*Proceedings of the 2018 Winter Simulation Conference*
*M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.*

# ENABLING CONTROL SYSTEM AND CLOUD-BASED SIMULATION SERVICE INTEROPERABILITY

Albert Jones
Guodong Shao
Frank Riddick

Engineering Laboratory
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, MD 20899, USA

## ABSTRACT

The latest innovations in Information and Communication Technologies (ICT) will change manufacturing and simulation forever. One of those changes involves the use of simulation-based cloud services and the integration of those services with operational problems on the shop floor. In this paper, we focus on the control problems, which we represent as a multilayer process graph. The nodes in this graph are control functions, which are cognitive in nature and can be executed by different agents. The links in the graph are the information objects needed by each agent to execute its assigned functions. We also describe a generic sets of control functions and the roles that simulation can play in executing those functions. This means that simulation is now part of an integrated process and can no longer be thought of as a stand-alone technology. In this paper, we discuss issues and propose approaches to addressing that integration.

## 1 INTRODUCTION

It is widely believed that we are in the Fourth Industrial Revolution, sometimes called *Industry 4.0* or *Smart Manufacturing* (Thoben et al. 2017). The goal of "Industry 4.0" is the information-intensive cyber-physical transformation of the manufacturing sector, resulting in a network of data warehouses, people, processes, services, systems, and production assets, which we call agents (Sameer et al. 2017). In this paper, we cluster agents into two groups: cognitive and physical. Each such agent can generate, leverage, and utilize actionable information to make real-time, cooperative decisions.

Throughout most of history, only human agents have had the capabilities to perform both cognitive and physical functions. Beginning with the First Industrial Revolution ("Industry 1.0"), however, and continuing with accelerated pace to this day, humans have been working collaboratively with "automated" machines, and robots. Initially, the term *automated* suggested the performance of a single, discrete, physical task and/or process. Over time, hardware technologies continued to improve, thereby, increasing the *automation of physical tasks*. In fact, today, many machines can execute those physical functions automatically with little or no-human intervention. Some machines can even make simple decisions (i.e., actuators). Even so, until recently, humans have remained the sole executors of all *cognitive functions.*

*Cyber-technologies,* which first emerged in the latter part of last century, have exploded since the dawn of the 21st century. As these "digital technologies" grew in functionality, sophistication, quality, and quantity, engineers began to realize that they could be used to "automate" *cognitive functions* as well. This realization is at the heart of Industry 4.0 or Smart Manufacturing Systems.

In our view, therefore, all *smart systems* will comprise networks of *smart cognitive agents*. Each such agent fills predefined roles and executes predetermined "cognitive" functions, often through interactions with other agents. The capabilities required of each agent will be implemented "jointly" by software agents,

hardware agents, artificial (machine) agents, human agents, and organizational agents, among many others (Romero et al. 2016; Romero et al. 2017). In this paper, we focus how to model such multi-agent systems for one role: *control.*

## 2 SMART SYSTEMS AND AGENTS

In this section, we describe two distinct ways of modeling different human and ICT agents and their interactions: human-machine systems and joint-cognitive systems (Woods et al. 2000; Rasmussen et al. 1994).

### 2.1 Modeling Agents as Human-Machine Systems

The first model represents systems in which each task is performed by some combination of a human operator "interacting with a machine" (Sheridan et al, 2006). "Interacting with" covers a wide spectrum of activities (Fasth-Berglund, 2013). At one end of that spectrum, is the *human-machine system (HMS),* where the machine performs all activities under a predefined set of normal circumstances. The human may need to take-over when there is a problem that the machine cannot solve. This type of interaction requires what is commonly called a *Human-Machine Interface (HMI).* This *interface* must be designed and built so that the human can (a) build an accurate simulation model of the real-world problem and (b) actually correct the problem, if needed, based on that model. Typically, these human-machine interfaces have four basic properties:

- Transition-oriented to capture, simulate, and display events and sequences
- Future-oriented to reveal what should/can happen next
- Pattern-oriented to support quick recognition of unexpected or abnormal conditions
- Abstract-oriented to capture the state and trends in higher order system properties

### 2.2 Modeling Agents as Joint Cognitive Systems

At the other end of the spectrum is the *Joint Cognitive System (JCS)* (Hollnagel et al. 2005). Currently available *cyber-technologies* have dramatically increased the *cognitive capabilities* of machines (i.e., cognitive computing). As these technologies advance, so will those cognitive capabilities - from reactive to self-aware. This means that the human and the machine can work more collaboratively, as joint partners, to execute cognitive functions. Because of this, we argue that "human-machine-interface" view is no longer the appropriate view. The appropriate view is a "joint-cognitive-system" view. This change in view is critical because the "human-machine-interface" view introduces a distinction between the human and the machine that is no longer necessary, or even desirable. The human and machine are no longer thought of as separate components interacting through some type of interface. They can now execute tasks as an integrated team, working on the same tasks and in the same temporal and physical spaces (Romero et al. 2016). A number of real-world examples of JCS can be found (Hollnagel et al. 2005).

