Usage of Cognitive Architectures in the Development of Industrial Applications

Utilization of a General Cognitive Process in the Domain Building Automation

Alexander Wendt[1], Stefan Kollmann[1], Lydia Siafara[1] and Yevgen Biletskiy[2]
[1]*Institute of Computer Technology, TU Vienna, Vienna, Austria*
[2]*Department of Electrical and Computer Engineering, University of New Brunswick, New Brunswick, Canada*
{alexander.wendt, stefan.kollmann, lydia.siafara}@tuwien.ac.at, biletski@unb.ca

Keywords: Cognitive Architecture, AI, Building Automation, Decision-making, KORE, Industrial Application, ACONA, Control Strategy.

Abstract: Cognitive architectures, which originate from the field of Artificial Intelligence, implement models for problem-solving and decision-making. These architectures have a wide room for implementation in industrial applications. The goal is to adapt a cognitive architecture to the demands of an application in the area of building automation. It is analyzed, why cognitive architectures are difficult to apply in industrial domain. The result of the analysis is a cognitive process, which is applied to an application in the building automation domain. The use of the architectures is demonstrated within a Java-based based middleware. There, the cognitive architecture is applied for the automatic generation and improvement of control strategies in building automation, which have the goal to minimize energy consumption with minimal reduction of the comfort.

1 INTRODUCTION

Cognitive architectures provide a general framework for developing computational decision-making applications and are often, but not necessarily, based on theories of the human mind (Langley et al., 2009). Autonomous decision-making ability is demanded in the context of the growing complexity of industrial applications. Therefore, they have a potential to contribute to such applications. Unfortunately, up to now, the few examples of industrial applications. (Kotseruba et al., 2016) raise the question whether cognitive architectures are suitable to apply for software development besides of experiments. This problem is addressed by proposing an approach to enhance the systematic application of cognitive architectures in the field of industrial systems.

As a method, we initially review well-known examples of cognitive architectures and discuss their functionality and usage in industrial applications. Then, we specify the types of software applications where cognitive architectures fit into and identify problems that may emerge during the adaptation of the architectures to a certain application. Based on this analysis, we propose in the last part of the paper our solution that consists of a cognitive process, which is common for all studied architectures. The process is implemented as an architecture. Finally, the functionality of the architecture is demonstrated within the project KORE (Cognitive Optimization of Control Strategies for Increasing Energy-efficiency in Buildings) (Zucker et al., 2016). KORE is applied in the domain of building automation, which has the purpose to optimize energy consumption under the constraints of comfort.

2 EXISTING APPLICATIONS OF COGNITIVE ARCHITECTURES

Cognition, according to Vernon et al., "can be viewed as the process by which the system achieves robust, adaptive, anticipatory, autonomous behaviour, entailing perception and action" (Vernon et al., 2007). It implies that the cognitive system is able not only to understand the current situation but also to function efficiently in situations for which it was not intended.

Among cognitive architectures, SOAR (State Operator and Result) (Langley et al., 2009) and LIDA (Learning Intelligent Distribution Agent)

(Ramamurthy et al., 2006) are prominent examples with different origins. While SOAR origins from the domain of logical problem solvers in classical artificial intelligence, LIDA tries to model the human mind and origins from neurological theories. Each cognitive architecture has its advantages and drawbacks.

SOAR is a general-purpose architecture that implements cognitive functionality and defines system behaviours by rules. LIDA is a cognitive architecture that aims to model the human mind. It provides a framework for cognitive architectures, where modules can be arbitrary created and linked. LIDA uses a partly bottom-up approach, where activated content or ideas of what to do are competing for attention. The winning content receives the attention and gets its action developed and executed. The cognitive architecture ICARUS (Langley et al., 2011) origins from the area of autonomous robots. It differs from SOAR as it uses several different memories to store skills, concepts and beliefs.

