Cognitive Architectures for Process Monitoring – an Analysis

Alexander Wendt[(1)], Stefan Kollmann[(2)], Aleksey Bratukhin[(3)], Alireza Estaji[(2)], Thilo Sauter[(2,3)], Axel Jantsch[(1)]

(1) TU Wien, Christian Doppler Laboratory for Embedded Machine Learning, Vienna, Austria, <firstname>.<lastname>@tuwien.ac.at
(2) TU Wien, Institute of Computer Technology, Vienna, Austria, <firstname>.<lastname>@tuwien.ac.at
(3) Danube University Krems, Department of Integrated Sensor Systems, Wr. Neustadt, Austria, <firstname>.<lastname>@donau-uni.ac.at

***Abstract*— In smart manufacturing, the demand increases to be able to monitor and adjust process execution during production. It has led to a shift towards distributed, modular automation. A solution seems to be to use a cognitive architecture. It turns out that their generality often makes them unsuitable for specific industrial problems. In this paper, we propose a cognition-inspired architecture design for health monitoring tasks. The problem class is represented by a conveyor belt use case. We then discuss to what extent this architecture matches common implementations of cognitive theories by following a generalized cognitive process.**

Keywords—cognitive manufacturing, cognitive architecture, cyber-physical system, distributed system, multi-agent systems

I. Introduction

An important task addressed by smart manufacturing and Industry 4.0 is the ability of automation systems to cope with the rising complexity and dynamism of manufacturing environments. It also includes being able to monitor and adjust process execution appropriately during the production to ensure the desired quality and ultimately to increase yield. With the growing complexity of the shop floor and the dynamic nature of production processes, state-of-the-art monolithic centralized systems no longer meet the needs of the industry. These requirements have led to a shift towards distributed, modular automation, giving more authority to the field level controllers for autonomous decision-making [1]. However, a global challenge of distributed control is its lack of a global overview, resulting in possible miss-assessments, inefficient or faulty operations, and potential convergence problems, which still limit the take-up of such solutions by industry [2].

Conventional approaches to system monitoring and error detection mostly rely on individual machines to determine any deviation from their expected performance and take corrective actions if possible [3]. Furthermore, this information typically stays locally inside the machine and is not available to other process steps. Process adaptation across machine boundaries involving several process steps is thus not possible.

In our particular problem, we will look into a health monitoring application for a series production line for conrods involving several machines interconnected by a conveyor system. Its purpose is to predict conrod errors and perform preventive maintenance. This problem is representative of predictive maintenance by monitoring time series.

A natural solution to monitoring, anomaly detection, and decision-making problems seems to be a cognitive architecture [4], which provides a general theory of cognition, often inspired by the human mind. Their holistic approach seems ideal for capturing and assessing a comprehensive overview of a complex system. Indeed, cognitive architectures may be well suited for human-like applications like robots or simulations [5]. However, their generality makes them often unsuitable for specific industrial problems because the requirements of such applications differ too much [5], [6], [7]. Generic cognitive architectures entail a significant overhead in the actual translation of functionality that causes costly and inefficient workarounds for the system to fulfil its tasks.

Our problem involves many components suitable for cognitive architectures. We define a health monitoring system for typical industrial use cases like the conrod series production inspired by the idea of a cognitive architecture. Then, we want to discuss how much a proposed architecture for the health monitoring use case matches common implementations of cognitive theories and whether the architecture can be classified as a cognitive architecture. Our contribution is twofold. It can be summarized as the following:

- Provide a design for a for health monitoring architecture, whose goal is to reduce the number of faulty pieces in distributed production systems
- Provide a methodology for designers, who want to use cognitive architectures in industrial applications to estimate to what extent a given software architecture can be classified as a cognitive architecture

The main takeaway is a qualitative study of how well the provided architecture design fits into the concepts of cognitive architectures. It provides insights in designing cognitive architectures for industrial use cases. Quantitative evaluation of the proposed architecture is not the scope of this paper.

The paper is structured as follows. After the definition of our problem in section II, we analyze common cognitive architectures and other solutions in the area of cognitive manufacturing in section III. In section IV, we design our health monitoring system to fit the use case. Finally, in section VI, we map modules of our solution to a cognitive process. We determine what parts of the cognitive process we can apply and which parts of our architecture do not fit.

II. Conrod Quality Measurements Use Case

In the particular health monitoring problem, a series production line for combustion engine conrods involves seven machines interconnected in series by a conveyor system. In each stage, machines modify some of the 24 physical properties of the metal piece, e.g., the thickness by grinding in Machine 1 and hole size by drilling in the succeeding Machine 2. After the last manufacturing step, a measurement station automatically measures the physical end properties, here called features. Predefined measurement thresholds and acceptable tolerances indicate the acceptance or rejection of a conrod if any feature is outside of the tolerance boundaries. Fig. 1 shows a conrod and some of the measured features.