This *JCS* view signifies a fundamental change in how humans and machines can now work together. The integrated view changes the emphasis from the "interaction" between "humans and machines" to "human-machine symbiosis" (Tzafestas, 2006). *JCSs* are characterized by three aspects or principles: (a) goal-orientation, (b) control, and (c) co-agency. The first principle states that all agents are "goal-oriented"; the second aspect refers to the principle of "working together" to enhance control and minimize entropy (i.e., disorder in the system), and the third aspect is about the interdependent and inter-related nature of all the agents action within a JCS. Hence, *Co-Agency* can be defined as a "collaborative agent system of software agents, hardware agents, artificial (machine) agents, human agents, and organizational agents, among many others, in the service of jointly held goals". Therefore, integrated cognitive simulation models need to be developed to support the JCS. This will make the behavior of the system more intuitive, the derived decisions more cognitive. There are various functionalities for this kind of simulation: (1)

displaying the system behavior for human to understand the problems, (2) cognitive controlling of the machines, and (3) deciding simulation parameters and variables.

## 3 EVOLVING NOTIONS OF CONTROL

The earliest notion of control was labeled "direct" control (Wiener 1948). A direct controller executes functions to control a single device (See Figure 1). It receives a setpoint from its superior that quantifies the goal(s) its subordinate device is expected to achieve. The controller then determines the right commands to achieve that goal and sends them. In direct control, the commands directly adjust the necessary device parameters to meet predetermined goals and stay within acceptable tolerance limits.

In direct control, questions about 'being in control' and 'amplifying control' are best answered in terms of control theory. Control theory focuses on optimizing the controller's ability to handle the dynamics in the device being controlled. Those dynamics, which are inferred from feedback, are used to determine whether new commands are necessary.

In today's world, direct control is increasingly implemented using sophisticated mathematical models, usually based on differential equations, and continuous simulation/analysis tools such as LabVIEW, Modelica, and Simulink. The controller simulation is used to verify/validate the design of the controller and can also be used to control a simulated device. These models and tools analyze feedback from physical sensors on the device to predict behavior. This prediction is used to determine automatically if the setpoint is still valid. If it is not, then some parameters are adjusted, sometimes automatically using the models, simulation, and tools, and sometimes manually by human operator.

### 3.1 Supervisory View

A supervisory controller is capable of controlling more than one device simultaneously (see Figure 2). Consequently, supervisory controllers must perform two different levels of control. The lower level controls the individual devices. Control is done, conceptually at least, using two instantiations of the direct-control architecture. The upper level controls the system comprising those individual devices. We now discuss what functions, and their associated inputs, that this system-level control must perform.

There are two types of inputs: setpoints and feedback. Setpoints, as before, are quantifications of the system's goals, which are usually expressed as performance metrics. The controller, then, must determine (1) the contribution each device will make to those metrics and (2) the commands each device must execute to meet that contribution. The former can be thought of as an internal setpoint for the devices. If each device meets its designated setpoint, the system will meet its goals. The feedback comes directly from the devices and in two types. The first type provides the information that is needed by the device controllers to estimate device performance - same as above. The second type provides the information needed for the system controller to estimate system performance. This information is used by the simulation to evaluate and confirm the control strategy at both levels, which leads to the desired system performance.

### 3.2 Hierarchical Control

The evolution of control was to have separate controllers for each level and link them together. The supervisory controller did only system-level control and would connect only to device-level controllers - with n direct connection to the devices themselves. The device-level controllers would then connect to the devices. That structure is called a hierarchy (see Figure 3) (Simon 1962; Albus 1981).

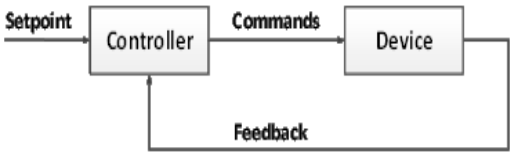


Figure 1: Direct control architecture.

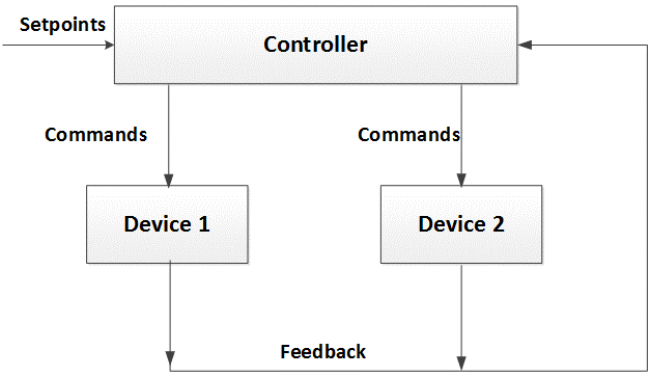


Figure 2: Supervisory control architecture.

The ability to implement such a structure relies on a simple, and already stated, observation. In supervisory control, the focus is on the system, not the devices. It is this focus that allows the supervisor to delegate and reassert authority, when necessary. It is this focus that enables the supervisor to achieve its own goals. And, it is this focus that gives the supervisor the responsibility for outcomes relative to those goals. So, it is not necessary for the supervisor to control the devices directly. Hence, the responsibility can be relinquished.