The cognitive architecture BDI (Belief, Desire, Intention) (Gottifredi et al., 2008) adds the component of a desire to cognitive architectures, where desires represent the goals of the system. SiMA (Simulation of the Mental Apparatus & Applications) (Schaat et al., 2017) extends the desires further into drives, emotions and feelings, which are used as evaluation mechanisms of different options of the system. Multi-agent solutions have gained attention within the community due to their ability to scale and allow partitioned development. An approach is the ACNF Cognitive Framework (Crowder et al., 2014).

SOAR and BDI (Gottifredi et al., 2008) have been applied as the decision-making in robots. ICARUS (Choi et al., 2009), BDI (Dignum et al., 2009), LIDA (Sandsmark and Viktil, 2012) and SiMA (Schaat et al., 2017) have been applied to games or simulations of virtual human-like actors. The agent TAC-Air-Soar (Heinze et al., 1999) shows the potential of cognitive architectures as virtual pilots in the modelling of fighter pilots in air combat scenarios. Since the theory of cognitive architectures often origins from psychology, some of them are used to mimic human behaviour in psychological experiments (Anderson et al., 2004), (Wendt et al., 2015), (Gobet and Lane, 2010). In addition, there exist real-world applications, where the predecessor of LIDA has been deployed. In the US Navy, it manages jobs for sailors, where the task is to offer jobs for sailors depending on the sailor's preferences, the Navy's policies, the needs of the tasks and the urgency (Franklin and Patterson Jr, 2006).

3 ANALYSIS OF APPLICATIONS

Cognitive architectures tend to be more suitable for particular application classes. The criteria for such applications are analyzed in the following.

3.1 Suitable Applications

As the human mind is claimed to be the most complex biological system that we know about, it would be expected that the same decision process would benefit industrial applications that are applied in complex environments (Dietrich and Zucker, 2008). These applications do not have access to all information about their environments and have to make decisions based on judgment instead of deterministic inputs.

Current applications can be categorized into two main groups: controllers for physical robots and virtual human in simulations. The domains are close to the area of the human mind. An application that differs from the others is the LIDA sailor application. Their common denominator is that they have to select one action out of several possible, comparable and competing actions, to fulfil certain goals. Due to the risk of applying a massive overhead, applications that operate only with complicated problems, where the environment is completely known are therefore not appropriate.

Because decisions in these applications are based on judgment and not on determinism, the evaluation of options plays a major role. The program logic together with stored data determines how to evaluate option. Compared to a straightforward coded program, a cognitive architecture has the advantage that a lot of necessary program logic is transferred from the code into knowledge. Due to the provided infrastructure, it can be claimed that if a cognitive architecture is used in a certain complexity of an environment, the implementation should be possible with less effort than using a direct implementation of a state machine.

The claim can be understood in the following context: Suppose that a game like Pacman is developed. The goal is for a player to eat food and avoid the ghosts. In the simplest case, decision-making consists of some rules that define the behaviour for each situation. The more inputs are available, the more rules have to be written. At the point of competing options, the code gets messy. Evaluation of each option regarding some criteria is necessary. Here, a cognitive architecture makes sense. If such a system is extended, only to the way options are created and evaluated has to be addressed.

3.2 Problems with Common Cognitive Architectures

Although cognitive architectures seem to be very useful, most of them have never left the laboratory (Kotseruba et al., 2016). The importance of extracting the correct problem is given through an example in the project VKT GOEPL (Wendt et al., 2012). The purpose was to develop a decision support system for the collaboration between agencies to protect critical infrastructure in case of an earthquake. It should answer queries like "how many hospital beds are available within 10 km radius from the epicentre?" A cognitive architecture only make sense, if there are several competing methods to answer the question.