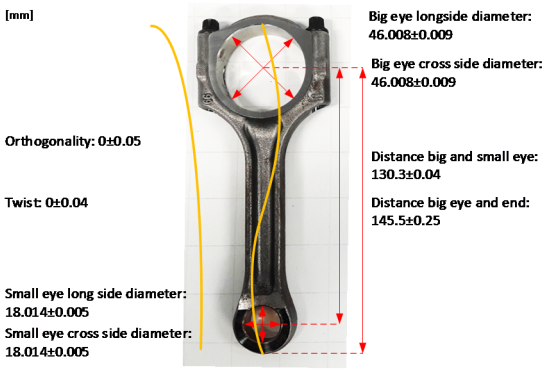

Fig. 1. Conrod quality measures with nominal values and tolerances

This system is a classic approach with independently working machines without any networking and minimal corrective actions. Although the machines have no knowledge of each other, errors can propagate through the features of the conrods. The goal is to reduce the number of faulty conrods and to minimize the cost of interrupting the system, e.g., by replacing a worn-out driller. To predict errors, the health monitoring system of each machine analyzes the measured time series of features. The time series show patterns like drifts, which end up in errors. If it would be possible to recognize the drift, then a maintenance action could be applied before the values exceed the tolerance boundaries.

Further, a problem to consider is that a drift or particular error pattern may be reinforced in a specific machine, but caused by other processes in a previous machine. For example, the conrod thickness varies due to the tilt of the workpiece in the grinding, which causes the driller to drill holes with higher variance. It requires cooperation between several machines.

The interaction and influence of several physical machines on a large-scale shop floor results in a system, which underlying structure cannot be known at design time because the underlying physical conditions are unknown. First, we cannot fully define the physical condition of the machines, which is limited by the limitations in available sensor equipment, e.g., no internal sensors. Second, we cannot fully determine the interactions between machines. Therefore, we suggest a system that is capable of identifying the relevant information, indicators, and patterns in the system by itself or at least with minimal user interaction.

III. Related Work

Cognitive architectures in industrial automation are an interdisciplinary topic. Therefore, we will first provide an overview of advances in cognitive manufacturing. Then, we look into common cognitive architectures as well as the problems that occur when we want to apply a general cognitive architecture schema on a specific use case.

A. Recent Approaches in Cognitive Manufacturing

The emergence of cognitive manufacturing became possible due to the availability of the huge amount of data that can be analyzed – a requirement for cognitive systems. According to [8], cognitive manufacturing applies cognitive computing, the Industrial IoT, and advanced analytics to optimize manufacturing processes beyond state of the art in the areas of Asset Performance Management, Process and Quality Improvement, Resource Optimization and Supply Chain Optimization. While traditional control systems use explicit "cause - reaction" logic, we try to address the situations when it's unclear neither the cause nor the possible reaction.

In [4], authors show an emerging trend in the introduction of cognitive approaches in manufacturing architectures, such as CogNetCon and PLANTCockpit that support the concept of Open architecture providing generic integration solution but requires flexibility that can be fulfilled by cognitive architectures. The authors propose to introduce a Cognition layer to the generic SOA that would consist of Cognition Engine, Model Repository, and Knowledge Representation Component and introduce cognition-based functionality to deal with the complexity and dissimilarity of the data.

Other solutions similar approaches adding some sort of cognitive engines with associated models to analyze, predict and improve manufacturing processes through cognitive functionalities by the seamless integration of different sources of information and merging. There are two areas of research of cognitive manufacturing present: Solutions such as [4] and [9] focus on the integration aspect attempting to merge the different sources of information under an umbrella of the cognitive engine. [4] propose an abstract cognitive architecture schema as the body of the "machine agents" without getting to detailed functionality design. However, they do not show how the cognitive architecture is actually implemented. Others, such as [10] and [11] focus on the model and engine development while the former aim to create a generic framework that can serve as a playground for the latter, which are often tuned to a particular manufacturing application area.

B. Common Cognitive Architectures

Cognitive architecture is an ambiguous term, and its usage is twofold [12]. One goal of cognitive architectures is to define and validate a particular theory of cognition. These theories are often inspired by psychology, biology, or decision theory. Another goal is to provide advancements in the area of artificial intelligence. It is argued that human is the most complex being that we know and human can solve lots of complicated problems. If we manage to design an architecture that represents the human cognitive process, then we should have a general tool for complex industry applications [13].