Relinquishing the direct control over the devices, however, does not mean that that control is not necessary. It is. In fact, the device controllers, which were formally "internal" to the supervisory controller, now become external to it. They become real, direct controllers as described above. As a result, the supervisory controller's main job is to derive and monitor setpoints for those device controllers. These derived setpoints are chosen so that the supervisory controller can meet its own goals. The individual device controllers can then determine the commands necessary for the individual devices to meet those derived setpoints.

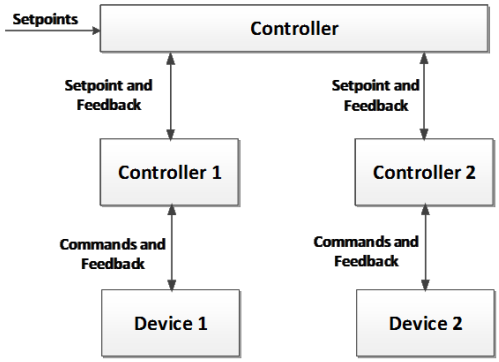


Figure 3: Hierarchical control architecture.

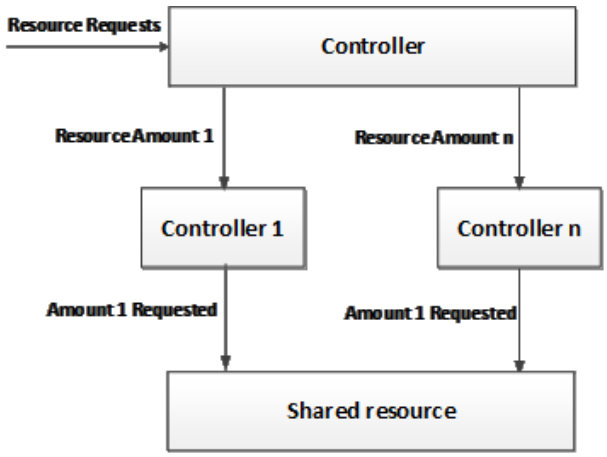


Figure 4: Polycentric control architecture.

The hierarchical control approach can be expanded to arbitrary number of levels. That number depends on a decomposition of the control problem into discrete problems. One continues to decompose until the problems are simple enough to be solved. The number of steps it takes to find those problems is the number of levels. This creates a tree structure of problems and solutions where the flow of control in this hierarchy is 1) strictly vertical and 2) between adjacent neighbors only. In an architectural sense, a hierarchy can be thought of as multiple layers of supervisory controllers "sitting on top of" a collection of direct controllers

and their associated devices. Simulation of hierarchical control is used to optimize the design of the whole control systems. The simulation studies help determine the optimal system operating conditions and balance the number of devices. The simulation can also help evaluate the performance of the controllers and the feedbacks at the regulating control level.

### 3.3 Polycentric Control

Hierarchical tree structures, like those shown above, are too rigid to deal with smart systems, which are dynamic, multi-layered networks of JCS enabled by, and empowered by, advances in information technology. Each node, often called an actor, in the network can perform a fixed and predetermined set of cognitive and/or physical functions, each with an associated expected performance goal. Each node in the network has multiple, direct, network connections to many other JCS: some acting as supervisors and some acting like subordinates. Each node in the network, therefore, is either a direct controller or a supervisory controller. Additionally, each node has many dependencies with other, non-directly, connected nodes.

Interactions within such complex systems were first explored by (Ostrom 2000). She noted that when a complex system depends on a common resource – such as single water source – individual controllers often make short-term, rational decisions that attempt to maximize their use of that resource. She also noted that those short-term decisions can deplete or destroy the common resource, on which they depend, in the long run. Ostrom argued that the best way to manage such systems is to create a higher level of organization responsible for the resource over its entire range and over longer periods of time. This organization then needs local authorities to change their behavior. That change involved sacrificing short-term return and autonomy so that this higher-level controller could analyze, plan, and dictate long-term behaviors.

The resulting architecture (See Figure 4) divides up the resource by giving each user a setpoint, which in this case represents each user's "share" of the resource. So, in this sense, the architecture has a supervisory control view. In return, the supervisor promised to sustain or grow the resource over the long term. Each user is now responsible for controlling its "share" of the resource. In other words, this user then determines how its own share can be used (See Figure 4). Dynamic simulation models or multiple integrated simulations are required to evaluate the performance of such a control system.

