Agent-Based Cognitive Architecture Framework

Implementation of Complex Systems within a Multi-Agent Framework

Alexander Wendt
Institute of Computer Technology, TU Wien
Gußhausstraße 27-29 E384,
1040 Vienna, Austria
alexander.wendt@tuwien.ac.at

Thilo Sauter
Danube University Krems
Dr.-Karl-Dorrek-Str. 30
3500 Krems, Austria
thilo.sauter@donau-uni.ac.at

***Abstract*—Due to their increasing complexity, the implementation of automation systems faces more challenges. Cognitive architectures have been designed to deal exactly with that. The purpose of a cognitive architecture is to find an action to every possible sensor data to move the system closer to a defined goal. The method is to utilize stored knowledge to reason about the best solution. The challenge for the implementation of such an architecture is to manage the overwhelming complexity of functionality and data. This paper presents the Agent-based Cognitive Architecture (ACONA) Framework, which aims at providing a general infrastructure, allowing the implementation of various cognitive architectures. It considers the encapsulation of functionality and provides a flexible composition of building blocks by applying a multi-agent approach.**

Keywords—cognitive architecture; framework; acona; software architecture; multi-agent system;

I. INTRODUCTION

In the area of automation systems, there is an increasing need for software that can deal with environments of high complexity. An example can be found in the field of building automation. Buildings are the most energy intensive sector in Europe, and there is a significant potential to save energy [1] if automation systems are applied. However, building automation systems of today are rigid and tailored to their specific building. Therefore, in the project KORE [2], a cognitive architecture is being developed to be applied to a building automation system.

A cognitive architecture is a piece of software that can be compared to the "brain" of a robot. It is connected to sensor data, has a knowledge base and executes actions. Its purpose is to find an action to every possible sensor data. This action shall then move the system closer to one or more system goals [3]. Stored knowledge is utilized to achieve the goal. In that way, the behavior of the system is not hard-coded into programming logic, but into data, which can be trained, parameterized and modified.

There are a wealth of different cognitive architectures [3] with a different origin of design. Some of them, like the SiMA architecture [4] strictly try to model the human mind based on psychology and neurology. Other architectures like SOAR [5], are mainly based on problem-solving theories from classical artificial intelligence. However, they all have in common that they are complex systems consisting of several modules, which interact through system processes. Usually, there is one main system process, where information is passed serially and where other processes operate on the incoming information.

Because many cognitive architectures share similar functionality, the idea is to create a generic framework, based on which cognitive architectures can be implemented. The main challenge in the implementation is to build a system that can handle the complexity of the functions of a cognitive architecture. Therefore, this paper proposes the Agent-Based Cognitive Architecture Framework (ACONA). The novelty of the approach is the usage of only one basic agent type and the splitting of the framework into three module types: memories, services, and processes. In many cognitive architectures e.g. in ACT-R [9], functions and processes do not always show a clear separation. The added value is that there are clear interfaces between functions processing data and processes that execute functions. In that way, the order of the functions can be configured arbitrarily. Further, the cognitive architecture can be instantiated from these three module types

II. RELATED WORK

There is a large number of different cognitive architectures available. SOAR [5] and LIDA [6] are well-known examples. Some of the cognitive architectures have found potential applications, e.g., in the modelling of fighter pilots in air combat scenarios by the agent TAC-Air-Soar [7]. Cognitive architectures are also commonly implemented in robots [8]. Finally, also, real-world applications exist. The predecessor of LIDA is operational in the US Navy, where it distributes tasks to sailors depending on their preferences, the Navy's policies and urgency.

Each architecture is implemented in its specific way, which is adapted to the purpose. Production rule based cognitive architectures like SOAR [5] or ACT-R [9] usually are constructed around one or more memories, with a central rule engine that alters the state of the memories by firing the rules. Except for the memories and the rules, there is not much predefined functionality. The core architecture is implemented using a few loosely coupled modules that interact through the rule engine.

LIDA [6] and SiMA [4] are both very comprehensive architectures with a few memories and many functional modules that access the memories and transport data between them. Additionally, LIDA offers a software framework of

memory modules, data structures, and codelets. Codelets are almost agents, which run in the background. They activate concepts from a memory if certain conditions are fulfilled. Other architectures could use the LIDA framework. The modules are executed as threads through a scheduler. However, on a first try with SiMA, it was clear that the data structures were not compatible. Finally, the ACNF Cognitive Framework [10] realizes a complete cognitive architecture by using a multi-agent system. There, specialized agents act within a service-oriented architecture.