In the famous survey by Vernon et al. [14], they described the common cognitive architectures SOAR, ACT-R, and ICARUS, among others. SOAR originates from the domain of logic problem solvers in artificial intelligence. Initially, it had some basic cognitive functionalities. Hence, system behaviours were entirely defined by rules. While ACT-R [15] is used widely in psychological experiments, ICARUS [16] and SOAR [17] are used in robotic applications or simulations of human decision making [17], [14]. The architectures LIDA [18] and SiMA [19] both modelled the human mind more extensively.

We observe that there is a convergence of functionality through the evolution of the different cognitive architectures. For instance, both SOAR and ICARUS were initialized only with a procedural memory. They were extended by an episodic memory [17], [16]. This type of memory can be found in LIDA [18] and SiMA [19] as well. The same can be said about the attention mechanism in BDI, which eventually also got one in this implementation of the basic model [20]. Although architectures are converging, there are still too many unsolved high-level problems to be able to bring them into a grand unified theory of cognition [16].

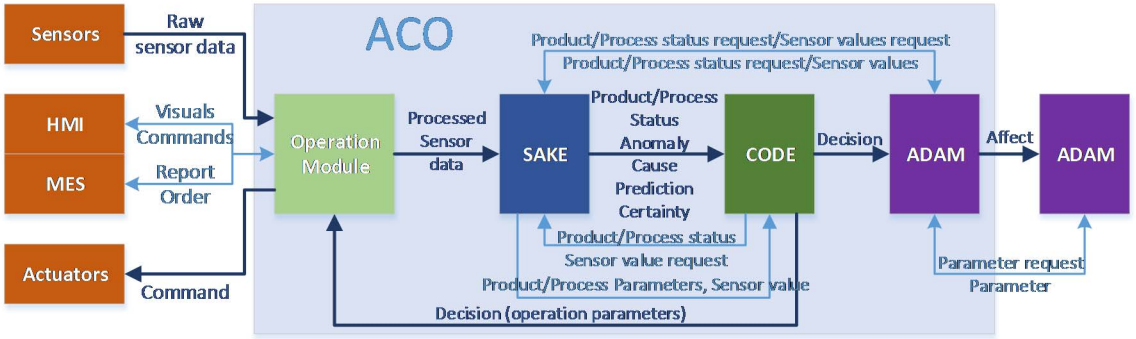

Fig. 2. The Autonomous Cooperating Object (ACO) system design

The main problem in designing cognitive architectures for industrial problems is highlighted in [7] and [5]. One either create a general schema for a specific theory of cognition, called *design by desiderata,* or one designs a cognitive architecture around a particular use case, called *design by use case* [7]. These possibilities are usually exclusive as design by desiderata is a top-down based approach, and design by use case is bottom-up. On the one side, if a general theory of cognition is used for a particular use case, there is a significant overhead. On the other hand, a specialized architecture will probably not support a theory of cognition. In [5], the implementation of a cognitive architecture in the building automation domain is demonstrated. A simulation planner is implemented as deliberative decision-making architecture, handing incoming HTTP-requests as goals. Although it fulfils its task as a cognitive architecture, it comes with a significant overhead of workarounds to fit the task to the way a cognitive architecture process information. It was not competitive with a use case-based, standard solution. Instead of forcing the use of a cognitive architecture as in [5], we want to start from the use case in Section II and then see how much "cognition" is left in the proposed architecture.

IV. The SAVE Architecture

The basic idea is to design a health monitoring system based on our use case. From Section III, we chose the design by use case paradigm. However, we want to introduce cognitive aspects to make our system more robust and adaptive. We intend to deploy an instance of our health monitoring system, as *Autonomous Cooperating Objects* (ACO), on each machine in the production line. Each machine has its own, specialized task and communicates with other ACOs to exchange information and make requests. Similar to a cognitive architecture, each ACO is designed to meet its own objectives regarding behaviour and performance.

A. Top-Level Functionality

The use case requires consideration of the following functionality, to lower the number of faulty conrods at a minimum of maintenance costs: Autonomous knowledge extraction by automatically recognizing value ranges; prediction of faulty conrods by identifying trends and drifts in time series; decision-making where each ACO shall have the possibility to choose between multiple actions, e.g. to replace a driller or to adjust grinding strength; and finally dynamic clustering that allows effective negotiation and propagation of mitigation measures as well as representation to other ACOs.

We designed the following ACO architecture inspired by cognitive architectures, but not bound to them, as shown in Fig. 2. Each ACO is an agent within a multi-agent system. It consists of four modules: The Operation Module (OM), the Self-aware Autonomous Knowledge Extraction unit (SAKE), the COgnitive Decision-making unit (CODE), and the Autonomous environment Discovery and Management unit (ADAM). In the following, each module and their interactions are explained.

