Liu et al., 2019) .

oneM2M was originally designed to standardize a common M2M (machine-to-machine) service layer platform for globally applicable and access-independent M2M services (Swetina et al, 2014). Seven standards development organizations (SDOs) joined forces to combine existing M2M protocols and discarded redundant and overlapping M2M service layer operations. oneM2M makes use of existing protocols such as Message Queue Telemetry Transport (MQTT), Hyper Text Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) etc. and standards such as Broadband Forum (BBF) and Open Mobile Alliance (OMA) as much as possible. This standard has been widely adopted in different IoT based large-scale systems (Datta

et al., 2015; 2016; Kim, 2017; Park et al., 2016; Wu et al., 2017; Yun et al., 2016). Wilner et al. (2017) proposed to use oneM2M to establish semantic interoperability among components of smart factories. An industrial device monitoring and control system was developed using oneM2M at the edge (Um, Lee, & Jeong, 2018) . Yun et al. (2019) presented a reference framework for CMfg and industrial CPSs based on oneM2M and argued that oneM2M would be advantageous for large network-based systems for its high scalability and ontology service.

AutomationML (Automation Markup Language) is another promising upcoming open standard series (IEC 62714) for the description of production plants and plant components (Drath et al., 2008). AutomationML describes the contents – what is exchanged between the parties and systems involved. It helps to model plants and plant components with their skills, topology, interfaces, and relations to others, geometry, kinematics, and even logic and behavior. A joint working group of the AutomationML e.V. and the OPC Foundation deals with the creation of a companion specification 'AutomationML in OPC UA' (Henßen & Schleipen, 2014) . Schroeder et al. (Schroeder et al., 2016) used AutomationML to develop digital twin of a manufacturing device represent its physical components as well as some attributes.

Apart from abovementioned attempts, ROY-G-BIV published a patent for controlling machine tools over a network and developed a machine communication platform called XMC (Brown & Clark, 2003).

2.2 Recent Advancements in Manufacturing Domain

2.2.1 Cloud Manufacturing (CMfg)

Cloud Manufacturing (CMfg) is an emerging manufacturing technology which makes manufacturing resources and services available over the internet by integrating cloud computing,

big data analysis, IoT, and manufacturing (Li, B. et al., 2010; Tao, Zhang, Venkatesh, Luo, & Cheng, 2011; Xu, 2012b). Several architectures of CMfg have been proposed by applying cloud computing to manufacturing (Holtewert et al., 2013; Luo et al., 2011; Tao et al., 2011; Xu, 2012). Wu et al. (2013) reviewed a few cloud manufacturing architectures in their paper. The architectures that were discussed focus on virtualization of manufacturing resources and services and offer them as online services to consumers. Tao et al. (2011) proposed a CMfg architecture consisting of ten layers. Rauschecker et al. (2011) proposed a uniform representation of manufacturing resource and services across multiple service providers in cloud manufacturing. Li, Liu, and Xu (2012) discussed heterogeneous systems and integration access method in cloud manufacturing and proposed a solution for the adaptation of manufacturing equipment based on fiber grating sensing technology. Wu et al. (2012; 2015) proposed a method of cloud-based design and manufacturing which enables sharing of manufacturing resources as cloud services, similar to infrastructure as a service (IaaS) or software as a service (SaaS). Xiang and Hu (2012) and Tao et al. (2014) discussed an intelligent approach for perceiving and accessing manufacturing resources in cloud manufacturing using IoT-based technology. Wang (2013) proposed a tiered architecture to service oriented manufacturing and connected it to a shop-floor environment to enable real time availability and monitoring. He also discussed machine availability monitoring and machining process planning towards cloud manufacturing and mentioned that closed-loop information flow makes process planning and monitoring feasible services for the CMfg. Mai et al. (2016) proposed a framework for 3D printing service platform for cloud manufacturing. Balta et al. (2018) presented a Production as a Service (PaaS) framework to connect both consumers and manufacturers by abstracting manufacturing steps of a product as individual service requests. Saez et al. (2018) developed an IoT enabled data

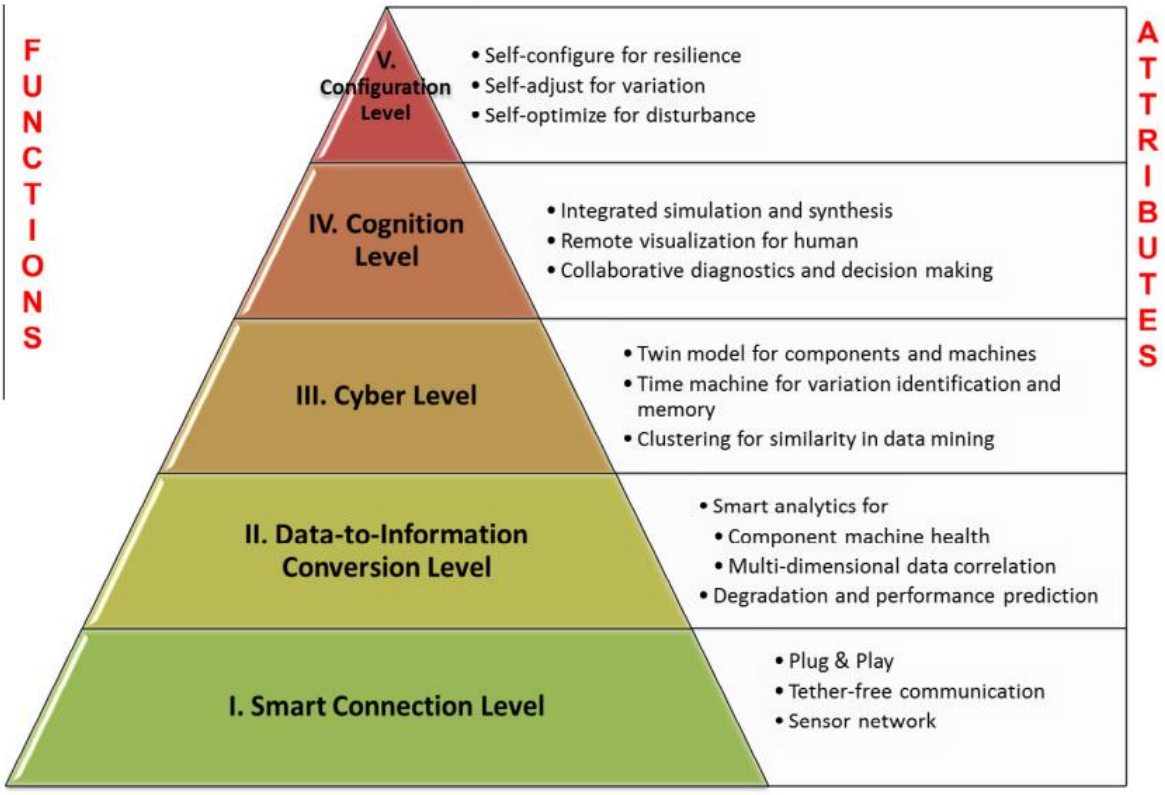


Figure 2. 5C architecture for implementation of industrial CPS (Lee, Bagheri, & Kao, 2015)

processing cloud infrastructure to monitor machine health, detect anomalies, and evaluate throughput of manufacturing systems.

2.2.2 Cyber-Physical System (CPS)

Cyber-Physical Systems (CPSs) are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the Internet (Geisberger & Broy, 2012; Hellinger & Seeger, 2011; Monostori, 2014; Monostori et al., 2016). In most CPSs, various embedded devices are networked to sense, monitor and actuate physical elements in the real world. CPS is being applied in a wide range of domains including advanced manufacturing. Although there exist numerous physical

manufacturing machines which are network-ready, very few of them are operated in a networked environment. Lee, Bagheri, and Kao (2015) proposed a unified five-level architecture (5C architecture) as a guideline for implementation of industrial CPS. Figure 2 illustrates the five levels in a bottom-up fashion – Connection, Conversion, Cyber, Cognition, and Configure. Cyber-physical production systems (CPPS) were proposed that consisted of autonomous and cooperative elements and subsystems that are connected based on the context within and across all levels of production, from processes through machines up to production and logistics networks (Monostori, 2014; Monostori et al., 2016). CPPS, relying on the latest and foreseeable further developments of computer science (CS), information and communication technologies (ICT), and manufacturing science and technology (MST) may lead to the 4th industrial revolution, frequently noted as Industrie 4.0 (Kagermann et al., 2013). Within global supply networks, machinery, warehousing systems and production facilities will incorporate in the shape of CPPS. These systems will autonomously exchange information, triggering actions and controlling each other independently within a smart factory (Zuehlke, 2010). Ridwan and Xu (2013) presented an intelligent machine monitoring system that allows parameter optimization based on MTConnect data before, during, and after machining in order to reduce process time and increase quality. Leitão, Colombo & Karnouskos (2016) described four prototype implementations (SOCRADES, GRACE, IMC-AESOP, and ARUM) for industrial automation based on CPSs technologies. They also identified key challenges of implementing CPS prototypes and divided them into six major areas - (i) CPS Capabilities, (ii) CPS Management, (iii) CPS Engineering, (iv) CPS Ecosystems, (v) CPS Infrastructures and (vi) CPS Information Systems. Zhang et al. (2016) proposed a data-driven CPS architecture for CNC machine tools and discussed three key issues – node configuration, interconnection technology of

heterogeneous nodes, and data-driven adaptive configuration. Lu and Ju (2017) introduced cyber-physical manufacturing services (CPMS) for service-oriented smart manufacturing system using standardized functional taxonomies and a reference ontology to separate service requests and service consumptions. Morgan and O'Donnell (2018) proposed a reconfigurable, flexible, and extensible CPS monitoring system for machine tools. Recently, a research group presented several approaches to develop cyber-physical machine tools (CPMT) platform using different industrial communication methods including MTConnect, STEP-NC, and OPC-UA (Deng et al., 2018; Kubota et al., 2018; Liu, Chao & Xu, 2017; Liu et al., 2018a; 2019c) . They implemented a cloud-based control algorithm for industrial CPS systems using Windows Communication Foundation (WCF) services and Microsoft Azure IoT Hub (Papcun et al., 2018). They also presented a systematic development method for CPMT using a cyber twin of machine tool in a cloud environment (Liu et al., 2018b).

2.2.3 Edge and Fog Computing in Manufacturing

For last couple of decades, computation and storage are being shifted from stationary desktop computers to remote cloud servers. Cloud computing provides better efficiency, availability, and scalability for large-scale systems. However, since cloud computing is centralized and there exist multiple hops between data being produced (in end devices) and data being processed (in cloud servers), it is now always capable of meeting all requirements of a distributed system, such as real-time response, localized security, resilience etc. This issue has become significant since the rapid growth of IoT devices generating terabytes of data every minute. The total number of interconnected physical devices is estimated to reach 50 billion by the end of 2020 (Al-Doghman et al., 2016). Transferring this huge amount of data from thousands of end devices to a central cloud server and processing them often leads to network

bottleneck increasing latency further. To overcome this, the concepts of edge and fog computing were introduced to move cloud capabilities to network edges in order to process data closer to end devices.

The first formal definition of fog computing was stated in 2012 by Bonomi et al. (2012) from CISCO as: "*Fog computing is a highly virtualized platform that provides compute, storage and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of the network.*" Varghese et al. (2017) defined fog computing as "*a model to complement the cloud for decentralizing the concentration of computing resources (for example, servers, storage, applications and services) in data centers towards users for improving the quality of service and their experience.*" It aims to minimize the request-response latency between applications and to enable local computing resources for the end-devices and network connectivity to centralized services (Iorga et al., 2018) . The elements of fog computing are typically network devices such as routers, switches, base station, servers etc. and are located in the network layer, between physical resources and cloud servers.

The term edge computing was first coined around 2002 and was mainly associated with the deployment of applications over Content Delivery Networks (CDN) (Garcia et al., 2015) . The main objective of this approach was to benefit from the proximity and resources of CDN edge servers to achieve massive scalability. Placing data and data-intensive applications at the edge reduced the amount of data transmission and also distance that data must travel (Escamilla-Ambrosio et al., 2018). Eventually, this concept became broader and more popular, specially for IoT domain. Nowadays edge computing refers to the control and management of a small number of standalone interconnected end-point devices at the edge of an environment (Nebbiolo Technologies Inc., 2019). With the availability of smaller and cheaper but yet powerful compute

devices, such as Raspberry Pi, Arduino, SoC FPGAs, the edge computing paradigm brings the processing back closer to end devices (McKee et al., 2018). Edge computing devices are primarily embedded controllers situated at the edge of the physical resource layer. However, in recent years, edge computing domain has extended to network devices too, and thus often considered as a sub-set of fog computing domain (Nebbiolo Technologies Inc., 2019).

Although often used interchangeably, edge computing and fog computing are fundamentally different (Escamilla-Ambrosio et al., 2018; Iorga et al., 2018; Nebbiolo Technologies Inc., 2019). Fog computing typically aids the cloud, whereas edge computing is defined by the exclusion of cloud. Fog computing is multilayer and hierarchical, where edge tends to be limited to three or four layers. Fog computing runs applications in a multi-layer hierarchical architecture allowing for dynamic reconfigurations for different applications while performing intelligent computing and transmission services. Edge computing runs specific applications for a small number of physical resources in a fixed logic location and provides a direct transmission service. Moreover, in addition to computation, and networking, fog computing also addresses storage, control and data-processing acceleration.

In recent years, several researchers focused on utilizing edge and fog computing solutions to address several issues in the advanced manufacturing domain. A fog-based adaptive operations platform was presented to interface equipment manufacturers and operational administrators of SCADA infrastructures (Gazis et al., 2015). Peralta et al. (2017) proposed a fog-based architecture for Industry 4.0 applications by placing MQTT broker in an intermediary fog layer between IoT devices and cloud applications to minimize energy requirements of IoT nodes. Aazam, Zeadally, and Harras (2018) discussed several prospects and research challenges of deploying fog computing middleware nodes in Industrial IoT systems. De Brito et al. (2018)

proposed to deploy fog computing nodes with the capability of internode peer-to-peer communication and programmability via a centralized service orchestration in smart factories and CPSs. Wan et al. (2018) developed a multiagent based fog computing system for smart factories to optimize load balancing and scheduling and reduce transmission delay. Yin, Luo, and Luo (2018) developed a container-based task scheduling and resource allocation model for delay-sensitive and high-concurrency functions of fog computing in smart manufacturing. Zhou et al. (2018) presented a fog computing-based cyber-physical machine tool system to improve performances and intelligence of CNC machine tools. Chen et al. (2018) introduced a reference architecture for IoT-based manufacturing and discussed the impact of edge computing from four aspects – edge-specific equipment, network communication, information fusion, and cooperative mechanism with cloud computing. They also presented a case study to implement active maintenance of manufacturing equipment using a prototype platform. Derhamy et al. (2018) proposed edge-based automation services to provide functionalities required to plan, sequence, and interconnect manufacturing CPSs. Govindaraj and Artemenko (2018) designed a novel redundancy live migration scheme for container-based edge computing in latency critical industrial applications and were able to reduce downtime by a factor of 1.8. Qi et al. (2018) proposed to utilize edge, fog, and cloud computing to deploy COS and digital twins of physical machines in different levels of smart manufacturing. Um, Lee, and Jeong (2018) presented an industrial device monitoring and control system based on oneM2M standard for an edge-based smart factory environment. Li et al. (2019) proposed a two-phase scheduling strategy on the edge computing layer for allocating computing resources to meet low-latency requirements of different AI tasks.

2.3 References

Aazam, M., Zeadally, S., & Harras, K. A. (2018). Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics, 14*(10), 4674-4682.

Al-Doghman, F., Chaczko, Z., Ajayan, A. R., & Klempous, R. (2016). A review on fog computing technology. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* 1525.

Ayatollahi, I., Brier, J., Mörzinger, B., Heger, M., & Bleicher, F. (2018). SOA on smart manufacturing utilities for identification, data access and control. *Procedia CIRP, 67*, 162-166.

Balta, E. C., Lin, Y., Barton, K., Tilbury, D. M., & Mao, Z. M. (2018). Production as a service: A digital manufacturing framework for optimizing utilization. *IEEE Transactions on Automation Science and Engineering, 15*(4), 1483-1493.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing,* 13-16.

Brooks, P. (2001). Ethernet/IP-industrial protocol. In *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. no. 01TH8597), 2,* 505-514.

Brown, D. W., & Clark, J. S. (2003). *Distribution of motion control commands over a network.* U.S. Patent 6,513,058.

Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H., & Zhang, Q. (2018). Edge computing in IoT-based manufacturing. *IEEE Communications Magazine, 56*(9), 103-109.

Claassen, A., Rohjans, S., & Member, S. L. (2011). Application of the OPC UA for the smart grid. In *2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies,* 1-8.

ControlNet International and Open DeviceNet Vendor association. (2001). Ethernet/IP adaptation of CIP specification, release 1.0. Accessed March 25, 2020. Retrieved from http://read.pudn.com/downloads166/ebook/763212/EIP-CIP-V2-1.0.pdf

Danjou, C., Le Duigou, J., & Eynard, B. (2016). Closed-loop manufacturing, a STEP-NC process for data feedback: A case study. *Procedia CIRP, 41*, 852-857.

Datta, S. K., Da Costa, Rui Pedro Ferreira, Bonnet, C., & Härri, J. (2016). oneM2M architecture based IoT framework for mobile crowd sensing in smart cities. In *2016 European Conference on Networks and Communications (EuCNC),* 168-173.

Datta, S. K., Gyrard, A., Bonnet, C., & Boudaoud, K. (2015). oneM2M architecture based user centric IoT application development. In *2015 3rd International Conference on Future Internet of Things and Cloud,* 100-107.

de Brito, M. S., Hoque, S., Steinke, R., Willner, A., & Magedanz, T. (2018). Application of the fog computing paradigm to smart factories and cyber-physical systems. *Transactions on Emerging Telecommunications Technologies, 29*(4), e3184.

Decotignie, J. (2009). The many faces of industrial ethernet [past and present]. *IEEE Industrial Electronics Magazine, 3*(1), 8-19.

Deng, C., Guo, R., Zheng, P., Liu, C., Xu, X., & Zhong, R. Y. (2018). From open CNC systems to cyber-physical machine tools: A case study. *Procedia CIRP, 72,* 1270-1276.

Derhamy, H., Andersson, M., Eliasson, J., & Delsing, J. (2018). Workflow management for edge driven manufacturing systems. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS),* 774-779.

Drath, R., Luder, A., Peschke, J., & Hundt, L. (2008). AutomationML-the glue for seamless automation engineering. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation,* 616-623.

Edrington, B., Zhao, B., Hansel, A., Mori, M., & Fujishima, M. (2014). Machine monitoring system based on MTConnect technology. *Procedia Cirp, 22,* 92-97.

El Zawawi, A., & El-Sayed, A. (2012). Integration of DCS and ESD through an OPC application for upstream oil and gas. In *2012 IEEE Power and Energy Society General Meeting,* 1-5.

Erickson, K. T. (1996). Programmable logic controllers. *IEEE Potentials, 15*(1), 14-17.

Escamilla-Ambrosio, P. J., Rodríguez-Mota, A., Aguirre-Anaya, E., Acosta-Bermejo, R., & Salinas-Rosales, M. (2018). Distributing computing in the internet of things: Cloud, fog and edge computing overview. *Neo 2016* (pp. 87-115) Springer.

Fieldbus Foundation. (2001). High speed ethernet specification documents FF-801, 803, 586, 588,589, 593, 941. Accessed March 25, 2020. Retrieved from http://www.fieldbus.org

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., . . . Riviere, E. (2015). *Edge-centric computing: Vision and challenges* ACM New York, NY, USA.

Garcia, M. V., Irisarri, E., Perez, F., Estevez, E., & Marcos, M. (2016). OPC-UA communications integration using a CPPS architecture. In *2016 IEEE Ecuador Technical Chapters Meeting (ETCM),* 1-6.

Gazis, V., Leonardi, A., Mathioudakis, K., Sasloglou, K., Kikiras, P., & Sudhaakar, R. (2015). Components of fog computing in an industrial internet of things context. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops),* 1-6.

Geisberger, E., & Broy, M. (2012). *Agenda CPS-integrierte forschungsagenda cyber-physical systems. acatech STUDIE, 1--297*

Govindaraj, K., & Artemenko, A. (2018). Container live migration for latency critical industrial applications on edge computing. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 1,* 83-90.

Hannelius, T., Salmenpera, M., & Kuikka, S. (2008). Roadmap to adopting OPC UA. In *2008 6th IEEE International Conference on Industrial Informatics,* 756-761.

Hellinger, A., & Seeger, H. (2011). Cyber-physical systems. driving force for innovation in mobility, health, energy and production. *Acatech Position Paper, National Academy of Science and Engineering, 1*(2)

Henßen, R., & Schleipen, M. (2014). Interoperability between OPC UA and AutomationML. *Procedia Cirp, 25*, 297-304.

Hirvonen, M. S. (2017). *Streamlining manufacturing data integration.* Master's Thesis. Tampere University of Technology.

Hoefling, M., Heimgaertner, F., Fuchs, D., Menth, M., Romano, P., Tesfay, T., . . . Gronas, V. (2015). Integration of IEEE C37. 118 and publish/subscribe communication. In *2015 IEEE International Conference on Communications (ICC),* 764-769.

Hoffmann, M., Thomas, P., Schütz, D., Vogel-Heuser, B., Meisen, T., & Jeschke, S. (2016). Semantic integration of multi-agent systems using an OPC UA information modeling approach. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN),* 744-747.

Holtewert, P., Wutzke, R., Seidelmann, J., & Bauernhansl, T. (2013). Virtual fort knox federative, secure and cloud-based platform for manufacturing. *Procedia CIRP*, *7*, 527-532.

Iatrou, C. P., & Urbas, L. (2016a). Efficient opc ua binary encoding considerations for embedded devices. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN),* 1148-1153.

Iatrou, C. P., & Urbas, L. (2016b). OPC UA hardware offloading engine as dedicated peripheral IP core. In *2016 IEEE World Conference on Factory Communication Systems (WFCS),* 1-4.

IEC. (2004a). *IEC 65C/353/NP. real-time ethernet: TCnet (time-critical control network).* International Electrotechnical Commission (IEC).

IEC. (2004b). *IEC 65C/355/NP. real-time ethernet: ETHERCAT.* International Electrotechnical Commission (IEC).

IEC. (2004c). *IEC 65C/356/NP. real-time ethernet: POWERLINK.* International Electrotechnical Commission (IEC).

IEC. (2004d). *IEC 65C/358/NP. real-time ethernet: SERCOS III.* International Electrotechnical Commission (IEC).

IEC. (2004e). *IEC 65C/359/NP. real-time ethernet: PROFINET IO. application layer service definition & application layer protocol specification.* International Electrotechnical Commission (IEC).

IEC. (2004f). *IEC 65C/361/NP. real-time ethernet: EtherNet/IP with time synchronization.* International Electrotechnical Commission (IEC).

IEC. (2012). *IEC 62541 - OPC unified architecture* International Electrotechnical Commission.

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N., & Mahmoudi, C. (2018). Fog computing conceptual model, recommendations of the national institute of standards and technology. *NIST Special Publication,* 500-325.

ISO. (2002). ISO 14649-1 - data model for computerized numerical controllers: Part 1 overview and fundamental principles.

Kagermann, H., Helbig, J., Hellinger, A., & Wahlster, W. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of german manufacturing industry; final report of the industrie 4.0 working group* Forschungsunion.

Kožár, S., & Kadera, P. (2016). Integration of IEC 61499 with OPC UA. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA),* 1-7.

Kubota, T., Liu, C., Mubarok, K., & Xu, X. (2018). A cyber-physical machine tool framework based on STEP-NC. In *Proceedings of the 48th International Conference on Computers and Industrial Engineering (CIE 48).*

Lee, J., Bagheri, B., & Kao, H. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters, 3,* 18-23.

Lei, P., Zheng, L., Wang, L., Wang, Y., Li, C., & Li, X. (2017). MTConnect compliant monitoring for finishing assembly interfaces of large-scale components: A vertical tail section application. *Journal of Manufacturing Systems, 45*, 121-134.

Leitão, P., Colombo, A. W., & Karnouskos, S. (2016). Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry, 81,* 11-25.

Leitner, S., & Mahnke, W. (2006). OPC UA–service-oriented architecture for industrial applications. *ABB Corporate Research Center, 48*, 61-66.

Li, B., Zhang, L., Wang, S., Tao, F., Cao, J. W., Jiang, X. D., . . . Chai, X. D. (2010). Cloud manufacturing: A new service-oriented networked manufacturing model. *Computer Integrated Manufacturing Systems, 16*(1), 1-7.

Li, R., Liu, Q., & Xu, W. (2012). Perception and access adaptation of equipment resources in cloud manufacturing. *Computer Integrated Manufacturing Systems, 18*(7), 1547-1553.

Li, X., Wan, J., Dai, H., Imran, M., Xia, M., & Celesti, A. (2019). A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. *IEEE Transactions on Industrial Informatics, 15*(7), 4225-4234.

Lin, Y., Lin, C., & Chiu, H. (2015). The development of intelligent service system for machine tool industry. In *2015 1st International Conference on Industrial Networks and Intelligent Systems (Iniscom)*, 100-106.

Lindström, O. M. (2015). *Standardized collection of production data in factory environment*

Liu, C., Hong, X., Zhu, Z., & Xu, X. (2018). Machine tool digital twin: Modelling methodology and applications.

Liu, C., Vengayil, H., Lu, Y., & Xu, X. (2019a). A cyber-physical machine tools platform using OPC UA and MTConnect. *Journal of Manufacturing Systems, 51*, 61-74.

Liu, C., Vengayil, H., Lu, Y., & Xu, X. (2019b). A cyber-physical machine tools platform using OPC UA and MTConnect. *Journal of Manufacturing Systems, 51*, 61-74.

Liu, C., Vengayil, H., Lu, Y., & Xu, X. (2019c). A cyber-physical machine tools platform using OPC UA and MTConnect. *Journal of Manufacturing Systems, 51*, 61-74.

Liu, C., Vengayil, H., Zhong, R. Y., & Xu, X. (2018). A systematic development method for cyber-physical machine tools. *Journal of Manufacturing Systems, 48*, 13-24.

Liu, C., & Xu, X. (2017). Cyber-physical machine tool-the era of machine tool 4.0. *Procedia Cirp, 63*, 70-75.

Liu, C., Xu, X., Peng, Q., & Zhou, Z. (2018). MTConnect-based cyber-physical machine tool: A case study. *Procedia Cirp, 72*, 492-497.

Liu, X. F., Shahriar, M. R., Al Sunny, S. N., Leu, M. C., & Hu, L. (2017). Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed. *Journal of Manufacturing Systems, 43*, 352-364.

Liu, X. F., Sunny, S. M., Shahriar, M. R., Leu, M. C., Cheng, M., & Hu, L. (2016). Implementation of MTConnect for open source 3D printers in cyber physical

manufacturing cloud. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,*

Lu, Y., & Ju, F. (2017). Smart manufacturing systems based on cyber-physical manufacturing services (CPMS). *IFAC-PapersOnLine, 50*(1), 15883-15889.

Luo, Y. L., Zhang, L., He, D. J., Ren, L., & Tao, F. (2011). Study on multi-view model for cloud manufacturing. In *Advanced Materials Research, , 201* 685-688.

Luo, Z., Hong, S., Lu, R., Li, Y., Zhang, X., Kim, J., . . . Liang, W. (2017). OPC UA-based smart manufacturing: System architecture, implementation, and execution. In *2017 5th International Conference on Enterprise Systems (ES),* 281-286.

Mahnke, W., Leitner, S., & Damm, M. (2009). *OPC unified architecture.* Springer Science & Business Media.

Mai, J., Zhang, L., Tao, F., & Ren, L. (2016). Customized production based on distributed 3D printing services in cloud manufacturing. *The International Journal of Advanced Manufacturing Technology, 84*(1-4), 71-83.

Maka, A., Cupek, R., & Rosner, J. (2011). OPC UA object oriented model for public transportation system. In *2011 UKSim 5th European Symposium on Computer Modeling and Simulation,* 311-316.

Mazak Corporation. (2017). Mazak SmartBox. Accessed March 25, 2020. Retrieved from https://www.mazakusa.com/machines/technology/digital-solutions/mazak-smartbox/

McFarlane, A. (1997). Fieldbus review. *Sensor Review.*

McKee, D. W., Clement, S. J., Almutairi, J., & Xu, J. (2018). Survey of advances and challenges in intelligent autonomy for distributed cyber-physical systems. *CAAI Transactions on Intelligence Technology, 3*(2), 75-82.

Michaloski, J., Lee, B., Proctor, F., Venkatesh, S., & Ly, S. (2009). Quantifying the performance of MT-connect in a distributed manufacturing environment. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 533-539.

Miller, B. A., Nixon, T., Tai, C., & Wood, M. D. (2001). Home networking with universal plug and play. *IEEE Communications Magazine, 39*(12), 104-109.

Modbus Organization. (2006). Modbus application protocol specification. Accessed March 25, 2020. Retrieved from http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

Monostori, L. (2014). Cyber-physical production systems: Roots, expectations and R\&D challenges. *Procedia Cirp, 17,* 9-13.

Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., . . . Ueda, K. (2016). Cyber-physical systems in manufacturing. *Cirp Annals, 65*(2), 621-641.

Morgan, J., & O'Donnell, G. E. (2018). Cyber physical process monitoring systems. *Journal of Intelligent Manufacturing, 29*(6), 1317-1328.

Müller, M., Wings, E., & Bergmann, L. (2017). Developing open source cyber-physical systems for service-oriented architectures using OPC UA. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 83-88.

Myung, S., & Kim, S. (2017). The design of open IoT platform based on oneM2M standard protocol. *Journal of the Korea Institute of Information and Communication Engineering, 21*(10), 1943-1949.

Nebbiolo Technologies Inc. (2019). Fog vs edge computing. Accessed March 26, 2020. Retrieved from https://www.nebbiolo.tech/wp-content/uploads/2019/11/whitepaper_fog-vs-edge-v1.1.01_WEB.pdf

Neumann, P. (2007). Communication in industrial automation—What is going on? *Control Engineering Practice, 15*(11), 1332-1347.

Palm, F., Grüner, S., Pfrommer, J., Graube, M., & Urbas, L. (2015). Open source as enabler for OPC UA in industrial automation. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, 1-6.

Papcun, P., Kajáti, E., Liu, C., & Zhong, R. Y. (2018). Cloud-based control of industrial cyber-physical systems.

Park, H., Kim, H., Joo, H., & Song, J. (2016). Recent advancements in the internet-of-things related standards: A oneM2M perspective. *Ict Express, 2*(3), 126-129.

Pauker, F., Ayatollahi, I., & Kittl, B. (2015). Service orchestration for flexible manufacturing systems using sequential functional charts and opc ua. *Dubrovnik, 9*, 11-09.

Pauker, F., Frühwirth, T., Kittl, B., & Kastner, W. (2016). A systematic approach to OPC UA information model design. *Procedia CIRP, 57*, 321-326.

Peralta, G., Iglesias-Urkia, M., Barcelo, M., Gomez, R., Moran, A., & Bilbao, J. (2017). Fog computing based efficient IoT scheme for the industry 4.0. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, 1-6.

Qi, Q., Zhao, D., Liao, T. W., & Tao, F. (2018). Modeling of cyber-physical systems and digital twin based on edge computing, fog computing and cloud computing towards smart manufacturing. In *ASME 2018 13th International Manufacturing Science and Engineering Conference*,

Rauschecker, U., Meier, M., Muckenhirn, R., Yip, A. L. K., Jagadeesan, A. P., & Corney, J. (2011). Cloud-based manufacturing-as-a-service environment for customized products.

Rentschler, M., Trsek, H., & Dürkop, L. (2016). OPC UA extension for IP auto-configuration in cyber-physical systems. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 26-31.

Ridwan, F., & Xu, X. (2013). Advanced CNC system with in-process feed-rate optimisation. *Robotics and Computer-Integrated Manufacturing, 29*(3), 12-20.

Ridwan, F., Xu, X., & Liu, G. (2012). A framework for machining optimisation based on STEP-NC. *Journal of Intelligent Manufacturing, 23*(3), 423-441.

Saez, M., Lengieza, S., Maturana, F., Barton, K., & Tilbury, D. (2018). A data transformation adapter for smart manufacturing systems with edge and cloud computing capabilities. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, 519.

Sauter, T. (2010). The three generations of field-level networks—Evolution and compatibility issues. *IEEE Transactions on Industrial Electronics, 57*(11), 3585-3595.

Schlechtendahl, J., Keinert, M., Kretschmer, F., Lechler, A., & Verl, A. (2015). Making existing production systems industry 4.0-ready. *Production Engineering, 9*(1), 143-148.

Schroeder, G. N., Steinmetz, C., Pereira, C. E., & Espindola, D. B. (2016). Digital twin data modeling with automationml and a communication methodology for data exchange. *IFAC-PapersOnLine, 49*(30), 12-17.

Seilonen, I., Tuovinen, T., Elovaara, J., Tuomi, I., & Oksanen, T. (2016). Aggregating OPC UA servers for monitoring manufacturing systems and mobile work machines. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1-4.

Swales, A. (1999). Open modbus/tcp specification. *Schneider Electric, 29*

Swetina, J., Lu, G., Jacobs, P., Ennesser, F., & Song, J. (2014). Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wireless Communications, 21*(3), 20-26.

Tao, F., Zhang, L., Venkatesh, V. C., Luo, Y., & Cheng, Y. (2011). Cloud manufacturing: A computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 225*(10), 1969-1976.

Tao, F., Zuo, Y., Da Xu, L., & Zhang, L. (2014). IoT-based intelligent perception and access of manufacturing resource toward cloud manufacturing. *IEEE Transactions on Industrial Informatics, 10*(2), 1547-1557.

ThomasNet. (2010). OPC foundation and MTConnect institute announce a memorandum of understanding. Accessed March 25, 2020. Retrieved from http://news.thomasnet.com/companystory/memorandum-of-understanding-announced-between-opc-mtconnect-584167

Thomesse, J. (2005). Fieldbus technology in industrial automation. *Proceedings of the IEEE, 93*(6), 1073-1101.

Tindell, K., Hansson, H., & Wellings, A. J. (1994). Analyzing real-time communications. In *Real-Time Systems Symposium, San Juan, Puerto Rico,*

UI Labs. (2017). O3 – operate, orchestrate, and originate. Accessed March 25, 2020. Retrieved from http://www.uilabs.org/project/o3-operate-orchestrate-and-originate-14-06-05/

Um, C., Lee, J., & Jeong, J. (2018). Industrial device monitoring and control system based on oneM2M for edge computing. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI),* 1528-1533.