## 4 WHAT IS "CONTROL" IN SMART MANUFACTURING SYSTEMS?

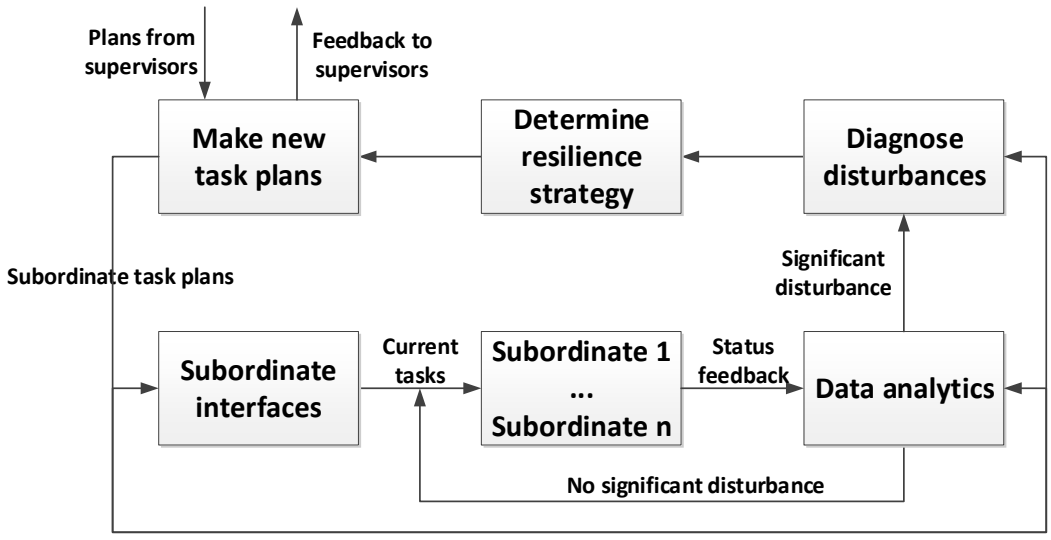


Figure 5: Generic controller.

A smart manufacturing system – in fact, any smart system – can be viewed as a network of agents that perform various combinations of cognitive and physical tasks. These agents work together to achieve both system-level and local-level performance objectives. Consequently, all agents are performance-based, rather than state-based. Locally, agents can be part of any of the architectures we described above. And, each such agent can be viewed as a JCS. This means that each JCS is simultaneously a supervisor to some nodes and a subordinate to other nodes. As a supervisor, the JCS controller provides broader temporal, spatial, and functional perspectives than its immediate subordinates. As a subordinate, the JCS controller must deal with issues they directly confront. All levels of control in smart systems then are in constant interplay as situations evolve. In Figure 5, we propose a generic architecture for any such controller.

### 4.1 A Generic Control Architecture

As a supervisor, each controller is responsible for creating task plans for each subordinate. Each subordinate task plan contains two major pieces of information: a list of partially-ordered activities and performance objectives. The list can be represented as an AND/OR graph where the nodes are the activities, including any required resources, and the arcs are time-based precedence relations. The objectives for each subordinate includes a description of the goals the subordinate is expected to meet and the allowable uncertainties. Collectively, these subordinate task plans are generated so that the controller, which is also a subordinate itself, can execute the activities and meet the goals assigned by its supervisors. Furthermore, as a subordinate, the controller provides feedback on those activities and goals to those supervisors.

In addition to creating these tasks plans, the supervising controller is responsible for monitoring its subordinates' execution of those plans. To do this, the controller uses its own performance trajectory and uncertainty band (See Figure 5) and the subordinates' feedback. As events occur, each subordinate communicates feedback to the supervisor. This feedback is analyzed using the Data Analytics function to determine its impact on the supervisor's performance goal. If the predicted performance lies within the tolerance band, then the supervisor does not need to change the current subordinate task plans.

Occasionally, subordinate events will cause the supervisor's predicted performance to go outside of that uncertainty band – see the heavy dashed line in Figure 6. When this happens, the supervisor's Diagnostic function must determine the problem and its Resilience function must determine a strategy for fixing the problem. Both functions use prior knowledge and deep learning to execute their assigned functions. Once a strategy is found, a new, subordinate, task plan will be generated. This new plan will be sent to the subordinates and the cycle starts again.

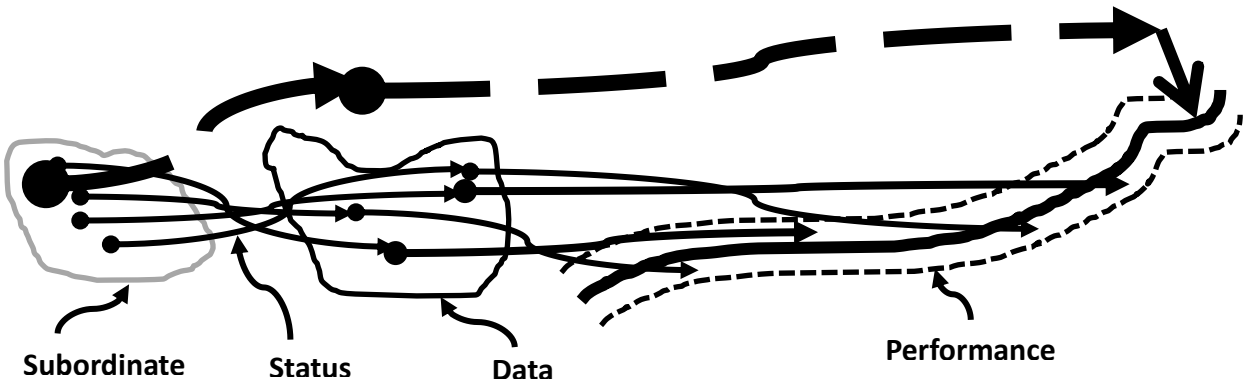


Figure 6: Graphical notion of control.