B. Operation Module (OM)

The OM represents the interface to the machine and contains the hardware-specific drivers of the system. It converts machine input to an internal representation, e.g. to extract sensor data from system messages. In the direction from ACO to the machine, it converts abstract action-commands into machine commands, e.g. converting the action "replace driller" to Modbus register values.

C. Self-aware Autonomous Knowledge Extraction (SAKE)

SAKE is designed as a self-aware unit to collect necessary information about itself and the conrod features. Its outputs are the base for decision making. This component intends to maximize automatic information extraction. As this module only contains single-machine signal analysis functionality, it fits well as a separate module.

SAKE is partially based on CCAM (Confidence-based Context-Aware condition Monitoring) [21], with adaptions to drift analysis. The primary purpose of CCAM is unsupervised, model-free machine-health monitoring. CCAM treats target devices, like water pipe systems or electrical motors, as black boxes. It uses only contextual information to trace the system states and is applied as a change point detection algorithm. CCAM also provides simple drift detection using a Discrete Average Block (DAB) algorithm. SAKE receives an input stream of raw data with conrod measurements. It performs data analysis and generates a report for the decision-making module CODE. Drift detection algorithms like Exponentially Weighted Moving Average (EWMA) and Cumulative Sum (CUSUM) are screening the data to determine drifts. It aggregates the results of multiple parallel running algorithms and generates a report per input sample.

D. COgnitive DEcision-making unit (CODE)

The CODE component is designed with flexibility and reusability in mind. CODE does not need to learn how to control a particular production line. It only learns to optimize control within the boundaries of a given abstraction.

A CODE component can be implemented by means Deep Q-Learning. [22] combines traditional Reinforcement Learning (RL) with Deep Neural Networks (DNN) to produce Deep Q-Networks (DQN) for artificial agents.

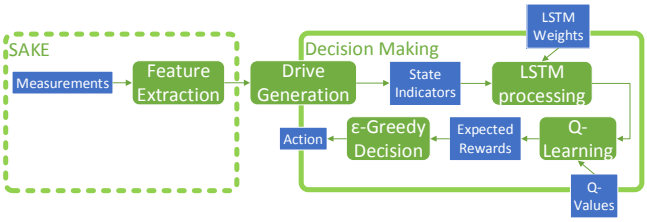

Fig. 3. Overview of the CODE module

The DNNs employed in this approach use adaptive convolutional layers to extract hierarchical features from the high dimensional inputs, without user interaction.

The approach in [22] is evaluated on a series of Atari video games and illustrates the potential of Convolutional Neural Networks (CNNs) for hidden state detection in high dimensional data. We expect the networks to perform comparably in our setup, identifying hidden states based on patterns present in the input data. CODE will extend the state estimation of DQNs with additional confidence information, which will be integrated into the Q-Learning concept. See Fig. 3 for an architecture overview of the CODE setup.

In this architecture, the symbolization step is reduced to feature extraction and preprocessing in the SAKE component, see the dotted box labelled SAKE in the figure above. The resulting state-indicators are passed into *Drive Generation* where the internal motivations for the agent are generated. Drives represent the system goals and system metrics. There is a monetary drive, which will be derived from the weighted, averaged value of the production line. A second drive will be a predictive drive that will provide rewards depending on the predictability of faulty conrods. We combine conrod features and drives as input for reasoning.

Long-term memory activation is covered by one or more Long-Term-Short-Term (LSTM) networks or potentially other, possibly deep, convolutional neural networks, trained for specific cases of production line states. Long-Term Short-Term Memories (LSTM) [23] are a form of recurrent Neural Networks that can train on temporal correlations in data with longer delays than traditional recurrent approaches.

To exemplify the setup in Figure 3, we could start by offline training LSTMs on typical drift sequences that are characteristic for tool wear-out. During online application, these LSTMs create predictions. The accuracy of these predictions, in combination with confidence information provided by the LSTMs, will then be used to determine if any predictors fit well and if so, which one fits the best to the current situation. The Q-Learning algorithm then uses this candidate for hidden state detection. Based on the detected state, the RL algorithm, e.g. Q-Learning, will determine the expected rewards for all known actions and deliver these to a simple decision-making strategy.

E. Autonomous environment Discovery and Management unit (ADAM)

ADAM primary focus is to facilitate the communication between the ACOs using the Relevance map to find appropriate communication partners. The functionality of ADAM can be divided into two distinct phases: 1) offline phase where ADAM operates on historical measurement data and prepares for the online operation and 2) the online phase where ADAM is primary a passive component reacting on the incoming request and reports.