Varghese, B., Wang, N., Nikolopoulos, D. S., & Buyya, R. (2017). Feasibility of fog computing. *arXiv Preprint arXiv:1701.05451,*

Vijayaraghavan, A., Sobel, W., Fox, A., Dornfeld, D., & Warndorf, P. (2008). Improving machine tool interoperability using standardized interface protocols: MT connect.

Vincent, S. J. (2001). FOUNDATION fieldbus high speed ethernet control system. *Fieldbus Inc,*

Wan, J., Chen, B., Wang, S., Xia, M., Li, D., & Liu, C. (2018). Fog computing for energy-aware load balancing and scheduling in smart factory. *IEEE Transactions on Industrial Informatics, 14*(10), 4548-4556.

Wang, L. (2013). Machine availability monitoring and machining process planning towards cloud manufacturing. *CIRP Journal of Manufacturing Science and Technology, 6*(4), 263-273.

Wang, L., Gao, R., & Ragai, I. (2014). An integrated cyber-physical system for cloud manufacturing. In *ASME 2014 International Manufacturing Science and Engineering Conference Collocated with the JSME 2014 International Conference on Materials and Processing and the 42nd North American Manufacturing Research Conference,*

Wikipedia. (2019). CC-link industrial networks. Accessed March 25, 2020. Retrieved from https://en.wikipedia.org/wiki/CC-Link_Industrial_Networks

Willner, A., Diedrich, C., Younes, R. '. B., Hohmann, S., & Kraft, A. (2017). Semantic communication between components for smart factories based on oneM2M. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA),* 1-8.

Wosnik, M., Kramer, C., Selig, A., & Klemm, P. (2006). Enabling feedback of process data by use of STEP-NC. *International Journal of Computer Integrated Manufacturing, 19*(6), 559-569.

Wu, C., Lin, F. J., Wang, C., & Chang, N. (2017). OneM2M-based IoT protocol integration. In *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, 252-257.

Wu, D., Greer, M. J., Rosen, D. W., & Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems, 32*(4), 564-579.

Wu, D., Liu, S., Zhang, L., Terpenny, J., Gao, R. X., Kurfess, T., & Guzzo, J. A. (2017). A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing. *Journal of Manufacturing Systems, 43*, 25-34.

Wu, D., Rosen, D. W., Wang, L., & Schaefer, D. (2015). Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation. *Computer-Aided Design, 59*, 1-14.

Wu, D., Thames, J. L., Rosen, D. W., & Schaefer, D. (2012). Towards a cloud-based design and manufacturing paradigm: Looking backward, looking forward. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 315-328.

Xiang, F., & Hu, Y. F. (2012). Cloud manufacturing resource access system based on internet of things. In *Applied Mechanics and Materials, 121*, 2421-2425.

Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing, 28*(1), 75-86.

Yin, L., Luo, J., & Luo, H. (2018). Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics, 14*(10), 4712-4721.

Yun, J., Teja, R. C., Chen, N., Sung, N., & Kim, J. (2016). Interworking of oneM2M-based IoT systems and legacy systems for consumer products. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, 423-428.

Yun, S., Kim, H., Shin, H., Chin, H. S., & Kim, W. (2019). A novel reference model for cloud manufacturing CPS platform based on oneM2M standard. *KIPS Transactions on Computer and Communication Systems, 8*(2), 41-56.

Zhang, C., Jiang, P., Cheng, K., Xu, X. W., & Ma, Y. (2016). Configuration design of the add-on cyber-physical system with CNC machine tools and its application perspectives. *Procedia Cirp, 56*, 360-365.

Zhao, F., Xu, X., & Xie, S. (2008). STEP-NC enabled on-line inspection in support of closed-loop machining. *Robotics and Computer-Integrated Manufacturing, 24*(2), 200-216.

Zhou, Z., Hu, J., Liu, Q., Lou, P., Yan, J., & Li, W. (2018). Fog computing-based cyber-physical machine tool system. *IEEE Access, 6*, 44580-44590.

Zuehlke, D. (2010). SmartFactory - towards a factory-of-things. *Annual Reviews in Control, 34*(1), 129-138.

Zurawski, R. (2014). *Industrial communication technology handbook.* CRC Press.

CHAPTER 3

MACHINE TOOL COMMUNICATION (MTCOMM) METHOD FOR CYBER MANUFACTURING

To address the heterogeneity issue of machine tools for developing cyber-physical manufacturing cloud systems, we developed the first Internet scale service-oriented communication method named Machine Tool Communication (MTComm) which enables both monitoring and operating of many heterogeneous types of machine tools over the Internet. MTComm allows machine tools to exchange manufacturing services and status data in XML format with other machine tools and with web and cloud applications through RESTful web services across the Internet. It is an application level communication method that uses a semantic ontology for representation of manufacturing machine configuration, status, and operational information at both machine and component level. MTComm is a significant improvement over the MTConnect, as it offers remote direct operation capabilities alongside status monitoring. MTComm has an agent-adapter based architecture that interacts with different machine tools using their own languages and exchanges manufacturing service information with client applications over the Internet. It not only facilitates communication between the Internet based applications and heterogeneous machine tools, but also supports interoperability between multiple types of machines located in different locations. MTComm is an application level communication method and is fundamentally based on HTTP. The messages in between are constructed in XML formats. This chapter describes the basics of MTComm method, its architecture, semantic ontology, services, and basic security measures.

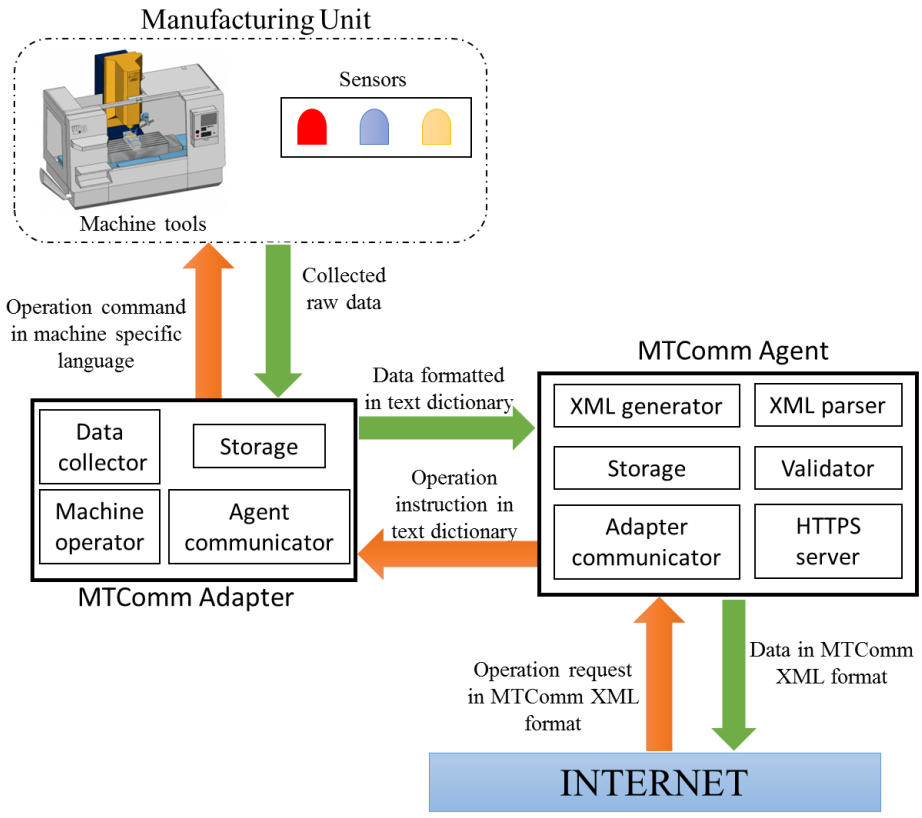


Figure 3. Architecture of MTComm method

3.1 Architecture of MTComm

MTComm interacts with different machine tools using their own languages and exchanges manufacturing service information with client applications over the Internet in a common XML message format (Sunny et al., 2017; Liu, Sunny & Shahriar, 2018). To achieve this, MTComm uses an agent-adapter based architecture, as shown in Figure 3. There are two components required to establish MTComm services – an adapter which communicates with the machine tool, and an agent which communicates with Internet-based client applications. The responsibilities of the agent and the adapter are mutually exclusive. Both the agent and the adapter provide a lot of flexibility for designers as the method only specify the way of

communication, not the way the agent and the adapter should be implemented. Therefore, MTComm is robust, scalable, and easy to be made compatible with various kinds of machines.

3.1.1 MTComm Adapter

The adapter is directly connected to the manufacturing resources. It works as a controller of the machine. Every bit of data flowing inward and outward of the machine goes through the adapter. The role of the adapter is very important as it not only controls and collects data from the machine, but also prevents from direct access to the machine from outside as it only accepts specific sets of commands. Each machine requires its own adapter as different machines' operation principles and mechanisms are different. Adapters are custom written because the meaning, units and values of data usually differ from machine tool to machine tool and device to device. The adapter can be implemented as a software application, or as a combination of software and hardware if the associated machine requires special hardware for data collection or operation. The interface between adapter and machine can be over TCP/IP, RS-232, RS-485, serial I/O etc. The adapter requires access to the machine's core system in such a way through which it can collect data from the machine and also operate the machine.

As shown in Figure 3, an MTComm adapter has four major modules. The 'data collector' module acquires raw status data from the associated sensors and machine tool in their own specific languages and formats by performing periodic queries. After receiving the data, the 'data collector' module converts the available data from its own format to a key-value pair-based text dictionary. The key refers to the name or type of the data and the value is the data itself. The 'machine operator' module is responsible for performing machine tool operation. It receives operation requests and associated parameters in a key-value pair-based text dictionary from the agent. Then it either sends a command to the machine tool in its own language to perform the

requested operation or executes a program, which may contain many operation commands, to complete the requested operation. For instance, if the operation is to move the X axis of a CNC machine to 15mm from its current position, a '`G0 X15`' command is sent to the machine. On the other hand, if the operation is to drill into a wooden block following a specific tool path, a G-code file with many lines of G-code commands are executed and sent to the machine tool. For file-based operations, associated files are stored in a 'storage' module. Other temporary files can also be stored if necessary. Size of the storage varies depending on type, size, and number of operational files. An 'agent communicator' module handles communication between the adapter and the agent. It sends the status data dictionary to the agent and also collects operational instruction dictionary from the agent. The communication interface between agent and adapter is implementation dependent and can be Ethernet, Bluetooth, Wi-Fi, USB etc.

For reporting the status of the machine, the adapter collects raw monitoring data directly from a machine toll in its own specific language. The numbers and types of data vary depending on what data the machine can provide and what data the manufacturer wants. Some common data types are machine's availability, the position of the axes, temperatures, progress rate, estimated time of the ongoing process etc. The data collection process is also dependent of machine type. For example, most 3D printers and CNC machines understand G-code, therefore their adapters send G-code based query and collect the responses. A typical 3-axis CNC machine adapter sends specific G-code based query '`M114`' to collect the current status of the machine axes. Upon successful acquisition, the data is converted into a simple key-value pair-based text dictionary (Figure 4). The importance of this conversion is twofold. Firstly, it guarantees that different types of data from different machines become consistent and are presented to the agent in one common structure. This facilitates the possibility of using one generic agent for all

machines. Secondly, the dictionary makes it easier for the agent to convert the available data into XML format. Once the dictionary is created, it is stored and forwarded to the agent for additional processing. The process of operating machine tools using the adapter is done in a reverse order. The adapter receives details of requested operations from the agent. To perform an operation, the adapter either sends corresponding commands to the machine or executes a program. For file

```
'availability'        :       'busy'
'x axis position'     :       '92.7'
'y axis position'     :       '-12.2'
'z axis position'     :       '152.3'
'nozzle temperature'  :       '212.5'
'heatbed temperature':        '59.2'
'process progress'    :       '30.0%'
'estimated time'      :       '55 minutes'
```

Figure 4. Example of key-value pair-based data dictionary created by MTComm Adapter

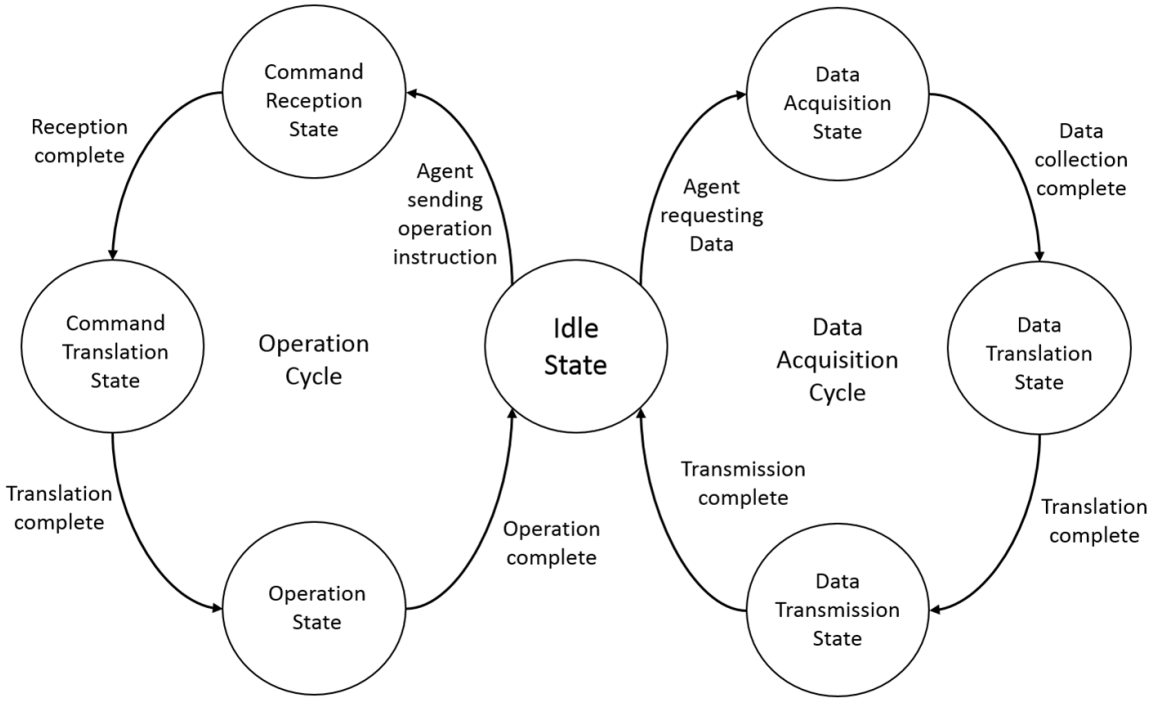


Figure 5. State transition diagram of MTComm Adapter

based operation, the adapter gets the model file from the agent. The data acquisition and operation of machines are performed in parallel. Data acquisition and operation cycles of an adapter are shown in Figure 5.

3.1.2 MTComm Agent

The agent is primarily a software program that bridges manufacturing machine tools and Internet-based client applications. It works as a translator on an http server. It consists of six major modules, as shown in Figure 3. Each agent hosts a RESTful 'HTTP server' that handles all the incoming and outgoing requests. The agent uses RESTful protocol – meaning it is stateless on the server side. Functionalities of agents are provided as RESTful web services. The server receives monitoring service requests from client applications over the Internet through HTTP GET method and sends corresponding XML responses. Operation requests are received through HTTP POST method. An 'adapter communicator' module handles communication with the adapter. It receives the status data dictionary from the adapter and sends the operational instruction dictionary to the adapter. An 'XML generator' module receives the status data dictionary and converts it into XML format using MTComm XML schemas. These schemas are based on the semantic ontology of MTComm, which is described in next section. Data in XML format is stored in a 'storage' module with a sequentially increasing number. If the storage becomes overfilled, the newest data replaces the oldest data. When a monitoring service request is received, the server collects corresponding XML message from storage and sends it as response to the client. For operation services, server module receives an XML message containing necessary information and parameters. A 'validator' module examines incoming operation requests to ensure that the request is in right format and the requested operation is compatible with the associated machine tool. This is to prevent the machine tool from damage

due to erroneous and malicious operation requests. The verification process can either be simple XML verification or very complex, depending on the developer's intent. Security measures to detect malicious requests and intrusion attacks can also be added to the validator module. If the incoming request is appropriate, it is validated and forwarded to an 'XML parser' module which uses MTComm schemas to parse the operation request and creates a key-value pair based text dictionary containing associated information and parameters, if any, about the requested operation. The 'adapter communicator' module sends this text dictionary to the adapter for further processing. The agent also sends back acknowledgments and error messages to the client application. As all MTComm agent operate in exactly the same way, one agent can be connected to multiple MTComm adapters. Additional functionality modules can be added in the agent if necessary. Both data acquisition and operation cycles of an agent are shown in Figure 6.

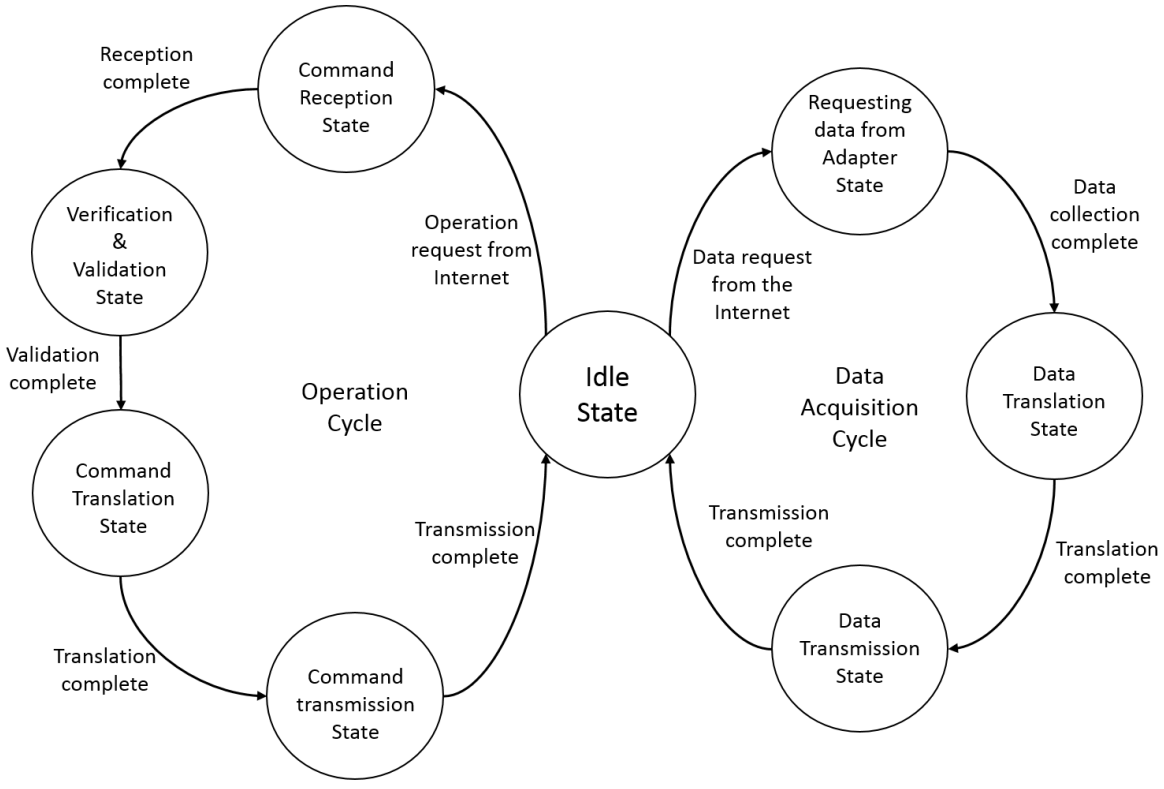


Figure 6. State transition diagram of MTComm Agent

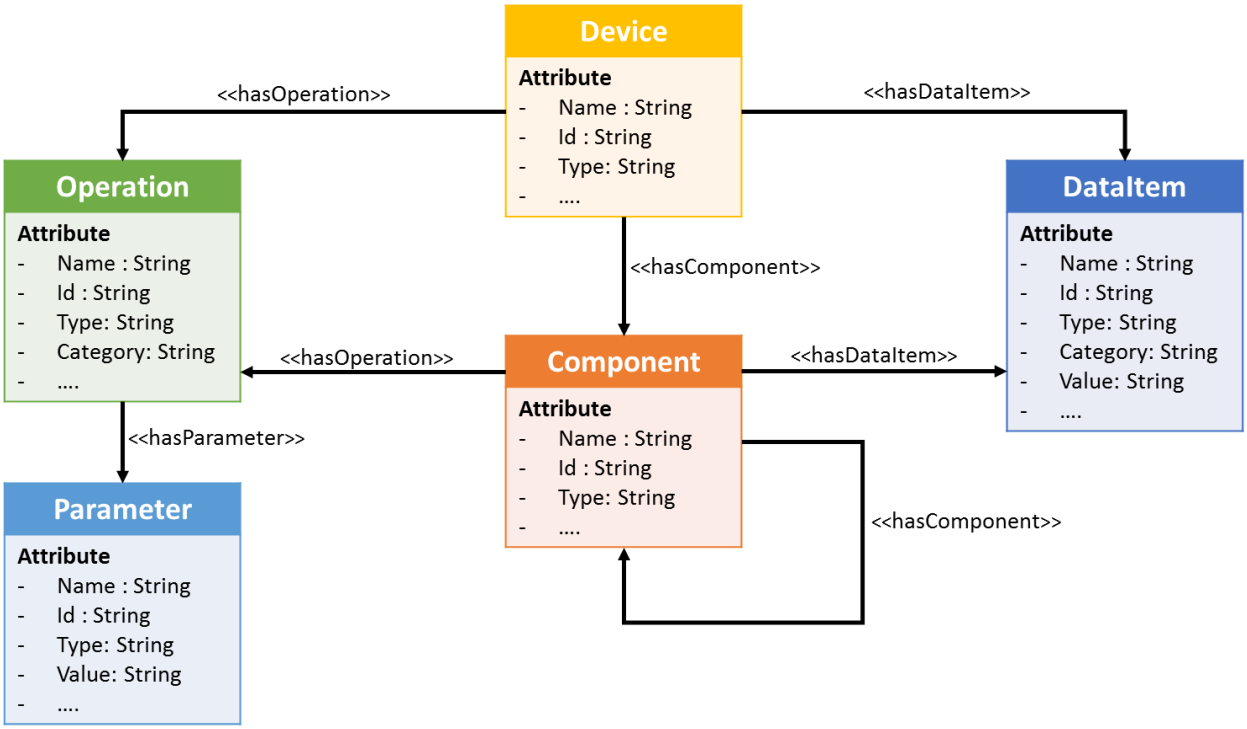


Figure 7. MTComm semantic ontology

3.2 MTComm Semantic Ontology

MTComm uses a semantic ontology to represent a manufacturing machine tool, its parts, connected equipment such as sensors, actuators, controllers etc., and all associated information. The monitoring data of a machine tool may include but not limited to configuration data, structural or organizational information, diagnostics, manufacturing service data, and manufacturing process data. Operational data include operation instructions and parameters. MTComm ontology represents structure of machine tools and their components and contains detailed information of their data and operations in a top-down hierarchical fashion. This ontology is regarded as an upper ontology. Upper ontology is a lightweight ontology limited to concepts that are abstract and generic enough to address a broad range of objects in the domain of interest (Ameri & Dutta, 2008). Therefore, while providing some level of standardization, an

upper ontology has sufficient flexibility and extendibility necessary for the conceptualization of highly heterogeneous and dynamic domains. The MTComm ontology defines the semantics meanings of machine components and of all the data that will be communicated from and to manufacturing machines. The semantic ontology is illustrated in Figure 7. As the figure shows, this ontology provides information about machine tool's status and operations not only at machine tool level, but also at its component and sub-component levels. It provides a foundation for translation of data sent from/to machine tools. The ontology enables hierarchical representation of machine tool information and thus makes the transformation of data to XML format easier.

In MTComm ontology, each machine tool is represented as a *device*. A *device* has *attributes* that provides general identifying information of the machine tool such as device name, device identifier number, device type, model, manufacturer name etc. A *device* is composed of one or more *components*. Parts of a machine tool and other connected equipment such as sensors, actuators, controllers etc. are represented by *components*. Each c*omponent* can have *components* of its own, if required. *devices* and *components* contain *dataitems* and *operations.*
A *dataitem* refers to a piece of status information that can be collected from a machine tool or its components. It provides a detailed description for each piece of data that is collected from a device - the type of data being collected, an array of optional attributes that further defines that data, and the value of the data. In most cases, manufacturing data is of two forms – a value (numeric or alphabetic) and a health status. So *dataitems* are divided into three categories, described below –

1) SAMPLE – A SAMPLE is the reading of the numerical value of a continuously variable or analog data value that can be measured at any point-in-time and will

always produce a result. The data provided for a SAMPLE category *dataitem* is always a floating-point number or integer with an infinite number of possible values. Examples of such data values are position of a linear X axis, temperature of a machine tool part etc.

2) EVENT – An EVENT is a data value representing a discrete piece of information with limited number of possible values. This category does not have intermediate values that vary over time, as does SAMPLE. An EVENT is information that, when provided at any specific point in time, represents the current state of a machine tool or its parts. There are two types of EVENT – those representing state, with two or more discrete values; and those representing messages that contain plain text data. An example of a state type EVENT is the value of the data item DOOR_STATE which can be OPEN, UNLATCHED, or CLOSED. An example of a message type EVENT is the value for a *dataitem* PROGRAM which can be any valid string of characters.
3) CONDITION – A CONDITION is a data value that communicates information about the health of a device and its ability to function. A valid value for a data item in the category CONDITION can be one of UNAVAILABLE, NORMAL, WARNING, or FAULT. A *dataitem* of category CONDITION may report multiple values at one time, unlike SAMPLE or EVENT.

An *operation* represents a manufacturing activity or process that can be performed by a machine tool or its components. Similar to d*ataitem*, it includes an array of optional attributes that provides detailed description of the associated manufacturing task. Most manufacturing processes require input arguments, e.g. material type to be used, required temperature etc., and often have specific constraints and conditions, e.g. minimum and maximum limit of extruder

temperature of a 3D printer, maximum spindle speed of a CNC machine etc. Therefore, an *operation* can have multiple *parameters* that contains operational arguments, constraints, or input values. *operations* are divided into two categories based on their level of association –

1) ACTION – An ACTION refers to a manufacturing task that is associated with or performed by a specific *component* of a machine tool. Examples of this category of *operation* are moving a linear X axis to a specific co-ordinate, change the temperature of a machine tool part, make a robotic hand to grab or release etc.
2) JOB – A JOB is a manufacturing process that is done by a machine tool as a whole, or in other words, involves multiple machine tool parts or systems. This category of *operation* is usually associated with the *device* itself, instead of its *components*. A JOB may involve running a batch of process commands, executing a machine specific program, or performing multiple ACTIONs sequentially. Often it involves use of a machine language program files like G-code files, .CSV files with motion path co-ordinates etc., or CAD model files. Example of such tasks are printing a 3D object in a 3D printer, drilling process in a CNC milling machine, autonomous navigation of a mobile robot etc.

Dataitems and *operations* are grouped based on their logical organization, instead of their physical organization. All *components*, *dataitems*, *operations*, and *parameters* have their own *attributes*. All *attribute* values are strings. The number of elements of each type depends on a machine tool's structure, configuration, and capabilities. MTComm ontology is simple yet generic and robust enough to be applied to heterogeneous machine tools and manufacturing systems. Such hierarchical representation of a machine tool with MTComm eases the virtualization process of a machine tool and its capabilities in a cloud environment.

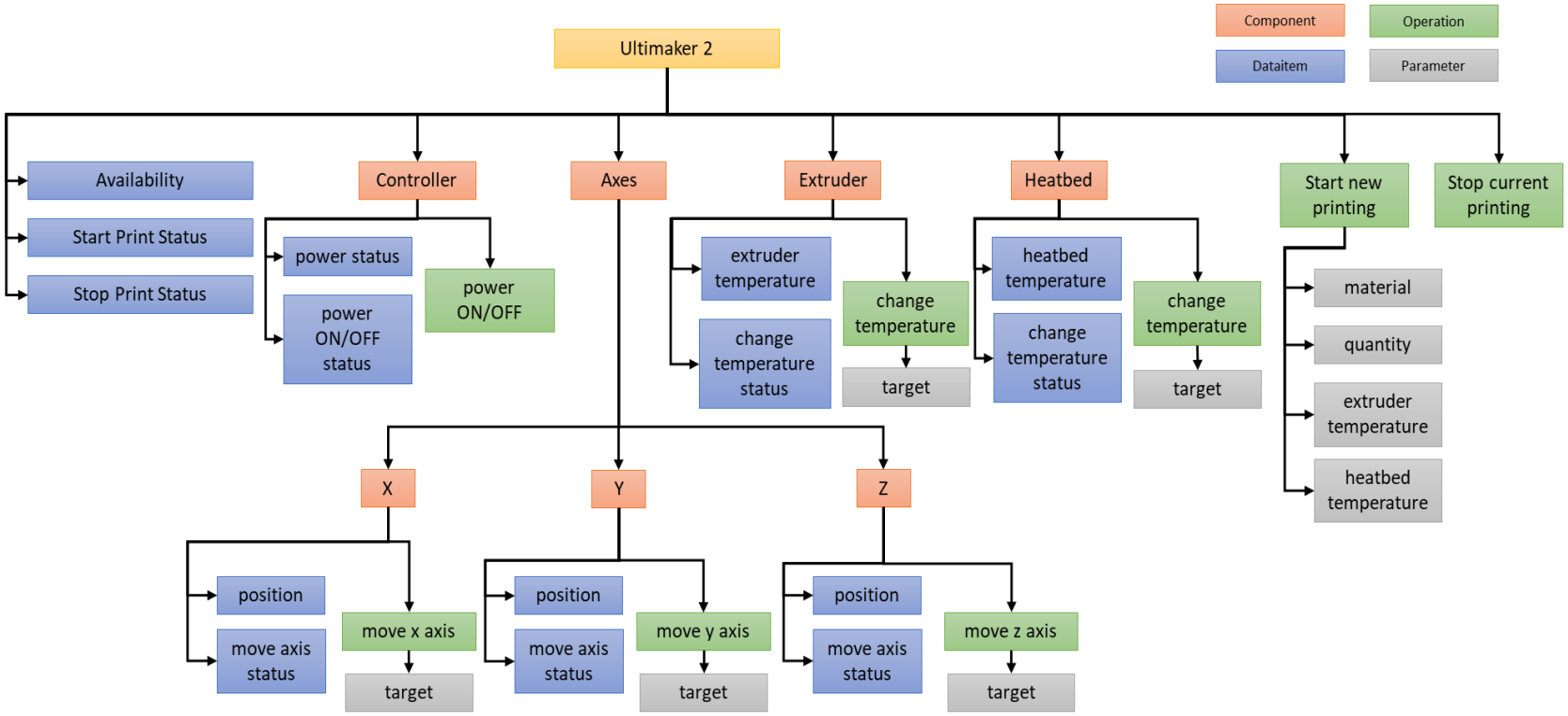


Figure 8. Hierarchical representation of ‘ Ultimaker 2’ using MTComm ontology

Figure 8 illustrates an example of hierarchical representation of a 3D printer - 'Ultimaker 2' using MTComm semantic ontology. Here the printer has three device level *dataitems* and two device level *operations*, one of which requires multiple *parameters*. It has four *components* – controller, axes, extruder, and heatbed. The axes *component* is composed of three additional *components*. Each *component* has its own *dataitems* and *operations*.

3.3 MTComm Services

MTComm enables manufacturing machine tools to exchange information for both data acquisition and operation through RESTful web services over the Internet. As of now, MTComm provides six services – `probe`, `current`, `sample`, `operate`, `error`, and `notification`. This section includes details of these MTComm services and how these can be used for cyber manufacturing. All services are provided as RESTful web services by MTComm agents via HTTP servers. URLs of these services are created by adding the service name after the address of an MTComm agent, e.g. "`https://10.5.1.7:1080/probe`". Each service has its own message in XML format. The structure, headers, and elements of these XML messages differ from one service to another. Details about these services are discussed below.

3.3.1 `Probe`

`Probe` service provides structural and configuration information of a machine tool including its *components* and all available *dataitems* and *operations*. It depicts a *device's* detailed organization in a hierarchical representation using MTComm ontology, as described is Section 3.2. Client application requests `probe` service via HTTP GET method. The response XML message is divided into two sections – 'Header' and 'Device'. 'Header' contains protocol related information like creation time, version, sender etc. and 'Device' provides the descriptive

```
<Device id="Ultimaker2" uuid="P2673" name="Ultimaker2 3D Printer" type="3D printer">
    <DataItems>
        <DataItem category="EVENT" id="avail" name="availablility" type="AVAILABILITY"/>
    </DataItems>
    <Operations>
        <Operation id="startPrintingJob" category="JOB" name="Start new job" type="PRINT">
            <Parameters>
                <Parameter id="material" name="Material type" type="MATERIAL"/>
                <Parameter id="targetExtTemp" name="Target Extruder Temp"
                    units="CELSIUS" type="TEMPERATURE">
                    <Constraints>
                        <Minimum>0</Minimum>
                        <Maximum>215</Maximum>
                    </Constraints>
                </Parameter>
                <Parameter id="targetBedTemp" name="Target Bed Temp"
                    units="CELSIUS" type="TEMPERATURE">
                    <Constraints>
                        <Minimum>0</Minimum>
                        <Maximum>65</Maximum>
                    </Constraints>
                </Parameter>
                <Parameter id="objectName" name="Printing object name"/>
            </Parameters>
        </Operation>
        <Operation id="stopJob" category="JOB" name="Stop current job" type="PRINT" />
    </Operations>
```

Figure 9. Example of a partial `probe` service response message

information of each machine tool served by the agent. As a machine's physical composition and capabilities do not change frequently, the response of `probe` for a particular machine tool remains static mostly. Figure 9 presents a portion of an example `probe` response message of a 3D printer. A complete response message from a 3D printer's agent in given in Appendix A.1.