### 4.2 The Role of Simulation in this Control Architecture

Note, that every operational function – production process planning, shop scheduling, inventory management, among others - is associated with a similar control architecture. The "make new task plans", in particular, applies to every such operational function. Simulation applications have been used extensively in making such "plans." Moreover, these same applications have also been used for implementing the other generic functions in Figure 5. Simulation can be used to make predictions using feedback data. It can be used to diagnose the source of problems when those predictions go awry. And, it can be used to evaluate the control strategies, the system performance, and other proposed solutions to those problems.

So, simulation has an important role to play in this notion of networked-based, multi-layered control of smart manufacturing systems. This is not new. What is new, however, is the emerging effort in the vendor community to make simulation a cloud service.

## 5 SIMULATION AS A CLOUD SERVICE

Simulation is moving from a locally-based application to a cloud-based service. Cayirci (2013) defines Modeling & Simulation as a Service (MSaaS) as a ''model for provisioning modeling and simulation services on demand from a Cloud Service Provider (CSP). This provisioning keeps the underlying infrastructure platform and software requirements/details hidden from the users.'' The user of a MSaaS systems is not responsible for the system maintenance, software licensing, or infrastructure scaling.

In a survey, Cayirci (2013) discusses virtualization, cloud computing, infrastructure, platform, and software as a service (SaaS); explains MSaaS architectures; describes security, privacy, accountability, risk, and trust concerns; and then discusses the service composition and interoperability of relevant simulation federation technologies. Three types of MSaaS are considered: (1) modeling as a service, (2) model as a service, and (3) simulation as a service. Users may use any of these type of MSaaS and store the results in the CSP for later use.

Guo et al. (2011) propose a service specification that formally defines how simulation software can be a service (SSaaS) and formally specifies service-oriented simulation experiments. This specification enables the generation of meta-models for simulation experiments.

Taylor et al. (2014) introduce the CloudSME Simulation Platform (CSSP), which enables simulation applications to be deployed as service (SaaS) that is supported by a cloud platform (PaaS). The case study shows how easily that CSSP can be used for agent-based simulation.

There have been some efforts focused on parallel, discrete event simulation in the cloud (Fujimoto et al. 2010; Ledyayev and Richter 2014; Li et al. 2013; Liu et al. 2012). The performance of the simulation deteriorates as the size of the cluster distributed across the cloud increases. This is due to the limited bandwidth and overhead of the time-synchronization protocols required. Thus, the cloud deployment for this category of simulations is still limited. Shekhar et al. (2016) proposed a cloud middleware for Simulation-as-a-Service (SIMaaS) that leverages a Linux container-based infrastructure with lower runtime overhead, a higher level of resource sharing, and low setup and tear down costs. Key requirements include: (1) SIMaaS needs to be able to elastically scale the number of simulations that need to be executed; (2) SIMaaS must ensure bounded response time to users' requests; (3) SIMaaS needs an ability to accept user-supplied result aggregation logic, apply it to the results of the simulation, and present the results to the users, and; (4) SIMaaS needs to provide a web-based user interface to users so they can load the simulation model, input parameters, and receive the results (Shekhar et al. 2016).

Rak et al. (2012) developed mJADES to support simulations in the cloud. It is a Java-based architecture that is designed to run concurrent simulations by automatically acquiring resources from an ad hoc federation of cloud providers.

Al-Zoubi and Wainer (2011) proposed the RESTful interoperability simulation environment (RISE), a cloud middleware that applies RESTful APIs to interface with the simulators and allows remote management through Android-based handheld devices.

For large-scale simulations in the cloud, users will have challenges such as (1) Usability- graphical user interfaces that are specifically designed for cloud-based simulation services are needed and (2) Data Confidentiality- the data used by the simulation may be sensitive and data providers may not permit the use of cloud services due to confidentiality concerns. Zehe et al. (2015) propose an architecture called the Scalable Electro-Mobility Simulation (SEMSim) Cloud Service, which allows the integration of multiple simulations using the High-Level Architecture (HLA)/ Run-Time Infrastructure (RTI).

## 6 SIMULATION SERVICE - JCS CONTROLLER INTEGRATION SCENARIO

Even if a smart manufacturing system can be expressed as a network of agents/controllers operating as nodes in a JCS, each agent must be able to translate the plans it receives from superiors into the actions necessary to implement that plan, including sending plans to all of its subordinate agents. For low-level agents, such as those that directly control a simple device, plans received from a supervisor might directly translate into actions. For example, the supervisory agent instructs the subordinate agent to turn its associated device off and the subordinate complies. For agents that operate at a "higher" level, implementing a plan might involve complex analysis and execution planning. For example, a high-level agent controlling a production system involving many machines might receive from its supervisor a plan that calls for the production of n widgets by the end of the shift. The problem this agent must actually solve is, given the current state of production, what activities do I need to undertake to carry out this plan *in addition* to the plans I am currently carrying out, while also taking into account the activities (plans) that my subordinate agents are able to carry out. Simulation has been shown as a viable method to solve problems characterized in this way and a cloud-based simulation service would make the technology available for use by multiple agents in a JCS. But if simulation is to be used solve the problems a JCS agent might face and if the execution of the simulation is to take place in the cloud, several issues must be resolved including: how to describe an agent's problem; how to exchange that problem with the simulation service, and; how to quickly convert the problem into an executable form and return the results to the agent.