Another problem is that although cognitive architectures claim to be very general, they tend to be highly tailored to a certain problem. As an industrial application is often very specific, the cognitive architecture must not be too specialized, in order to cover the required functionality. For instance, SiMA models the human mind with high detail according to a model derived from psychoanalysis. Compared to a generic architecture like SOAR, SiMA contains much pre-programmed functionality. In the project ECABA (Zucker et al., 2016), the idea was to apply SiMA with minimal changes to a problem in automated building control. The SiMA model assumes that a drive is independent. Because the proposed drives of the building controller are interdependent, a workaround with bad benefit/cost ratio had to be used.

SOAR and BDI use a minimal cognitive cycle. If an industrial application has a need for an attentional functionality, which filters relevant from non-relevant content like in SiMA or LIDA, it may not be possible to use these architectures because an attentional mechanism is not a part of their concepts. Perhaps, it is possible to implement this functionality with high effort. General-purpose architectures, which are more general problem solvers often lack the flexibility needed for a certain application.

4 THE COGNITIVE PROCESS

To be able to use the potential of cognitive systems in industrial applications, the shortcomings described previously have to be addressed. The method proposed in this paper is to create a meta-architecture that consists of a common cognitive process, which executes customized functions. According to (Wendt, 2016), a general cognitive process can be extracted and common cognitive architectures can be mapped to it. In this paper, the idea is to use a modified version of that process.

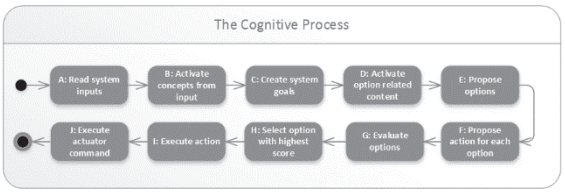

Figure 1: The cognitive process.

Figure 1 shows an overview. The cognitive process describes one cognitive cycle, i.e. the path from input to an action. In the following, each step is described:

The first step is to "perceive" the input data (*A: Read system input*), which can be a user request in an application. It corresponds to the neural layer in SiMA that contains raw data. Sensor data is transformed into the internal representation (*B: Activate Concepts from Input)*. A knowledge base is used to classify the data and to load the matching symbol. No additional reasoning is performed. It corresponds to the Perceptual Associative Memory of LIDA or the Perceptual Buffer in ICARUS.

Then, the activated symbols are enhanced with inferred knowledge. System goals (*C: Create System Goals*) are activated by the sensor content if they are not predefined. In addition, inferred knowledge about the environment is activated (*D: Activate Option Related Content*). Belief templates are tested and beliefs are instantiated in a working memory. Implicit knowledge is made explicit. In SiMA, the system goals are drives, which rely on sensor data from its body. In BDI, desires are tested against the beliefs that originated from the sensor data (steps C and D are swapped).

Based on the beliefs and the goals, ways of fulfilling the system goals are proposed (*E: Propose options*). These options define what the system is able to do. They may contain possible actions that the system can execute. In some applications, this step may be optional if the options are equal to the actions (*F: Propose Action for each Option*). Options can also be interpreted as directives that can be fulfilled by actions. In SOAR, operators and in BDI, intentions are proposed. In LIDA, the options are presented through coalitions of attention codelets with the beliefs. All cognitive architectures have some means to evaluate the proposed options, in order to rank them (*G: Evaluate Options*). There are two sorts of evaluations: Degree of goal fulfilment and the effort. In SiMA, a rich set of evaluation methods is used. Options are evaluated against the drives, the feelings

and the effort. Other methods used by e.g. SOAR is the usage of preferences for certain operators like "operator1 is better than operator2".

Through evaluation, options receive a score. One option is selected based on its score (*H: Select Option with Highest Score*). In the architectures SiMA and LIDA, a second cognitive process would start to develop a plan for each option. An action that is associated with the option is executed (*I: Execute Action*). It can be an action that alters the state of the environment or an action that alters the internal state of the system itself. In SOAR, that is what the operators are doing. They only alter the internal state of the working memory. An external action is transformed into actuator commands, in order to make a change in the system environment (*J: Execute Actuator Command*).