3.3.2 `Current`

MTComm `current` service provides the most recent values of *dataitems* at the time when the service is requested via HTTP GET method. It basically provides a snapshot of a machine tool's status at a certain time. The response message consists of two sections – 'Header' and 'Streams'. The data values are given as CDATA of corresponding XML elements. Sequence

number is a significant *attribute* each element in 'Streams' section, as it helps to differentiate between machine data captured at different instances and times. It is also used to fetch status of

```
<DeviceStream name="Ultimaker2" uuid="P2673">
  <ComponentStream component="Device" componentId="Ultimaker" name="Ultimaker2">
    <Events>
      <Availability dataItemId="availability" name="availablility" sequence="66"
          timestamp="2017-02-09T01:21:46">BUSY</Availability>
    </Events>
    <Actions>
      <Printing operationId="startJobStatus" name="Start new job Status"
          sequence="66" timestamp="2017-02-09T01:21:46">ONGOING</Printing>
      <Printing operationId="stopJobStatus" name="Stop current job Status"
          sequence="66" timestamp="2017-02-09T01:21:46">AVAILABLE</Printing>
    </Actions>
  </ComponentStream>
  <ComponentStream component="Linear" componentId="x" name="X">
    <Samples>
      <Position dataItemId="xPos" name="Actual X Position" sequence="66"
          subType="ACTUAL" timestamp="2017-02-09T01:21:46">105.3</Position>
    </Samples>
    <Actions>
      <Position operationId="moveXStatus" name="Move X Axis Status"
          sequence="66" timestamp="2017-02-09T01:21:46">UNAVAILABLE</Position>
    </Actions>
  </ComponentStream>
  <ComponentStream component="Linear" componentId="y" name="Y">
    <Samples>
      <Position dataItemId="yPos" name="Actual Y Position" sequence="66"
          subType="ACTUAL" timestamp="2017-02-09T01:21:46">-95.6</Position>
    </Samples>
    <Actions>
      <Position operationId="moveYStatus" name="Move Y Axis Status" sequence="66"
          timestamp="2017-02-09T01:21:46">UNAVAILABLE</Position>
    </Actions>
  </ComponentStream>
  <ComponentStream component="Linear" componentId="z" name="Z">
    <Samples>
      <Position dataItemId="zPos" name="Actual Z Position" sequence="66"
          subType="ACTUAL" timestamp="2017-02-09T01:21:46">-216.9</Position>
    </Samples>
    <Actions>
      <Position operationId="moveZStatus" name="Move Z Axis Status" sequence="66"
          timestamp="2017-02-09T01:21:46">UNAVAILABLE</Position>
    </Actions>
  </ComponentStream>
  <ComponentStream component="Sensor" componentId="extruder" name="Extruder">
    <Samples>
      <Temperature dataItemId="extruderTemp" name="Extruder Temp Sensor"
          sequence="66" timestamp="2017-02-09T01:21:46">209.7</Temperature>
```

Figure 10. Example of `Current` service response message

the machine at a particular moment. MTComm also allows the use of attributes as path parameters in the service URL to perform element specific queries and actions. This feature enables to acquire data or perform operation of a particular *component*. For instance, the service URL to collect only the value of the extruder temperature of a 3D printer is as below –

```
http://10.5.55.7:10090/current?path=//Sensor//Dataitem[@id="extruderTemp"]
```

This feature allows to create service URLs using the ontology dynamically based on user requests, instead of storing every service URL in database. It is also possible to collect data of a specific moment using timestamp or sequence number. For example –

```
http://10.5.55.7:10090/current?path=//Axes//Dataitem[@type="POSITION" and @subtype="ACTUAL"]&at=54573
```

This service request collects positions of the axes with sequence number 54573. Figure 10 contains a portion of an example `current` request response message of a 3D printer during a printing JOB. The complete example is available in Appendix A.2.

3.3.3 `Sample`

`Sample` service gives a list of values of *dataitems* for a certain time interval. The response message is similar to `current` service, following the same structure. The only difference is that `sample` provides multiple continuous values of *Dataitems* instead of just one. Sequence number can be used to retrieve data of a particular time range, specifically useful to collect, store, and analyze historical. If no parameter is specified in the service request, `sample` responds with values of a default time interval.

3.3.4 `Operate`

`Operate` service enables performing machine tool operations remotely over the Internet. A machine tool *operation* is requested from the client application by sending an `operate` request in XML format to the machine's agent via HTTP POST method. An `operate` request message includes details of the requested operation and associated parameters, if any. The primary XML elements describing the operation is termed based on its category and type, as defined in its `probe` message. Figure 11(a) depicts an example of `operate` request of a 3D printing JOB. An `operate` request may contain information for multiple *operations* where each *operation* element has a sequence number attribute referring to the order of execution. Figure 11(b) shows an example of such `operate` request.

When an MTComm agent receives an incoming `operate` request, its 'validator' module conducts a two-step verification process - checks if the XML message is a valid one, and then determines whether the requested operation is meant for and supported by the machine tool. This second step is very crucial as it prevents the machine from performing potentially harmful operations and damaging itself. The agent uses the `probe` service to verify the operational information. A *parameter* element may include *constraints* (as shown in Figure 9 and Appendix A.1) that defines limitations and range of accepted values. If the operation or any of the parameters does not match with `probe` information, the request is rejected. For example, if a machine's X axis can move to a maximum position of 200 mm and its current position is 120 mm, an operation request for moving the x axis 100 mm more will be rejected by the agent's 'validator' module. Once the request is validated and verified, the agent parses the message and sends the operation information to the adapter. The adapter than initiates and completes the operation by sending appropriate commands to the machine tool. Figure 12 illustrates the

```
<MTCommOperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../Schemas/MTCommOperations_0.3.xsd">
    <Header bufferSize="10" instanceId="1" creationTime="2017-01-18T12:00:00"
            sender="Ultimaker2" version="0.1"/>
    <Operations>
        <Device uuid="P2673" name="Ultimaker2 3D Printer">
            <DeviceOperation id="P2673" name="Ultimaker2 3D Printer">
                <Jobs>
                    <Printing operationId="startJob" name="Start new job" sequence="1"
                              timestamp="2017-01-18T05:45:40">
                        <Parameters>
                            <Material id="material" name="Material Type"
                                timestamp="2017-01-18T05:45:40">PLA</Material>
                            <Temperature id="targetExtTemp" name="Target Extruder temperature"
                                units="CELSIUS" timestamp="2017-01-18T05:45:40">210</Temperature>
                            <Temperature id="targetBedTemp" name="Target Bed temperature"
                                units="CELSIUS" timestamp="2017-01-18T05:45:40">58</Temperature>
                            <Object id="objName" name="Object Name"
                                timestamp="2017-01-18T05:45:40">Clip</Object>
                        </Parameters>
                    </Printing>
                </Jobs>
            </DeviceOperation>
        </Device>
    </Operations>
</MTCommOperations>
```

(a) Single operation

```
<MTCommOperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../Schemas/MTCommOperations_0.2.xsd">
    <Header bufferSize="10" instanceId="1" creationTime="2017-01-18T12:00:00"
        sender="Ultimaker2" version="0.1"/>
    <Operations>
        <Device uuid="cxz" name="Utlimaker 3D Printer">
            <ComponentOperation component="Linear" componentId="x" name="X">
                <Actions>
                    <Position operationId="moveX" name="Move X Axis" sequence="2"
                        units="MILLIMETER" timestamp="2017-01-18T06:16:39">-50.0</Position>
                </Actions>
            </ComponentOperation>
            <ComponentOperation component="Linear" componentId="y" name="Y">
                <Actions>
                    <Position operationId="moveY" name="Move Y Axis" sequence="3"
                        units="MILLIMETER" timestamp="2017-01-18T06:16:39">-50.0</Position>
                </Actions>
            </ComponentOperation>
            <ComponentOperation component="Sensor" componentId="extruder" name="Extruder">
                <Actions>
                    <Temperature operationId="changeExtTemp" name="Change Extruder Temperature"
                        sequence="1" units="CELSIUS" timestamp="2017-01-18T06:16:39">110.0</Tem
                </Actions>
            </ComponentOperation>
        </Device>
    </Operations>
</MTCommOperations>
```

(b) Multiple operations

Figure 11. Examples of `operate` request message

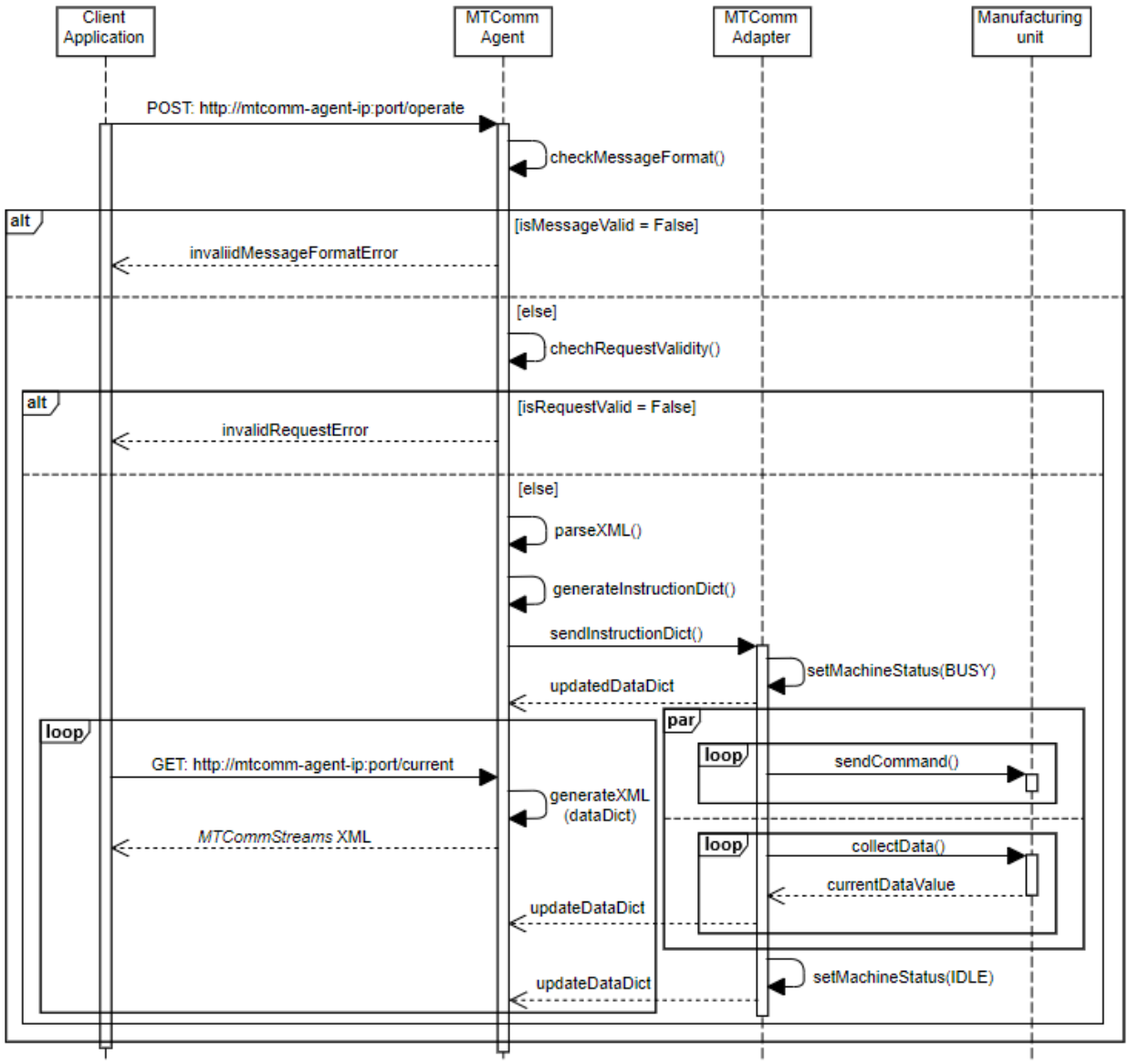


Figure 12. Sequence diagram of an `operate` service execution

execution of an `operate` service in a sequence diagram. As illustrated, the adapter can send commands to a machine and also collect monitoring data at the same time. The diagram considers a normal *operation* only, meaning no failure occurs during the execution. In case of multiple operations, the agent uses sequence numbers attribute to send operational details to the agent in the order requested by the client application.

3.3.5 `Error`

MTComm `error` service is used to send error reports when the agent cannot recognize or handle the incoming request. This service is generated automatically by an MTComm agent's HTTP server upon failure and cannot be requested externally. If the request is malformed or the agent cannot return any XML response for some internal error, `error` service generates a response message stating the type and details of the error for the client application. The response XML message contains one *Error* element for every error that has occurred. An *Error* element has an error code as an attribute and the CDATA of the element is the details of the error in plain text. For example, if a `current` request is made with wrong path and invalid sequence number, its `error` service response message will contain two *Error* elements – one for `invalid_path` and one for `out_of_range` error.

For an incoming `operate` service request, if an agent's 'validator' module fails to validate the request or find invalid parameters that may result in machine failure, an `ack` message with appropriate error code and description is generated.

3.3.6 `Notification`

MTComm `notification` service is used to send operational acknowledgments and notifications to the client applications. Using this service, an MTComm agent notifies a client application about all messages associated with an MTComm *operation* including acknowledgements, error messages, periodic status updates etc. For an incoming `operate` service request, if an agent's 'validator' module fails to validate the request or find invalid parameters that may result in machine failure, an `ack` message with appropriate error code and description is generated and sent to the client application. Similar to `error` service,

`notification` service is also auto generated and cannot be requested by client applications. Their response message structure is also alike – a *Notification* element with a notification code as an attribute and the CDATA with details of the notification in plain text such as 'request received', 'operation completed', 'material not found' etc. However, `error` and `notification` each serves separate purpose. For example, if an `operation` is requested via an invalid XML message, `error` message is generated and sent to the client application. If the requested operation is halted due to overheating of a machine component, this fault is reported via a `notification` message.

3.4 Security measures in MTComm

A major concern of implementing cyber manufacturing systems is the assurance that proprietary information about the intellectual property owned by the organization or information about the company's operations is available only to authorized individuals. It is also important to avoid security disasters for machine tools at factory floor level. Performing malicious operations or operations with erroneous parameters can damage, even destroy machine tools. Monitoring and operating machine tool over the Internet involve sharing information in the form of detailed run-time operations and critical hardware controls. For general acceptance of MTComm, the secrecy of the proprietary information must be properly maintained. MTComm can be implemented in a vast variety of manufacturing environments and therefore security has to be tailored for corresponding situation. The focus of this research is general security specifications, which should serve as guidelines that cover most of the basic security requirements. As all MTComm services except `operate` are read-only and thus in no way can harm machine tools, the security mechanisms are primarily related to the `operate` service.

Client application uses username-password based authentication to make sure only authorized users can use the application. Because of the agent-adapter architecture of MTComm, a client application only gets indirect access to a machine through its agent, without violating factory floor security. The adapter operates a connected machine tool and the agent makes sure that only valid operational commands and parameters are sent to the adapter. All communications between client application and a machine go through the agent's HTTP server. To strengthen security, an agent can include HTTPS server, instead of HTTP, as the former encrypts all messages with SSL (Secure Socket Layer) encryption. Connection between a client application and the agent's server is established by exchanging SSL certificate and private key. Therefore, only the client application and the agent server know how to decrypt the messages and access the data. This reduces the possibility of interception-based attacks like eavesdropping and man-in-the-middle attack. As described in Section 3.3.4, an MTComm agent performs a two-step verification with every incoming request to make sure that the request is valid, and the requested operation is supported by the associated machine tool. Once the agent is convinced that all operational information is correct and the machine can perform this operation, only then the request is forwarded to the adapter. Also, the agent does not have direct access to the machine, only the adapter can send commands directly to the machine. The adapter only accepts a very specific list of operational instructions, set by the developer, from the agent. Therefore, even if the agent's server is compromised somehow, the adapter would not execute any command it does not recognize.

More security measures can be added if required. For example, a centralized local server can be added as a gateway for factories with many machine tools. All communication between agents and client applications will go through this local server. It may include security components

like session handling, token-based authentication, anti-malware software etc. It physically separates the machine tools in factory floor by using segmented networks and includes protective measures against different types of attacks.

3.5 References

Ameri, F., & Dutta, D. (2008). A matchmaking methodology for supply chain deployment in distributed manufacturing environments. *Journal of Computing and Information Science in Engineering*, *8*(1).

Liu, X., Sunny, S. M. N. A., & Shahriar, M.R., (2018). Semantic Ontology-Based Internet Scale Communication Method of Machine Tools for Providing Remote Operational Services. *U.S. Patent Application 16/020,795*.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

CHAPTER 4

REMOTE AND COLLABORATIVE MANUFACTURING IN CYBER-PHYSICAL MANUFACTURING CLOUD

As mentioned in Chapter 1, the primary objective of developing MTComm was to facilitate remote monitoring and operation services in a scalable service-oriented cloud-based cyber-physical manufacturing system managing different types of physical resources. Therefore, the first application of MTComm was to design and develop these capabilities for such a system as well as implementing a testbed for experimentations and evaluation. During development, it was also noticed that these functionalities could be used to establish intercommunication between manufacturing machine tools and enable collaborative manufacturing processes where multiple machines can work together towards a common objective. This chapter describes the methodologies for achieving both remote and collaborative manufacturing using MTcomm in a cyber-physical manufacturing cloud. It also discusses experimental outcomes to evaluate MTComm's performance and feasibility for cyber manufacturing.

4.1 Cyber-Physical Manufacturing Cloud (CPMC)

MTComm was designed and developed primarily to enable exchange manufacturing monitoring and operation services remotely over the Internet in a paradigm named cyber-physical manufacturing cloud (CPMC) that integrates the concepts of cyber-physical system and cloud manufacturing (Sunny et al., 2017). Figure 13 illustrates a conceptual framework of CPMC (Liu et al., 2016, 2017). Users interact with CPMC to request manufacturing services using multiplatform applications from desktops or mobile devices via HTTP. Cloud manufacturing services and applications are hosted in cloud servers. Communication between cloud services and manufacturing machine tools are done using MTComm via controllers and local servers. A

controller component consists of MTComm adapter and agent programs. Local servers work as gateways between the cloud servers and the machine networks and are optional. Manufacturing resources inside the factory floor can have any type of network and infrastructure, as the cloud does not communicate with machine tools directly. The controller translates in-between messages (XML to machine language and vice versa) using MTComm. Based on MTComm `probe` service data, a virtual copy or digital twin of each machine tool is generated in the cloud. Cloud applications interact with these digital twins instead of directly connecting to machine tools. Users can thus monitor and control machines remotely in CPMC. CPMC has a four layered architecture – Application layer, Core cloud layer, Virtualization layer, and Resource layer. MTComm is the core technology of virtualization layer and bridges cloud layer with resource layer.

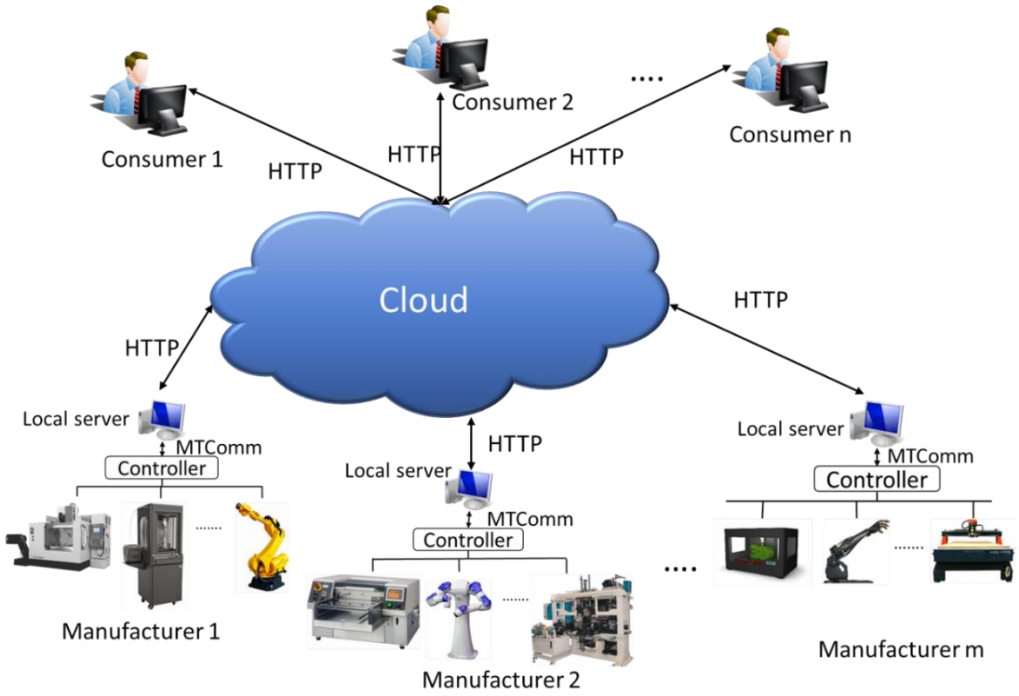


Figure 13. Conceptual framework of CPMC (Liu et al. 2017)

4.1.1 Remote monitoring and operation of machine tools using MTComm

In a cyber-physical cloud environment, virtualization of resources is very crucial. In the CPMC, each machine tool is represented virtually by publishing its characteristics and capabilities through MTComm's `probe` service. A `probe` response XML message provides all available information about the machine tool including machine configuration data, characteristics data, details (name, identifier, type, units etc.) of every *dataitem* and *operation* associated with the machine, operational parameters etc. Also, the semantic ontology of MTComm provides a hierarchical representation of a machine tool and its component in a top-down fashion. The virtualization service of CPMC acquires these data from a machine's agent through `probe` and creates and stores an interactive virtual copy or digital twin of the machine in the cloud. Status information of a machine tool are periodically collected via `current` service and synchronized with corresponding digital twin in real-time. This virtualization process is usually done when a machine tool is registered and added to the cloud for the first time. MTComm service URLs of a machine tool is also stored in the cloud database at this stage and linked to corresponding elements of the digital twin. Whenever a service of a manufacturing machine tool is required, cloud applications in CPMC interact with its digital twin which then communicates with the machine's agent and responds accordingly.

Each machine tool in the CPMC is connected to a controller where its MTComm agent and adapter programs are deployed. A controller can be any small-scale computing device or system, such as Arduino, raspberry pi, FPGA development board etc., with sufficient processing power and storage. The adapter program continuously acquires status data from the machine tool, updates the values in a key-value pair based text dictionary, and forwards it to the agent. The agent converts incoming data in dictionary to XML messages and stores them sequentially

in its buffer storage. It also publishes MTComm RESTful services via its HTTP server. When a cloud application in CPMC receives a monitoring request from a user, it queries the digital twin of corresponding machine tool. The digital twin sends a `current` (or `sample`, depending on type of request) service request to the agent via HTTP GET method. The agent responds with the most recent XML message, which is forwarded to the cloud application. The XML message is then parsed, and data values are extracted from it and presented to the user in tabular format. Figure 14 shows an example response of a monitoring request to observe a 3D printer from users in CPMC client application.

To perform an operation of a machine tool upon user request in CPMC, a cloud application generates an `operate` request XML message based on its `probe` response message

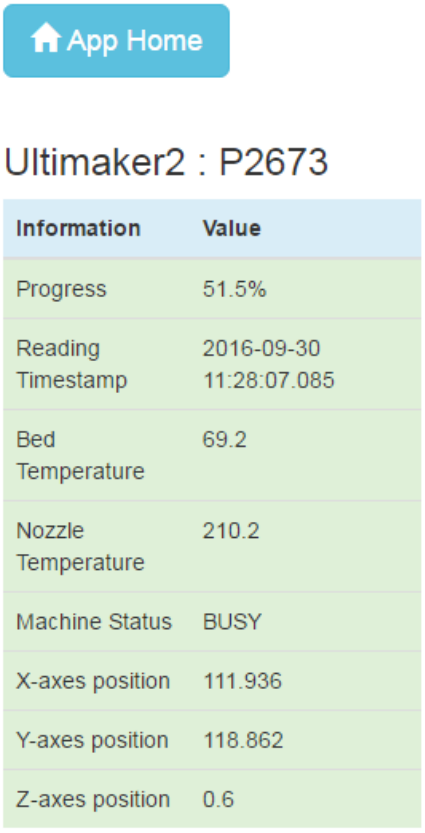


Figure 14. Example response of a monitoring request in CPMC client application

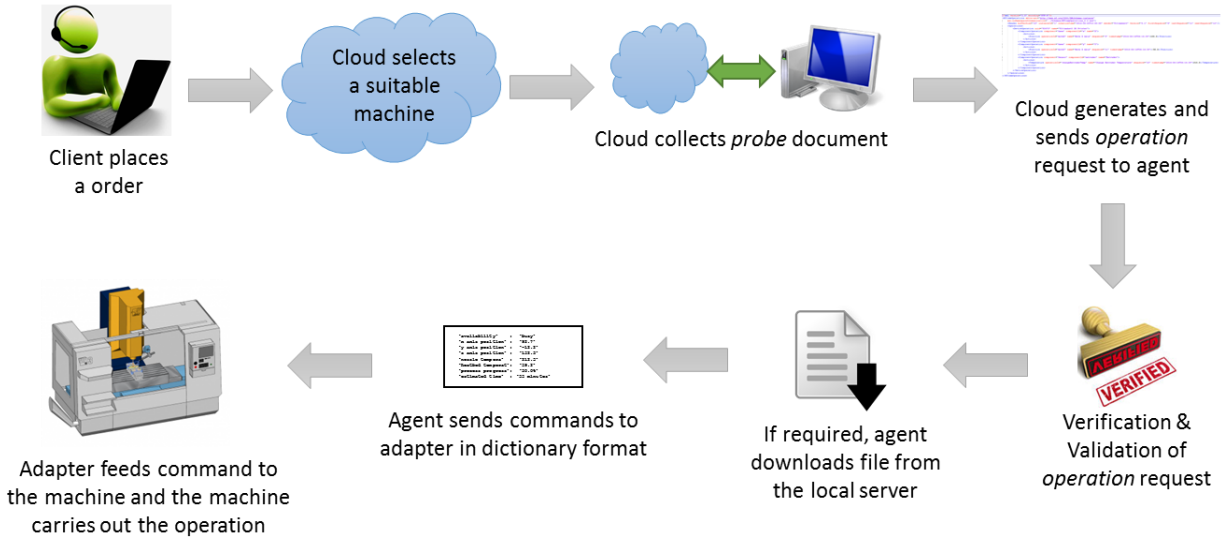


Figure 15. Operation procedure in CPMC using MTComm

and user inputs (parameters, arguments, constraints etc. of requested operation given by the user), which is then forwarded to the digital twin to call corresponding `operate` service. The XML message is sent to the machine tool's agent via HTTP POST method. The agent stores the message, verifies and validates it, extract necessary information upon success, and forwards instructions to the adapter in a text dictionary. The adapter takes parameter values and initiate the operation by sending appropriate commands to the machine tool. The agent sends a `notification` message with acknowledgment of starting the operation or with error description in case of a failure. As described above, some manufacturing processes require model or program files. How these files are made available to MTComm adapters is determined by manufacturers; MTComm does not specify any particular mechanism for this. Files can be manually stored inside controllers and users are only allowed to choose from already existing files. Although it limits flexibility of available manufacturing operation choices, this method strengthens security and privacy. Another way to do this is to store the files in local servers or in the cloud and have the controllers download them as necessary by providing the download URL

as a *parameter* in `operate` request. In this case, however, manufacturers are responsible to establish secured and encrypted communication channel between local server or cloud and controllers for downloading the required file. Figure 15 shows the operation procedure in CPMC using MTComm.

4.1.2 Implementation of a CPMC testbed using MTComm

To evaluate the performance and effectiveness of CPMC and MTComm, a fully operational CPMC testbed was developed, as shown in Figure 16. The CPMC testbed consists of three separate testing sites. Two sites are located at the University of Arkansas (Uark) and another is located at the Missouri University of Science & Technology (MST). One Uark site has three manufacturing machines – an X-Carve CNC machine from Inventables, a small robotic arm named uArm from UFactory, and a RepRap (Jones et al., 2011; Bowyer, 2014) 3D printer named Ultimaker 2, while the other Uark site is consisted of two machine tools – a RepRap 3D printer named Bukito from Bukobot and another uArm. The site in MST has a uArm robotic arm and Bukito 3D printer. All machines used are open-source. Each machine tool is connected to a Raspberry Pi (RPi) 3 which works as an MTComm controller of that machine tool. MTComm agent and adapter programs, developed in `python`, of a machine tool are deployed in the associated RPi. RPis have network access via Ethernet and Wi-fi. Connection types between the RPis and the machines are varied, e.g. via USB, Bluetooth, or wireless network card. Using a RPi as a controller offers several advantages. One advantage is its low cost, small size and low power consumption rate. The newest RPis contain enough memory space to provide adequate buffer storage needed by the agent to hold data. It has enough computation power to run multiple agent and adapter programs for several machines simultaneously. For simplicity, one RPi is used for one machine tool in the testbed. Besides it provides a unique scalable and plug-n-play feature

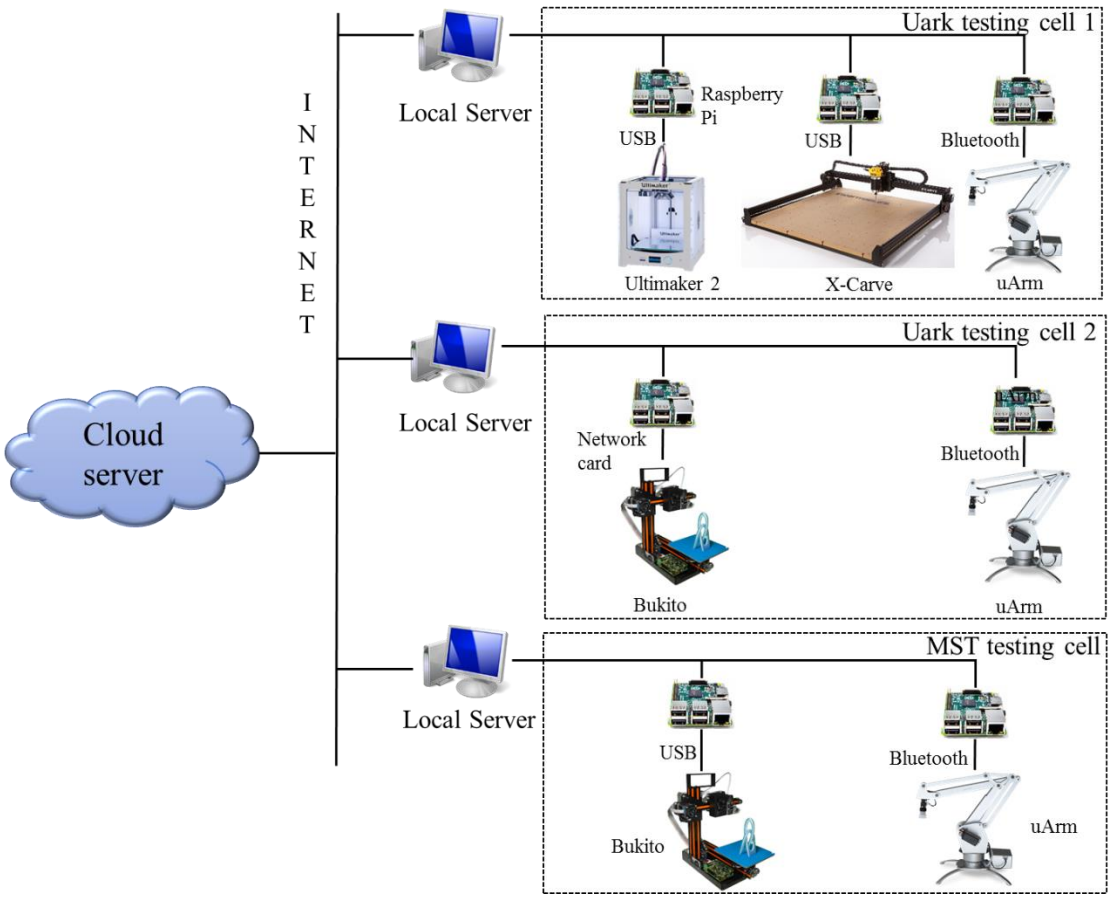


Figure 16. The implemented CPMC testbed

to the system. A new machine can be added to the CPMC system just by connecting it to a RPi where the adapter and agent for that machine is deployed. The RPi also supports various types of standard communication interfaces. Each testing site has its own local server where the virtualized manufacturing web services are hosted. Each machine tool is connected to its own controller (RPi). There are two cloud servers deployed in virtual machines in Uark network for hosting manufacturing cloud services and cloud applications. Data are stored in the cloud using HBase, which is a NoSQL database. A web application named “Application Center” is hosted in the cloud which contains all published manufacturing applications and works as a marketplace for both customers and service providers. A user (customer or manufacturer) can create a profile by providing his/her information and setting a username and password. After logging in, he/she

can view available manufacturing services and subscribe to the desired ones. Once subscribed to manufacturing services, corresponding access links are showed in a dashboard, using which users can access their desired manufacturing services and monitor or operate associated machine tools. While placing an order, the users can choose from available options and parameters. For instance, users can select the color and material type of a 3D printed object. Cloud applications make HTTP calls to the monitoring web services to acquire data in XML format. Both file sharing methods described in 3.1.2 (locally and in the cloud), has been implemented and tested. Multiple iterations of manufacturing monitoring and operation processes in different scenarios were conducted using the testbed.

4.2 Collaborative manufacturing using MTComm

As monitoring data of a machine is available in XML format over the Internet, manufacturing machine tools using can monitor the status of other machines using MTComm. In addition to that, machines can also initiate operations of other machines via MTComm `operate` service. All communications are actually carried out by the agents of the associated machine tools. Thus, MTComm facilitates machine-to-machine communication not only inside a factory floor, but also across multiple manufacturing plants situated in geographically different locations over the Internet. If a machine tool (M_1) requires service of another machine tool (M_2), its agent inspects the current status of M_2 from its data in XML format using `current` service over the Internet (assuming M_1 knows the service URL of M_2's agent). If available, M_1's agent generates an `operate` request message and sends it to M_2's agent. M_2's agent then processes the request and completes the operation. If M_2 is not available, M_1's agent has the option of either waiting until M_2 becomes available or communicating with another machine which is available

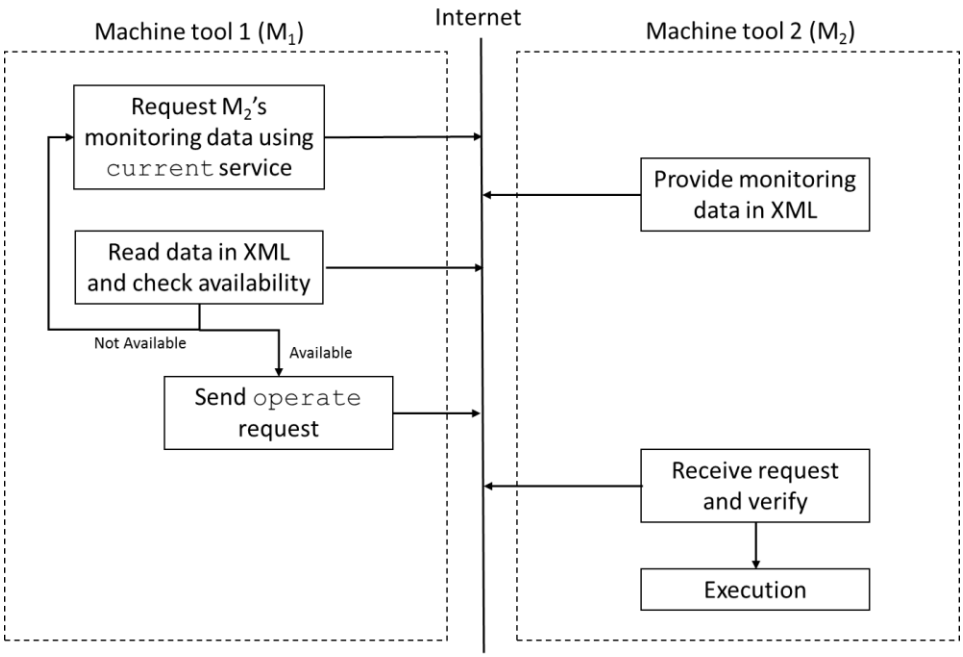


Figure 17. Communication between two machine tools using MTComm

and can provide the service M_1 is looking for. Figure 17 illustrates the communication process between two machines using MTComm over the Internet.