A central question in this paper is to find out which functionality the smallest building block in the framework needs. A suitable base for the smallest building block can be the widely used multi-agent-system JADE [11]. It allows programming individual agents with defined behavior. In JADE, the agents are registered in a container. There, they perform their tasks. The agents interact through the exchange of messages. However, the JADE framework has to be extended to be used in a cognitive architecture.

III. From a Complicated to a Complex System

The simplest cognitive architecture can be described by one process, according to the capture of Fig. 1. A process consists of several process steps, in which at least one function is executed. These functions either modify data in a memory or pass modified data to the next functions according to a predefined scheme. A cognitive cycle starts when sensor values are perceived and ends when an action has been selected. However, as more functionality is added to the system, it gets too difficult to represent it. A series of modules cannot represent the whole complexity of the system, especially if some functionalities are used more than once or if several parallel processes are running in the system. It gets complicated, and there is a high risk of implementing the software antipattern "Spaghetti code".

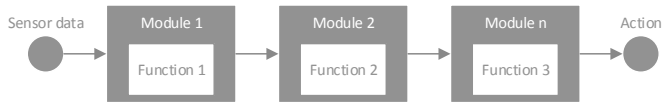

Fig. 1. Simplest cognitive architecture

The cognitive architecture LIDA [6] solves this problem by using memories for data exchange and codelets to implement specialized, hard-coded functionality. It means that no data is passed from function to function. Codelets are threads, which are running in certain intervals polling the memories and modifying the data there and passing the data to other memories. In that way, many processes can run in parallel as only the content of memories is checked. The memories and codelets are connected through a central configuration file. Memories, functions, and processes are clearly separated here. It has the advantage that functions can be easily replaced by just changing the process in the configuration file.

The ACNF Cognitive Framework [10] takes this even further and implements every functionality as an agent in a multi-agent system. Agents offer services, which are accessed by other agents. It provides excellent encapsulation of functionality and also dynamic access to services (different to LIDA).

IV. The ACONA Framework

Inspired by these architectures, the main building blocks for the ACONA framework are defined: memory modules, service modules, and process modules. It has the advantage that business logic is completely separated from system functionality, data structures, and communication. Processes call services and services alter data of the memories.

A. Memory Module

Memory modules are the databases of the system. Together, they represent the system state. The cognitive architecture can either use several short-term memory modules to represent different parts of the system, like ACT-R or use one single working memory like SOAR and KORE in Fig. 2. Additionally, memory modules can either just be access points to external databases. It is the case for long-term memories as storage of knowledge.

How the memories are programmed, depends on the design of the system. For instance, a working memory can be divided into several areas, which contain their kind of data. A reason to divide data into different areas of a working memory is that only the relevant area is searched through in a query. The memory module provides write, read and subscribe functions.

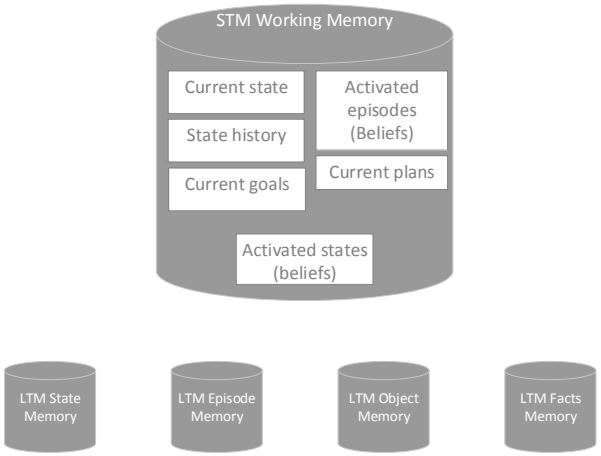

Fig. 2. Example of memory modules for the cognitive architecture KORE [2] (LTM=long-term memory; STM = short-term memory)

Finally, a memory module does not do anything itself. Data is only allowed to be modified by services. In that way, everything that happens in the system is triggered by a process module.

B. Service Module

Service modules are the functions of the system without any hierarchy. Similar to SOAR, they are the operators that alter data. Processes, which are represented as procedural knowledge, do not alter data. In SOAR, it is seen as a significant advantage that the production rules are separated from the operators because the system gains flexibility. In the

ACONA framework, the actual code of how the system does anything is put in the service module.

To recognize the state of the service module, it provides state messages about the state of the executed functions. For instance, if the service module did process some data and it failed, then it gets into the state erroneous. These states or meta-data are important to know for the process modules.

C. *Process Module*

While service modules are applied to data in the memory modules, something has to execute the services. It is the task of process modules. By default, every system contains a top process. This system process can then start other process modules. They are then working in parallel or sequential order. The capture of Fig. 3 can be either a cyclic or triggered process. The cyclic processes run in a certain interval as soon as the super process starts them. Triggered processes start at a certain trigger and return their result to the super process.