Architectures like SOAR and BDI execute the described process once, while due to an attentional mechanism SiMA and LIDA executes it twice. The winning option is further developed into detailed plans.

5 TRANSFORMATION INTO A COGNITIVE ARCHITECTURE

In case of a reactive system, the cognitive process can be implemented straightforward. In the general case, however, a *deliberative system* is applied, which needs multiple cognitive cycles to decide about an action. Figure 2 shows the architecture.

In most cognitive architectures, the execution of actions is sequential. While an *external action* is an action that alters the state of the environment, an *internal action* only alters the internal state. For instance, if data is loaded from a long-term memory, it does only change the internal state. It may be necessary to execute multiple internal actions before an external action is executed. The key to handle this is to keep track of the system's own decisions in a working memory to know what has already been done in a sequence of actions.

The system needs functions. In LIDA, the concept of codelets was introduced. A codelet is a small piece of code that executes independently on the content of the working memory, e.g. to test sensor data and activate an internal representation in the Perceptual Associative Memory. Inspired by LIDA, codelets will be implemented as the functions of the system. They wait for a trigger to start. All codelets are assigned a process step in Figure 1. The idea is that instead of having fixed functions in the architecture, every function is a codelet that can be added or removed, in order to allow complete customization of the cognitive process.

Every architecture needs memories. The long-term memory can be in any format, e.g. an ontology or a relational database, depending on the purpose of the system. Through the codelets, its content is loaded and converted into the internal representation. The internal representation is defined in two memories: the working memory and the internal state memory. In the *working memory*, all content, which is relevant for the current situation is stored, similar to SOAR. It keeps actual instances of input data, data from the long-term memory and data, which is generated through codelets. In the *internal state memory*, only decision-making relevant data like goals and options is kept. It makes sense to separate the memories as one of them only handles meta-data, which is linked to the real data.

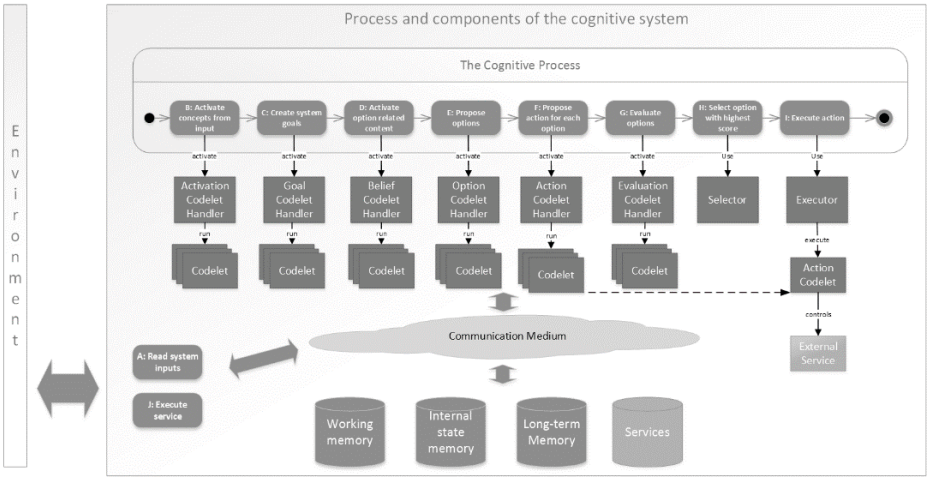

Figure 2: The general cognitive architecture that implements the cognitive process.

6 THE APPLICATION KORE

Building Energy Management Systems (BEMS) monitor and control the operation of the building systems to maintain acceptable indoor comfort levels under the constraint of energy efficiency. For the control of these systems, automated decisions are made using a control logic, which consists of a set of rules defined by an expert. These rules are static or updated during a re-commissioning phase. To reduce the engineering effort, the focus in the project KORE (Cognitive Optimization of Control Strategies for Increasing Energy-efficiency in Buildings) (Zucker et al., 2016) is to automatically generate and propose rule-sets to the building operator.