The procedure of such machine-to-machine communication is very useful in different manufacturing scenarios. It allows manufacturers to achieve factory floor automation, respond to production process failures, perform collaborative manufacturing, improve production scheduling etc. For instance, let there be two identical machines in a factory floor. One machine is assigned a job by the CPMC. During the manufacturing procedure, a failure occurs, and the machine becomes incapable of completing the job. At this point, instead of pausing the production process, the agent of machine can forward job to another identical machine in the factory (assuming there is one). Thus, MTComm enables machine tools to participate in collaborative manufacturing actively. They can carry out parts of a single manufacturing job or perform multiple iterations of the same job. This unique capability also allows a CPMC to create

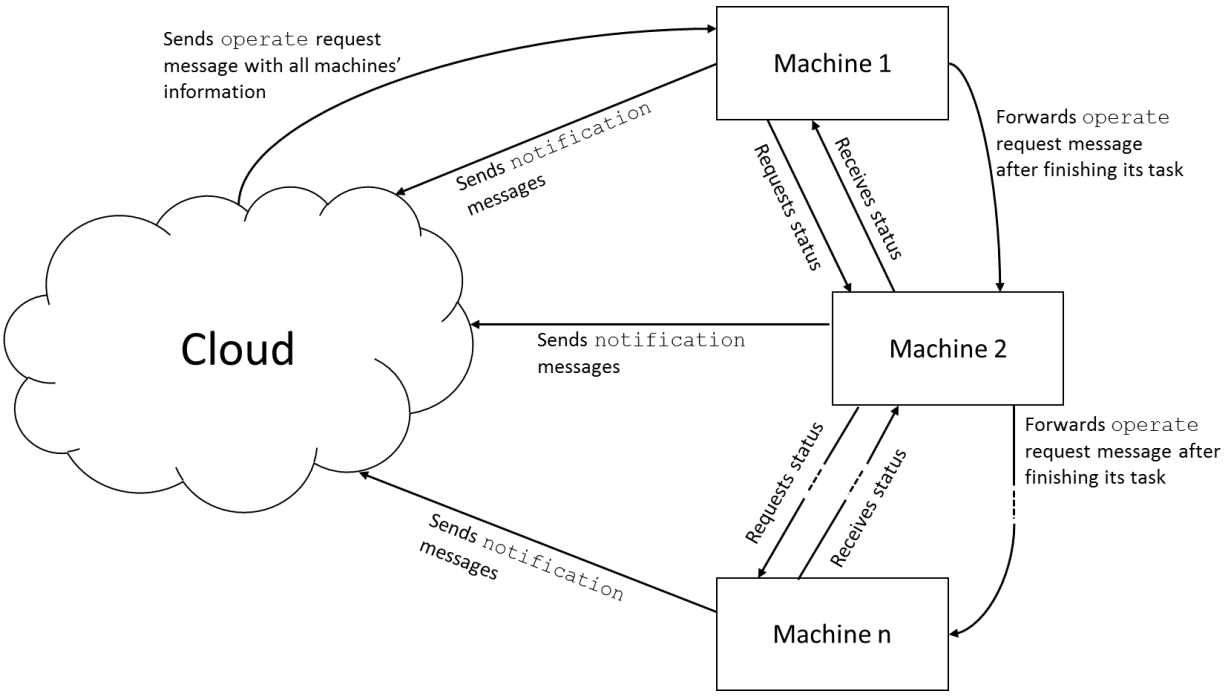


Figure 18. Collaborative manufacturing process using MTComm in a CPMC

collaborative manufacturing processes that involve participation of different types of machine tools over the Internet.

4.2.1 Process of collaborative manufacturing in CPMC

Figure 18 demonstrates the workflow of a collaborative manufacturing process with n machine tools. The machine itself, its adapter and agent are collectively shown as a machine entity. Like other manufacturing operations discussed above, a collaborative manufacturing operation is initiated from a client application in the CPMC. Manufacturers or machine tool owners has the flexibility to design specialized collaborative operations by pre-selecting which machines should be used for a particular operation, or to let the cloud application choose from available machine tools. When there is a request for collaborative process, a cloud application, designed specifically for performing collaborative operations, collects `operate` service URLs from database, generate a single `operate` request XML message containing *opeartion*

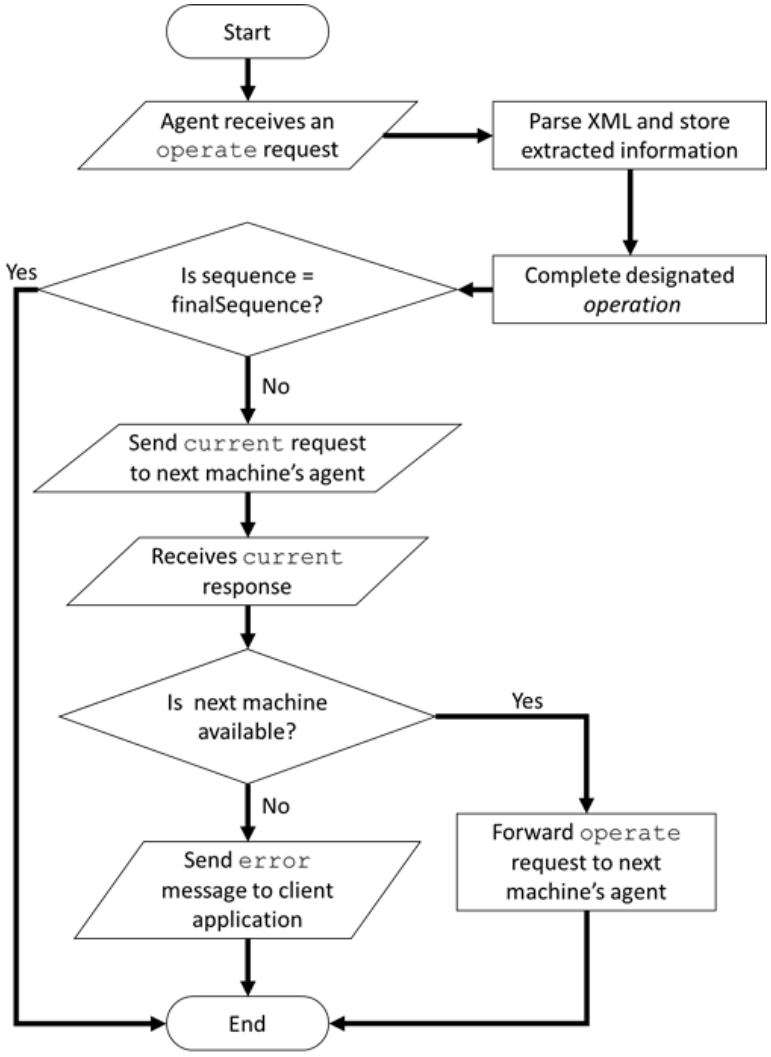


Figure 20. Process flow chart of an MTComm agent for collaboration

information of all associated machine tools. A collaborative `operate` request is slightly different than the one showed in Figure 11(b) – instead of *ComponentOperation*, it has multiple *DeviceOperation* elements with an additional *Parameter* providing the `operate` service URL of the next machine tool (except the last one); and the header contains an additional *attribute* named 'finalSequence' which is the number of machine tools involved in the requested operation. Once generated, the `operate` request message is sent to the agent of the first machine tool (M_1) via HTTP POST method. The agent stores the XML message, validates its

format, finds its own operation using *deviceId*, and verifies the operation. It also checks if its own operation's 'sequence' attribute is less than or equal to the 'finalSequence' value. If latter, it identifies itself as the last machine tool of the chain and terminates after finishing its operation. If an error is encountered, it sends a `notification` message with description of the error, and the process is terminated. Otherwise, it sends a `notification` message to the cloud application, acknowledging that it has received and started the operation. Then operational information is forwarded to the adapter, which carries out the operation. Once completed, the agent sends another `notification` message confirming successful completion. Then it identifies the next machine (M_2) using the 'sequence' number (if it is not the last machine), collects the service URL of M_2's agent (either provided in the `operate` message by the cloud application or pre-loaded in the local storage), and requests `current` service of M_2 to check its availability. If M_2 is available, it forwards the `operate` XML message to the M_2's agent. If unavailable, M_1 may wait until available, or notify the cloud application which will wait for M_2 instead, making M_1 available, depending on user's intent. Figure 19 shows the flow chart of steps for an MTComm agent to participate in collaborative manufacturing.

The rest of the procedure is similar. M_2 completes its own operation, identifies next machine, checks its status, and forwards the `operate` message to M_3. The process continues until the message reaches M_n machine (considering *n* machine tools are required to complete the collaborative manufacturing process). When M_n completes its own operation, it compares its own 'sequence' with 'finalSequence' value and finds them to be equal, meaning there are no more machines. So, like a normal operation, it sends a `notification` message to the cloud application which marks the collaborative manufacturing operation as completed.

TABLE I. Sample case of assigning five consecutive collaborative operations (a, b, c, d, e) during five time cycles in the CPMC

M_1	M_2	M_3	M_4	M_5
a	b	c	d	e
e	a	b	c	d
d	e	a	b	c
c	d	e	a	b
b	c	d	e	a

Let a CPMC system be consists of *n* identical machines of which each one can complete a certain task *w* in time period *t.* Also let a collaborative operation require all machines to complete task *w* once. Here it is assumed that the time between transferring operation from one machine tool to another is negligible, i.e., one machine tool immediately starts its part as soon as the former machine completes its task. For CPMC, the total time for one such collaborative operation is *nt*.

Now let another scenario be considered where *m* collaborative operations are assigned at the same time. For the CPMC, all the operations can be initiated at once assuming the initial setup time required for the cloud application to request `operate` service is negligible, and all machines are available to work. As all machines are similar and conduct the same task, each can be assigned to a new operation. After the first *t* cycle, each machine becomes available and can start the next parts of all the jobs. The situation is shown in Table I for *n=5* and *m=5.* So, the total time required to complete all jobs is *nt* (Sunny, Liu & Shahriar, 2018).

4.2.2 Experiments in CPMC testbed and evaluation

Several collaborative manufacturing scenarios were simulated in the CPMC testbed. In the Uark testing cells, two types of collaborative manufacturing were implemented. The first one required participation of a 3D printer, a robotic arm, and a CNC machine. In this scenario, the 3D printer produced an object upon receiving an `operate` request from the cloud application. When it finished the production, it forwarded the XML message to a robotic arm sitting next to it which then extended its arm to the 3D printer's heatbed, picked up the produced object, and carried it to a CNC machine nearby. The next part of the collaboration required the CNC machine to do a drilling job onto the 3D printed object. When the CNC machine finished its part, the collaborative job is concluded. If any of the machine tools was busy, then the former machine kept observing its status and when it became available, proceeded with the operation. The described collaborative operation was carried out in two different scenarios – when all machines were available throughout the whole process i.e. not doing some other individual or collaborative task, and when the CNC machine was doing its own individual job. In the latter scenario, when the CNC machine was able to complete its own task before the 3D printer finished its printing, there was no waiting. Figure 20 demonstrates example of status monitoring during different stages of a collaborative manufacturing in the CPMC.

In the second collaborative manufacturing test case, two 3D printers worked together. When a printing process was assigned to one 3D printer and it could not complete the operation, because of a failure, then it forwarded the operation to the other 3D printer. In the experiments of this type of collaborative process, some intentional failure scenarios were orchestrated, e.g., too much high temperature of the extruder, unavailability of printing material etc. In each experiment, the second 3D printer successfully completed the assigned job.

App Home

Log out

Ultimaker2 : P2673

Information	Value
Progress	98.7%
Reading Timestamp	2016-09-30 11:28:39.094
Bed Temperature	69.1
Nozzle Temperature	210.5
Machine Status	BUSY
X-axes position	114.119
Y-axes position	114.7
Z-axes position	1.0

X-Carve : C153

Information	Value
Progress	0.0%
Reading Timestamp	2016-09-30 11:28:39.110
Machine Status	Alarm
X-axes position	0.000
Y-axes position	0.000
Z-axes position	0.000

Uarm : #1

Information	Value
Reading Timestamp	2016-09-30 11:28:39.125
Machine Status	available
Robot Grabber	2
Robot Grab Rotation	0
X-axes position	0
Y-axes position	0
Z-axes position	0

(a) 3D printer (Ultimaker2) working

App Home

Log out

Ultimaker2 : P2673

Information	Value
Progress	0.0%
Reading Timestamp	2016-09-30 11:29:50.093
Bed Temperature	62.5
Nozzle Temperature	123.1
Machine Status	AVAILABLE
X-axes position	0.0
Y-axes position	0.0
Z-axes position	0.0

X-Carve : C153

Information	Value
Progress	48.65%
Reading Timestamp	2016-09-30 11:29:50.106
Machine Status	Run
X-axes position	682.750
Y-axes position	580.000
Z-axes position	-4.191

Uarm : #1

Information	Value
Reading Timestamp	2016-09-30 11:29:50.117
Machine Status	available
Robot Grabber	2
Robot Grab Rotation	0
X-axes position	0
Y-axes position	0
Z-axes position	0

(b) CNC machine (X-Carve) working

Figure 20. Example of status data during a collaborative manufacturing experiment in the CMPC

One of the key advantages of using the presented communication method between machine tools is the fact that it enables interoperability not only among machines in a single factory floor, but also among machines situated in different locations. At present, in most cases the communication between machine tools is limited to machines that are situated in close proximity or in a certain factory which are connected with each other through a Local Area Network (LAN). But using MTComm, they are able to connect to other machines from other factories, from other cities, even from other countries. To protect from unauthorized access, only machine tools which were registered and connected to the cloud could access other machine tools in the CPMC.

4.3 References

Bowyer, A. (2014). 3D printing and humanity's first imperfect replicator. *3D printing and additive manufacturing*, *1*(1), 4-5.

Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C., & Bowyer, A. (2011). RepRap–the replicating rapid prototyper. *Robotica*, *29*(1), 177-191.

Liu, X. F., Shahriar, M. R., Sunny, S. M., Leu, M. C., Cheng, M., & Hu, L. (2016, January). Design and implementation of cyber-physical manufacturing cloud using MTConnect. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

Liu, X. F., Shahriar, M. R., Sunny, S. M. N. A., Leu, M. C., & Hu, L. (2017). Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed. *Journal of Manufacturing Systems*, *43*, 352-364.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2018). Communication method for manufacturing services in a cyber–physical manufacturing cloud. *International Journal of Computer Integrated Manufacturing*, *31*(7), 636-652.

CHAPTER 5

DESIGN AND DEVELOPMEMT OF MTCOMM EDGE MIDDLEWARE

One of the primary objectives of Industry 4.0 is to enable seamless integration of web/cloud based client applications and heterogeneous physical machine tools for real-time data acquisition and analysis. Nowadays manufacturing machine tools generate tremendous amount of data every second which are required to be transmitted at high-speed. The rapid increase in the number of devices and quantity of data generated, fueled by heterogeneity of data types, often results in complex scenarios where continuous data storing and processing become inefficient and expensive (McKee et al., 2018). This may also lead to network traffic bottleneck and scalability issues, specially for large-scale CMSs managing thousands of machines at the same time. Edge computing (Shi et al., 2016) can aid in this regard by storing and processing data locally before transmission. In most existing CMSs, data analysis is typically performed in cloud servers, as most analysis algorithms require extensive computational power. But this adds a considerable delay between data generation and event detection, which may lead to hazardous situation. Recently some researchers proposed to conduct data analysis in local/private cloud servers (Brito et al., 2018). However, deploying edge-clouds with sufficient computational performance is expensive and can also suffer from aforementioned issues, specially for large-scale CMSs.

To overcome such issues, we propose to extend MTComm's functionalities to offload repetitive and computationally inexpensive data processing and analysis tasks to low-cost constrained edge nodes, each responsible for only a few physical machines. Such edge nodes will assist cloud based applications through data caching, optimization, and preliminary fault detection and mitigation, and thus lessen the workload of both central and edge cloud servers and

improve overall system efficiency. But MTComm is not specifically designed to be readily used for edge computing, as it does not offer specifications for localized data caching or processing in order to minimize cost and maximize efficiency, specially for resource constrained edge platforms. Therefore, we designed and developed a novel scalable MTComm Edge Middleware (MEM) for CMSs by extending existing MTComm agent's capabilities beyond data transformation. Alongside transforming collected data and incoming operation requests, an MEM also optimizes data before transmission and performs preliminary data analysis for *in-situ* fault diagnosis. To enhance its efficiency and reduce communication latency while minimizing cost, we focused on two approaches – 1) developing optimization strategies and 2) using hardware acceleration through high performance embedded devices. For this purpose, we first adopted unique optimization strategies for localized data caching and transmission using a *hold-until-changed* approach. MEMs aim to minimize data storage requirements and cost while enabling high-speed transmission and processing, low bandwidth usage, and rapid fault detection without increasing information loss. Also, MTComm's remote operation mechanism was revisited and modified to enable overall faster initiation of manufacturing tasks. The performance of the optimization strategies was evaluated in the CPMC testbed and compared to the existing MTComm agent's performance in terms of average response time, bandwidth used or average message size, and required storage size. After that, we designed MTComm programs and hardware architecture for a System-on-Chip (SoC) Field Programmable Gate Array (FPGA) device as MEM to speed-up data processing and transmission. This chapter discusses the development, implementation, experimentation, and evaluation of the optimization strategies and the SoC FPGA based MEM as a low-cost edge computing solution for next generation cyber manufacturing systems.

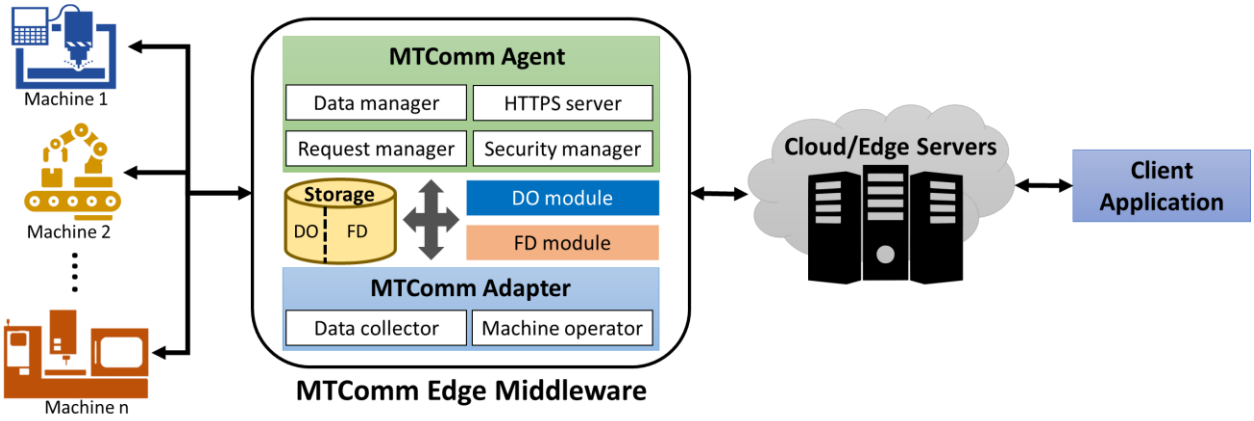


Figure 21. Architecture of MTComm Edge Middleware (MEM)

5.1 Architecture of an MTComm Edge Middleware

To enable edge computing using MTComm in a CMS, we revisited the existing MTComm architecture to extend the agent's capabilities beyond data transformation and service hosting and convert it to an MEM. Figure 21 illustrates the architecture of our proposed MEM. In addition to the modules of a typical MTComm agent (MTCagent) and adapter program, MEM also includes a local data storage, a data optimization (DO) module, and a fault diagnosis (FD) module. The cross-shaped arrow refers to the interconnect bus which is used by all MEM components to communicate with each other. Like an MTCagent, one MEM can acquire data from multiple machine tools and communicate with multiple clients over the Internet using HTTP. In a CMS, an MEM connects to core cloud servers either directly or via edge/fog servers, if available. It is designed as a standalone software program that is easily deployable in low-cost constrained embedded platforms. It can even be embedded into a machine's controller unit. The storage and processing power of an MEM depend on associated hardware platform. Functionalities of different MEM components are discussed below.

1) MTComm agent and adapter

The MTComm agent (MTCagent) and adapter modules function similarly as described in 3.1 – adapter is responsible for collecting data from physical resources and sending operational commands to machine tools, while the agent primarily works as a translator on an HTTP server and agent hosts all five MTComm RESTful services. However, there is slight difference in the data acquisition methodology. Also, the internal components of both modules have been simplified. Previously, data collected by adapter was directly forwarded to the agent and stored in its storage. In MEM, an adapter performs periodic data acquisition cycle via its data collector and the acquired data is processed by an optimization module before being stored. The agent's data manager gets data from storage when it receives a service request. It also works with the optimization module for keeping track of its clients and associated service requests. Details of these process are discussed in the next section. For executing requested operations, the interaction between adapter and agent is same as before, although the agent's request manager utilizes an optimization strategy to enable faster execution. An MTCagent may include a security manager for imposing security countermeasures.

2) Data storage

A local data storage is used by the MEM to temporarily store collected data before transmission. It is functionally different from the original buffer storage of an MTCagent, as acquired data are stored in it using a data caching strategy discussed in next section. It basically consists of two relational databases – one used by DO module and one used by FD module. The FD database is typically larger than the DO database, as more data are required to perform anomaly detection procedures.

3) Data optimization (DO) module

Responsibilities of the DO module are two-fold. Firstly, it processes data collected from machine tools to determine which data to be stored locally. Secondly, it also analyzes incoming service requests from client applications to determine which data to be transmitted to whom. Its primary focus is to reduce data transmission latency and bandwidth usage without increasing information loss and also to enhance overall system scalability by storing and processing data at the edge.

4) Fault diagnosis (FD) module

The FD module is responsible for monitoring and analyzing incoming machine tool data to detect possible failure events during a manufacturing operation. Based on previous operational data, it performs lightweight fault detection algorithms to determine if a new data is an anomaly or not. In case of a fault, it alerts the user using error messages and initiates any pre-defined mitigation tasks, such as raising alarms, immediately stopping ongoing processes etc., if available. Details of the FD module is discussed in chapter 6.

5.2 Optimization of MTComm for MEM

As MEMs are primarily designed for constrained edge devices, it is necessary to adopt optimization strategies for MTComm in order to reduce storage, latency, and processing power requirements. In this section, we present three optimization strategies of MTComm for proposed MEM -- data caching strategy, data transmission strategy, and operation optimization strategy. The primary objective of these strategies is to reduce data transmission and processing latency, bandwidth usage, and storage requirement of an MEM as much as possible without increasing information loss by performing iterative tasks at the edge level.

5.2.1 Data caching optimization

An MTConnect agent, thus MTCagent usually contains a circular buffer storage to temporarily store acquired data, where data are stored sequentially, and most recent data entry overwrites the oldest one. As a machine tool usually generates significant amount of data at high frequency and MTCagent stores every data collected from it, the buffer storage can fill up very fast. Therefore, the buffer storage should be large enough to allow for adequate space required to minimize information loss. Typically, MTConnect agents are deployed in desktop computers or servers, which contain enough storage to fulfill this requirement. However, for a constrained edge computing device, this can become a significant issue as large storage requirement leads to increased cost. So, it is necessary to optimize data caching in an MEM to minimize storage size and cost.

Therefore, MEM adopts a *"hold-until-changed"* approach where only unique *dataitem* values are stored in the OD database along with associated timestamp and sequence number while discarding repeating or redundant values. Algorithm 1 in Figure 22 illustrates our data caching procedure. For simplicity, let us assume one MEM is connected to one machine tool. At the beginning, MEM uses the `probe` document of associated *device* and creates an object for each *dataitem* with related attributes, such as id, name, type etc., and value. When MEM collects machine tool data for the first time, it stores all *dataitem* values in the database with id, timestamp, and sequence number (which is 1 at this point). When next batch of data arrives, unlike MTCagent which just stores these new data without any processing, the OD module checks whether the new value of a *dataitem* is different from its previously stored value or not. If same, the new value is discarded. Otherwise, a new entry is created in the database with this new *dataitem* value, latest timestamp, and incremented sequence number. If no *dataitem* has new

Algorithm 1 Data caching procedure

Require: new data available from a *device*

1: $device.currentSeq \leftarrow device.currentSeq + 1$
2: $timestamp \leftarrow$ current time
3: $hasNewEntry \leftarrow$ `False`
4: **for** each *item* object in *dataitemsList* **do**
5: **if** ($item.lastStoredData =$ `NULL` or
6: ($item.lastStoredData \neq$ `NULL` and
7: $item.currentData \neq item.lastStoredData$))
8: **then**
9: $hasNewEntry \leftarrow$ `True`
10: Add a new entry to database with values of
11: ($device.currentSeq, timestamp, item.id,$
12: $item.currentData$)
13: $item.lastStoredData \leftarrow item.currentData$
14: **end if**
15: **end for**
16: **if** $hasNewEntry =$ `False` **then**
17: $device.currentSeq \leftarrow device.currentSeq - 1$
18: **else**
19: commit database
20: **end if**

Figure 22. Data caching strategy in MEM

value, then whole batch is ignored, and current sequence number remains the same. This way the OD database only contains unique data for all *dataitems*. Sequence numbers and timestamps are used to retrieve *dataitem* values at a certain time. The time interval between two consecutive sequences indicate that the *dataitem* had the same value for that particular time period. Using this method, information loss is minimized as all unique values are stored and timestamped, yet the amount of data stored for a given time and thus size of local storage required are considerably smaller than those of a conventional MTCagent. To minimize database size further, a user can

specify a maximum memory limit. If the OD database reach that limit, next data entry will overwrite the oldest one.

5.2.2 Data transmission optimization

Usually when an MTCagent receives a `current` request, it immediately responds with most recent *dataitem* values. It also allows client applications to perform queries with specific parameters, such as *dataitem* id, sequence number, timestamp etc. Also using `sample` service, a client application can request for data of a certain time interval, e.g. from sequence 20 to sequence 85. Therefore, it is up to client applications to determine how to send requests to an MTCagent. If a client wants to ensure that it receives all available data, it has to keep track of what data sequences it previously received and also determine which sequences are to be requested and how often. For a large-scale CPMS, this may lead to significant scalability issue. Also, this may result in considerable data loss for simple client applications. In one experiment, we developed a simple remote monitoring web application which was only capable of sending consecutive `current` requests to an MTCagent and displaying data extracted from response messages. The results showed considerable gaps between sequence numbers of consecutive responses, ranging from tens to thousands, as data acquisition and conversion rate of the agent is typically much faster than transmission rate over the Internet.

To overcome such issues, our proposed MEM keeps track of its clients and determines which and how much data should be sent to a client upon `current` request. Figure 23 and 24 demonstrates the procedure used for preparing responses for a client. Alongside its database, MEM maintains a key-value pair-based dictionary which contains ip addresses or URLs of all client applications it has communicated with and sequence numbers of the most recent messages

Algorithm 2 Prepare response for incoming request

Require: receive a new `current` request for a *device*

1: $timestamp \leftarrow$ current time
2: $clientAddr \leftarrow$ address/url of client
3: Fetch $clientDict$ containing list of previous clients
4: Create an empty dictionary named $itemsToAdd$
5: **if** $clientAddr$ exists in $clientDict$ **then**
6: $responseSeq \leftarrow clientDict[clientAddress] + 1$
7: **if** $responseSeq \leq device.currentSeq$ **then**
8: $results \leftarrow$ all entries in database WHERE
9: $sequence = currentSeq$
10: **for** each $item$ in $results$ **do**
11: add $(item.id, item.data)$ to $itemsToAdd$
12: **end for**
13: **end if**
14: **else**
15: $responseSeq \leftarrow device.currentSeq$
16: **for** each $item$ in $dataitemsList$ **do**
17: add $(item.id, item.currentData)$ to
18: $itemsToAdd$
19: **end for**
20: **end if**
21: **if** $itemsToAdd$ is empty **then**
22: Send `HTTP 204` response
23: **else**
24: **for** each $item$ in $itemsToAdd$ **do**
25: Call function $createXmlElement$ with
26: $(currentSeq, timestamp, item.id, item.data)$
27: **end for**
28: Generate XML message
29: Send `HTTP 200` response with XML message
30: **end if**
31: add/update $(clientAddr, currentSeq)$ pair in $clientDict$

Figure 23. Data transmission algorithm used in MEM

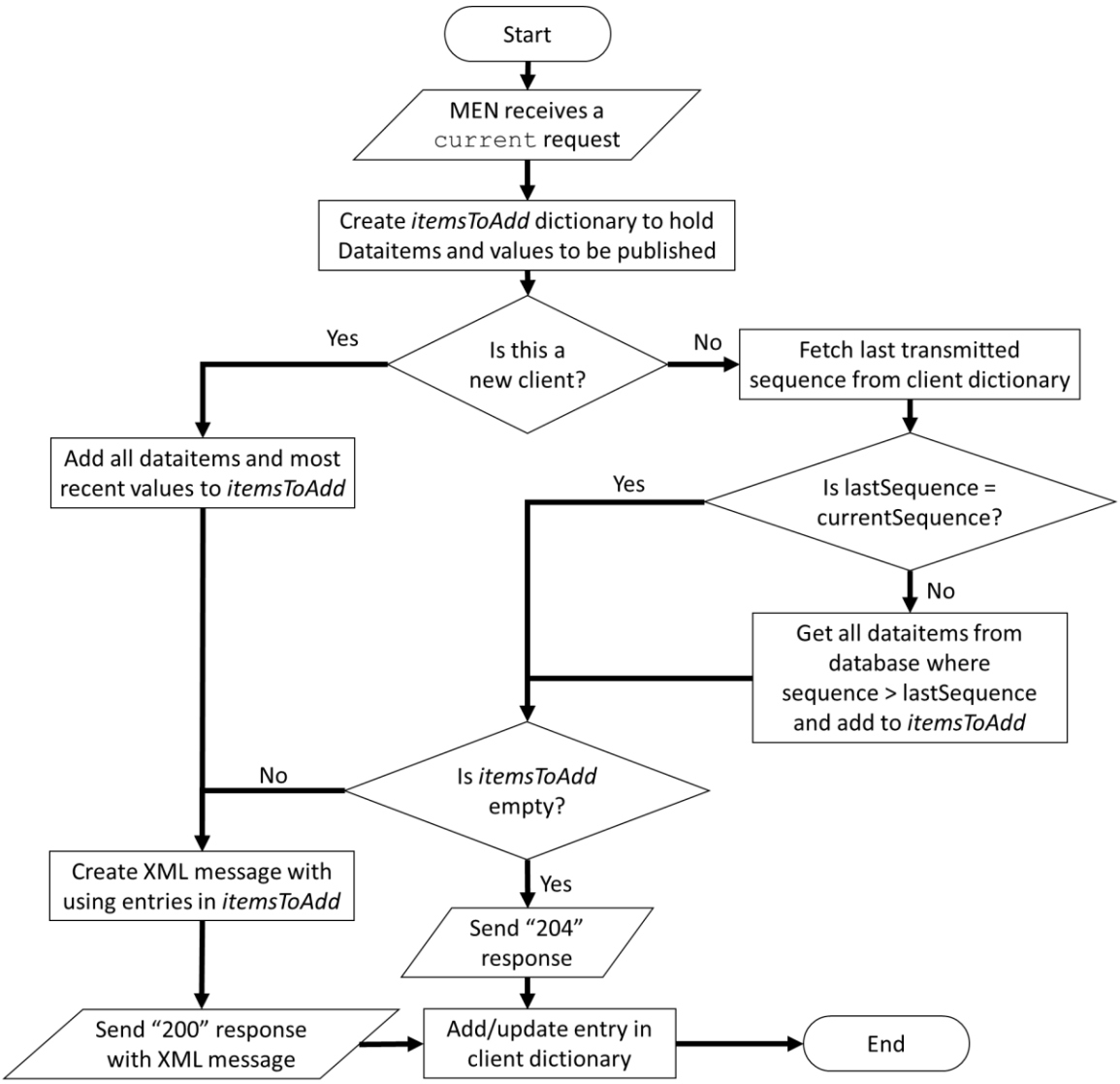


Figure 24. Process flow of data transmission strategy in MEM

transmitted to them. When MEM receives an incoming `current` request, the OD module uses the client's address to check if this client has any entry in its client dictionary. If no pair is found, then it identifies this client as a new one. Then an XML response message is created with most recent values of all *dataitems*. As all *dataitem* objects holds their most recent values, this particular step does not require fetching data from local OD database. The XML message is timestamped and sequenced with corresponding values at that point. It also contains additional

information such as header elements, attributes of *dataitems* etc., which are acquired from the `probe` document at the beginning. Then MEM sends an `HTTP 200` response with the generated XML message and adds an entry to its client dictionary with client's address as key and latest sequence number used to create the response as value. When MEM receives a new request from the same client, it finds its entry in the client dictionary and retrieves associated sequence number. Then it searches for *dataitems* in the OD database associated with the next sequence number. If no entry is found for this new sequence, it means the machine has not generated any new data since the client's last query. So, MEM sends an `HTTP 204` response with no content. In case where MEM finds relevant database entries, it fetches those *dataitem* entries and create an XML element for each entry using id and stored value of the *dataitem*, most recent timestamp and sequence number, and other associated information. As mentioned in the previous section, the OD database only stores *dataitems* with changed values, so the response XML only contains *dataitems* with new data that were not previously transmitted to the client. After XML generation is completed, MEM sends an `HTTP 200` response with it and updates this client's entry in the client dictionary with new sequence number. The client dictionary has a time-out functionality that removes a client entry when its last sequence is no longer available in the OD database, indicating that client has not communicated in a long time.