Figure 7 illustrates how an agent in a JCS might be integrated with a simulation service. In this scenario, the behavior of the controller can be described as evaluating the plans sent to it by its supervisor in the context of its current state, resulting in a target state (goal) and a set of actions that the controller will undertake to reach the target state. The controller's current state includes the current state of the physical (machines, tools, employees, etc.) and logical (process plans, orders, subordinate controllers, etc.) entities that make up the part of the production system that the controller is managing, the production-related capabilities of those entities, and the actions that may be undertaken to change the state of those entities and thus change the controller's state. When new plans are received and the new goal will require a complex or extensive reworking of the current plan, the controller can choose to use a cloud-based simulation service to assist it in evaluating the situation so that a set of actions can be developed that will enable it to move from its current state to its new target state.

A key need of this integration scenario is the development of an exchange standard, which we call a Generic Simulation Problem Representation (GSPR). GSPR is **not** a simulation model per se; but, it is a description of the problem that needs to be solved using a simulation service. GSPR provides a form that can be converted into a simulation and that is readily exchangeable between the controller and the simulation service. It must have a flexible means for describing system state, system capabilities, target goals, and result formats for the problems that could be solved through simulation.

If a simulation service is to be used to solve the controller's problem, the information related to the problem must be packaged and formatted based on the GSPR specification. Referred to as a GSPR instance, this data consists of the part of the controller's state necessary to describe the problem and a description of the target goal. Information necessary to enable the controller to access the results will also be present.

The GSPR instance information could be constructed manually, but that would restrict the integrated system to only solving problems with long lead times. Since the format for this data is specified by the GSPR specification and the controller must have some representation for its state, it should be possible to construct a software component that the controller can use to take its state and the target goal as input and produce a GSPR instance. At the top of Figure 7, this component is referred to as the Simulation Problem Generator.

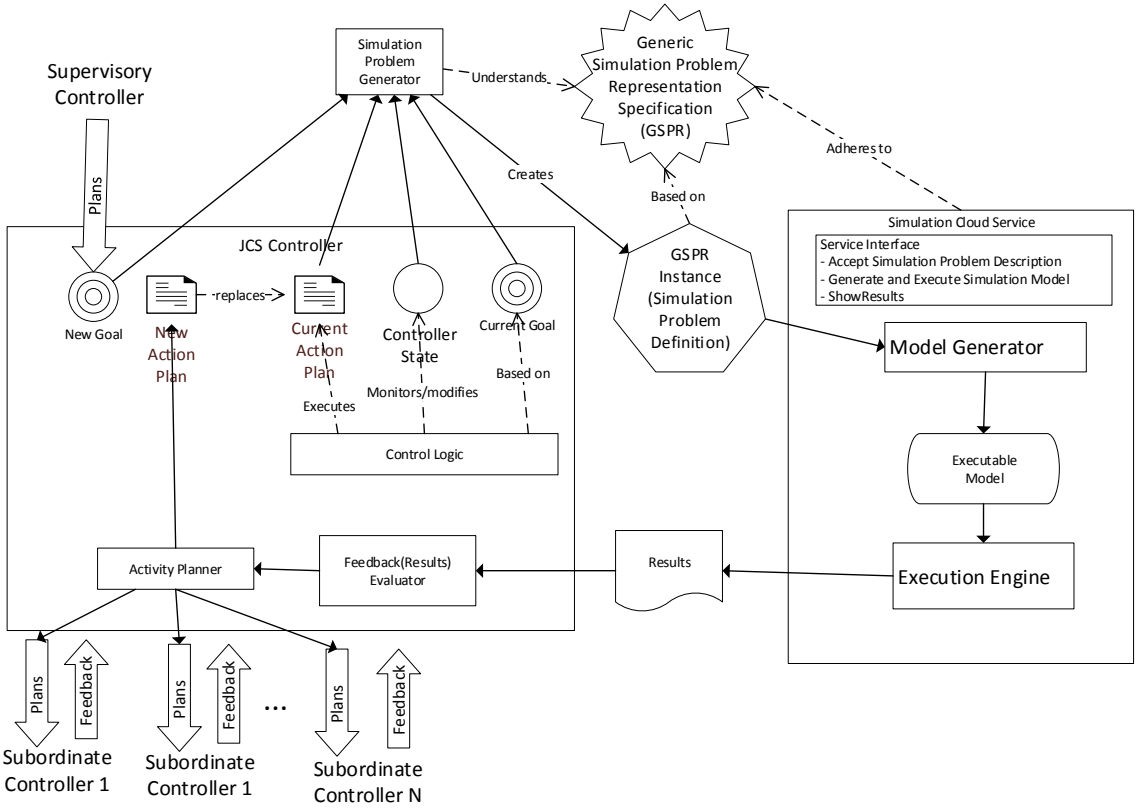


Figure 7: JCS Controller integrated with a Cloud-Based Simulation Service.