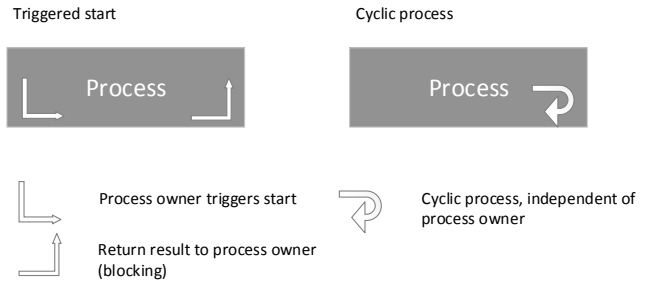

Fig. 3. Serial and parallel processes

Compared to the common state of the art architectures, the process modules are equal to a sequence of production rules. They represent the procedural knowledge of the system and act as the rule engine of the system. Different to SOAR, many processes can run parallel. It would be like having multiple rule engines of SOAR running within the same system. It allows a complete separation of system processes and memories, which is not possible in SOAR.

Finally, if the modules are put into one system, it could look like the abstract system design in Fig 4. The top process starts the cyclic process 1. Process 1 triggers the on-demand process 1.1 to execute the function of Service 1. Service 1 accesses Long-term memory 3 and writes some data into the working memory. When finished, process 1.1 activates process 1.2. It starts service 2 that reads data from the working memory and writes back. Service 1 and Service 2 are executed in sequence.

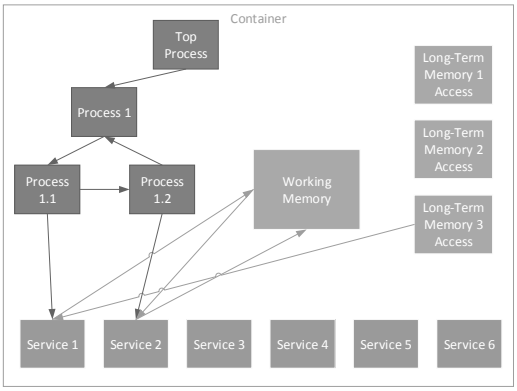

Fig. 4. An example of how the ACONA framework works

V. THE CELL

To be usable as a framework, the module types process, memory and function have to have some common functionality, such as common means of communication and data structures. It is the domain of multi-agent systems because they demand a complete encapsulation of functionality. Data is exchanged by messages. Therefore, the main question is which functionality is needed to realize the three modules described previously.

It is of benefit if the module types extend the same structure. In Fig. 5, the core module or agent, called the cell is shown. It defines all common functionality in the system. It consists of input and output services which are offered to other agents; a data storage; behaviors that are activated through activations, subscriptions of data by other agents; a subagent handler for agents, which are considered to be inside of this agent and the configuration of activations and behaviors.

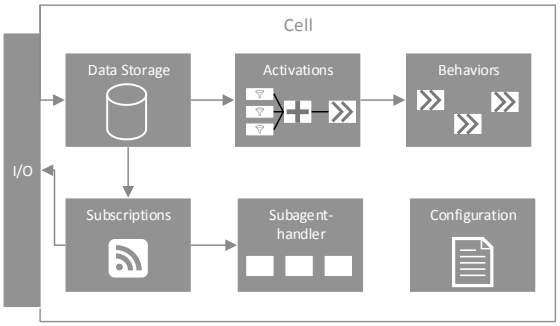

Fig. 5. Architecture of the cell

Through the I/O, messages from other agents reach the cell. The I/O consists of four services: write, read, subscribe and unsubscribe. Each of these services accesses the data storage. All data that is handled by the agent and shared between agents passes the data storage. Data is organized as data points. In that manner, the data storage represents the state of the agent. At the cell level, each agent has its own memory. At the higher level, the memory module either uses this memory or implements an own memory like a database.

Other agents can directly subscribe values of the data storage. The subscriptions can be seen as an agent implementation of the observer pattern. By calling the subscribe service of an agent, the caller gets notified every time a certain data point has changed. In that way, modules can be executed in a sequence.

The behaviors are the things that do something in the system. A behavior realizes any functionality that the cell shall have. Unless the behavior runs cyclic, it has to be triggered. The activations do exactly this. Activations are internal subscriptions of data points of the data storage, but with rule-based triggering. In Fig. 6, a schematic view of the activators, their conditions, and the behaviors is shown. The functionality

can be explained with an example: A cell can implement a calculator by creating a behavior that adds two operands. The adder behavior is only triggered if the data points of the operands are changed and they are both numbers. Else, the cell waits.