Using this strategy, MEM reduces bandwidth usage by preventing transmission of repeating or redundant *dataitem* values. It enables faster data transmission and processing as `HTTP 204` responses has no message body and thus can be processed more quickly. Also, not all *dataitems* change values in every data acquisition cycle, specially in idle states, so the `HTTP 200` responses of proposed method contains fewer elements and thus requires less bandwidth in

average than a typical MTCagent. Furthermore, MEM keeps tracks of its clients and thus enhance scalability of CPMSs.

5.2.3 Optimization of operation execution

In MTComm experiments, the average operation initiation time (OIT) referring to the time between a client application sending an `operate` request and the associated MTComm adapter initiating the requested operation was found to be about 1 second. This rather large delay can become an issue in specific manufacturing scenarios, specially for processes requiring real-time or near real-time execution, such as emergency stopping of a machine tool in case of a failure. Therefore, it is necessary to reduce this latency for the proposed MEM. In an `operate` service, two processes are responsible for majority of this delay – receiving the POST request containing the XML message and the two-step verification process. Both processes are necessary, as the former one acquires operational information from the client application and the later ensures associated machine's security by verifying whether the requested operation and provided information are safe to be executed. However, from our survey of different manufacturing processes we noticed that in many regular manufacturing scenarios, one machine tool typically performs similar operations iteratively with little to no change in parameters. For example, many 3D printers only support one material type and thus the temperatures (extruder and heatbed) are usually same for all printing jobs. Only information required for executing these printing operations is the 3D model source. To stop an ongoing operation, no parameter is required. For component operations, this phenomenon is more common. To move an axis of a machine tool, the only parameter required is the destination coordinate value. For these types of operations, sending an `operate` request with XML message is not necessary. Rather requesting

operations without sending an XML message can potentially lead to significant reduction in execution time as both receiving and verifying an incoming request would be much faster.

Therefore, we revisited our initial design of MTComm's `operate` service and introduced two execution modes – custom and default. The custom mode follows our initial design of using HTTP POST method and XML messages and can be used for both single and multiple *operations*. In default mode, an `operate` service for a single *operation* is requested via HTTP GET method, similar to the `current` service. An MTComm agent contains default configurations for most, if not all, associated *operations*. To initiate an *operation*, a client application sends an HTTP GET request with unique id of the *operation* in the path of the service URL as below –

```
http://10.5.55.7:10090/cnc_01/operate?path=//Operation[@id="stop
Drilling"]
```

Here, `cnc_01` is the *device* name and the first path element refers to the `probe` XML element associated with the requested *operation* (*Device* in the above example, as stopping a drilling process is a device-level *operation*). *Parameters* can be added at the end of the request URL by using the '&' operator. Unique id of a *parameter* and its target value are combined using the '=' operator. Here are two examples of parameterized `operate` GET request –

```
http://10.5.55.7:10090/3dp_01/operate?path=//Operation[@id="star
tPrinting"]&material=ABS&model=Box
```

```
http://10.5.55.7:10090/3dp_01/operate?path=//Actuators//Operatio
n[@id="changeExtruderTemp"]&value=210
```

When an MTComm agent receives an `operate` GET request, it extracts the *operation* id and parameters (if any) from the request URL. The extraction process is almost instantaneous compared to that of the custom mode, as it does not require to parse an XML message. Even if an operation has hundreds of parameters (which is rather unlikely) and the service URL becomes very large, the extraction will still be faster. If there is no parameter given, it loads the default configuration for the requested *operation* and forwards instructions. No verification is needed here, as the configuration is pre-defined by the user and thus is trusted. If the request URL contains parameter values, those are verified against associated constraints to make sure they are within acceptable range. This verification process is much faster than that of the custom mode, as it does not include validating and parsing an XML message. Once verified, the agent sends these parameters with the operation instructions to the adapter for initiating the requested *operation*.

As an example, let us consider a 3D printer which uses two types of materials (PLA and ABS) to print five different objects (ring, ball, box, triangle, and logo). The target extruder temperatures for PLA and ABS are 200 and 230 degree Celsius respectively. The default object and material type are set to be ring and PLA respectively. When the printer's agent receives an `operate` request via GET method without any parameter specified, default configurations are loaded, and the printer starts printing a PLA ring at 200 degree Celsius. The client can choose to print an ABS box by adding respective parameters and values in the service URL, as shown in the URL example above. If a service request states extruder temperature to be 230 degree Celsius, but does not specify ABS as material type, the verification process will identify the request to be hazardous as the requested parameter value exceeds accepted temperature for default material (PLA) and therefor will reject the operation. Thus, even with this simplified default mode, all features of MTComm operations are retained.

This does not eliminate the need for custom mode, however, as not all manufacturing processes can be done in this manner, specially those that require interactions between multiple machine tools or components. An example of such manufacturing scenario is performing a series of testing operations of different machine components for fault diagnosis or regular maintenance. Adopting both execution modes for MTComm `operate` service not only reduces the overall average OIT, but also improves its robustness and provides users with the flexibility to choose between them based on their requirements and preference.

5.3 Development and Implementation of a SoC FPGA based MEM

To fully realize its feasibility and potential, implementation of an MEM prototype was necessary. In our earlier experiments, we used Raspberry Pi (RPi) devices running a Linux based Raspbian operating system to deploy agent and adapter programs and established the proof-of-concept. RPis were found to be capable enough to function as an MEM. However, in a real manufacturing plant, RPis will not be a suitable and efficient choice for MTComm deployment. Firstly, a RPi is small scale computer with its own operating system designed for generic uses. Running MTComm programs in a RPi suffers from time delays caused by system functions and procedures. Therefore, it may not support high-speed data exchange using MTComm, which is often crucial for manufacturing processes. Secondly, RPis have limited I/O connectivity and may not be enough for large scale manufacturing machine tools, which often contain complex bus based communication interfaces. Last, but not the least, RPis do not promote plug-n-play feature, which is crucial for developing a robust cyber-physical cloud framework. Using a dedicated device will also provide better security and reduced latency. Therefore, we explored different high-performance off-the-shelves embedded devices to implement a dedicated MEM prototype. Considering the software and hardware flexibility that the proposed MEM architecture should

have, we found high-performance and reconfigurable SOC FPGA devices to be the best candidates for prototype implementation.

We chose a Xilinx Zynq SoC FPGA as our development platform because of its wide acceptability and high performance. Zynq devices integrates a 667 megahertz ARM Cortex-A9 dual-core processor (PS) with powerful silicon peripherals, a high performance low-power consuming Programmable Logic (PL) of 28 nm, and 1GB DDR3L memory in a single embedded device. SoC PS is capable of running lightweight Linux operating system (OS). Figure 25 illustrates the block diagram of the architecture of proposed MEM prototype. Components of MEM are distributed between PS and PL based on their functionalities. AS PL is capable of faster computation, computationally extensive tasks should be performed in PL as much as possible. MTComm adapter is divided into two parts -- one for PS and one for PL. This is because in Zynq devices some peripherals are only available to PS. Therefore, depending on the type of communication interface a machine tool has, it can be only be connected to PS or PL. For example, Zynq-7000 devices only have USB ports with PS, so machine tools with USB connectivity need to be connected to PS. Analog sensors and actuators which can communicate via General Purpose I/O (GPIO) pins can connect to PL. So, to support a wide range of machine tools, it's better to deploy adapter programs for both PL and PS and leverage them appropriately. Additional data acquisition modules like analog-to-digital converters (ADCs), data samplers etc. are also a part of PL. Optimization module which requires to perform comparisons frequently is deployed in PL. MTComm agent is deployed in the Linux OS of PS, as it requires protocol stacks and library packages to host an HTTP server and process XML messages and also because Ethernet ports are connected to PS only. Resource manager and security manager are also parts of PS as they may require running sophisticated algorithms. PL, PS, and the memory are connected through a high-

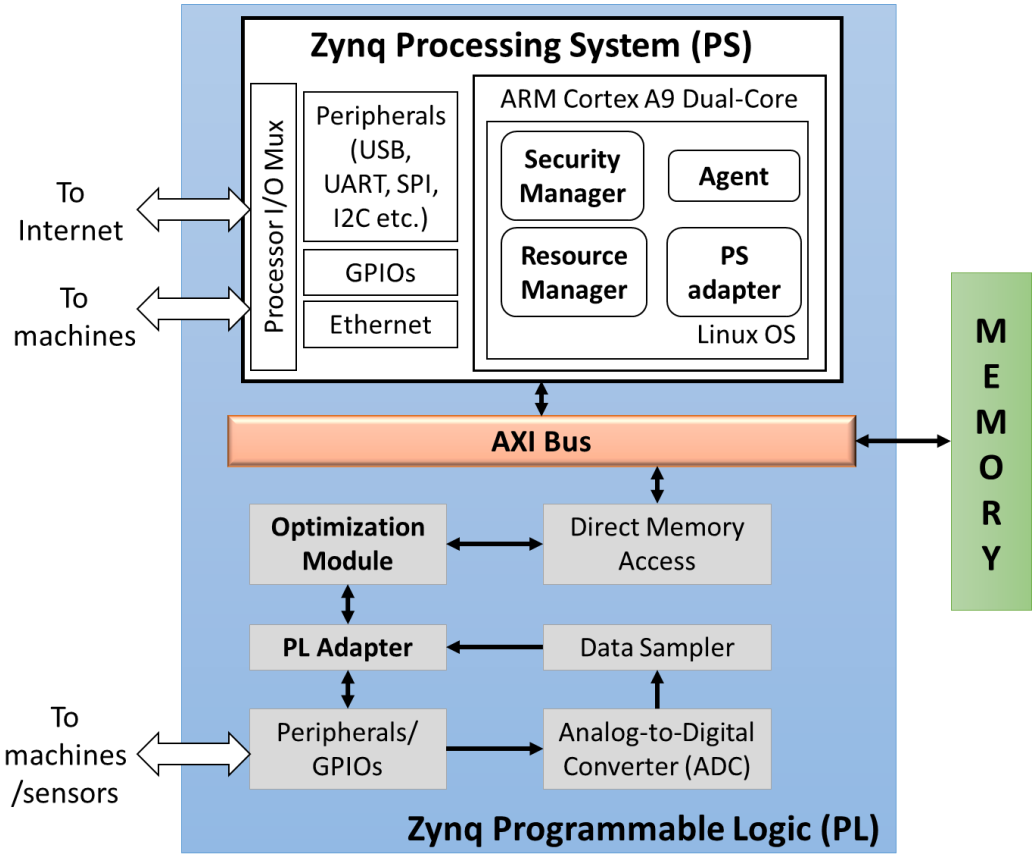


Figure 25. Block diagram of Zynq SoC based MEM prototype

speed Advanced eXtensible Interface (AXI) interconnect bus. As the memory is shared, both PL and PS modules can it via direct memory access modules.

For our implementation, we chose the Xilinx Zybo Z7 (Zynq-7010) FPGA development board as it comes with different communication interfaces (GPIOs, UART, USB, JTAG, SPI, I2C, and Gigabit Ethernet) and is only $200 (off-the-selves MTConnect adapters cost atleast $1000). A lightweight Linux OS called Petalinux was installed in the PS. All peripherals except GPIOs were connected to PS, so machine tools were connected to PS via USB/SPI/I2C and external analog sensors like pressure sensor, vibration sensor etc., were connected to PL via GPIOs. Analog data were converted using ADC and data sampler and sent to PL adapter, while PS adapter collected

data from machine controllers. These data were checked against previously stored data by the optimization module before being stored into the memory. Agent, resource manager, and resource manager programs along with all necessary packages were deployed in Petalinux OS. To reduce processing time, only necessary packages such as peripheral drivers, HTTP and XML libraries, encryption packages etc., were installed on a bare-minimum Petalinux distribution. Remote operation was performed in similar fashion, but in reverse order.

The software tools and programming languages used for the implementation are listed below:

- Vivado 2018.3: It is Xilinx's official programming tool and was used for hardware design, developing PL modules in VHDL language, and interconnecting the interfaces of PS and PL.
- Software Development Kit (SDK): This tool is also provided by Xilinx and was used to define and create the device tree, the FSBL (First Stage Boot Loader), and the Zynq boot image required to run the Petalinux OS on the SoC. It was also used to load these to the FPGA board.
- Python: This scripting language was used to program, compile, and execute programs for PS modules (agent, PS adapter, resource manager, and security manager) in the Petalinux OS. This was chosen particularly because all previous MTComm development were done using Python, hence, the transition was easier than choosing a new language.

5.4 Results and Analysis

5.4.1 Evaluation of optimization strategies

To evaluate the performance of aforementioned optimization strategies, experiments were conducted in a CPMC testbed with three machine tools, as shown in Figure 26. Ultimaker 2 and

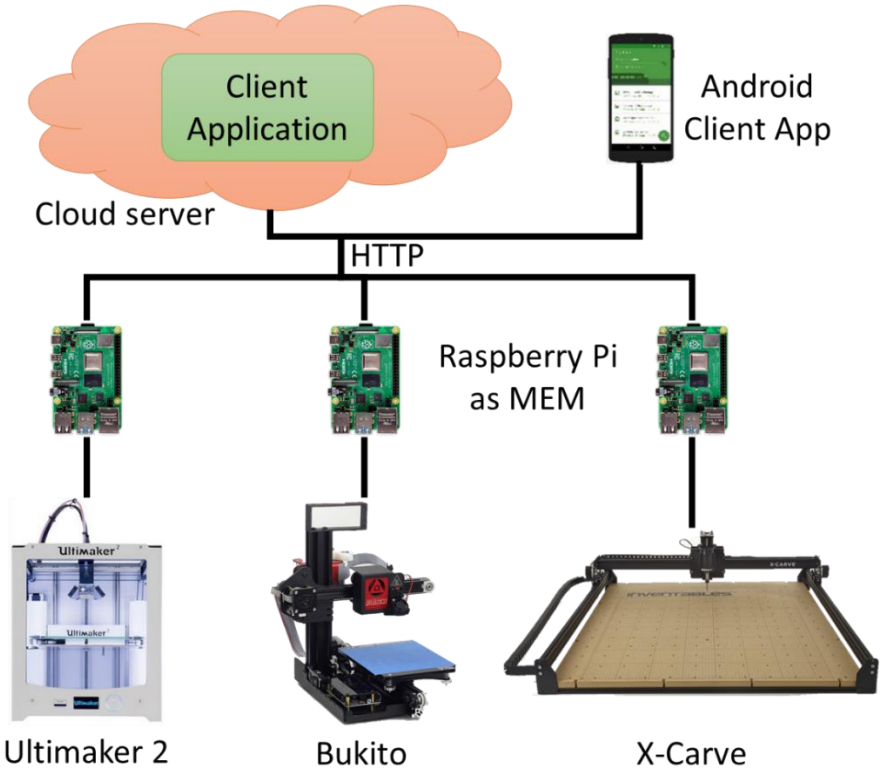


Figure 26. Experimental setup for MEM experimentations

Bukito are 3D printers, and X-Carve is a CNC machine. Each machine tool was connected to a Raspberry Pi (RPi) 3, which contained both MEM and legacy MTCagent programs developed in Python. SQLite3 databases were used as local storages. Although one RPi had sufficient storage and processing capacity to support all three machines, individual RPis were used to simulate a manufacturing environment with multiple MTCagents. Ultimaker 2, Bukito, and X-Carve had 18, 14, and 10 *dataitems* respectively. A client application was developed and deployed in a virtual machine (cloud server). Also, another web-based client application was developed for Android smartphones, which communicated with MEMs directly. Both applications were designed to continuously send `current` requests to RPis via HTTP GET, store and analyze response messages, display extracted data values, and calculate response time and message size.

The performance of the implemented MEMs was compared with previously developed or "legacy" MTCagents. Data were collected using both legacy and new methods from all machine tools in different manufacturing scenarios. At first, only one machine was kept active at a time and clients sent continuous requests to its RPi in both idle and busy (running an operation) states. Duration of operations done by Ultimaker 2, Bukito, and X-Carve were about 5, 13, and 2 minutes respectively. Then data were collected in a 'Combined' state, where all three machines were in busy state and clients queried machines sequentially (one-at-a-time), e.g. Ultimaker 2 → X-Carve → Bukito → Ultimaker 2 and so on.

Performance of MEM was evaluated in three categories -- response time (RT) depicting the time between client sending a `current` request and getting response back, minimum size of local storage, and bandwidth usage or size of response messages. Table II and Figure 27 present the comparison of average RT for each machine in both states calculated from 10000 consecutive

TABLE II. Average response time (RT) in milliseconds (ms)

Machine name	Machine state	Avg. RT for legacy method (ms)	Avg. RT for MEM (ms)	RT reduction (%)
X-Carve	Idle State	7.365 ± 1.2	3.813 ± 0.9	**48.228**
	Busy State	7.756 ± 1.3	4.002 ± 1.1	**48.401**
Ultimaker 2	Idle State	13.053 ± 2.7	6.438 ± 2.4	**50.678**
	Busy State	16.057 ± 2.9	7.237 ± 2.6	**54.929**
Bukito	Idle State	12.931 ± 1.3	5.706 ± 1.7	**55.874**
	Busy State	14.608 ± 2.9	6.255 ± 2.5	**57.181**

requests. In each case, MEM provided about 50-55 percent reduction in RT compared to legacy method, 52.5 percent in average. In one study (Lynn et al., 2017), average time to complete an MTConnect HTTP GET request using conventional method was 4.2 ms in a setup where MTCagent and its client were close and part of the same local area network (LAN). In our setup, all RPIs were given global static IP addresses and client applications communicated with MTCagents and MEMs over Internet. Still average RT of MEMs was very close (5.6 ms).

As HTTP 204 responses contain no message body, clients can process such responses quicker than HTTP 200 responses. Also, this reduces amount of data transferred or bandwidth used, which is supported by Table III showing total local storage size required and total number and size of messages transferred during an operation. For combined state, data were collected from all machines, but results were calculated only for Ultimaker 2, to understand how collecting data sequentially from additional machines affects performance of an agent. As MEM only stores

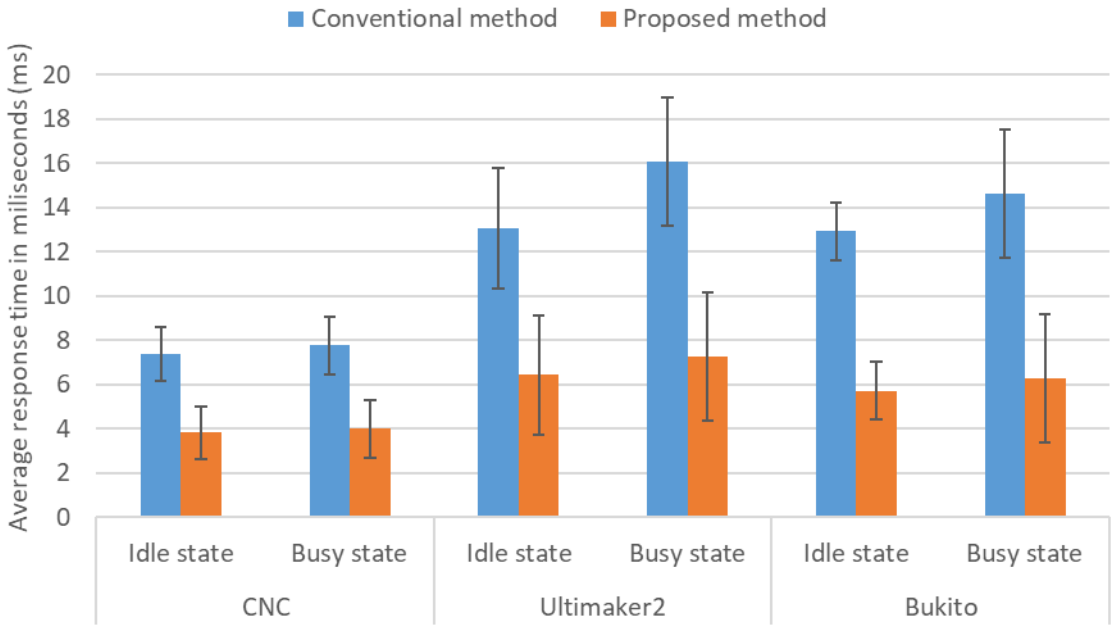


Figure 27. Comparison of average response time for legacy MTCagent and MEM

unique *dataitem* values, required size of local storage to store all available data during an operation was more than 90 percent less in all cases (96.24 percent in average), compared to a legacy MTCagent which stores all incoming data. This also shows that the minimum storage size required to ensure minimal information loss is significantly smaller for proposed MEM, which is very crucial for constrained edge devices. Table III also shows significant decrease in amount of data (bytes) being transmitted by MEMs during an operation – 85.32 percent in average. number of messages were also reduced in proposed method, except in combined state which showed a 19.32 percent increase. However, this increase actually indicates performance improvement, as it shows that proposed method was able to extract more information in combined state than conventional method. Legacy MTCagent has to process full-size responses from all three

TABLE III: Comparison of total size of local database and transferred messages and total number of messages

		X-carve	Bukito	Ultimaker 2	Combined
Total size of database (bytes)	Legacy	3147732	13085696	2787263	1171046
	MEM	8616	368640	102672	96635
	Reduction	**99.726**	**96.316**	**97.183**	**91.748**
Total bytes sent	Legacy	47739494	198461787	16489780	6928051
	MEM	111439	47679347	2854141	2686327
	Reduction	**99.767**	**97.598**	**82.691**	**61.225**
Total no. of messages	Legacy	17449	50728	4201	1765
	MEM	115	4699	2381	2106
	Reduction	**99.341**	**90.737**	**43.323**	**-19.32**

TABLE IV. Average size of response message in bytes

Machine name	Machine state	Avg. size for legacy method (bytes)	Avg. size for MEM (bytes)	Size reduction (%)
X-Carve	Idle State	2721	1.7	**99.938**
	Busy State	2725.677	4.047	**99.852**
Ultimaker 2	Idle State	3896	6.847	**99.824**
	Busy State	3923.496	64.616	**98.353**
Bukito	Idle State	3918	5.646	**99.856**
	Busy State	3906.316	33.212	**99.149**

machines before querying the first machine again. This time delay is much smaller for MEM, as its responses are shorter and, in some cases, empty. Therefore, in a given time, MEM allows clients to query one machine in a multi-machine factory floor more frequently and thus receive more messages leading to better information gain.

As depicted in Table IV, the average size of response messages also showed a significant reduction in proposed method – 99.5 percent in average. This is expected though, as MEM only sends changed values and thus generates smaller messages less frequently than a legacy MTCagent, specially in idle states. For instance, 3D printers have temperature sensors for extruders and heatbeds, which change values with room temperature. All other *dataitem* values remain unchanged in idle states. Even in busy state, not all *dataitems* generate new values in every clock cycle. For example, the "AVAILABILITY" *dataitem* of a machine only changes its value at the start and end of an operation, so is only required to be transmitted twice during a

manufacturing process. Legacy MTCagent still reports all values to client and thus wastes bandwidth.

5.4.2 Evaluation of SoC MEM prototype

To evaluate the performance of our SoC FPGA based MEM prototype, it was used with machine tools of two Uark cells of the CPMC testbed described in Chapter 4, as shown in Figure 28. The prototype replaced the previously used RPis and experiments were conducted to measure latency between client's requests and agent's responses for `current` and `operate` services.

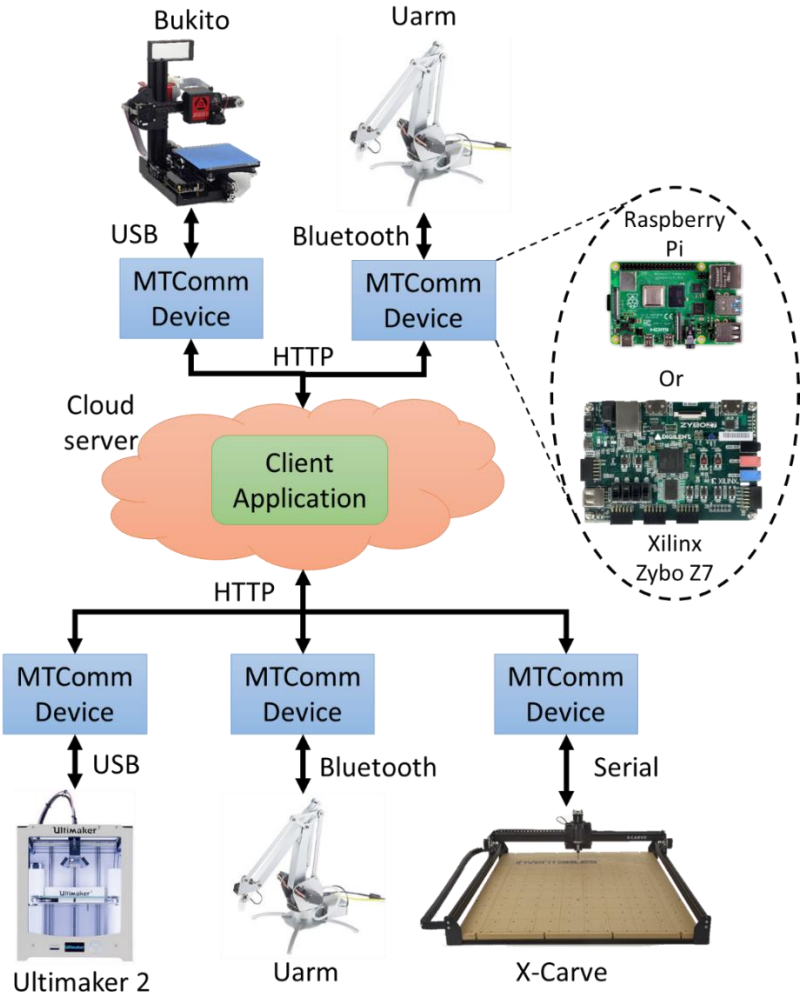


Figure 28. Experimental setup for evaluation of MEM prototype

The results at different stages of MTComm development were compared for evaluation. The stages are – stage I with initial design as described in Chapter 3 and 4, stage II after adopting optimization strategies discussed above, and finally stage III with our MEM prototype.

The first evaluation criterion was the average response time (RT) for monitoring using `current` service, which refers to the time between a client application sending a `current` request and getting a response back from an MTComm agent. Average RT was calculated for each machine type in CPMC testbed by sending 10000 consecutive requests ten times (total 100000 samples) in both idle and busy (while machine was executing an operation) states. Table V and Figure 29 show resultant average RTs in tabular and graphical formats respectively. RTs of the 3D printers were considerably higher than other machines as they had more *dataitems* and thus had larger XML response message. At each development stage, average RT was significantly reduced for all machines. Adopting the data caching and transmission optimization

TABLE V. Average response time (RT) at different stages of MTComm

Stages	**Data type**	**X-Carve**	**Ultimaker2**	**Buktio**	**Uarm**
I	Avg. RT (ms)	8.8106	14.5565	13.7675	7.5605
II	Avg. RT (ms)	4.1075	6.8374	5.9805	3.8149
	Decrease (%)	**53.38**	**53.0285**	**56.5607**	**49.5417**
III	Avg. RT (ms)	1.2184	2.534	2.0363	1.0792
	Decrease (%)	**86.1712**	**82.592**	**85.2094**	**85.7281**

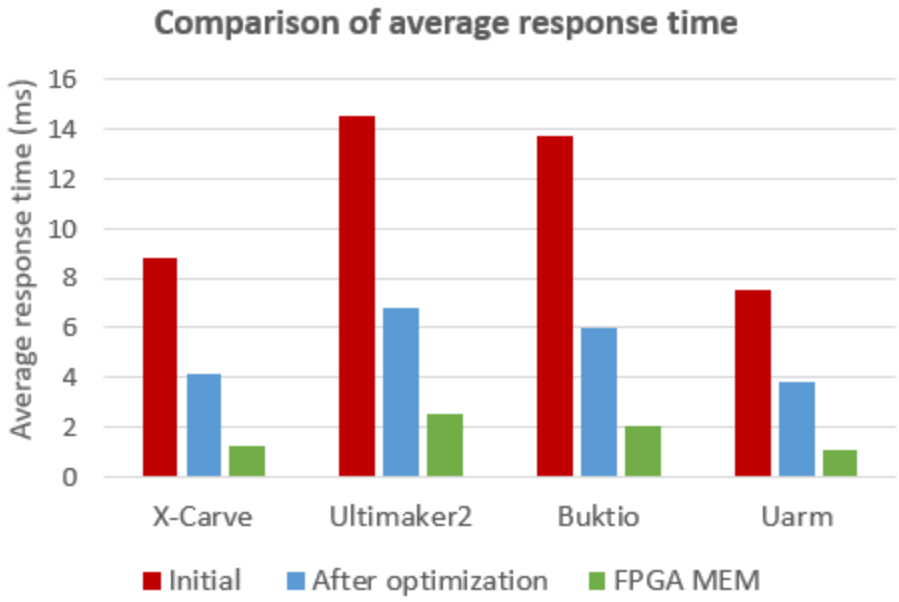


Figure 29. Comparison of average RT at different stages of MTComm

strategies resulted in an average reduction of about 53.13 percent. The reduction was even larger for our FPGA MEM prototype compared to the initial design, about 84.93 percent in average. As mentioned before, in a study by Lynn et al. (2017), average RT of an MTConnect agent in a LAN was found to be 4.2 ms In our setup, all RPIs were given global static ip addresses and cloud applications communicated with MTComm agents over Internet. Still average RT in stage II was very close (5.19 ms), while stage III RT was better (1.72 ms).

These experimental results depict that our design choices can not only enable faster data acquisition in MTComm, but also be applied to MTConnect systems and improve performance, as monitoring services are similar for both methods (Sunny, Liu & Shahriar, 2020).

Next, we measured the average operation initiation time (OIT) referring to the delay between client sending `operate` request and MTComm adapter initiating the requested *operation*. Like before, we conducted experiments in three stages. As our primary goal is to

TABLE VI: Comparison of average operation initiation time

Stage	Data type	Request with no parameter	Request with one parameter	Request with three parameters	Request with five parameters
I	Avg. OIT (ms)	988.3203	997.8515	1016.3986	1056.7108
II	Avg. OIT (ms)	5.0172	5.5094	6.1969	7.4606
	Reduction (%)	**99.4924**	**99.4479**	**99.3903**	**99.2940**
III	Avg. OIT (ms)	3.0063	3.2652	3.8735	4.8178
	Reduction (%)	**99.6958**	**99.6728**	**99.6189**	**99.5441**

compare between original (custom/POST) method with the optimized (default/GET) method, only default mode was used for stage II and III experiments. Four scenarios were simulated in each stage – `operate` request with 0, 1, 3, and 5 *parameters* respectively. Average OIT was calculated from 100 `operate` requests for each machine type in each scenario (total 400 samples). As shown in Table VI, our optimization led to remarkable reduction in OIT. Average reduction of OIT was about 99.41 percent at stage II and 99.63 percent at stage III. Average OIT for custom mode was similar in stage II with slight reduction (about 15%) in stage III. These results prove our design choices' efficiency and thus enhance MTComm's feasibility and performance.

5.5 References

de Brito, M. S., Hoque, S., Steinke, R., Willner, A., & Magedanz, T. (2018). Application of the fog computing paradigm to smart factories and cyber-physical systems. *Transactions on Emerging Telecommunications Technologies*, *29*(4), e3184.

Lynn, R., Louhichi, W., Parto, M., Wescoat, E., & Kurfess, T. (2017, June). Rapidly deployable MTConnect-based machine tool monitoring systems. In *ASME 2017 12th International Manufacturing Science and Engineering Conference collocated with the JSME/ASME*

2017 6th International Conference on Materials and Processing. American Society of Mechanical Engineers Digital Collection.

McKee, D. W., Clement, S. J., Almutairi, J., & Xu, J. (2018). Survey of advances and challenges in intelligent autonomy for distributed cyber-physical systems. *CAAI Transactions on Intelligence Technology*, *3*(2), 75-82.

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, *3*(5), 637-646.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2020). Development and optimization of an MTConnect based edge computing node for remote monitoring in cyber manufacturing systems. In 2020 IEEE International Conference on Fog Computing (ICFC). IEEE. In Press.

CHAPTER 6

FAULT DIAGNOSIS USING MTCOMM IN THE CLOUD AND AT THE EDGE

Fault diagnosis of machine tools is a crucial aspect of manufacturing as accurate and timely detection of failures and effective maintenance action can significantly reduce unplanned downtime and improve productivity (Saez et al., 2017). However, due to noise caused by components and part interactions during a manufacturing process, implementation of accurate diagnosis has not been completely feasible (Isermann, 2011). Also, health states of machine tools are typically measured and diagnosed by both visual inspection and manual maintenance which mainly relied on professional experience and knowledge of individual experts. Whenever a fault is detected, or a maintenance action is required, one or more personnel needs to physically go to the factory floor and test the machine tool in question manually. In cases of false positive detection, it often leads to waste of time, money, labor, and productivity. No research was found to offer remote online testing of manufacturing machine tools directly from cloud applications, mostly due to lack of feasible communication method and heterogeneity of machine tools.

To address this issue, we developed a fault diagnosis mechanism in CPMC that allows users to perform diagnosis and maintenance of manufacturing machine tools through both remote monitoring and online active testing using MTComm (Sunny et al., 2017; Sunny, Liu & Shahriar, 2018). With its remote operation capability, MTComm provides manufacturers with options for performing available testing operations of machine tools and associated equipment over the Internet for regular maintenance and fault diagnosis at both *device* and *component* level. Thus, manufacturers can both monitor and perform online testing and maintenance operations of different machine tools situated in geographically different locations form cloud applications without being physically present in the factory floor. MTComm allows user to confirm a

machine's condition through remote testing before sending someone to factory floor and saves time, labor, and cost. It also enhances production automation and efficiency through a cloud platform. Experiments were conducted in CPMC testbed to evaluate its performance to diagnose faults and test machines tools during various manufacturing scenarios. The results demonstrated excellent feasibility to detect anomaly during manufacturing operations and perform active testing operations remotely from cloud applications using MTComm.