A *rule-set* is defined as a parameterized set of interconnected control blocks like the example in Figure 3. A *control block* is function of the building, e.g. an actuator for a heating element or a CO_2 sensor. Defined control blocks are instantiated in Matlab Simulink. A Simulink model is then used to simulate the rule-sets within a virtual building and get feedback on how well they performed.

The task of the KORE application is to automatically generate rule-sets, test them, evaluate them and decide about the best method to continue the optimization process. The system consists of three components: A cognitive system for rule-set generation, a simulator to test the generated rule-sets and an ontology to store test results as well as building information. The ontology is the long-term memory of the cognitive system. An algorithm inside of the cognitive system does the rule-set generation. It arranges predefined control blocks corresponding to a problem definition, which are later parameterized.

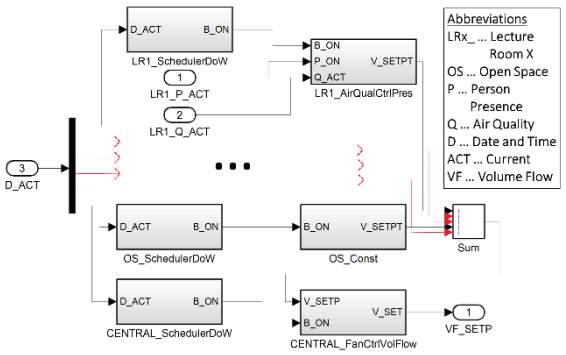

Figure 3: Example of a rule structure in KORE.

There are three parts of information used. The *problem definition* consists of the building structure, the environmental setup, e.g., the season of the year to be tested, and user requirements regarding comfort and energy. The problem definition is stored in an ontology, which has been created by domain experts. The *available utilities* include a collection of available control blocks and semantic knowledge. The *solution space* consists of rules, generated by interconnecting and parameterizing the available control blocks. A further concept used is the *episode*. It is the evaluation of a particular generated and tested rule structure and parameters. Each episode is evaluated regarding the fulfilment of the system goals, i.e., energy efficiency, comfort and penalty that describes the fulfilment of external rules applied to the system.

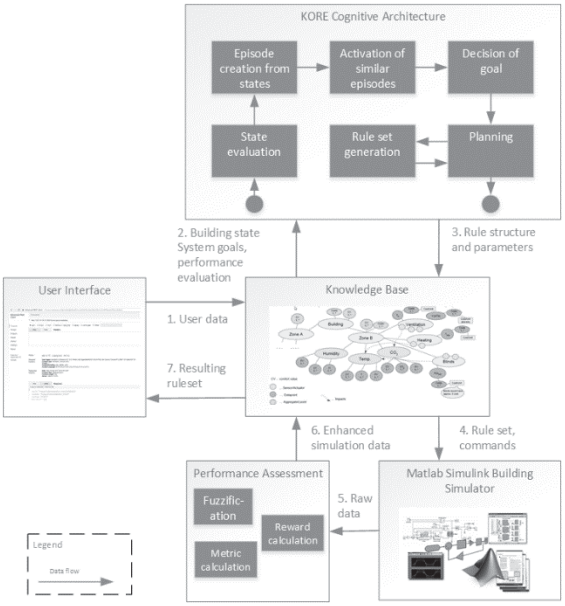

Figure 4: The process of the KORE Application.