The Relevance map is the main database ADAM operates on. For each monitored parameter of the ACO, there is the corresponding pair of values: "ACO" and "relevance". ACO is a unique identification of ACO in the system, and relevance is a float value ranging from 0 to 1, where 1 defines absolute certainty that the respective ACO causes deviations of the parameter in question.

Initially, ADAM uses historical measurements for the creation of the Relevance map by 1) finding a deviation point in one of the Pearson distribution parameters of the measurements and looking through the other ACO measurements if at this point of time there is a change in the any of the other ACOs parameters. Each correlation adds to the relevance value between the ACO for the particular parameter.

During the system run, ADAM subscribes to all the parameters of all relevant ACOs. In a broader case, ADAM should subscribe to all the parameter changes. Still, to reduce the communication overload, only ACOs that have above 0.5 relevance value for a parameter will be of interest. Upon receiving the measurements, ADAM attempts to find a correlation between the local performance and the ACOs in question to establish what parameter of what ACO possibly affect local ACO performance. The evaluation metrics are based on the extract values comparison as well as Pearson distribution values (mean and sigma). After each cycle, ADAM adjusts the Relevance map. At the same time, ADAM collects information about local and remote actions taken and associate it with the effect of its local parameters.

In a nutshell, the goal of ADAM is to provide implicit connectivity between ACOs without forcing an explicit structure of the system. Upon receiving a request from CODE, ADAM has to decide which ACO can perform an action and what type of action it should be. To do so, ADAM evaluates the options in terms of the effects remote ACOs actions affected the parameter in question as well as ACO's costs, success rate, and success chance of a possible action. ADAM finds the possible ACOs that can act based on the relevance of ACOs to this particular parameter and publishes cost estimation requests to its topics related to the ACOs in question. It waits for the responses and chooses the best option that has the highest accumulated value for success rate and chance and the lowest value for the costs factor.

V. Cognitive Architecture Evaluation

We designed our health monitoring system based on the use case-based approach [7]. Now we want to estimate to which extent this architecture can be mapped to a cognitive architecture schema. We can summarize what a cognitive architecture does regarding an industrial application: A cognitive architecture is a software program that selects and executes an action from possible competing actions, which satisfies its goals, for any given input state.

We apply the cognitive process that was introduced in [5]. The model has been derived as a common denominator from the existing cognitive architectures BDI, SOAR, LIDA, ICARUS, and SiMA. We made minor naming modifications of the process in Fig 4. The cognitive process abstracts standard components in cognitive architectures into a general process. All cognitive architectures extend these steps within their cognitive theories, and they can be mapped to the process.

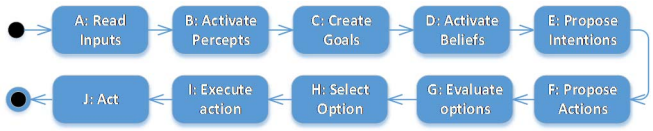

Fig. 4. The cognitive process of most common architectures [5]

A. Architecture Mapping

First, we go through each step of the process and try to map the ACO's functionality to that. We then compare the overlap of the process to other cognitive architectures.

A. Read Inputs: All architectures need a defined interface for transferring raw data from the sensor to the "body". For that purpose, we have the *Operation Module* (OM). All input from the connected machine arrives here, where drivers convert them into the internal representation. Because the connection to other machines uses another technology, the *Autonomous environment Discovery and Management* unit (ADAM) does not use OM for communication. Instead, it has its own external interfaces to other ACOs.

B. Activate Percepts: This is where symbolization of raw data is transferred into the internal representation. In LIDA, specialized *codelets* scan the input and activate concepts if conditions are matching. Equivalent, a deep neural network could extract objects from images and put in some visual buffer. We use the SAKE for this task. SAKE converts the raw signal data into states, i.e. fuzzifying the raw data into symbols that the system knows. In case of incoming data from ADAM, no extra activation of concepts happen here as ADAM puts already activated symbols directly into the communication. ADAM here acts as a sort of "telepathy" sensor, which is not common in cognitive architectures that mimic the human mind.

C. Create Goals: A common denominator of all cognitive architectures is that they use one or more explicitly declared goal states, which they want to reach, see, e.g. *desires* in BDI. In some architectures like SiMA, a *homeostasis* generates the goals, which are to restore a balance in the body. Other architectures like SOAR, use predefined top-goals. All actions of the system aim to reach the goal state. The concept is related to the reward in the domain of reinforcement learning. However, the value function used is not very flexible and needs to be entirely defined by the developer for a specific use case.