6.1 Fault Diagnosis in the Cloud

6.1.1 Diagnosis Center in CPMC

Diagnosis center is a cloud application in CPMC that monitors machine tools, detects fault/anomaly if any, identifies probable component(s) causing fault/anomaly, and provides remote testing options. Depending on the number and complexity of connected machine tools, the cloud can contain multiple instances of the diagnosis center, each responsible for one or more machine tool(s). It is divided into three modules based on functionality as below (Figure 30) –

- Data monitoring (DM) module – This module constantly monitors streaming data of associated machine(s).
- Fault/anomaly detection (FAD) module – This module uses statistical techniques or machine learning algorithms to detect anomaly in the acquired data.
- Remote active testing (RAT) module – Using MTComm ontology, this module determines which component(s) is the most probable candidate for causing fault/anomaly and provides dynamic options for performing online active testing of that component(s) over the Internet.

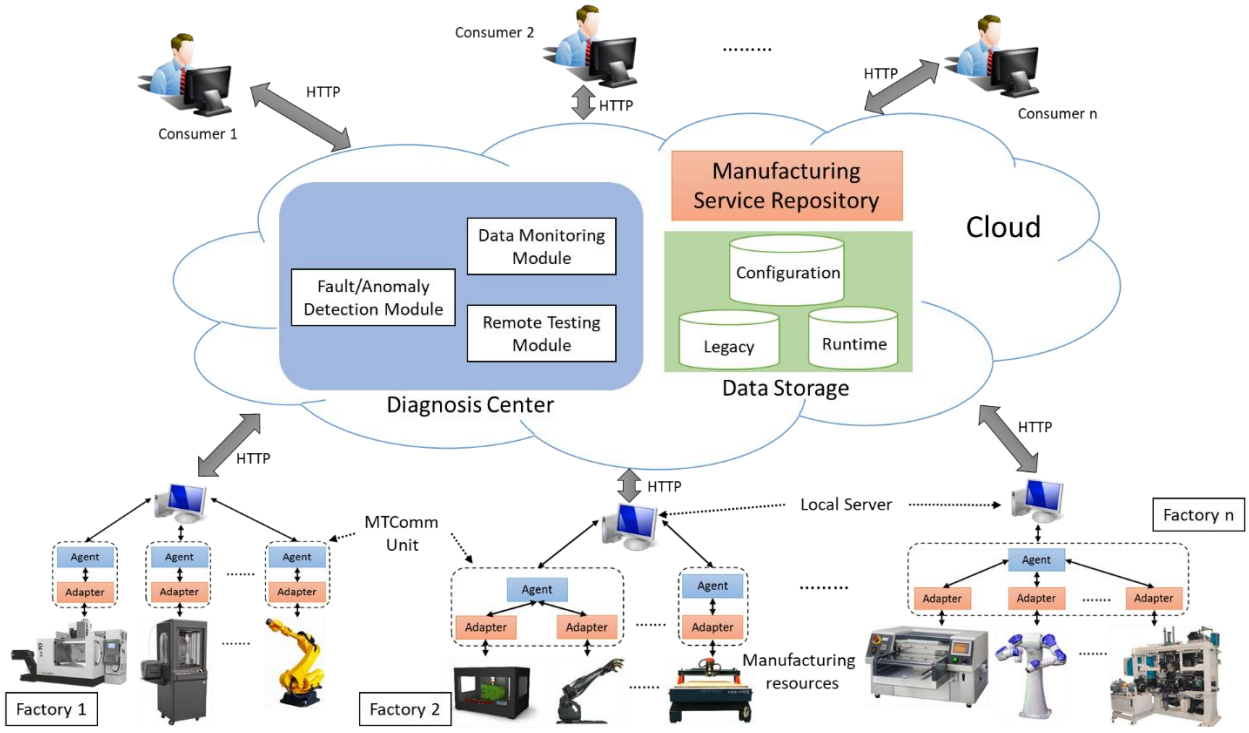


Figure 30: Conceptual framework for fault diagnosis center in CPMC

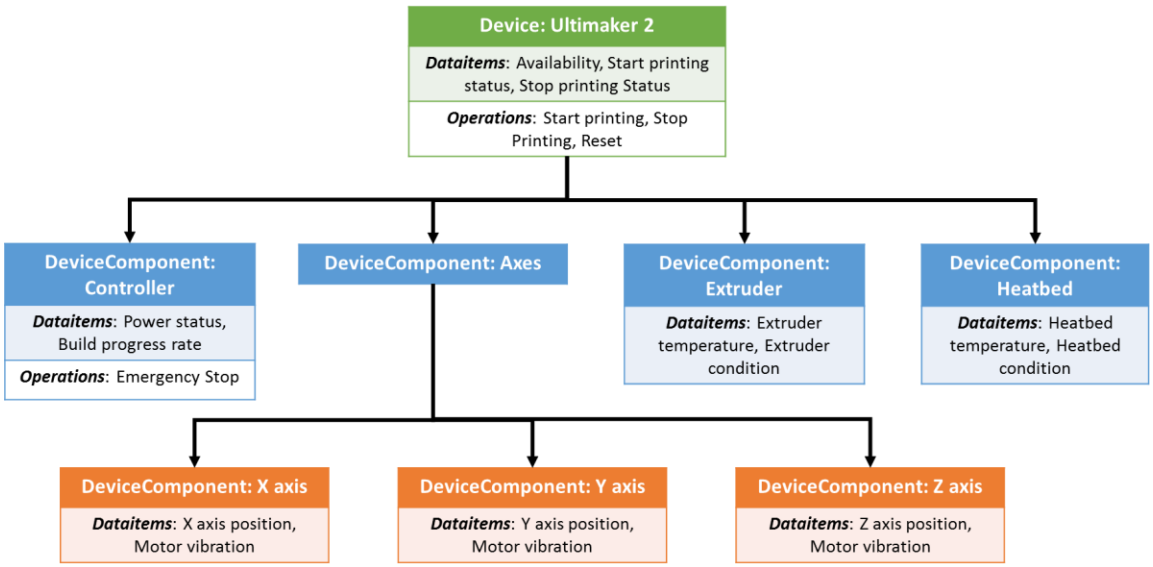
6.1.2 Fault/anomaly detection using MTComm

For efficient fault/anomaly detection using MTComm, we extended existing MTComm mechanism by adding two operational modes to MTComm `probe`, `current`, and `sample` services - NORMAL and DIAGNOSIS. In NORMAL mode, MTComm shows only machine tool and its components' selective data elements and operations. In DIAGNOSIS mode, however, MTComm provides all available information of factory floor including interconnections, software (agent and adapter), machine tool and its components. It also offers more operations than NORMAL mode, operations that are to be used for testing and maintenance. The reason behind having two different mode is two-fold. Firstly, to perform usual machine operations and monitor machine health, not all available data are necessary. For example, to print objects with a 3D printer, information about specific component operations like

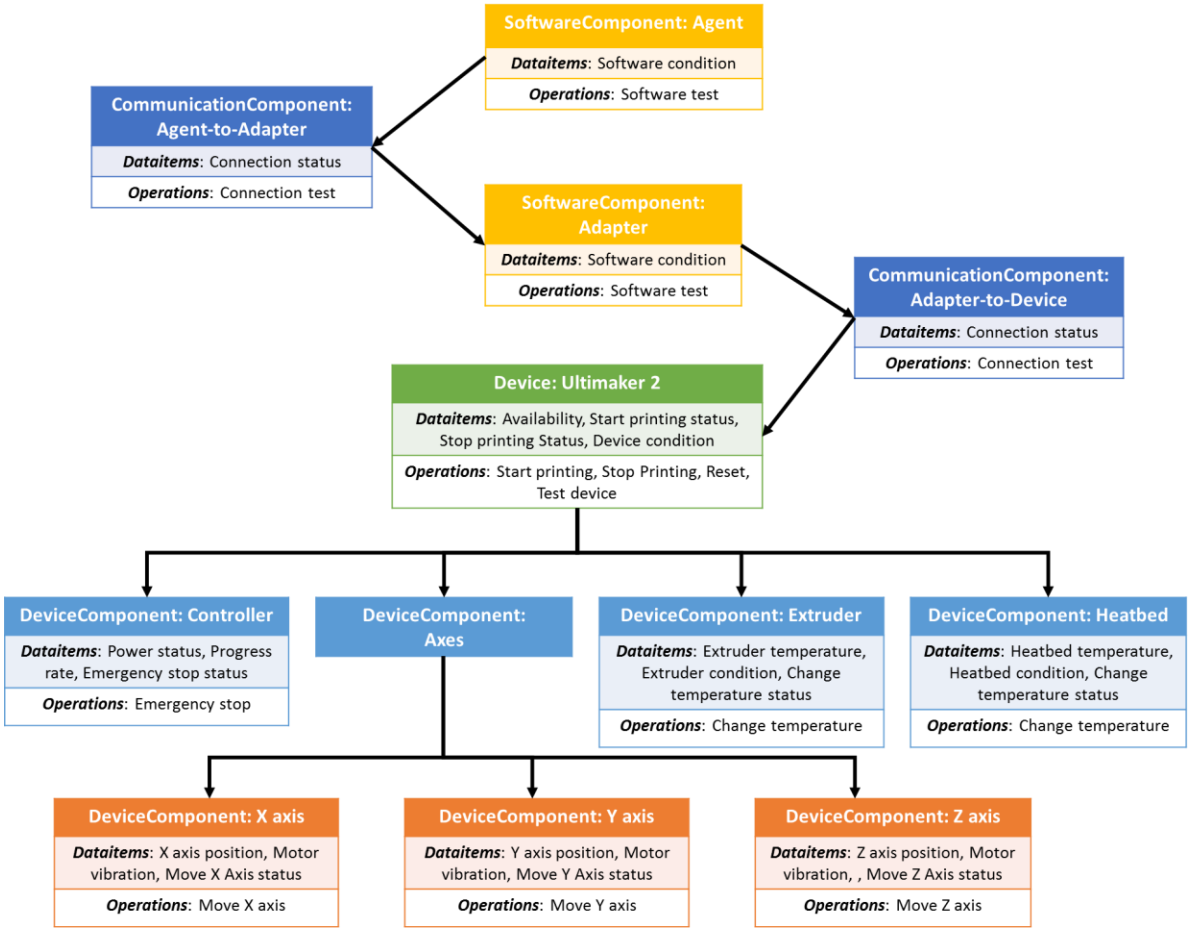
axes movement are not required. Secondly, it gives manufacturers the flexibility to choose which data to be transmitted and how frequently. For diagnosis purpose, it may require acquiring data at a higher frequency than during a normal operation. Therefore, streaming the primary data not only reduces the time required for data acquisition, conversion (to XML), and transmission, but also optimizes bandwidth use and factory floor network traffic. However, having two modes is not a must; it is up to a manufacturer to select how many modes there would be, and which information would be transmitted when. Thus, MTComm provides scalability and flexibility to manufacturers. Figure 31 illustrates what type of information is provided by `probe` service of a 3D printer for both modes.

In MTComm, the type of an EVENT or SAMPLE *dataitems* is either ACTUAL or INSTRUCTIONAL (INST). ACTUAL type *dataitems* are data representing actual value collected by from sensors. INST type *dataitems* show values given by a machine's controller unit. Most manufacturing machine tools require a set of machine commands to perform an operation. When queried, the machine's controller reports certain data, like axes positions, based on the command being executed. These data are INST values. In normal conditions, the difference between ACTUAL and INST value of a *dataitem* should be within an acceptable threshold. This feature is used to identify anomaly with EVENT and SAMPLE *dataitems*. As EVENT category *dataitems* can only have a few pre-defined string values, anomaly is detected when INST and ACTUAL values do not match. Anomaly with SAMPLE *dataitems*, on the other hand, is more complex to detect and requires anomaly detection techniques.

Let a machine tool has *n* components. The machine itself and each component has *m* unique *dataitem*s. For simplicity, let all unique *dataitems* have both ACTUAL value (Da_1, Da_2, …, Da_m) and INST value (Di_1, Di_2, …, Di_m). ACTUAL values can be represented as



(a) NORMAL mode



(b) DIAGNOSIS mode

Figure 31. Information provided by `probe` service of a 3D printer in both modes

$$Da_k{}^t = Di_k{}^t \pm Th_k \tag{1}$$

where $Da_k{}^t$ and $Di_k{}^t$ are ACTUAL and INSTRUCTIONAL value of *k*-th *dataitem* at time *t*, and Th_k is the threshold value for *k*-th *dataitem*. For each *dataitem*, deviation between two values at time *t* is calculated –

$$d_k{}^t = Da_k{}^t - Di_k{}^t \tag{2}$$

An anomaly flag (*a*) is calculated for each *dataitem* –

$$a_k{}^t = \begin{cases} 1, & if\ d_k{}^t < LTh_k\ or\ d_k{}^t > UTh_k \\ 0, & otherwise \end{cases} \tag{3}$$

where 0 indicates no anomaly was detected and 1 means an anomaly was flagged, UTh_k and LTh_k refer to upper and lower threshold values respectively. Determining these threshold values is very crucial, as it is used as the ultimate decider to identify a data as anomaly. Having a pre-determined threshold value can cause erroneous calculation. Therefore, the threshold values are determined by statistical techniques or machine learning algorithms for anomaly detection. The proposed system does not specify which detection technique to be used.

For data that can only be acquired from sensors, not from machine controller (e.g. motor vibration, acceleration, material weight etc.), outliers are detected by three-sigma rule using only ACTUAL values. In practice, instead of calculating anomaly flag for a single time instant, the calculation is done over a small time window to reduce occurrence of false positive incidents. For a time window of length t_w, the overall anomaly flag (*A*) –

$$A_k{}^t = \begin{cases} 1, & if \sum_{t=t-t_w}^{t} a_k{}^t > ATh_k \\ 0, & otherwise \end{cases} \tag{4}$$

Here, ATh_k refers to anomaly threshold which is the acceptable number of anomalies for a certain time window. Anomaly scores are calculated for 'Event' *dataitem* s as well.

A machine's total anomaly score is calculated by summing anomaly flags of all *dataitem* s for a given time. However, some *dataitems* are more sensitive than others. For example, slight variation of axis position of a CNC machine is acceptable during an operation, but if the vibration of the drill motor is off the chart, the operation needs to be terminated immediately. For this reason, each *dataitem* is assigned a certain weight (*w*) which is multiplied to the anomaly flag of that *dataitem*. The value of *w* increases with the priority and sensitivity of the *dataitem.* Therefore, a machine's anomaly score (*MAS*) at time *t* is given by –

$$MAS = \sum_{j=1}^{n} \sum_{k=1}^{m} {w_{jk}}^{t} . {A_{jk}}^{t} \tag{5}$$

If the FAD module detects a high *MAS,* an *alarm* is raised, and an emergency stop command is sent to the machine's agent. If *MAS* value is in medium range, FAD issues a *warning*, but the operation continues. In either case, FAD initiates DIAGNOSIS mode after an anomaly is detected and the operation is stopped or finished.

6.1.3 Online Active Testing using MTComm

Using MTComm's remote operation capability, the remote active testing (RAT) module in the diagnosis center enables testing of machine tools and associated components over the Internet. When a machine is connected to the CPMC for the first time, its corresponding RAT module collects its ontological representation and all available information including its organization, available *dataitems* and *operations* via MTComm `probe` service in DIAGNOSIS mode. RAT uses these information to create a virtual copy of the machine and associated

components using its MTComm ontology. The hierarchical representation clearly defines a factory floor's organization and allows RAT module to easily pinpoint components related to an anomalous *dataitem* and provide model driven dynamic online testing options. RAT can be used to perform testing of a machine tool regardless of occurrence of a fault. In normal conditions, it offers user to perform any available testing operation remotely. In case of a fault, RAT provides dynamic testing options – it tries to identify the most probable component(s) responsible for the fault and shows specific related operations available to test the component(s). This is done using weighted anomaly scores calculated by FAD. Components with high anomaly scores are most likely to be causing faults.

Online active testing operations are divided into three categories – software testing, interconnection testing, and machine testing. All testing operations are performed via RESTful web services hosted by the agent. Software testing operations involve running pre-defined scripts to test various functionalities of the MTComm agent and adapter programs. The agent testing is performed from the cloud application, while the adapter testing is conducted by the agent, as there is no direct link between the cloud and the adapter. There are three primary interconnections in a CPMC setup – between cloud application and agent, between agent and adapter, and between adapter and machine. Each of them has its own interconnection testing operation. Machine testing operations include testing of a machine as-a-whole and its components. A machine tool as-a-whole is tested either by running a set of pre-defined commands or by performing a certain machine level operation. Component testing operations involve performing operations with a specific component. Operations are performed using MTComm's `operate` service. It involves sending description and parameters of an operation in

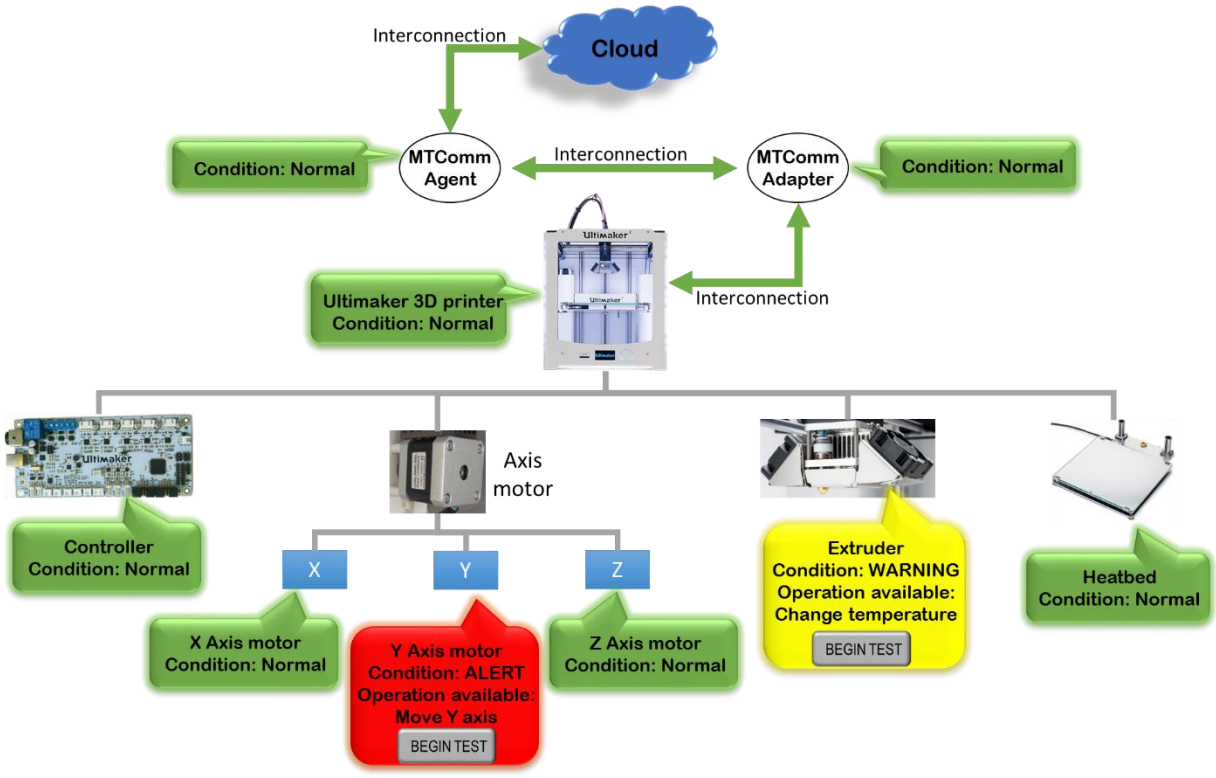


Figure 32. Example of user interface for RAT module of a diagnosis center in CPMC

MTComm XML format to the machine's agent. Specific testing services can be added if necessary.

Figure 32 illustrates an example user interface of a RAT module for a 3D printer. It represents a state when Y axis motor was disconnected, and extruder temperature was slightly drifting from its normal values. When performing testing operations over the Internet, the data monitoring and FAD module respectively acquires and analyzes data in DIAGNOSIS mode. The collected data and analysis results are shown to the user to help remotely identifying which component is behaving erratically and is causing fault(s).

6.2 Fault diagnosis at the edge

Data-driven fault diagnosis and prognosis is typically performed in central cloud servers

or local servers (private/edge cloud), as data analysis algorithms with better accuracy require extensive computational power. However, despite radical improvement in data transmission speed and hardware processing time, there still exists considerable delay between fault occurrence in factory floor and fault detection in servers. To improve current scenario, we propose to include lightweight *in-situ* fault diagnosis mechanisms in MEMs to provide faster fault detection and response. It should be noted that, MEMs are not a replacement of existing FD systems. Rather an MEM aims to assist cloud-based data analysis programs by performing small-scale repetitive data processing tasks close to physical machines. Thus in a CPMC, MEMs perform preliminary fault detection, while FD system in cloud servers conduct more advanced analysis. The responsibilities of an MEM FD module is three-fold - data cleaning, fault detection, and mitigation.

A. Data cleaning/filtering

Not all changes in sensor data are significant for fault diagnosis. For example, a typical temperature sensor can measure the temperature of a 3D printer's extruder up to 45 decimal places. Changes after 2 decimal places can often be disregarded, specially in idle states. So, instead of sending all changed data to cloud servers, an MEM performs data cleaning or filtering before transmission by discarding unnecessary data values. This filtering can be based on manual specifications or statistical computation. In the former method, a user specifies criteria for data cleaning (e.g. sending temperature values with 2 decimal places only). An MEM can also perform simple statistical data analysis to calculate threshold values and identify certain anomalous data points that are within acceptable range and can be ignored. Thus MEMs can not only reduce bandwidth usage, but also improve overall data processing efficiency of FD in a CPMC.

B. Fault detection

The FD module monitors streaming time series sensor data to identify possible anomalous data points. Training with data from previously normal operations can be done in two ways - user can provide old data files at the beginning, or MEM can collect data from several operations and use those as training dataset (assuming normal operations). Once training is complete, FD module uses the trained model to analyze new data values and identify possible anomalies. Each incoming data value is given one of three anomaly scores based on a threshold upper and lower limit - normal (0), warning (0.5), fault (1). If warning thresholds are not available, then a data point is either normal or fault. Threshold values *(th)* for each category are computed using pre-selected anomaly detection algorithm. To improve accuracy, anomaly scores *(A)* are calculated by adding scores over a small time window *(tw)* (6). Using MTConnect's semantic ontology, an MEM calculates individual anomaly scores for different *dataitems* of a *device* and finally compute an overall machine anomaly score (MAS) to determine if the machine is normal or faulty. In most manufacturing processes, *dataitems* have different levels of priority and sensitivity. So each *dataitem* is assigned a certain weight (*w*) which is multiplied with its anomaly score to compute MAS (5). Thus even if multiple sensors have "warning" values, the MAS can be either faulty or normal depending on their weights.

$$A_k^t = \begin{cases} 1, \text{ if } \sum_{t-t_w}^{t} a_k^t \geq fTh_k \\ 0.5, \text{ if } wTh_k \leq \sum_{t-t_w}^{t} a_k^t \leq fTh_k \\ 0, \text{ otherwise} \end{cases} \quad (6)$$

As MEMs are designed for constrained embedded platforms, it is necessary to determine which FD algorithms are best suited for them. The most efficient FD method for MEMs is the one that can autonomously detect anomalies with best accuracy with least amount of training data (storage) and lowest delay. As most sensors generate univariate one-dimensional time series

data, simple statistical methods can be used to identify point anomalies. Instead of limiting to develop and fine-tune a single algorithm, we explored different commonly used approaches described below as proof-of-concept.

1) Three-sigma (3σ) *rule:* The concept of three-sigma rule of thumb is that nearly all values of a distribution lie within three standard deviations (σ) of the mean (μ) (Pukelsheim, 1994). Any value beyond this 3σ limit is considered as an outlier. For normal distributions, 99.73% values lie within 3σ limit and 95.45% within 2σ limit. From previous experiments with our testbed, we found that most of the sensor data are normally distributed, thus were suitable to be used with the three-sigma rule. In MEM, we identified 2σ limit as warning threshold and 3σ limit as fault threshold, as shown in (9).

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2 \tag{8}$$

$$a_k^t = \begin{cases} 1, & \text{if } value^t < \mu - 3\sigma \text{ or } value^t > \mu - 3\sigma \\ 0.5, & \text{if } \mu - 3\sigma \leq value^t \leq \mu - 2\sigma \\ & \quad \text{or } \mu + 2\sigma \leq value^t \leq \mu + 2\sigma \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

2) Interquartile range (IQR): Interquartile range (IQR) refers to the difference between 75th and 25th percentiles, or between upper (Q3) and lower (Q1) quartiles of a distribution (Upton & Cook, 1996). IQR is often used to find mild and extreme outliers in data with median as center. In MEM, IQR is used as below –

$$a_k^t = \begin{cases} 1, & \text{if } v^t < Q1 - 3IQR \text{ or } v^t > Q3 + 3IQR \\ 0.5, & \text{if } Q1 - 3IQR \leq v^t \leq Q1 - 1.5IQR \\ & \quad \text{or } Q3 + 1.5IQR \leq v^t \leq Q3 + 3IQR \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

3) Moving Average Filter (MAF) and Moving Median Filter (MMF): Moving average filter is widely used for anomaly detection and eliminating white noise in sensor data. When a new data arrives, average and standard deviation (σ) is calculated of last W data values, where W is a sliding window of fixed size (Upton & Cook, 1996), as shown in Equation (7) and (8). If new data differs too much from moving average (more than σ), it is registered as outlier. A hyper-parameter (between 0 and 1) is used to regulate sensitivity to anomalies, and thus identify mild and extreme outliers.

The Moving Median Filter is similar, but uses median instead of average, as median is more robust against anomalies (Hochenbaum, Vallis & Kejariwal, 2017). One advantage of using MAF or MMF in MEM is that we can add new normal data points to existing window and calculate new moving average and (σ) keeping the size of window, and thus storage, fixed.

4) Seasonal Hybrid ESD (SH-ESD): This algorithm was developed by Twitter as an extension of the Extreme Studentized Deviate (ESD) test (Rosner, 1975) for finding outliers in univariate time series data (Hochenbaum, Vallis & Kejariwal, 2017). It uses an upper bound on the number of anomalies. In the worst case, the number of anomalies can be at most 49.9% of the total number of data points in the given time series. In practice, our observation, based on production data, was that the number of anomalies was typically less than 1% in the context of application metrics and less than 5% in the context of system metrics.

ESD test is defined for the hypothesis:

H_0: There are no outliers in the data set

H_1: There are up to k outliers in the data set

A test statistic is computed for a given time series data set which is defined as follows –

$$C_k = \frac{max_k\,|x_k - \bar{x}|}{s} \tag{11}$$

Where,

$\bar{x}$ = mean of the time series

s = standard deviation of the time series

The test statistic is then compared with a critical value, computed using Equation (12), to determine whether a value is anomalous (Rosner, 1983). The number of anomalies is determined by finding the largest *k* such than $C_k > \lambda_k$.

$$\lambda_k = \frac{(n-k)t_{p,n-k-1}}{\sqrt{(n-k+1+t^2_{p,n-k-1})(n-k+1)}} \tag{12}$$

$$p = 1 - \frac{\alpha}{2(n-k+1)} \tag{13}$$

Where,

n = total number of data points

$t_{p,n-k-1}$ = the 100p percentage point from the *t* distribution with *n-k-1* degrees of freedom

α = significance level

SH-ESD first decomposes a time series using Seasonal-Trend Decomposition (STL) (Cleveland et al., 1990) to extract seasonality (*S*). Then the residual (*R*), which is the distance between observed data and the best-fit curve calculated by STL, is calculated using Equation (14), where *X* is the raw time series and $\tilde{X}$ is the median of the raw time series.

$$R = X - S - \tilde{X} \tag{14}$$

The residual has a normal distribution and thus is compliant with anomaly detection techniques like ESD. However, mean and standard deviation are sensitive to anomalous data. Therefore, the better choice here is to use median and median absolute deviation (MAD), as these metrics are more robust against anomalies. For a univariate data set *X1, X2, ..., Xn*, MAD is

defined as the median of the absolute deviations from the sample median, as shown below –

$$MAD = median_i(|X_i - median_j(X_j)|) \quad (15)$$

So, the test statistics of ESD becomes

$$C_k = \frac{max_k\,|x_k - \tilde{x}|}{MAD} \quad (16)$$

To identify point anomaly in MEM, a new data point is added to testing dataset and then SH-ESD is used to find two anomaly points in the new dataset. If the new data is found among anomalies, it is considered as an anomaly.

5) Univariate Gaussian Predictor (UGP): The Univariate Gaussian Predictor creates a historical model of time series data and then predict and compare new values (Hayes & Capretz, 2014). It calculates mean (μ) and variance (σ^2) from training data and then classifies each new value (x^t) based on its probability in the distribution, $p(x^t)$ which is defined by Equation (17). A threshold value is set to decide which probabilities are lower than the threshold value to be considered as anomalies.

$$p(x^t) = \frac{1}{\sqrt{2\pi\sigma}}\exp - \frac{(x^t - \mu)^2}{2\sigma^2} \quad (17)$$

C. Mitigation

When a fault is detected, an MEM automatically initiates pre-defined emergency mitigation tasks such as immediately pause/stop the current operation, power off faulty machines or tools, sounding an alarm etc. MEM also adopts active testing mechanism described in Section 6.1.3 in small-scale. The FD module uses MTComm ontology and anomaly scores to pin-point faulty component(s) and reports to client. If testing routines for specific components are available, MEM can also perform autonomous testing of fault component(s). As an MEM is in

close proximity of a machine, it can complete such actions much faster than a cloud-based FD system, and thus prevent possible hazardous situations.

6.3 Experiments in CPMC testbed and evaluation

6.3.1 Experiments of fault diagnosis in the cloud

A cloud application for diagnosis center was developed and deployed in the CPMC testbed for each machine tool in Uark sites. Each diagnosis center was trained with data acquired from five normal operations of the associated machine. Table VII lists *Dataitems*, faults, and available testing operations of the machines in the testbed. Alongside MTComm's six primary services, two more services were added to each agent – `ping`, which tested interconnections, and `log`, which kept a historical log of events. Faults were created intentionally, both before and during operations. MTComm allows machine tools to communicate with each other autonomously, which facilitates to perform collaborative manufacturing operations involving multiple machines. The bottom row of Table VII refers to collaborative operation scenarios. Experiments conducted with the prototype system used two statistical anomaly detection method – three sigma rule (Pukelsheim 1994) and Tukey's method (Tukey 1949). The concept of three-sigma rule of thumb is that nearly all values (99.7%) lie within three standard deviations (σ) of the mean (μ). Any value beyond this 3σ limit is considered as an outlier. For this method, anomaly scores are calculated with following equation –

$$a_k{}^t = \begin{cases} 1, & if\ Da_k{}^t < \mu_k - 3\sigma_k \quad or \quad Da_k{}^t > \mu_k + 3\sigma_k \\ 0, & otherwise \end{cases} \tag{18}$$

Tukey's method uses the inter-quartile range ($|Q_3\text{-}Q_1|$, where Q_1 and Q_3 are first and third quartile respectively) to determine upper and lower threshold value. Using this method, equation (19) becomes –

TABLE VII. *Dataitems*, faults, and available testing *Operations* in CPMC testbed

Machine tool	Primary *Dataitem*s	Faults	Testing *Operation*s
3D printers (Ultimaker 2, Core XZ, Bukito)	Machine availability, Axes positions, Material weight, Motor vibration, Progress rate, Extruder temperature, Heatbed temperature (not for Bukito)	Broken connection (between adapter and machine, between agent and adapter)	Use `ping` service to test interconnections
		No material loaded	No test available
		X, Y or Z axis motor disconnected	Move corresponding axis
		Very High/low extruder or heatbed temperature	Change extruder or heatbed temperature
CNC Machine (X-carve)	Machine availability, Axes positions, Axes Motor vibration, Drill motor vibration, Progress rate	Broken connection (between adapter and machine, between agent and adapter)	Use `ping` service to test interconnections
		Drill motor has no power	Run drill motor
		X, Y or Z axis motor disconnected	Move corresponding axis
Robotic Arms (U-arm)	Machine availability, Axes positions, Axes Motor vibration, Progress rate	Broken connection (between adapter and machine, between agent and adapter)	Use `ping` service to test interconnections
		X, Y or Z axis motor disconnected (before and during operation)	Move corresponding axis
Collaborat-ive operation	All dataitems of associated three machines (Ultimaker 2, X-Carve, one U-arm)	Broken connection (between Ultimaker 2 and U-arm, between U-arm and X-carve)	Test connection with the faulty agent

$$a_k{}^t = \begin{cases} 1, & if\ Da_k{}^t < Q_1 - 3|Q_3 - Q_1|\ \ or\ \ Da_k{}^t > Q_3 + 3|Q_3 - Q_1| \\ 0, & otherwise \end{cases} \quad (19)$$

Figure 33 illustrates examples of anomaly detection with time series data for two *dataitems* (Extruder temperature of Ultimaker 3D printer and Y axis position of CNC machine). The response time to detect an anomaly/fault by the diagnosis center was found to be around 200-500 milliseconds. The average time to initiate an operation was about 1 second. This higher value was expected as verification of incoming operation request by agent added some delay. Experiments were also conducted to determine effectiveness of fault detection with INSTRUCTIONAL value and remote testing capabilities using MTComm over the Internet. 40 operations were conducted with four target *dataitems* (10 operations each, 5 with fault and 5 without fault) of two machines – x axis position and extruder temperature of a 3D printer, and y axis position and drill motor vibration of a CNC machine. Both anomaly detection techniques were used in each case. To evaluate the efficiency, F-scores were calculated by using following set of equations (Hochenbaum, Vallis & Kejariwal, 2017) –

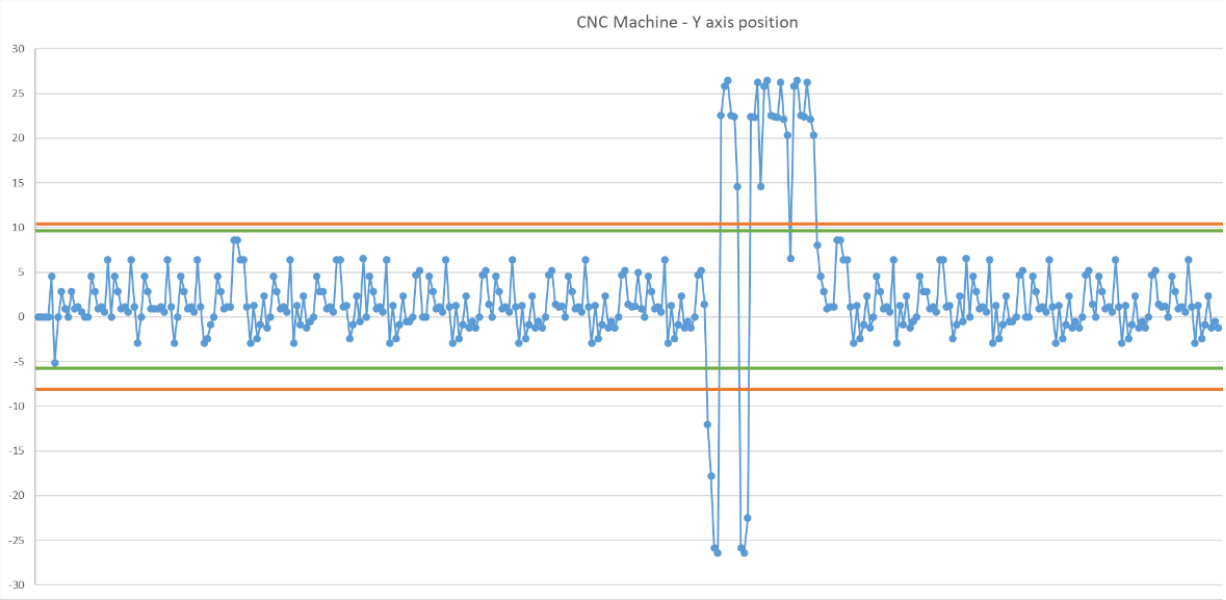
$$Precision = \frac{number\ of\ true\ postives}{number\ of\ true\ positives + number\ of\ false\ positives} \quad (20)$$

$$Recall = \frac{number\ of\ true\ postives}{number\ of\ true\ positives + number\ of\ false\ negtives} \quad (21)$$

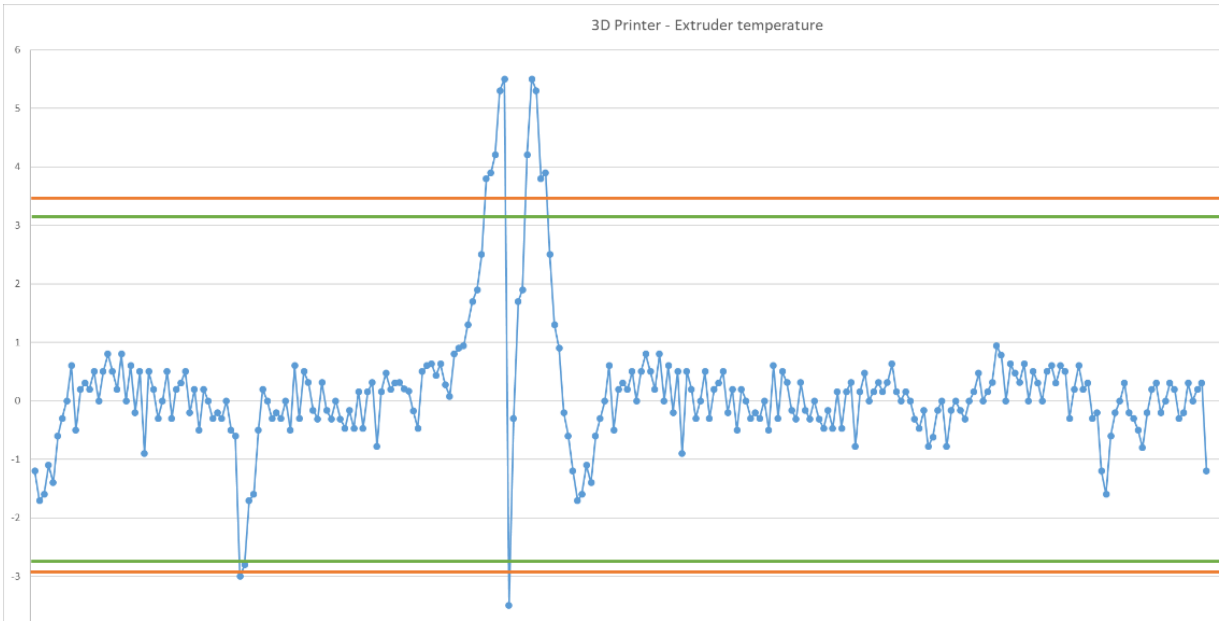
$$F = 2 \times \frac{precision \times recall}{precision + recall} \quad (22)$$

Table VIII summarizes the outcomes. Analysis with only ACTUAL values was unable to determine faults with axes positions. The results indicate that using INSTRUCTIONAL value increases overall accuracy for both anomaly detection methods. When remote testing capability was used to confirm faults, accuracy was even higher. It was also observed that while remote testing significantly reduced number of false positive cases, it was not useful in false negative

cases as no error was detected. To reduce occurrence of false negatives, anomaly detection method with higher accuracy is required.