The process of the KORE application is visualized in Figure 4. The process steps are described with the numbers 1 to 7. It starts with a user request, which contains the problem description address, the season to optimize and the evaluation criteria (1 in Figure 4). The problem description is enriched with information stored in the knowledge base and is sent as input to the cognitive system (2). First, the system retrieves episodes from similar problems to find matching rule-sets, using case-based reasoning. Rule-sets that resulted in episodes with high returns have higher probabilities of being selected by the system as potential solutions in the future. The cognitive system provides options to start rule generation from scratch or to vary parameters of existing episodes. The generated rule-structure (3) is sent to the building simulator (4). It is returned as raw data to an evaluator

that adds the evaluations as meta-data (5). The episode is stored in the ontology (6). Then, the process starts over again or returns to the user if a rule-set satisfies the input conditions (2), (7).

7 IMPLEMENTATION OF KORE

he cognitive process is to the architecture in Figure 4. The main problem is divided into subsystems. They are marked with a dark colour in Figure 5. Each subsystem is defined as a separate cognitive problem. In this paper, one exemplary subsystem will be presented to show how a cognitive process is applied outside of the standard uses like robots and artificial life simulators.

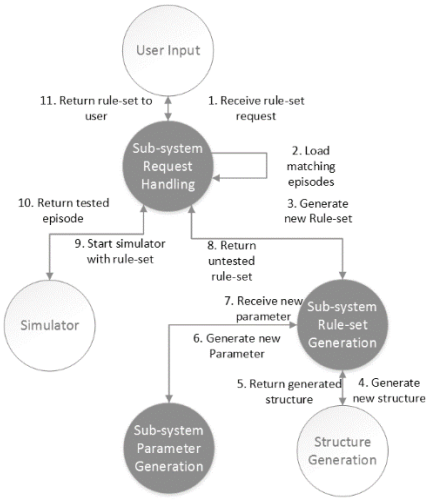

Figure 5: Subsystems of the KORE application.

7.1 Subsystem Request Handling

When a user request is received by the subsystem <Subsystem Request Handling>, the system must select among existing episodes. The fulfilment of the request by highest evaluated episode is the goal of the system. In case the goal is not fulfilled, the second best alternative is selected: to generate a new rule-set.

In Table 1, goals, options and actions of the subsystem <Request Handling> are listed. Goals is the fulfillment of a request, where <request interrupted> is the more important goal. Actions are predefined. The options are matched with the actions. For each activated episode, an option is generated <episode 1..n> that is connected to the action <return episode>. In addition, an option is generated to generate a new rule-set <new rule-set> with the action <generate rule-set>.

Table 1: Overview of the subsystem Request Handling.

Goals	Options	Actions
request fullfilled	episode 1..n	return episode
request interrupted	new rule-set	generate rule-set
	untested rule-set	test rule-set
	interruption	

Generated, untested rule-sets return to the <Request Handling> as <untested rule-set> connected to the action <test rule-set>.

7.2 Software Implementation

In (Wendt and Sauter, 2016), the ACONA framework for implementing cognitive architectures in the Java-based multi-agent platform Jade was presented. On the lowest level, Jade agents are located. To allow synchronous calls like reading a value from another agent while blocking the method, having multiple behaviours running in parallel and to get more control over the external communication, the ACONA framework adds a layer on top of Java Jade.

ACONA introduces cell functions, which allow remote procedure calls for functions in other agents and the communication is completely separated from the function logic. Each agent also receives a memory with the structure of datapoints for Json strings, in order to provide a flexible internal representation.

Processes are implemented as codelet handlers, where a *codelet handler* is an engine, which runs codelets. In Figure 2, the setup is shown. The top process is a codelet handler, where the processes steps of the cognitive process are also codelet handlers. Each process step allows customized codelets to be executed. With this software setup, a flexible and highly customizable cognitive system has been designed.

7.3 Test Results

A request is provided through a RESTful web service to the <Request Handling Subsystem>. A request consists of CO_2, energy and penalty requirements. The system is demonstrated with two requests: <Request 1> and <Request 2>. The goal conditions to fulfil is an evaluation in the range [0, 1] of CO_2, energy and penalty, In <Request 1>, all conditions are set to 0.9. In <Request 2>, they are 0.5. In Figure 6, dashed lines show the request conditions.