The explicit top goal of the ACO is to reduce the number of faulty pieces at minimal costs. The value function of CODE defines what it shall do with recognized faulty states. However, different from symbol processing architectures like SOAR and BDI, the goals of how to get to the end goal are implicit. Further, the epsilon greedy strategy also replaces a part of the goal concepts. A cognitive architecture would also strive to accumulate knowledge about unknown, or poorly known aspects of the world. Epsilon greedy does the same be picking randomly between established and random actions. A typical epsilon greedy algorithm, however, bases the tradeoff between exploration and exploitation on the length of the simulation sequence and thereby simplifies the directed aspect of exploration away in favour of a purely statistical approach.

D. Activate Beliefs: While step B only activates symbols of perception based on, perceivable, external inputs, step D infers beliefs about the world. It is additional knowledge that the system uses for defining options. This knowledge is usually stored in a long-term semantic- or episodic memory. It makes implicit knowledge explicit to the system, e.g. adding relative positions to the belief buffer in ICARUS or BDI. As SAKE infers states from processed raw data, it also adds information if the current state was an error state or not. It compares the values of a state with the predefined value tolerances. Additional information about drifts or causes of error in the data that is inferred is the beliefs of the system.

E. Propose Intentions: Intentions or options are generated based on beliefs and goals. It defines what the system can do. In BDI, they are called intentions. All analyzed, cognitive architectures require that there are no fixed programmed paths, but that multiple dynamically created ways of reaching goals can be instantiated. Intentions are "competing" for being selected.

In CODE, decision-making takes place. As it uses reinforced learning, the tables used to store the probabilities and expected rewards for reinforcement learning can be considered simplified proto-concepts for planning and decision making. A very fundamental difference here is the lack of entry points for advance moment evaluations. The tables only store averaged results for experienced probabilities and rewards, and do not allow for, for example, situation-specific evaluations of the memorized moments. For example, the expected reward for being in the kitchen will be very different if you plan to cook, in comparison to when you plan to bath. Reinforcement learning methods attempt to rectify this in Deep Learning setups by generating additional states for these specific situations, e.g. having two states "kitchen-batching" and "kitchen-cooking" with different values. However, these workarounds are still purely statistical and cannot, yet, reflect semantic knowledge about the world, or consider set of parallel long- or short-term plans.

E. Propose Actions: Usually, the intentions are the high-level means of reaching some goal, while the action the explicit next actuator execution to alter the external world or the internal memory. For each intention, there shall be at least one action that can be executed. In reasoning architectures like ICARUS, the process of finding actions is done by reasoning down from a situation to an executable action that can be applied.

In the previous paragraph, we discussed the option handling in CODE. As reinforcement learning is used, the actions selection is based on the Q values that are learned over time, for each state. The Q values for each proposed action then represent the sum of rewards expected when following this action.

G. Evaluate options: All architectures that have competing intentions must compare them to each other to be able to select one of them. Decision-making, therefore, evaluates them regarding effort and the possibility to satisfy a goal. An example is preferences in SOAR, which tell the best of two options. The reinforcement learning concept of CODE uses the Q-values as a method of comparing options. Due to the statistical mechanics underlying reinforcement learning, the Q-values already combine any expected short-term and long-term reward for a given action.

H. Select Option: Intentions are ranked, and then the highest one is selected. If attentional mechanisms are applied, they require a second cycle (cycle 2) of the cognitive process. LIDA is such an example. Attention codelets choose some intention, which triggers the second cycle to generate action proposals for that intention only. CODE selects its action by

selecting the action with the highest Q-value. No attention mechanism is implemented that sort out intentions in the preprocessing.

I. Execute Action: The system executes the selected action. As mentioned before, internal as well as external actions can be proposed. In reactive systems, there are no internal actions as a trigger immediately produces a response. BDI works in the manner. In deliberative systems like SOAR or SiMA, internal actions can be proposed that just alter the state of an internal memory. In our case, no internal actions exist, as no explicit memory is used to keep track of the state. CODE sends a command to the OM or ADAM, depending on the destination.

J. Act: The driver of OM translates the action command into machine code, or ADAM sends the message to the addressed ACO.