(a) Data of Y axis position of 'X-Carve' CNC machine



(b) Data of extruder temperature of 'Ultimaker 2' 3D printer

Figure 33. Graphs showing example time series data machine tools. Sudden spikes are due to intentional faults. Green and red lines are thresholds calculated by three-sigma and Tukey's method respectively

TABLE VIII. F-score (%) in different scenarios

Anomaly detection methods used	without INSTRUCTIONAL values		with INSTRUCTIONAL values	
	before remote testing	after remote testing	before remote testing	after remote testing
three-sigma	54.55	57.14	91.89	97.44
Tukey's	57.14	59.65	96.10	97.47

6.3.2 Experiments of fault diagnosis at the edge

Experiments of fault diagnosis in MEM were conducted with the same CPMC testbed described in Section 4.4.1. Fault diagnosis experiments were conducted with two 3D printers. For simplicity, we focused on two *dataitems* - vibration of x axis motor and temperature of the extruder, and categorized data points as either normal or fault. Both sensor values were found to be normally distributed. Three training datasets with data from one(tr-1), five(tr-5), and ten(tr-10) normal operations respectively were provided a-priori. Two types of faults were manually induced during a 3D printing operation - hitting the motor (sudden change in vibration) and changing extruder temperature (both sudden and gradual change). Once training was done, same operation was run five times and two faults, one of each type, were induced each time. Overall F-score and average detection delay (time between faulty data arrival and detection) for different FD methods are shown in Table IX.

All FD methods were able to detect induced faults in almost all cases, as faulty data had significant gap with normal data, but numbers of false positive detection were widely varied. In case of 3σ, IQR, and UGP, parameters like mean, variance etc. were calculated before an operation started, so anomaly detection was almost instantaneous (delay in UGP was higher as

TABLE IX. Overall F-score (%) with different training datasets and average detection delay (ms) for different FD methods

Method name	F-score (%) with tr-1	F-score (%) with tr-5	F-score (%) with tr-10	Average delay(ms)
3σ	38.461	54.054	68.966	<< **0.001**
IQR	43.478	52.941	71.429	<< **0.001**
MAF	51.428	54.545	58.824	0.045
MMF	54.054	58.065	64.516	0.105
SH-ESD	**78.261**	**83.333**	**90.909**	269
UGP	58.064	66.667	74.074	1.795

its calculation was more complex). But their accuracy increases with large training data, meaning they require more storage size. For MAF and MMF, storage size is fixed as both use a pre-defined window size. Thus increasing training dataset had little effect on accuracy. Their delay was rather small, due to their simple statistical computation. SH-ESD showed best result accuracy in all cases, but also had the worst average delay, which became even larger with increased training data. So, the results clearly indicate that performing lightweight statistical FD methods in MEMs can achieve acceptable accuracy; however, there is a trade-off between accuracy and required storage and computational delay. So, it is up to manufacturer's discretion to choose between accuracy and cost of implementation.

6.4 References

Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. (1990). STL: A seasonal-trend decomposition. *Journal of official statistics*, *6*(1), 3-73.

Hayes, M. A., & Capretz, M. A. (2014, June). Contextual anomaly detection in big sensor data. In *2014 IEEE International Congress on Big Data* (pp. 64-71). IEEE.

Hochenbaum, J., Vallis, O. S., & Kejariwal, A. (2017). Automatic anomaly detection in the cloud via statistical learning. *arXiv preprint arXiv:1704.07706*.

Isermann, R. (2011). *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Springer Science & Business Media.

Pukelsheim, F. (1994). The three sigma rule. *The American Statistician*, *48*(2), 88-91.

Rosner, B. (1975). On the detection of many outliers. *Technometrics*, *17*(2), 221-227.

Rosner, B. (1983). Percentage points for a generalized ESD many-outlier procedure. *Technometrics*, *25*(2), 165-172.

Saez, M., Maturana, F., Barton, K., & Tilbury, D. (2017, August). Anomaly detection and productivity analysis for cyber-physical systems in manufacturing. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)* (pp. 23-29). IEEE.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

Sunny, S. M. N. A., Liu, X., & Shahriar, M. R. (2018, July). Remote Monitoring and Online Testing of Machine Tools for Fault Diagnosis and Maintenance Using MTComm in a Cyber-Physical Manufacturing Cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 532-539). IEEE.

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, 99-114.

Upton, G., & Cook, I. (1996). *Understanding statistics*. Oxford University Press.

CHAPTER 7

IOT ENABLED ON-DEMAND GROCERY SHOPPING AND DELIVERY CLOUD USING MTCOMM AT THE EDGE

MTComm enables interoperability of heterogeneous machine tools and associated equipment over the Internet. To expand its capabilities to other domains, we investigated its feasibility to establish cross-domain collaboration between IoT devices and machine tools. As a result, we designed and developed an integrated IoT enabled on-demand grocery shopping and delivery cloud (IGSDC) using MTComm at the edge (Sunny, Liu & Shahriar, 2019). A study in 2015 found that 25 percent of U.S. consumers ordered groceries online, and 55 percent were willing to do so. Internet-of-Things (IoT) is making significant contributions to improve the current trends of online grocery services (Fagerstrøm, Eriksson & Sigurðsson, 2017). As grocery purchases are usually done on needbasis, IoT is aiding online grocery services by allowing consumers to keep track of grocery items, set reminders for expiration, and even order directly from built-in interfaces (Desai et al., 2017). Moreover, suppliers and retailers are using IoT technologies to improve management and distribution of products and promote better shopping experience. The Internet-of-Shopping is considered as the future of supermarkets where no human intervention is required to manage it. Innovations in robotics, artificial intelligence, and computer vision made product delivery by self-driving robots a reality (Cho et al., 2014). Integrating in-home and in-warehouse IoT devices and robots through cloud services has the promise of creating a robust and efficient on-demand grocery shopping and delivery platform that connects consumers and suppliers based on their needs and offered services respectively and allows them to monitor, manage, and even operate their physical resources over the Internet. Heterogeneity of communication protocols used by such resources is a major challenge to

develop such a system. IoT devices use various communication protocols, such as MQTT, CoAP, Z-wave, Zigbee etc., data formats, definitions, and system architectures (Sill, 2017). Moreover, industrial resources use their own communication protocols, e.g. Ethernet/IP, Modbus, PROFINET, EtherCAT etc., and standards (Sunny, Liu & Shahriar, 2018). Edge computing can aid by transforming data near end devices to a common format and provide cloud applications a uniform interface to interact with them (Shi et al., 2016).

In IGSDC, we developed an MTComm based edge computing hub (MEH) for IoT home appliances and devices, as well as industrial machine tools to enable service-oriented cloud based on-demand grocery product tracking and ordering capabilities. An MEH communicates with different types of IoT devices and robot systems using their own languages and protocols. The collected data is converted to a common MTComm XML format and sent to cloud services. Based on IoT data, the cloud notifies consumers when they are running low on grocery products and offers available ordering options from different suppliers through mobile app. On the other hand, suppliers connect their warehouse IoT devices and delivery robots using MEH to the cloud by publishing their monitoring and operation capabilities as RESTful web services and perform product delivery remotely upon receiving an order. As both in-home and in-warehouse resources communicate with the cloud through MEH using MTComm XML messages over the Internet, the IGSDC system does not require to deploy separate cloud services and applications for consumers and suppliers, hence provides better scalability and efficiency. As many robot systems used in manufacturing and warehouse environments share similar operation principles, MTComm can be easily applied to in-warehouse robot systems. However, in-home IoT devices are not readily compliant with MTComm. Therefore, we developed a mechanism to adopt MTComm for IoT resources. Experiments with a prototype IGSDC system demonstrated

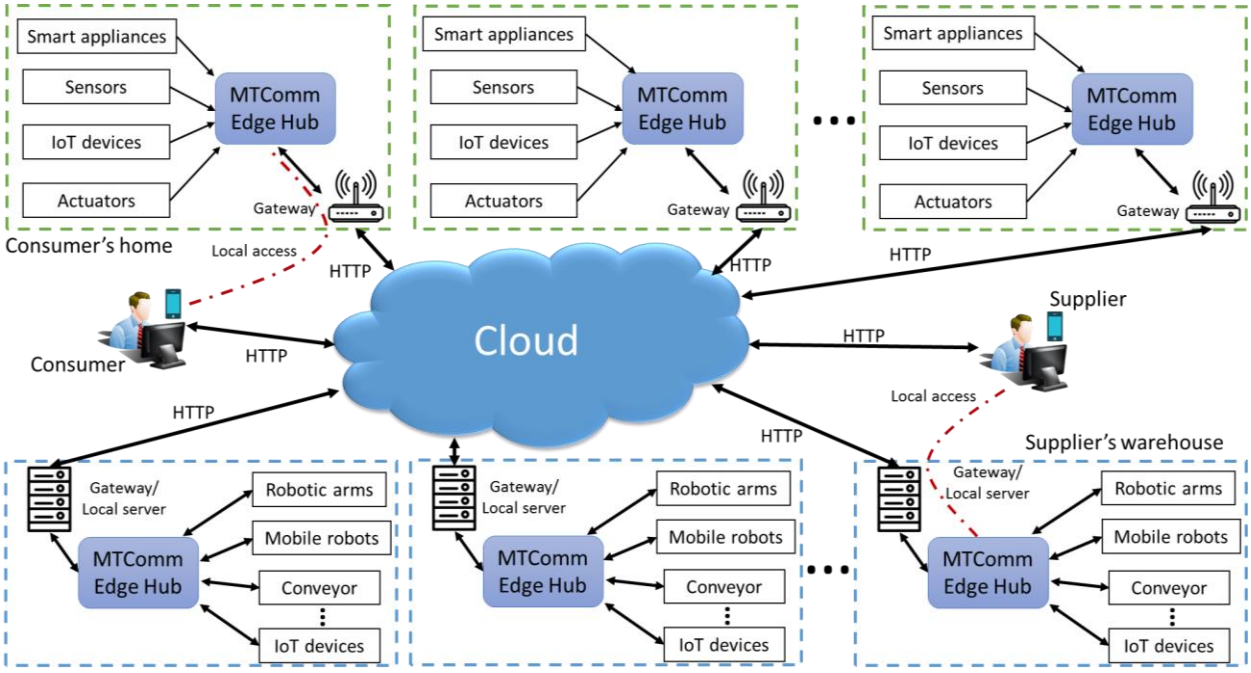


Figure 34. Conceptual framework of IGSDC

excellent feasibility and effectiveness of MTComm based edge computing to integrate heterogeneous resources from different environments with cloud services in order to facilitate cross-domain collaboration and provide unique on-demand grocery product shopping and delivery services for both consumers and suppliers.

7.1 Architecture and components of IGSDC

Figure 34 illustrates a conceptual framework of the IGSDC system. IGSDC is designed in such a way that both consumers and suppliers interact with the system in similar fashion to enhance scalability and robustness of the system. Users (consumer and supplier) interact with the cloud via mobile or web browser applications using HTTP. All physical resources, including in-home IoT appliances devices and in-warehouse machines, are connected with cloud services using MTComm edge hubs. An MEH is an embedded system, like Raspberry Pi, ASIC, FPGA etc., with sufficient computational and network capabilities to establish bi-way communication

between different types of physical resources and cloud services using MTComm method. MEHs communicate with the cloud using HTTP via gateways. Users can also interact with an MEH locally (shown by red dotted lines in Figure 34) via a smartphone app or on-board display interface (if available). All communications to and from an MEH are encrypted. The presented system follows a four-layered architecture. The resource layer at the bottom contains different types of physical end devices, such as sensors, actuators, IoT devices, smart home appliances, robotic arms, mobile delivery robots, computing resources, conveyors etc. Communication layer establishes communication between resource layer and central cloud layer using MTComm. Central cloud layer refers to the cloud infrastructure that hosts necessary cloud services to connect consumers with suppliers and manage web services of their subscribed physical resources. It includes subscription services (SS) for managing user and MEH subscriptions, virtualization service manager which use MTComm `probe` service to acquire structural and configuration data of MEHs and create virtual copies, security manager for ensuring security and privacy, monitoring center (MC) which uses MTComm `current` service to acquire most recent status of resources, remote operation center (ROC) for executing operations of resources remotely through MTComm `operate` service, service broker for running recommendation algorithms to match potential customers and service providers, service repository, and data storage. At the top resides application layer that manages multiplatform applications for users to interact with the IGSDC cloud.

7.2 Utilization of MTComm in edge computing

7.2.1 MTComm Edge Hub (MEH)

MTComm Edge Hub is an embedded device positioned at the edge of an environment that uses MTComm method to integrate web and cloud services with physical resources. This

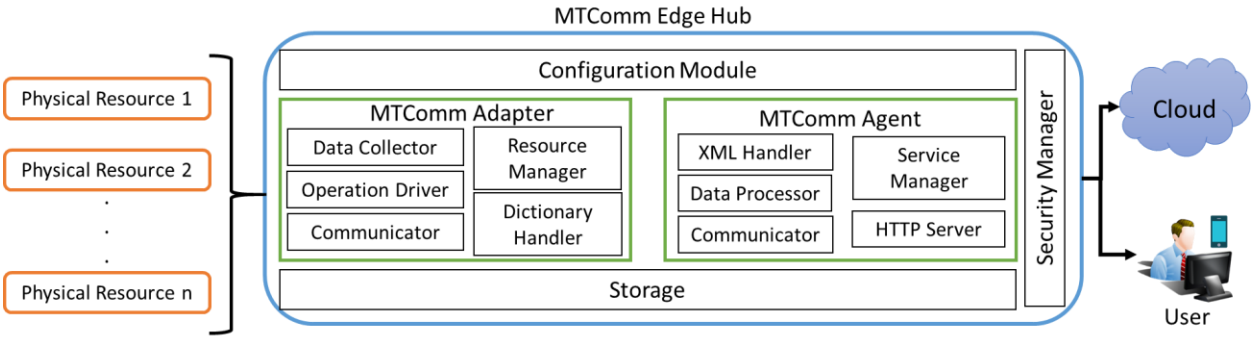


Figure 35. Components of MTComm Edge Hub (MEH)

MEH is slightly different than the MEM described in Chapter 5, as MEM is primarily build for industrial machining tools, while MEH is rather generalized to be used in different domains. As depicted in Figure 35, an MEH primarily consists of MTComm Adapter and MTComm Agent. An adapter module in MEH has five components – 'Data Collector' that acquires data from resources periodically, 'Operation Driver' performing requested operations by sending commands or executing a program, 'Resource Manager' responsible for administering connected resources, 'Dictionary Handler' which handles both incoming (operational information) and outgoing (collected data) key-value pair based dictionary, and 'Communicator' that establishes communication with resources and the Agent module's 'Communicator'. The Agent module contains an 'XML Handler' to generate, translate, and parse XML messages. 'Data processor' is used to perform computation and analysis with received information (both monitoring and operational) including identifying critical events, generating alerts, verifying incoming operation requests etc. The type of information processing depends largely on hardware constraints and capabilities as well as developer's intent. The Agent module hosts an 'HTTP Server' for communication over the Internet with a 'Service Manager' that manages MTComm RESTful web services and maps them with corresponding physical resources. An MEH also contains a 'Security Manager' module for authentication and encryption, a 'Configuration module' through

which users can add, remove, and configure their resources, and a ‘Storage’ unit that is shared by all other components. An MEH includes multiple interface modules, such as USB and Serial ports, Ethernet, Bluetooth, Wi-Fi etc., and support for different communication protocols to connect with heterogeneous physical resources. As these protocols vary from one environment to another, MEHs are required to be custom-made for different environments. However, majority of such modifications are adapter-related, as working mechanisms of the Agent is same for all MEHs.

7.2.2 Adopting MTComm for IoT resources

The primary goal of MTComm ontology was to represent heterogeneous machine tools and associated data in a generic model easily understandable by cloud and web services. This is also highly desirable in IoT domain. Therefore, we developed a mechanism to incorporate this ontology with IoT resources. Manufacturing machine tools usually have complex organizations with multiple components generating large amount of data. But typical IoT devices used in a home environment have simpler data structure with fewer or even no components. Many IoT sensors generate only a single type of data values. So, presenting a single IoT device using MTComm ontology is not very efficient. Therefore, we considered multiple IoT resources together as an MTComm *device* based on their functionality, location, or user choice. Each individual IoT resource is a *component* of the high-level *device* entity. If an IoT device consists of multiple elements, those are considered as lower level *components*. *Dataitems* refer to sensor values and *operations* refer to actuation activities or tasks. Characteristic information or metadata are described by attributes. A home may have one or more *device*. After a user confirms the organization of a *device*, the agent creates a `probe` document accordingly. Any IoT resource can be added to or removed from an existing *device* as a *component*. A `current`

message includes most recent values of all *dataitems* of a *device*. Data streams are timestamped and sequenced. MTComm allows to query individual *dataitems*. To explain the mechanism more, let us consider a user whose home has one IoT enabled smart refrigerator (Rf), one IoT light bulb (Lb), and one IoT humidity sensor (Hmd). Rf consists of two parts – freezer and refrigerator, each with a temperature sensor and a temperature changing activity. Rf also shows overall operation mode. Lb provides current status and performs switching action upon user input. Hmd gives current humidity level of the room. The user can create a single MTComm *device* for his entire home, as shown in Figure 36. As MTComm was not designed for IoT resources, modifications to its XML schemas is necessary to incorporate IoT data information. However, structure or composition of the schemas are not changed, more data types and related information are added to existing MTComm schemas.

The number of physical resources in the bottom layer and their associated web services

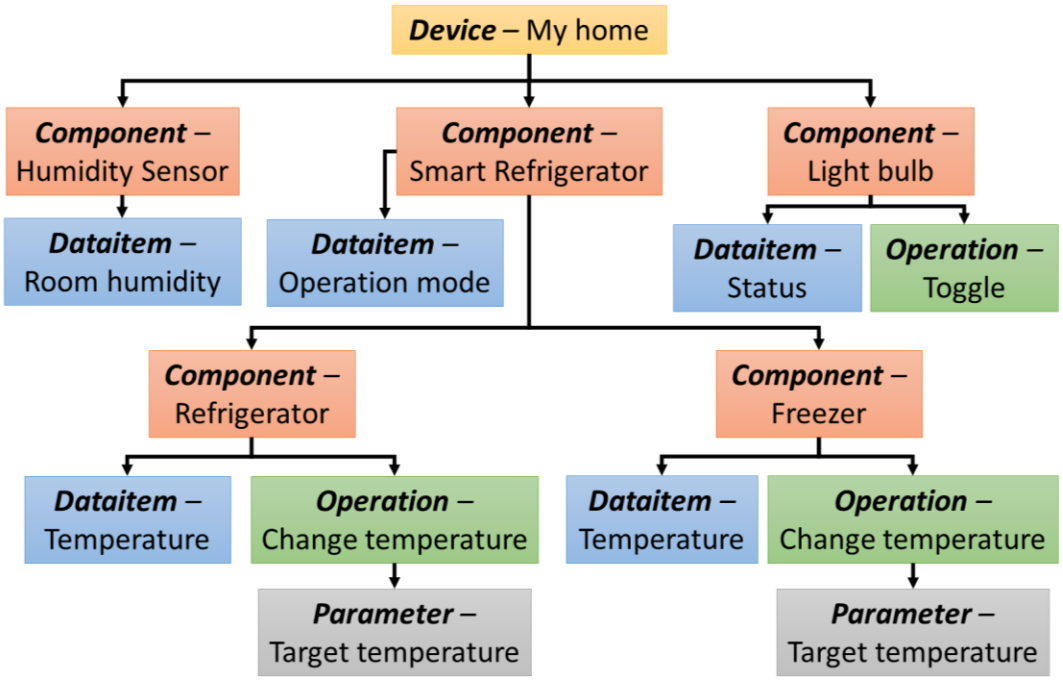


Figure 36. Example of an MTComm *device* hierarchy for IoT

can be very large. Therefore, the naming scheme in edge computing is very important for proper addressing, resource identification, scalability, service management, and communication. MEH employs a two-step naming scheme to address this issue. 'Resource Manager' in MEH Adapter assigns unique (locally) identifier and network address for all connected resources. These information alongside other resource metadata (if any) are stored in a relational table in the MEH storage. The 'Service Manager' of MEH agent assigns user and service friendly unique (locally) name based on the MTComm onotlogy using dot convention and maps them with corresponding RESTful services. For instance, considering the aforementioned example of IoT home, the *dataitem* of Lb is addressed as "`my_home.light_bulb.status`". This naming mechanism makes service and resource management easier for both users and cloud services. Each MEH is given a unique (globally) identifier by the cloud after registration, so each name assigned by an MEH becomes unique globally as well (e.g. "`meh01b_my_home.light_bulb.status`").

Naming of the RESTful web services with MTComm is also efficient. The format of the web service URLs is simple – '`address_of_MEH/`*`device`*`_id/service_name`'. For example, '`http://10.0.5.5:7777/my_home/probe`' is the `probe` service URL of the example home *Device*. MTComm also allows use of attributes as path parameters in service URL to enable acquiring data or perform operation of a particular *Component*. For instance, the service URL to collect the value of Hmd only is –

```
http://10.0.5.5:7777/my_home/current?path=//humidity_sensor
```

Timestamp or sequence is used to collect data of a specific moment or time interval –

```
http://10.0.5.5:7777/my_home/sample?path=//smart_refrigerator//f
reezer//Dataitem[@id="freezer_temp"]&from=50&count=100
```

This service request collects freezer temperature with sequence number 50 to 150. Similarly, service URL to toggle the light bulb is –

```
http://10.0.5.5:7777/my_home/operate?path=//bulb//Operation[@id=
"toggle"]
```

Any attributes of an XML element can be used as path parameters. This feature allows to create service URLs using the ontology based on user requests. Therefore, the cloud only requires storing MEH identifier and web address information in the SR and can dynamically generate necessary service URLs, instead of storing individual service URLs for all available services.

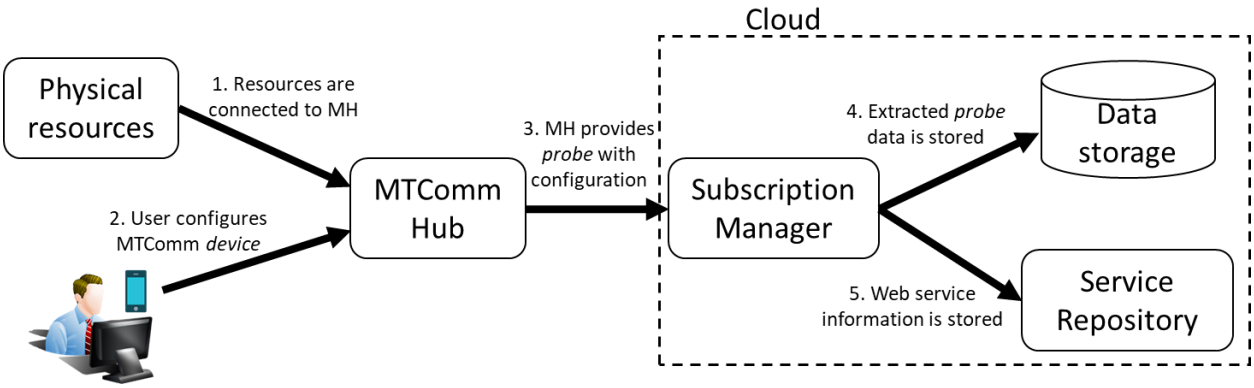
Adopting MTComm method and its ontology for in-home IoT resources has several advantages. Firstly, this represents heterogeneous IoT devices in a common hierarchical structure using XML messages. It facilitates exchange of information between disparate pieces of IoT resources using different languages or data formats and cloud services over the Internet. Using *probe*, cloud services can easily recognize data outputs, configuration, and capabilities of IoT devices in a home environment. As all MEH communicates with the cloud in same manner using XML, there is no need to modify cloud services and applications each time a resource is added or removed. Therefore, MTComm enhances scalability and efficiency for IoT enabled clouds. XML is a rich data format with the flexibility to add multiple metadata as attributes if necessary. MTComm XML messages enables efficient data query, extraction, and ingestion process. Lastly, being an industry compliant communication method, MTComm also supports the possibility to integrate home IoT services with other industry based value-added cloud services.

7.3 Service execution processes in IGSDC using MTComm services

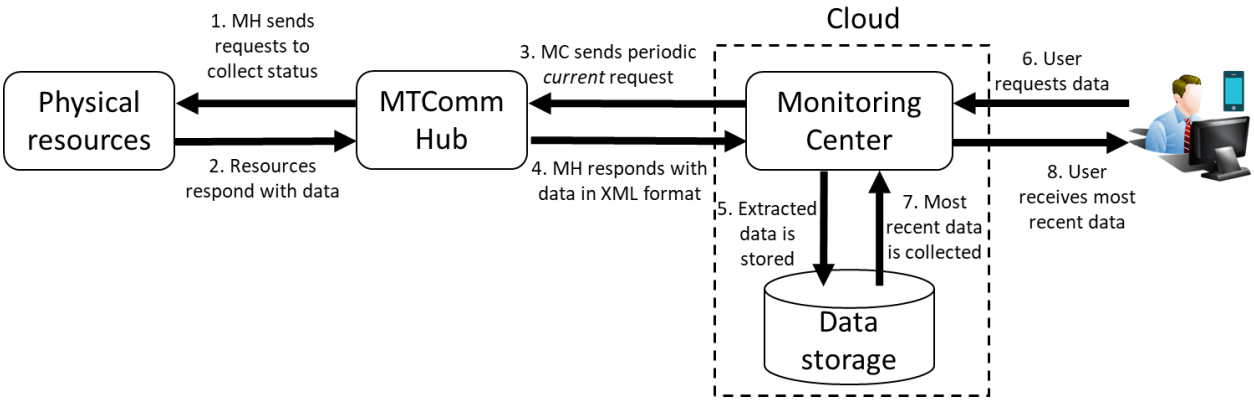
The method of providing IoT data driven grocery shopping and autonomous home delivery in IGSDC system consists of four primary processes – subscription and publication, monitoring, service matching, and operation execution. This section describes these processes briefly.

The first step for both consumers and suppliers of IGSDC is to subscribe as a user and create a user profile. Once done, a user connects his/her physical resources and configure corresponding MTComm *device(s)*. The MEH creates the `probe` XML document with detailed organization data. Then the user subscribes his MEH to the cloud providing its web address. Then the SM assigns a unique id for MEH, requests its `probe` service, parses the response XML, creates a virtual copy of the MEH and its resources using extracted data, and stores it in data storage. It also stores MEH's service information as a tuple in SR. For consumers, this tuple only contains user id, MEH's unique id, and its web address, while for suppliers it contains additional information like what type of product they are selling, how much products are available, in which areas they can deliver etc. Figure 37(a) shows the flow of service subscription and publication.

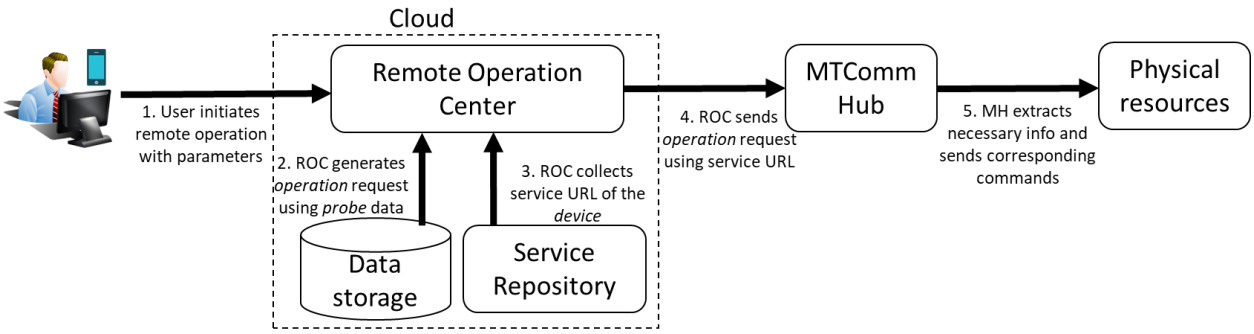
The monitoring process (Figure 37(b)) involves two type of interactions – resource-MEH-cloud interaction and user-cloud interaction. An MEH collects status data from its connected resources and stores them in MTComm XML format at regular intervals. The MC in the cloud sends periodic `current` or `sample` requests to MEH, receives XML responses, extracts data, and stores them in data storage. When a user requests to view the status of his resources, the MC responds with the most recent data available in storage. If a user sets any alert for a specific event, e.g. low supply of grocery, completion of an operation etc., the MC notifies user when conditions



(a) Subscription and Publication



(b) Monitoring



(c) Operation execution

Figure 37. Different processes in IGSDC using MTComm

of the alert are fulfilled.

The service matching process only involves service broker and service repository modules in the central cloud layer and is responsible for finding suitable delivery services for client's needs. When low supply is identified, the broker searches for potential supplier services using recommendation algorithms and provides consumer with resulting service options. Criteria like type, amount, service location, supplier's reputation etc. are used to match consumers with best possible suppliers.

For executing an operation of a physical resource, related instructions and parameters are collected from its user by the ROC. Using these information, the ROC generates an *operation* request based on the `probe` data. Then the ROC collects corresponding MEH's address, creates specific `operate` service URL, and sends the request to the MEH. After verification and validation, MEH extracts the instructions and parameters from the message and sends necessary commands to the physical resource to execute the operation. The process is illustrated in Figure 37(c).

Apart from the service matching, other three processes are identical for both consumers and suppliers. The amount of data may vary, but these processes are irrespective of user type, physical resource type, and operation type. Adding or removing a user or resource is simple and require little to no modification of cloud services. All services are capable of handling heterogeneous users, requests, and physical resources. This way IGSDC ascertains better scalability, efficiency, performance, and user experience.