The last component of this scenario is the Cloud-based Simulation Service. The interface to this service is quite simple: accept a simulation problem description; generate and execute the simulation model, and; show results. The key components of the service are the model generator (built to understand information assembled according to the GSPR specification) and the execution engine. The model generator takes as input the GSPR instance and uses it to create an executable simulation model. The execution engine runs the simulation and collects the results.

To complete the integration, the controller receives the results from the simulation service. It then evaluates the results and uses the information to come up with a new set of actions to reach the target state. Some of these actions might only involve local changes to the controller's state, but other actions might be sent to subordinate controllers for implementation.

The feasibility of this scenario may seem uncertain, but several of the developments necessary for success are achievable as extensions of previous research, including:

- *Controller State Representation* - The Core Manufacturing Simulation Data (CMSD) standard could be used to as a model to represent the controller's state with respect to the production system being managed (SISO 2010). CMSD defines the format and content for the information entities

necessary to enable information exchange between simulations and other shop floor applications, and has been used in many studies in this area (McLean et al. 2005; Lee et al. 2011).

- *Simulation Problem Description* – CMSD has both a UML and an XML representation (SISO 2012). As such it already provides a means for not just defining controller state but for exchanging controller state. To exchange Simulation Problem Descriptions, exchangeable representations for simulation configuration and the target goal would need to be developed and exchanged along with the CMSD supported information.
- *Model Generator* – There have been several studies that used CMSD Data, enhanced with simulation configuration and goal information, to generate simulation models that were then executed on a simulation execution engine (Bergmann et al. 2012; Bergmann and Strassburger 2015; Jain and Lechevalier 2016). These and other studies have been successful in generating executable models for different simulation engines.

## 6. SUMMARY

In this paper, we discussed the changes taking place in manufacturing as a result of the proliferation of new information and communication technologies (ICT). Those technologies are automating some of the cognitive functions that have been traditionally done by human beings. We argued that, today at least, humans collaborate with those technologies to collect and analyze data and make decisions based on that analysis. We call this resulting collaboration joint cognitive systems (JCS). We then describe a generic architecture for control based on the notion of a JCS. Finally, we discuss some research questions that arise when using cloud-based, simulation services to implement the functions in that architecture.

## DISCLAIMER

No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software systems are identified in this paper to facilitate understanding. Such identification does not imply that these software systems are necessarily the best available for the purpose.

## REFERENCES

Albus, J., A. Barbera, and R. Nagel. 1981. "Theory and Practice of Hierarchical Control". In *Proceedings of the 23rd IEEE Computer Society International Conference,* 18-39. Piscataway, New Jersey: IEEE.

Bergmann, S., S. Stelzer, S. Wüstemann, and S. Strassburger. 2012. "Model Generation in SLX using CMSD and XML Stylesheet Transformations". In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque et al., 3046 - 3056, Piscataway, New Jersey: IEEE.

Bergmann, S. and S. Strassburger. 2015. "On the Use of the Core Manufacturing Simulation Data (CMSD) Standard: Experiences and Recommendations". In *Proceedings of the Fall Simulation Interoperability Workshop 2015*, August 31st to September 4th, 2015, Orlando, Florida.

Cayirci, E. 2013. "Modeling and Simulation as a Cloud Service: A Survey." In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy et al., 389–400. Piscataway, New Jersey: IEEE.

Fasth-Berglund, Å. And J. Stahre. 2013. "Cognitive Automation Strategy - for Reconfigurable and Sustainable Assembly Systems". *Assembly Automation* 33(3):294-303.

Fujimoto R. M, A. W. Malik, and A. Park. 2010. "Parallel and Distributed Simulation in the Cloud". *SCS M&S Magazine* 3:1–10. The Society for Modeling and Simulation International (SCS).

Guo, S., F. Bai, and X. Hu. 2011. "Simulation Software as a Service and Service-Oriented Simulation Experiment." In *Proceedings of the 2011 IEEE International Conference on Information Reuse and Integration*, 113–116. Piscataway, New Jersey: IEEE.

Hollnagel, E. and D. Woods. 2005. *Joint Cognitive System: Foundations of Cognitive Systems Engineering*. London: Taylor and Francis.

Jain, S. and D. Lechevalier. 2016. "Standards based generation of a virtual factory model". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T.M.K. Roeder et al., 2762-2773. Piscataway, New Jersey: IEEE.

Ledyayev, R. and H. Richter. 2014. "High Performance Computing in a Cloud Using OpenStack". In *Proceedings of CLOUD COMPUTING 2014: The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization*, May 25th - 29th, 2014, Venice, Italy.

Lee, T., F. Riddick, and B. Johansson. 2011. "Core Manufacturing Simulation Data – A Manufacturing Simulation Integration Standard: Overview and Case Studies". *International Journal of Computer Integrated Manufacturing* 24(8):689-709.

Li, Z., X. Li, T. Duong, W. Cai, and S.J. Turner. 2013. "Accelerating Optimistic HLA-Based simulations in Virtual Execution Environments". In *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation*, 211-220, Ney York, New York: ACM.