B. Discussion

The use case-based architecture ACO was designed based on loosely coupled modules, of which each module fulfils its part of the use case requirements. It consists of four modules, which were independently defined by three different research groups. Additionally, the modules were supposed to be working for various subproblems autonomously. In that perspective, ACO is a union of modules, which work together. However, as analyzed previously and in Tab. 1. It provides all necessary functionality to fulfil the cognitive process. Therefore, we consider it to be a use case-based cognitive architecture. As it is a mixture of different concepts, it is clear that the ACO architecture cannot fulfil a cognitive theory as expected from the analysis in [5] and [7]. However, if the ACO would be derived from a particular architecture in a top-down manner, which one would we chose? In Tab. 1, we match BDI, SOAR, and LIDA to the cognitive process. Each architecture represents a level of comprehensiveness. BDI is the architecture with the simplest structure and LIDA, the most comprehensive one. We compared the architecture functionalities to find out which one is the most similar to our architecture.

The simplicity of BDI might be a reason why BDI is popular in multi-agent systems. It is simple, does not use a unique way of processing data and is therefore very general. Therefore, it would require many extensions and only serve as a framework. On the other side, LIDA, which has a complex activation system for percepts, would force us to make comprehensive workarounds to apply the SAKE functionality. Hypothetically, SOAR could be applied instead of CODE. However, it would require much fine-tuning of rules to get to the right conclusions. It turns out that our architecture seems to have most in common with BDI. BDI is simple enough to be extendible with our functions.

Cognitive design paradigms seem to introduce additional complexity and higher demands on data handling for very little to no immediate gain. In cases where multiple goals exist and warrant a cognitive attempt, the use case underlying the design should be changed to be compatible with cognitive approaches. For instance, instead of having a use case describing how to optimizing a machine, the use case should develop around an autonomous agent and its interactions with a machine.

In factory control, determinism is desired. The cognitive architecture SAVE is designed to be deterministic up to the extent of error detection and correction, simply by the fact that it will only react to problems. In this regard, the chance of false positives, i.e. taking unnecessary corrective action, can be kept reasonably low by choosing the conditions carefully. Furthermore, a cognitive system sit's "above" a fundamental control system. It only steps in when its confidence is high enough. It also means that we can still apply various fallback rules that prevent too drastic behaviour. For instance, a simple rule can ensure that we do not change a drill three times within 24 hours.

TABLE I. COMPARISON TO THE COGNITIVE PROCESS WITH LABELS OF MATCHING CONCEPTS

Process Step	**Architecture**			
	BDI [20]	***SOAR [17]***	***LIDA [18]***	***ACO***
A. Read Inputs	Yes	Yes	Yes	Yes, OM and ADAM
B. Activate Percepts	No, no explicit symbolization of perceived objects	Yes, symbolization to working memory from an input	Yes, sensory module perception codelets activate percepts	No, like BDI
C. Create goals	Yes, desires generated from beliefs, use cases	Yes, fixed top goal(s) and sub-goaling	Yes, percepts and memories activate emotions	Yes, like BDI and through greedy epsilon
D. Activate Beliefs	Yes, beliefs generated directly from raw data	Yes, state elaborations by firing rules performed	Yes, percepts activate content by association	Yes, like BDI
E. Propose Intentions	Yes, intentions derived from desires and beliefs	Yes, operators by firing rules proposed	Yes, activated and attached attention codelets	Yes, implicit through reinforcement learning
F: Propose actions	Yes, task handler/planner Manager (after step H)	Yes, plan instances activated by percepts	Yes, in cycle 2, by winning coalition	Yes, implicit through reinforcement learning
G. Evaluate Options	Yes, importance values applied to intentions	Yes, preferences compare proposed operators	Yes, highest evaluated attention codelet	Yes, through Q values from reinforcement learning
H. Select Option	Yes, select the highest evaluated intention	Yes, select the preferred operator	Yes, cycle 1: select intention, cycle 2: select action	Yes, through the highest Q value
I. Execute Action	Yes	Yes, the operator writes to working memory or external action	Yes	Yes, through OM or ADAM
J. Act	Yes	Yes	Yes	Yes
Cycle 2: E-Propose intentions: Attention	No, an optional feature	No, no attentional mechanism used	Yes, winning coalition of attention codelets can create actions	No, no attentional mechanism proposed

VI. Conclusion

Initially, we wanted to design a cognitive architecture for health monitoring tasks in the production industry. We got into the dilemma to apply existing cognitive architectures, which would cause a massive implementation overhead or to design an architecture based on our use case that would not follow any cognitive theory. As the application was our priority, we designed an architecture based on a use case of a factory production line.

We analyzed how well our architecture would fit into a cognitive architecture schema. We concluded that although all single components fulfil the requirements of a cognitive architecture, our architecture cannot be easily implemented by using an existing architecture as it would demand a holistic concept. Therefore, if we would give it a try, we would have to select the most straightforward possible architecture that we analyzed: BDI.