7.4 Implementation of IGSDC prototype and evaluation

As a proof-of-concept, we implemented a fully functional prototype of the IGSDC system which is shown in Figure 38. We developed a user application for an android smartphone to interact with the cloud. This demo app had the capability to view current status of MTComm *devices*, view available product ordering options, and place orders. Raspberry Pi (RPi) devices were used as MEHs. We created two zones in our lab – home zone and warehouse zone, fifteen meters apart. Each zone had one RPi as an MEH running MTComm adapter and agent programs developed in Python. In home zone, we had a GE Profile Series smart refrigerator (Serial – PYE22PMKES) and two Arduino boards, of which one was connected to a weight scale built using a square force-sensitive resistor (FSR) and published its data via MQTT, and the other was connected to a BMP180 barometric pressure sensor and transferred data to its RPi via USB. The refrigerator was not open-sourced and only displayed its sensor values through "GE Kitchen" smartphone app. However, a portion of the refrigerator data was available through IFTTT APIs. Therefore, we used IFTTT webhooks to collect data from the refrigerator. The warehouse zone consisted of two robots – one Uarm robotic arm and one Interbotix Turtlebot 2i mobile robot running Robot Operating System (ROS). The Turtlebot was capable of autonomous navigation, obstacle avoidance, path planning, and object manipulation with a Pincher MK3 robotic arm. The warehouse RPi was connected with Uarm using USB connection and with Turtlebot over Wi-Fi. The cloud services were deployed and hosted in a virtual machine on the university network.

Experiments were conducted emulating a hypothetical grocery shopping scenario using this prototype system. The refrigerator and two sensors were considered as components of a single MTComm d*evice*. The available *dataitems* were refrigerator operation mode, refrigerator

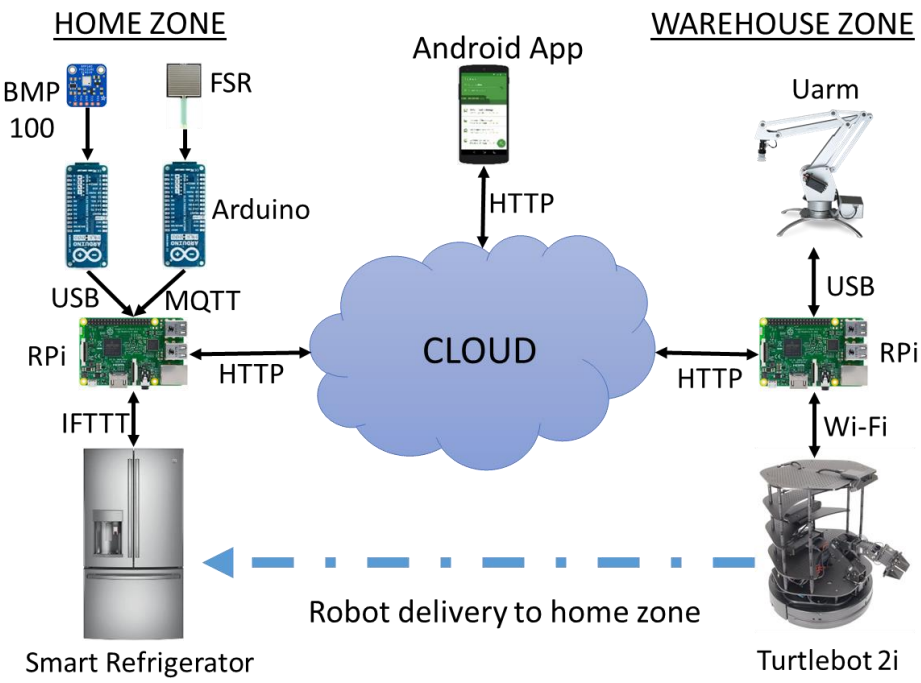


Figure 38. IGSDC prototype system

and freezer door status (open or close), temperature of refrigerator and freezer, barometric pressure, and value of the weight sensor. The home RPi collected data from these sources at 10 Hz and created corresponding XML messages. A full half-gallon milk carton was placed on the weight scale at the beginning. The logic was set so that when the weight was less than 25 percent of its initial weight, it would be considered as low supply. Then the carton was nearly emptied and placed on scale again. The cloud detected this incident and notified the user through android app. User could view available delivery services in the "Place order" page of the app. Once an order was placed, the cloud generated an `operate` request containing instructions for both robots to their RPi. First, the Uarm executed a pre-programmed co-ordinate file in .csv format to move from idle position to product location, grab the product, and then carry it to the Turtlebot. Once done, it forwarded the `operate` request to Turtlebot's agent which extracted destination co-ordinates (position of the refrigerator) from the message and sent them with necessary

commands to the robot. The Turtlebot was pre-trained to map the whole room. Based on the destination co-ordinate, it then travelled from the warehouse zone to the home zone navigating autonomously. The cloud was constantly collecting status of the robot. When it reached the refrigerator, the cloud detected the event through co-ordinate matching and sent a notification to user app. Using its robotic arm, the Turtlebot dropped the product in front of the refrigerator and then travelled back to its original position in the warehouse zone. Several test runs were performed and the whole process was successfully completed each time. Other than the interactions with the android app, no human intervention was needed at any time during tests.

To evaluate the effectiveness of MEH, WireShark software was used to monitor the data streams between MEHs and the cloud services. The performance was measured based on delta-time – the time interval between cloud sending a request and receiving resource data, and packet size – the total size of data packets received by the cloud. For comparison purpose, a separate setup was prepared where the cloud collected data from the home zone devices by querying them individually without using MEH. Table X shows the comparison between total average (of ten datasets) delta-time and packet size of both systems respectively. In the setup using direct communication, each resource sent its data separately while MTComm aggregated all *Dataitems* in a single data stream. Therefore, total delta-time and packet size were much smaller for MTComm based IGSDC system. As the number of IoT devices increases, the second setup had

TABLE X. Comparison of delta-time and packet size

Criteria	**System using direct communication**	**System using MTComm**	**Reduction (%)**
Average Delta-time (ms)	27.1817	9.9863	**63.2609**
Average packet size (byte)	4387.2	1952.6	**55.4933**

greater delay and bigger packet size, while MTComm based system experienced much smaller increase rate; hence improving overall latency, bandwidth usage, and system performance.

7.5 References

Cho, S., Lee, D., Jung, Y., Lee, U., & Shim, D. H. (2014). Development of a cooperative heterogeneous unmanned system for delivery services. *Journal of Institute of Control, Robotics and Systems*, *20*(12), 1181-1188.

Desai, H., Guruvayurappan, D., Merchant, M., Somaiya, S., & Mundra, H. (2017, February). IoT based grocery monitoring system. In *2017 Fourteenth International Conference on Wireless and Optical Communications Networks (WOCN)* (pp. 1-4). IEEE.

Fagerstrøm, A., Eriksson, N., & Sigurðsson, V. (2017). What's the "Thing" in Internet of Things in Grocery Shopping? A Customer Approach. *Procedia computer science*, *121*, 384-388.

Sill, A. (2017). Standards at the edge of the cloud. *IEEE Cloud Computing*, *4*(2), 63-67.

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, *3*(5), 637-646.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2018). Communication method for manufacturing services in a cyber–physical manufacturing cloud. *International Journal of Computer Integrated Manufacturing*, *31*(7), 636-652.

Sunny, S. M. N. A., Liu, X., & Shahriar, M. R. (2019, July). An Integrated IoT Enabled On-Demand Grocery Shopping and Delivery Cloud System Using MTComm at the Edge. In *2019 IEEE International Conference on Edge Computing (EDGE)* (pp. 51-55). IEEE.

CHAPTER 8

CONCLUSIONS

8.1 Summary

The historically conservative world of manufacturing is changing rapidly due to the emergence of disruptive technologies like CPS, CMfg, IoT etc. These technologies are often intertwined to improve overall efficiency and reduce production cost by streamlining factory floor data and automating manufacturing processes. The integration of two emerging manufacturing paradigms - cyber-physical systems and cloud manufacturing has the promise of transforming existing manufacturing systems. Existing communication methods do not support direct operations of manufacturing machine tools over the Internet, which is a major hindrance for realization of cloud-based manufacturing CPSs. MTComm, the Internet-scale communication method presented in this dissertation, can play a key role in addressing this issue with its scalable service-oriented remote monitoring and direct operation capabilities of heterogeneous machine tools over the Internet. It allows manufacturers to acquire status data and perform machining operations of different types of machines situated in geographically different locations across the world remotely from web and cloud based manufacturing applications through RESTful web services. Because of its agent-adapter based architecture and flexible interfacing options, MTComm can be used with modern network enabled machine tools as well as legacy manufacturing machines without existing network functionality. Heterogeneous manu facturing resources can communicate with each other and participate in collaborative manufacturing through MTComm services. Although designed and developed to be used for integrating CPS and CMfg systems, MTComm can also be used with connect any manufacturing resources to the Internet, transforming a legacy system into an Internet-enabled CPS. As MTComm is a service-

oriented method, additional services can be easily added if required. Adopting optimization strategies has greatly boosted MTComm's performance and efficiency. Thus, MTComm boosts machine interoperability and factory floor automation, and reduces human involvement in production processes. The edge middleware can become a potential game-changing platform for future manufacturing by offloading iterative computational tasks to network edges and reducing burdens of central cloud servers, hence can enable plug-n-play capability and enhance overall system scalability. Through its diverse applications in a CPMC testbed including remote and collaborating manufacturing and active testing based fault-diagnosis, MTComm has exhibited its potential and feasibility to be used in manufacturing CPSs and CMfg Utilization of MTComm to develop an IoT-enable grocery shopping and delivery cloud has proved its promise to deliver cross-domain interoperability. Successful experiments with the testbed further supported MTComm's potential for improving scalability, robustness, productivity, and efficiency of today's manufacturing systems.

8.2 Contributions

The above summary presents an overview of the research described in this dissertation. This particular section specifies the scientific contributions that are direct results of my efforts.

- I designed and developed MTComm method including its agent-adapter architecture, specifications, sematic ontology, RESTful services and methodology (Sunny, Liu & Shahriar, 2017). Initially I laid the groundwork by investigating the requirements of a cyber-physical manufacturing systems and existing communication mechanisms, specially MTConnect, which led to my initial adaptation of MTConnect for open-source 3D printers (Liu et al., 2016) and utilization of TCP commands alongside MTConnect to achieve both monitoring and operational capabilities of heterogeneous machine tools over

the Internet (Liu et al., 2017; Sunny, Liu & Shahriar, 2018). This naïve approach helped me realizing the shortcomings of the-then developed CPMC, as well as gaining enough insights to expand and modify MTConnect in order to incorporate direct operation of machine tools. To achieve this, I designed the agent-adapter architecture of MTComm in a way that it is compatible with MTConnect (made adapter a compulsory component, also redefined agent and adapter's internal organization), extended MTConnect semantic ontology by adding *operation* and *parameter* elements, added two new RESTful services named `operate` and notification for execution of manufacturing processes, created two new XML schemas – *MTCommOperations* for operate service and *MTCommNotifications* for `notification` service, and modified existing MTConnect schemas for *MTConnectDevices* (used for `probe`) and *MTConnectStreams* (used for `current` and `sample`) XML documents.

- I actively participated in designing the architecture of the CPMC and implementing the CMPC testbed described in Chapter 4 using MTComm. Particularly I was responsible for establishing the bi-directional communication between physical machine tools and the cloud. I solely developed MTComm adapters and agent programs for RepRap 3D printers, X-carve CNC drilling machine, and Uarm robotic arms in Python. I deployed these programs in Raspberry Pis and configured them appropriately so that they were able to interact with corresponding machine tools. I also conducted rigorous experiments with the testbed to evaluate and finetune MTComm method. Lastly, I developed and implemented a peer-to-peer collaborative manufacturing process using MTComm and tested its feasibility and performance with existing CPMC testbed.

- Investigating the results from previous experiments with MTComm and CPMC, I identified several scopes of improving MTComm's performance including communication latency and storage requirements. To achieve that, I infused edge computing methodologies with MTComm and created the concept of MTComm edge middleware. I designed three optimization strategies for MTComm, two of which (data caching and transmission optimization) can be adopted for MTConnect based systems as well. I also developed an SoC FPGA based hardware prototype using a Xilinx Zybo-Z7 development board with dual Cortex A9 ARM processors to reduce computation time even further. The experimental results showed considerable decrease in communication latency, local storage size, and bandwidth usage.
- To develop additional applications in CPMC, I explored several different features of existing manufacturing CPSs and identified fault diagnosis as a promising paradigm. Utilizing the remote operation capability of MTComm, I introduced the concept of active testing-based fault diagnosis and maintenance of manufacturing machine tools. I designed and implemented a fault diagnosis center in the CPMC. To improve the efficiency of the fault detection process, I revisited MTComm design and defined two running modes – NORMAL and DIAGNOSIS, the later specifically designed for fault diagnosis purposes. I also designed an anomaly detection process using MTcomm and INSTRUCTION type data. After developing the MEM and optimization strategies, I investigated several light-weight anomaly detection algorithms and evaluated their performance for edge-based fault diagnosis. I conducted several experiments in the CPMC testbed by injecting manual faults and the results were promising enough to suggest MTComm's strong feasibility for fault diagnosis in cyber manufacturing systems.

- To realize MTComm's potential for achieving cross-domain interoperability, I designed, developed, and implemented a novel IoT enabled cloud system for on-demand grocery shopping and autonomous delivery. Firstly, I modified exiting MTComm methodologies and schemas and implemented MTComm adapter programs to support acquiring data from heterogeneous IoT devices used in consumer homes. Secondly, I developed mechanisms to connect mobile robots running Robot Operating System (ROS) with cloud services using MTComm. This is a significant development as ROS is widely used in today's industries. Lastly, I designed the cloud architecture, created an Android application capable of interacting with machine tools using MTComm and cloud services, and implemented a testbed system to conduct experiments and provide proof-of-concept.

8.3 Comparison with existing works

To highlight the differences between the work described in this dissertation and other communication approaches adopted by manufacturing researchers in recent years, a high-level comparison is given in Table XI. There has been a significant number of research and development works in both academia and industry regarding CPS and CMfg in recent years. The comparison here focuses on works with similar features as MTComm, specifically those which adopted specific communication approaches, included remote operational capabilities, offered moderate to high scalability, performed computation at the edge layer, and provided measurements of communication latency (as improving latency was a major focus of my work discussed in Chapter 5). It is evident from the table that none of these works provided all features supported by MTComm for heterogeneous machine tools in a scalable cloud-based manufacturing CPS environment. Moreover, the communication latency of the presented method

clearly outperforms the other approaches, making MTComm a suitable choice for manufacturing processes with real-time communication constraints.

TABLE XI. Comparison of presented method with existing literature (shortened words: Comm. = Communication, MTC = MTConnect, Hetero. = Heterogeneous, Mod. = Moderate)

Features	Presented method	Wang, Gao & Ragai (2014)	Lin, Lin & Chiu (2015)	Liu et al. (2017)	Parto (2017)	José Álvares et al. (2018)	Liu et al. (2018)	Okwudire et al. (2018)
Comm. method used	MTComm	TCP	TMTC	MTC + TCP	MTC + MQTT	MTC / OPC	MTC	Low level
Monitoring	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Operation	Yes	Yes	Yes	Yes	No	Yes	No	Yes
Supported machine types	Hetero.	CNC, Robots	CNC	Hetero.	Hetero.	CNC lathe	CNC	3D printers
Optimized for edge	Yes	No	Yes	No	Yes	No	No	Yes
Latency (ms)	5-10	30	120	100-200	4000	500	100	100-250
Scalability	High	Mod.	High	High	Mod.	Mod.	High	Mod.

8.4 References

Hu, L., Nguyen, N. T., Tao, W., Leu, M. C., Liu, X. F., Shahriar, M. R., & Sunny, S. M. N. A. (2018). Modeling of cloud-based digital twins for smart manufacturing with MT connect. *Procedia manufacturing*, *26*, 1193-1203.

José Álvares, A., Oliveira, L. E. S. D., & Ferreira, J. C. E. (2018). Development of a Cyber-Physical framework for monitoring and teleoperation of a CNC lathe based on MTconnect and OPC protocols. *International Journal of Computer Integrated Manufacturing*, *31*(11), 1049-1066.

Lin, Y. L., Lin, C. C., & Chiu, H. S. (2015, March). The development of intelligent service system for machine tool industry. In *2015 1st International Conference on Industrial Networks and Intelligent Systems (Iniscom)* (pp. 100-106). IEEE.

Liu, C., Vengayil, H., Zhong, R. Y., & Xu, X. (2018). A systematic development method for cyber-physical machine tools. *Journal of manufacturing systems*, *48*, 13-24.

Liu, X. F., Shahriar, M. R., Al Sunny, S. N., Leu, M. C., & Hu, L. (2017). Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed. *Journal of Manufacturing Systems*, *43*, 352-364.

Liu, X. F., Sunny, S. M. N. A., Shahriar, M. R., Leu, M. C., Cheng, M., & Hu, L. (2016, January). Implementation of MTConnect for open source 3D printers in cyber physical manufacturing cloud. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection.

Okwudire, C. E., Huggi, S., Supe, S., Huang, C., & Zeng, B. (2018). Low-level control of 3d printers from the cloud: A step toward 3d printer control as a service. *Inventions*, *3*(3), 56.

Parto Dezfouli, M. (2017). *A secure MTConnect compatible IoT platform for machine monitoring through integration of fog computing, cloud computing, and communication protocols* (Doctoral dissertation, Georgia Institute of Technology).

Shahriar, M. R., Sunny, S. M. N. A., Liu, X., Leu, M. C., Hu, L., & Nguyen, N. T. (2018, June). MTComm based virtualization and integration of physical machine operations with digital-twins in cyber-physical manufacturing cloud. In *2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (pp. 46-51). IEEE.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2017, June). Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud. In *2017 IEEE International Congress on Internet of Things (ICIOT)* (pp. 121-128). IEEE.

Sunny, S. M. N. A., Liu, X. F., & Shahriar, M. R. (2018). Communication method for manufacturing services in a cyber–physical manufacturing cloud. *International Journal of Computer Integrated Manufacturing*, *31*(7), 636-652.

Wang, L., Gao, R., & Ragai, I. (2014, June). An integrated cyber-physical system for cloud manufacturing. In *ASME 2014 International Manufacturing Science and Engineering Conference collocated with the JSME 2014 International Conference on Materials and Processing and the 42nd North American Manufacturing Research Conference*. American Society of Mechanical Engineers Digital Collection.

APPENDICES

A. Example of response XML message of a `Probe` request from a 3D printer's MTComm agent

```
<?xml version="1.0" encoding="UTF-8"?>
<MTCommDevices xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="../Schemas/MTCommDevices_0.2.xsd"
>
    <Header bufferSize="10" instanceId="1" creationTime="2016-
06-14T12:00:00" sender="Ultimaker2" version="1.2"
               assetCount="0" assetBufferSize="10"/>
    <Devices>
        <Device id="Ultimaker2" uuid="P2673" name="Ultimaker2 3D
Printer" type="3D printer" buildType="Square">
            <DataItems>
                <DataItem category="EVENT" id="avail"
name="availablility" type="AVAILABILITY"/>
            </DataItems>
            <Operations>
                <Operation id="startPrintingJob" category="JOB"
name="Start new job" type="PRINT">
                    <Parameters>
                        <Parameter id="material" name="Material
type" supportedMaterial="PLA" type="MATERIAL"/>
                        <Parameter id="targetExtTemp"
name="Target Extruder Temp" units="CELSIUS" type="TEMPERATURE">
                                    <Constraints>
                                        <Minimum>0</Minimum>
                                        <Maximum>215</Maximum>
                                    </Constraints>
                                </Parameter>
```

```
                        <Parameter id="targetBedTemp"
name="Target Bed Temp" units="CELSIUS" type="TEMPERATURE">
                                    <Constraints>
                                        <Minimum>0</Minimum>
                                        <Maximum>65</Maximum>
                                    </Constraints>
                                </Parameter>
                        <Parameter id="objectName"
name="Printing object name"/>
                    </Parameters>
                </Operation>
                <Operation id="stopJob" category="JOB"
name="Stop current job" type="PRINT" />
            </Operations>
            <Components>
                <Axes id="axes" name="Axes">
                    <Components>
                        <Linear id="x" name="X">
                            <DataItems>
                                <DataItem category="SAMPLE"
coordinateSystem="MACHINE" id="xPos" name="Actual X Position"
subType="ACTUAL" type="POSITION"/>
                            </DataItems>
                            <Operations>
                                <Operation id="moveX"
category="ACTION" coordinateSystem="MACHINE" name="Move X Axis"
type="POSITION">
                                    <Parameters>
                                        <Parameter
id="targetPosition" name="Target Axis position"
units="MILLIMETER">

    <Constraints>
```

```
<Minimum>0</Minimum>

<Maximum>200</Maximum>

</Constraints>
                                                    </Parameter>
                                                </Parameters>
                                            </Operation>
                                        </Operations>
                                    </Linear>
                                    <Linear id="y" name="Y">
                                        <DataItems>
                                            <DataItem category="SAMPLE"
coordinateSystem="MACHINE" id="yPos" name="Actual Y Position"
subType="ACTUAL" type="POSITION"/>
                                        </DataItems>
                                        <Operations>
                                            <Operation id="moveY"
category="ACTION" coordinateSystem="MACHINE" name="Move Y Axis"
type="POSITION">
                                                <Parameters>
                                                    <Parameter
id="targetPosition" name="Target Axis position"
units="MILLIMETER">

<Constraints>

<Minimum>0</Minimum>

<Maximum>200</Maximum>

</Constraints>
                                                    </Parameter>
                                                </Parameters>
```

```
                                        </Operation>
                                    </Operations>
                                </Linear>
                                  <Linear id="z" name="Z">
                                    <DataItems>
                                        <DataItem category="SAMPLE"
coordinateSystem="MACHINE" id="zPos" name="Actual Z Position"
subType="ACTUAL" type="POSITION"/>
                                    </DataItems>
                                    <Operations>
                                        <Operation id="moveZ"
category="ACTION" coordinateSystem="MACHINE" name="Move Z Axis"
type="POSITION">
                                                <Parameters>
                                                    <Parameter
id="targetPosition" name="Target Axis position"
units="MILLIMETER">
     <Constraints>
     <Minimum>-220</Minimum>
     <Maximum>0</Maximum>
     </Constraints>
                                                    </Parameter>
                                                </Parameters>
                                        </Operation>
                                    </Operations>
                                </Linear>
                            </Components>
                        </Axes>
                        <Sensor id="extruder" name="Extruder">
                            <DataItems>
```

```
                        <DataItem category="SAMPLE"
id="extruderTemp" name="Extruder Temp Sensor"
type="TEMPERATURE"/>

                        <DataItem category="EVENT"
id="extruderReady" name="Extruder Ready State"
type="EXECUTION"/>

                    </DataItems>

                    <Operations>

                        <Operation id="changeExtruderTemp"
category="ACTION" name="Change Extruder Temperature"
type="TEMPERATURE">

                                    <Parameters>

                                        <Parameter
id="targetExtTemp" name="Target Extruder Temp" units="CELSIUS"
type="TEMPERATURE">

                                            <Constraints>


    <Minimum>0</Minimum>


    <Maximum>215</Maximum>

                                            </Constraints>

                                        </Parameter>

                                    </Parameters>

                                </Operation>

                    </Operations>

                </Sensor>

                <Sensor id="bed" name="Bed">

                    <DataItems>

                        <DataItem category="SAMPLE" id="bedTemp"
name="Bed Temp Sensor" type="TEMPERATURE"/>

                        <DataItem category="EVENT" id="bedReady"
name="Bed Ready State" type="EXECUTION"/>

                    </DataItems>

                    <Operations>
```

```
                        <Operation id="changeBedTemp"
category="ACTION" name="Change Bed Temperature"
type="TEMPERATURE">
                                    <Parameters>
                                        <Parameter
id="targetBedTemp" name="Target Bed Temp" units="CELSIUS"
type="TEMPERATURE">
                                            <Constraints>

     <Minimum>0</Minimum>

     <Maximum>65</Maximum>
                                            </Constraints>
                                        </Parameter>
                                    </Parameters>
                                </Operation>
                    </Operations>
                </Sensor>
                <Controller id="controller" name="Controller">
                                    <DataItems>
                                        <DataItem
type="EXECUTION" id="buildProgress" category="EVENT" name="Build
Progress State"/>
                                        <DataItem
category="EVENT" id="turnOffStatus" name="Turn machine off
Status" type="POWER"/>
                                    </DataItems>
                                    <Operations>
                                        <Operation id="turnOff"
category="ACTION" name="Turn machine off" type="POWER" />
                                    </Operations>
                                </Controller>
                    <Systems id="systems" name="systems">
                         <Components>
```

```
                                <Electric id="el" name="electric">
                                    <DataItems>
                                        <DataItem
category="EVENT" id="p2" name="power" type="POWER_STATE"/>
                                    </DataItems>
                                </Electric>
                            </Components>
                        </Systems>
                </Components>
            </Device>
        </Devices>
</MTCommDevices>
```

B. Example of response XML message of a `Current` request from a 3D printer's MTComm agent during a printing JOB

```
<MTCommStreams xmlns="urn:MTComm.org:MTCommStreams:1.2"
xmlns:m="urn:MTComm.org:MTCommStreams:1.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:MTComm.org:MTCommStreams:1.2
http://www.MTComm.org/schemas/MTCommStreams_1.2.xsd">
  <Header bufferSize="10" creationTime="2017-02-09T01:21:46"
firstSequence="57" instanceId="1" lastSequence="66"
nextSequence="67" sender="Ultimaker2" version="1.2"/>
     <Streams>
       <DeviceStream name="Ultimaker2" uuid="P2673">
          <ComponentStream component="Device"
componentId="Ultimaker" name="Ultimaker2">
            <Events>
               <Availability dataItemId="availability"
name="availablility" sequence="66" timestamp="2017-02-
09T01:21:46">BUSY</Availability>
            </Events>
            <Actions>
```

```
            <Printing operationId="startJobStatus"
name="Start new job Status" sequence="66" timestamp="2017-02-
09T01:21:46">ONGOING</Printing>

            <Printing operationId="stopJobStatus" name="Stop
current job Status" sequence="66" timestamp="2017-02-
09T01:21:46">AVAILABLE</Printing>

          </Actions>

        </ComponentStream>

        <ComponentStream component="Linear" componentId="x"
name="X">

          <Samples>

            <Position dataItemId="xPos" name="Actual X
Position" sequence="66" subType="ACTUAL" timestamp="2017-02-
09T01:21:46">105.3</Position>

          </Samples>

          <Actions>

            <Position operationId="moveXStatus" name="Move X
Axis Status" sequence="66" timestamp="2017-02-
09T01:21:46">UNAVAILABLE</Position>

          </Actions>

        </ComponentStream>

        <ComponentStream component="Linear" componentId="y"
name="Y">

          <Samples>

            <Position dataItemId="yPos" name="Actual Y
Position" sequence="66" subType="ACTUAL" timestamp="2017-02-
09T01:21:46">-95.6</Position>

          </Samples>

          <Actions>

            <Position operationId="moveYStatus" name="Move Y
Axis Status" sequence="66" timestamp="2017-02-
09T01:21:46">UNAVAILABLE</Position>

          </Actions>

        </ComponentStream>
```

```
          <ComponentStream component="Linear" componentId="z"
name="Z">
            <Samples>
               <Position dataItemId="zPos" name="Actual Z
Position" sequence="66" subType="ACTUAL" timestamp="2017-02-
09T01:21:46">-216.9</Position>
            </Samples>
            <Actions>
               <Position operationId="moveZStatus" name="Move Z
Axis Status" sequence="66" timestamp="2017-02-
09T01:21:46">UNAVAILABLE</Position>
            </Actions>
          </ComponentStream>
          <ComponentStream component="Sensor"
componentId="extruder" name="Extruder">
            <Samples>
               <Temperature dataItemId="extruderTemp"
name="Extruder Temp Sensor" sequence="66" timestamp="2017-02-
09T01:21:46">209.7</Temperature>
            </Samples>
            <Events>
               <Execution dataItemId="extruderReady"
name="Extruder Ready State" sequence="66" timestamp="2017-02-
09T01:21:46">BUSY</Execution>
            </Events>
            <Actions>
               <Temperature
operationId="changeExtruderTempStatus" name="Change Extruder
Temperature Status" sequence="66" timestamp="2017-02-
09T01:21:46">UNAVAILABLE</Temperature>
            </Actions>
          </ComponentStream>
          <ComponentStream component="Sensor" componentId="bed"
name="Bed">
            <Samples>
```

```
                <Temperature dataItemId="bedTemp" name="Bed Temp
Sensor" sequence="66" timestamp="2017-02-
09T01:21:46">62.4</Temperature>
            </Samples>
            <Events>
               <Execution dataItemId="bedReady" name="Bed Ready
State" sequence="66" timestamp="2017-02-
09T01:21:46">BUSY</Execution>
            </Events>
            <Actions>
               <Temperature operationId="changeBedTempStatus"
name="Change Bed Temperature Status" sequence="66"
timestamp="2017-02-09T01:21:46">UNAVAILABLE</Temperature>
            </Actions>
          </ComponentStream>
          <ComponentStream component="Path"
componentId="motherboardPath" name="Motherboard Path">
            <Samples>
               <Execution dataItemId="buildProgress" name="Build
Progress State" sequence="66" timestamp="2017-02-
09T01:21:46">23.3%</Execution>
            </Samples>
            <Actions>
               <Power operationId="turnOffStatus" name="Turn
Machine Off Status" sequence="66" timestamp="2017-02-
09T01:21:46">AVAILABLE</Power>
            </Actions>
          </ComponentStream>
       </DeviceStream>
     </Streams>
</MTCommStreams>
```

C. Example of XML message of a `Operate` request for starting a 3D printing JOB

```
<?xml version="1.0" encoding="UTF-8"?>

<MTCommOperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="../Schemas/MTCommOperations_0.3.x
sd">

    <Header bufferSize="10" instanceId="1" creationTime="2017-
01-18T12:00:00" sender="Ultimaker2" version="0.1"
firstSequence="9" lastSequence="9"

     nextSequence="10"/>

    <Operations>

     <Device uuid="cxz" name="CoreXZ 3D Printer">

          <DeviceOperation id="cxz" name="CoreXZ 3D Printer">

                <Jobs>

                   <Collaboration operationId="startJob"
name="Start new job" sequence="9" timestamp="2017-01-
18T05:45:40">

                        <Parameters>

                             <Material id="material"
name="Material Type" timestamp="2017-01-
18T05:45:40">PLA</Material>

                                  <Quantity id="quantity"
name="Number of objects" timestamp="2017-01-
18T05:45:40">1</Quantity>

                             <Object id="objName" name="Object
Name" timestamp="2017-01-18T05:45:40">Box</Object>

                        </Parameters>

                   </Collaboration>

                </Jobs>

             </DeviceOperation>
```

```
     </Device>
    </Operations>
</MTCommOperations>
```

D. Example of XML message of a `Operate` request for starting multiple ACTIONs

```
<?xml version="1.0" encoding="UTF-8"?>
<MTCommOperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="../Schemas/MTCommOperations_0.2.x
sd">
    <Header bufferSize="10" instanceId="1" creationTime="2017-
01-18T12:00:00"
          sender="Ultimaker2" version="0.1"/>
    <Operations>
     <Device uuid="cxz" name="Utlimaker 3D Printer">
          <ComponentOperation component="Linear" componentId="x"
name="X">
                    <Actions>
                         <Position operationId="moveX"
name="Move X Axis" sequence="2"
                              units="MILLIMETER"
timestamp="2017-01-18T06:16:39">-50.0</Position>
                    </Actions>
               </ComponentOperation>
               <ComponentOperation component="Linear"
componentId="y" name="Y">
                    <Actions>
                         <Position operationId="moveY"
name="Move Y Axis" sequence="3"
                              units="MILLIMETER"
timestamp="2017-01-18T06:16:39">-50.0</Position>
                    </Actions>
               </ComponentOperation>
```

```
                <ComponentOperation component="Sensor"
componentId="extruder" name="Extruder">
                    <Actions>
                        <Temperature
operationId="changeExtTemp" name="Change Extruder Temperature"
                            sequence="1" units="CELSIUS"
timestamp="2017-01-18T06:16:39">110.0</Temperature>
                    </Actions>
                </ComponentOperation>
    </Device>
    </Operations>
</MTCommOperations>
```

E. Example of XML message of a `Operate` request for starting a collaborative manufacturing Job involving three machines – Ultimaker 2, Uarm, and X-carve

```
<?xml version="1.0" encoding="UTF-8"?>
<MTCommOperations xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:noNamespaceSchemaLocation="../Schemas/MTCommOperations_0.3.x
sd">
    <Header bufferSize="10" instanceId="1" creationTime="2017-
01-18T12:00:00" sender="Ultimaker2" version="0.1"
firstSequence="9" lastSequence="9"
     nextSequence="10"/>
    <Operations>
          <Queue id="1">
               <DeviceOperation uuid="P2673" name="Ultimaker2 3D
Printer">
                    <Jobs>
```

```
                        <Collaboration operationId="startJob"
name="Start new job" sequence="9" timestamp="2017-01-
18T05:45:40">

                            <Parameters>

                                <Material id="material"
name="Material Type" timestamp="2017-01-
18T05:45:40">PLA</Material>

                                <Quantity id="quantity"
name="Number of objects" timestamp="2017-01-
18T05:45:40">1</Quantity>

                                <Object id="objName"
name="Object Name" timestamp="2017-01-18T05:45:40">Clip</Object>

                                <Temperature
id="changeExtruderTemp" name="Change Extruder temperature"
timestamp="2017-01-18T05:45:40">210</Temperature>

                                <Temperature
id="changeHeatbedTemp" name="Change Heatbed temperature"
timestamp="2017-01-18T05:45:40">60</Temperature>

                            </Parameters>

                        </Collaboration>

                    </Jobs>

                </DeviceOperation>

            <Queue>

            <Queue id = "2">

                <DeviceOperation uuid="ra01" name="U-arm">

                    <Jobs>

                        <Collaboration operationId="startJob"
name="Start new job" sequence="9" timestamp="2017-01-
18T05:45:40">

                            <Parameters>

                                <Name id="opName"
name="Operation Name" timestamp="2017-01-
```

```
18T05:45:40">Remove</Name>
                                </Parameters>
                           </Collaboration>
                      </Jobs>
                 </DeviceOperation>
           <Queue>
           <Queue id="3">
                 <DeviceOperation uuid="xc01" name="X-carve">
                      <Jobs>
                           <Collaboration operationId="startJob"
name="Start new job" sequence="9" timestamp="2017-01-
18T05:45:40">
                                <Parameters>
                                     <Material id="material"
name="Material Type" timestamp="2017-01-
18T05:45:40">PLA</Material>
                                     <Quantity id="quantity"
name="Number of objects" timestamp="2017-01-
18T05:45:40">1</Quantity>
                                     <File id="fileName"
name="File Name" timestamp="2017-01-18T05:45:40">Route1</Object>
                                </Parameters>
                           </Collaboration>
                      </Jobs>
                 </DeviceOperation>
           <Queue>
    </Operations>
</MTCommOperations>
```