For beliefs, a memory loader is triggered that loads only the metadata of episodes, which perfectly matches the scenario to be tested. Three episodes match the two requests. Their evaluations are shown

with filled lines in Figure 6. After option codelets have generated one option for each activated episode and an additional option to generate a new rule-set, evaluation codelets look for goal conditions and the evaluations in the episodes and add them to the total evaluation of that particular option. Each option has a current state and a next state. Action codelets match each option with a precondition state and a postcondition state and adds a proposed action to the matching option.

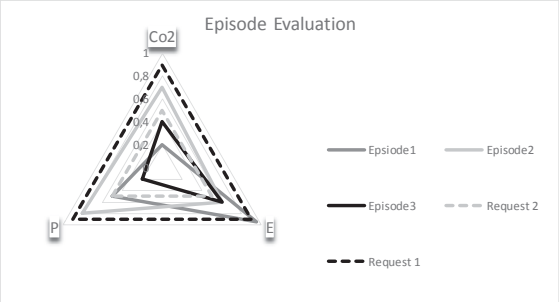

Figure 6: Evaluation of episodes regarding CO2, energy and penalty criteria.

After evaluation, the option with the highest score is selected. The results have been visualized in Figure 7. For <Request 1>, the option <GenerateRule> was selected, which means that no loaded episode fulfilled the requirements of the request. All evaluations are negative because they do not match the requirements. Therefore, a new rule-set and episode have to be generated. However, an interesting effect occurs if the possibility to generate new rules is removed. The system always execute the action of the best option and in such a case an episode would be returned although it does not fulfill the requirement.

In <Request 2>, the requirements were lowered to 0.5 each and the option <OptionEpisode2> was selected as it had the best rank. The corresponding action was to return the rule-set of that episode to the user.

Figure 7: Evaluation of available options for two requests.

8 DISCUSSION

A general cognitive process was extracted based on the analysis of common cognitive architectures. It should make it easier to apply cognitive architectures in industrial applications. The most important task is to find the suitable problem, i.e. problems, where a system has several competing options to choose from. The cognitive architecture has to be very flexible to be applied to industrial applications specialized for only one task. The more specialized a cognitive architecture gets, the harder it is to implement without violating the underlying cognitive model. The cognitive process was turned into a general cognitive architecture that allows more customization as all functions are defined as codelets. Codelets enhance and modify existing concepts in the working memory in a deliberative way.

The implementation shows that an architecture can be quickly setup and extended through codelets. Another advantage is that every codelet can be separately tested by unit tests, as the system is "open" to injections into the working memory and the internal state memory. It is easy to integrate new actions or option types with low effort.

A drawback noticed, which is common for all rule-based cognitive architectures, is that with increasing possibilities, the complexity of the system rises because codelets are generally interdependent. For instance, if new ways of evaluations are added, perhaps it makes the system unbalanced, which results in selecting the "wrong" option.

As future work, the architecture will be adapted and applied in the area of Industry 4.0 within the project Self-Aware health Monitoring and Bio-inspired coordination for distributed Automation systems (SAMBA). Apart from the decision-making module, the architecture includes other two modules for error detection and communication with other agents for distributed decision-making. The challenge here is to adapt the system in a distributed environment, therefore to exhibit collective behaviour that will be able to pursue the goals of the larger system. However, because the cognitive process is general enough, the effort for transformation in another domain is expected to be kept low.

As the common cognitive cycle can be extracted from the studied cognitive architectures, a perfect validation would be to implement an existing cognitive architecture with all specialized functions. Besides, of the cognitive process, the key to implementing an architecture would be to create the proper state machine, which is correctly represented

within the internal state memory. SiMA, which has much specialized cognitive functionality would be suitable for such a test.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support provided to us by the BMVIT and FFG (Austrian Research Promotion Agency) under the KORE project (848805).