The next steps will be to implement and test the SAVE architecture within a multi-agent system simulation. We need to see if our cognitive concepts help us achieving the flexibility we were hoping to achieve. In general, we only consider such an attempt promising when having multiple goals, beyond the immediate solution to a technical problem.

Acknowledgment

This work is performed within the project SAVE (FFG 864883). This project is co-funded by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) under the programs "Production of the Future".

References

[1] Bratukhin, A. and Sauter, T., "Functional Analysis of Manufacturing Execution System Distribution", in IEEE Transactions on Industrial Informatics, vol. 7, no. 4, pp. 740-749, Nov. 2011.

[2] Liu, Huaxin, "A dynamic bottleneck-oriented manufacturing control system", Vol. 13. GITO mbH Verlag, 2011.

[3] Mijailovic, V., "Probabilistic method for planning of maintenance activities of substation components", Electr. Power Syst. Res., vol. 64, no. 1, pp. 53–58, 2003.

[4] Iarovyi, S., Lastra, J. L. M., Haber, R., and del Toro, R., "From artificial cognitive systems and open architectures to cognitive manufacturing systems", in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), IEEE, pp. 1225-1232, 2015.

[5] Wendt, A., Kollmann, S., Siafara, L., and Biletskiy, Y., "Usage of cognitive architectures in the development of industrial applications", in proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018, 2018.

[6] Kotseruba, I., Gonzalez, O. J. A., and Tsotsos, J. K.: "A review of 40 years of cognitive architecture research", Focus on perception, attention, learning and applications. arXiv preprint arXiv:1610.08602, 1-74, 2016.

[7] Vernon, D., "Two ways (not) to design a cognitive architecture". Cognitive Robot Architectures, p. 42. 2017

[8] Mills, K., "What Is Cognitive Manufacturing?" Metrology.news, 16 January 2019.

[9] Hu, Long, et al., "iRobot-Factory: An intelligent robot factory based on cognitive manufacturing and edge computing", Future Generation Computer Systems 90 (2019): pp. 569-577, 2019.

[10] Zhao, Yaoyao F., and Xun X., "Enabling cognitive manufacturing through automated on-machine measurement planning and feedback", Advanced Engineering Informatics 24.3, 2010, pp. 269-284.

[11] Maier, Paul, et al., "Automated plan assessment in cognitive manufacturing", Advanced Engineering Informatics 24.3, 2010, pp. 308-319.

[12] Lieto, A., Bhatt, M., Oltramari, A., & Vernon, D., "The role of cognitive architectures in general artificial intelligence", 2018.

[13] Dietrich, D., and Zucker, G., "New approach for controlling complex processes: An introduction to the 5th generation of AI", in 2008 Conference on Human System Interactions (pp. 12-17). IEEE. 2008.

[14] Vernon, D., Metta, G., and Sandini, G., "A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents", IEEE transactions on evolutionary computation, 11(2), 151-180. 2007.

[15] Ritter, F. E., Tehranchi, F., and Oury, J. D., "ACT□R: A cognitive architecture for modeling cognition". Wiley Interdisciplinary Reviews: Cognitive Science, 10(3), e1488. 2019.

[16] Choi, D., and Langley, P., "Evolution of the ICARUS cognitive architecture", Cognitive Systems Research, 48, 25-38. 2018.

[17] Nuxoll, A. M., and Laird, J. E., "Enhancing intelligent agents with episodic memory", Cognitive Systems Research, 17, 34-48. 2012

[18] Madl, T., Franklin, S., Chen, K., Montaldi, D., and Trappl, R., "Towards real-world capable spatial memory in the LIDA cognitive architecture". Biologically Inspired Cognitive Architectures, 16, 87-104. 2016

[19] Schaat, S., Wendt, A., Kollmann, S., Gelbard, F., and Jakubec, M., "Interdisciplinary Development and Evaluation of Cognitive Architectures Exemplified with the SiMA Approach", In EAPCogSci. 2015.

[20] Sinclair, J., and Lee, I., "A generic cognitive architecture framework with personality and emotions for crowd simulation". In 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE) (pp. 1-6). IEEE. 2017.

[21] Götzinger, M., TaheriNejad, N., Kholerdi, H. A., Jantsch, A., Willegger, E., Glatzl, Rahmani, A. M., Sauter, T., Liljeberg, P., „Model-free condition monitoring with confidence", International Journal of Computer Integrated Manufacturing, 32(4-5), 466-481, 2019.

[22] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Hassabis, D., "Human-level control through deep reinforcement learning", Nature, 518(7540), pp. 529–533. https://doi.org/10.1038/nature14236, 2015.

[23] Hochreiter, S., & Schmidhuber, J., "Long Short-Term Memory. Neural Computation", 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735. 1997