Liu X., Q. He, X. Qiu, B. Chen, and K. Huang. 2012. "Cloud-Based Computer Simulation: Towards Planting Existing Simulation Software into the Cloud". *Simulation Modelling Practice and Theory* 26:135-150.

McLean, C., Y.T. Lee, G. Shao, and F. Riddick. 2005. "Shop Data Model and Interface Specification". Technical Report NISTIR 7198. National Institute of Standards and Technology, Gaithersburg, Maryland.

Ostrom E. 2010. "Beyond Markets and States: Polycentric Governance of Complex Economic Systems". *American Economic Review* 100(3):641-72.

Rasmussen, J., A. Pejtersen, and L. Goodstein. 1994. *Cognitive Systems Engineering*. Hoboken, New Jersey: Wiley.

Romero, D., P. Bernus, O. Noran, J. Stahre, and A. Fast-Berglund. 2016. "The Operator 4.0: Human Cyber-Physical Systems & Adaptive Automation towards Human-Automation Symbiosis Work Systems". In *Proceedings of the IFIP International Conference on Advances in Production Management Systems*, September 3rd-7th, Iguassu Falls, Brazil, 677-686.

Romero, D., T. Wuest, J. Stahre, and D. Gorecky. 2017. "Social Factory Architecture: Social Networking Services and Production Scenarios through the Social Internet of Things, Services and People for the Social Operator 4.0". In *Proceedings of the IFIP International Conference on Advances in Production Management Systems*, September 3rd-7th, 2017, Hamburg, Germany, 265-273.

Sameer, M., K. Muztoba, D. Romero, and T. Wuest. 2017. "Smart Manufacturing: Characteristics, Technologies and Enabling Factors". *The Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*. Special issue article: 1-20.

Shekhar, S., H. Abdel-Aziz, M. Walker, F. Caglar, A. Gokhale, and X. Koutsoukos. 2016. "A Simulation as a Service Cloud Middleware". *Annals of Telecommunications* 71:93-108.

Sheridan, T. and R. Parasuraman. 2006. "Human-Automation Interaction". *Human Factors and Ergonomics* 1(1):89-129.

Simon, H. 1962. "The Architecture of Complexity". *The American Philosophical Society* 106(6):467-482.

Simulation Interoperability Standards Organization. 2010. "SISO-STD-008-2010: The Core Manufacturing Simulation Data - UML Model". Simulation Interoperability Standards Organization, Orlando, Florida.

Simulation Interoperability Standards Organization. 2012. "SISO-STD-008-01-2012: Standard for Core Manufacturing Simulation Data - XML Representation". Simulation Interoperability Standards Organization, Orlando, Florida.

Taylor, S., A. Anagnostou, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini. 2014. "A Tutorial on Cloud Computing for Agent-Based Modeling Amp; Simulation with Repast". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk et al., 192-206. Piscataway, New Jersey: IEEE.

Thoben, K., S. Wiesner, and T. Wuest. 2017. "Industrie 4.0 and Smart Manufacturing - A Review of Research Issues and Application Examples". *International Journal of Automation Technology* 11(1):4-19.

Tzafestas, S. 2006. "Concerning Human-Automation Symbiosis in the Society and the Nature". *Interntaional Journal of Factory Automation, Robotics and Soft Computing* 1(3):16-24.

Weiner, N. 1948. *Cybernetics: Or Control and Communication in the Animal and the Machine.* Paris, (Hermann & Cie) and Cambridge, Massachusetts. MIT Press.

Woods, D. and N. Sarter. 2000. "Capturing the Dynamics of Attention Control from Individual to Distributed Systems: The Shape of Models to Come". *Theoretical Issues in Ergonomics Science* 11(1-2):7-28.

Zehe, D., A. Knoll, W. Cai, and H. Aydt. 2015. "SEMSim Cloud Service: Large-Scale Urban Systems Simulation in the Cloud". *Simulation Modelling Practice and Theory* 58:157-171.

## AUTHOR BIOGRAPHIES

**ALBERT JONES** is the Scientific Advisor for the System Integration Division at the National Institute of Standards and Technology. His current research interests include using Category Theory as the new mathematical foundation for integrating, and using cognitive science as the new foundation for controlling, smart manufacturing systems. He is on the Engineering Advisory Boards at Morgan State University and Loyola University. Before coming to NIST, he held faculty positions at Loyola University and Johns Hopkins University. His graduate degrees are in Mathematics and Industrial Engineering from Purdue University. His email address is albert.jones@nist.gov.

**GUODONG SHAO** is a computer Scientist in the Life Cycle Engineering Group in NIST's Systems Integration Division of the Engineering Laboratory. His current research topics include modeling, simulation, and analysis; data analytics; and optimization for Smart Manufacturing. He served as a *member* of the *WSC Board of Directors* and on the editorial board of the International Journal on Advances in Systems and Measurements. His email address is gshao@nist.gov.

**FRANK RIDDICK** is a computer Scientist in the Process Engineering Group of the Engineering Laboratory at NIST. He has participated in research and authored over 50 technical papers relating to distributed simulation, manufacturing simulation integration, information modelling, and smart manufacturing. His email address is frank.riddick@nist.gov.