REFERENCES

Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y., 2004. An integrated theory of the mind. *Psychol. Rev. 111*, 1036.

Choi, D., Kang, Y., Lim, H., You, B.-J., 2009. Knowledge-based control of a humanoid robot, in: Intelligent Robots and Systems, 2009. IROS 2009. *IEEE/RSJ International Conference On. IEEE*, pp. 3949–3954.

Crowder, J.A., Carbone, J.N., Friess, S.A., 2014. Artificial cognition architectures. *Springer*.

Dietrich, D., Zucker, G., 2008. New approach for controlling complex processes. An introduction to the 5 th generation of AI, in: *Human System Interactions, 2008 Conference On. IEEE*, pp. 12–17.

Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M., 2009. Games and agents: Designing intelligent gameplay. *Int. J. Comput. Games Technol*. 2009.

Franklin, S., Patterson Jr, F., 2006. The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, *software agent. pat 703*, 764–1004.

Gobet, F., Lane, P.C., 2010. The CHREST architecture of cognition: The role of perception in general intelligence, in: *Procs 3rd Conf on Artificial General Intelligence*. Atlantis Press.

Gottifredi, S., Tucat, M., Corbatta, D., García, A.J., Simari, G.R., 2008. A BDI architecture for high level robot deliberation, in: *XIV Congreso Argentino de Ciencias de La Computación*.

Heinze, C., Goss, S., Pearce, A., 1999. Plan recognition in military simulation: Incorporating machine learning with intelligent agents, in: *Proceedings of IJCAI-99 Workshop on Team Behaviour and Plan Recognition*. pp. 53–64.

Kotseruba, I., Gonzalez, O.J.A., Tsotsos, J.K., 2016. A Review of 40 Years of Cognitive Architecture Research: Focus on Perception, Attention, Learning and Applications. *ArXiv Prepr.* ArXiv161008602.

Langley, P., Choi, D., Trivedi, N., 2011. Icarus user's manual. *Inst. Study Learn. Expert.* 2164.

Langley, P., Laird, J.E., Rogers, S., 2009. Cognitive architectures: Research issues and challenges. *Cogn. Syst. Res. 10*, 141–160.

Ramamurthy, U., Baars, B.J., SK, D., Franklin, S., 2006. Lida: A working model of cognition.

Sandsmark, M.T.H., Viktil, K.B.M., 2012. Jantu: A Cognitive Agent Playing StarCraft: Brood War. Institutt for datateknikk og informasjonsvitenskap.

Schaat, S., Jager, W., Dickert, S., 2017. Psychologically Plausible Models in Agent-Based Simulations of Sustainable Behavior, in: Agent-Based Modeling of Sustainable Behaviors. *Springer*, pp. 1–25.

Vernon, D., Metta, G., Sandini, G., 2007. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Trans. Evol. Comput. 11*, 151–180.

Wendt, A., 2016. Experience-Based Decision-Making in a Cognitive Architecture. TU Vienna, *TU Vienna University Library.*

Wendt, A., Dönz, B., Mantler, S., Bruckner, D., 2012. Evaluation of Database Technologies for Usage in *Dynamic Data Models.* pp. 219–224.

Wendt, A., Gelbard, F., Fittner, M., Schaat, S., Jakubec, M., Brandstätter, C., Kollmann, S., 2015. Decision-making in the cognitive architecture SiMA, in: *Technologies and Applications of Artificial Intelligence (TAAI), 2015 Conference On. IEEE*, pp. 330–335.

Wendt, A., Sauter, T., 2016. Agent-Based cognitive architecture framework implementation of complex systems within a multi-agent framework, in: *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference On. IEEE*, pp. 1–4.

Zucker, G., Wendt, A., Siafara, L., Schaat, S., 2016. A cognitive architecture for building automation, in: *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE.* IEEE, pp. 6919–6924.