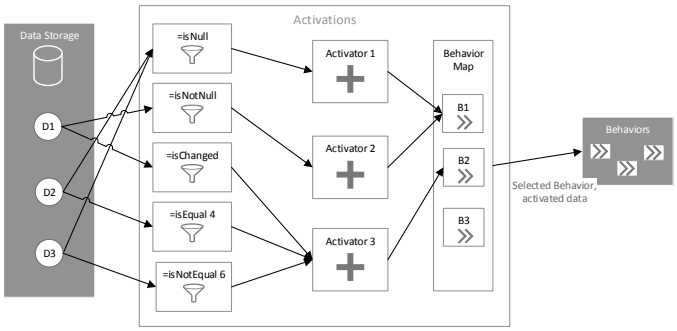

Fig. 6. An example of how the ACONA framework works

An optional functionality is to put agents within agents. The idea is to enable a system to create subsystems within each other. A particular application is to create rule engines, where services are subagents of the rule engine.

Finally, the configuration can be described by the analogy to a living cell. If the behaviors are the functionality of the living cell, the configuration is the DNA. The goal is to generate a cell entirely independent from the setup and to let agents replicate themselves, not just by byte-copy, but through the real instantiation of functionality. The purpose is to make it possible for a cell to change its behavior. In software terms, it means to load other behavior classes autonomously. It could be a way to enhance emergent behaviors within a cognitive architecture.

VI. First Implementation and Preliminary Results

After comparison of different means to realize the ACONA framework, the multi-agent framework JADE [11] was selected. It is a "living" multi-agent framework with suitable licensing, and it is highly customizable. The cell was built as an extension of the JADE framework. JSON data structures were implemented to maximize compatibility with other software. this allows the transformation of data structures to strings as well as into graphs. The configuration of the system was also implemented as JSON objects. It has the interesting effect that the configuration can be injected directly into a cell through a normal "write"-message from another cell, i.e., as the messages would be sent in the cognitive architecture. There, the configuration can reinitialize and change existing behaviors and in this way modify the system behavior.

As this is a work in progress, the current stage of research is that there has been a successful implementation of the cell concept. The next steps are to realize the three module types.

VII. Conclusion and Future Work

Within this paper, the Agent-based Cognitive Architecture Framework ACONA was introduced. Its purpose is to create a framework for the implementation of primarily cognitive architectures. However, the framework may also be suitable for other applications of complex systems. The framework relies on two principles: a multi-agent system approach to ensure maximal encapsulation and independence of functionality; and a strict division of system functionality into three types of modules: A memory module, a service module, and a process module.

Future work will include the implementation of a complete cognitive architecture in the project KORE [2] with approximately 30 different functional modules and at least three parallel running main system processes using this framework. The anticipated challenge is to be able to use external simulators as well as case-based reasoning to create predictions. These are two completely different ways of making decisions in the architecture, and it will test the flexibility of the framework.

Acknowledgment

This work was partly funded by the Austrian Research Promotion Agency FFG within project KORE (project number 848805).

References

[1] D. Bosseboeuf, Energy efficiency trends and policies in the household and tertiary sectors - An Analysis Based on the ODYSSEE and MURE Databases. 2015.

[2] Meisel, M., Xypolytou, E., Wendt, A.: Ergebnisquerschnitt durch ausgewählte Smart Grids Projekte, in proceedings of the 14. Symposium Energieinnovation, Graz, Austria, 2016

[3] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. IEEE transactions on evolutionary computation, vol. 11, no. 2:pp. 151–180, 2007. doi: 10.1109/TEVC.2006.890274.

[4] Schaat, S., Wendt A., Kollmann, S., Gelbard, F., Jakubec, M.: Interdisciplinary Development and Evaluation of Cognitive Architectures Exemplified with the SiMA Approach, in proceedings of EAPCogSci 2015, EuroAsianPacific Joint Conference on Cognitive Science, Torino, Italy, September 25-27, pp. 215-220, 2015

[5] John E. Laird, Clare Bates Congdon, Karen J. Coulter, Nate Derbinsky, and Joseph Xu. The soar user's manual version 9.3.1. Technical report, Computer Science and Engineering Department, University of Michigan, 2011.

[6] Uma Ramamurthy, Bernard J Baars, and Stan Franklin. Lida: A working model of cognition. 2006. CogPrints.

[7] Pat Langley, John E. Laird, and Seth Rogers. Cognitive architectures: Research issues and challenges. Cognitive Systems Research, 10(2):141–160, 2009.

[8] John E Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive robotics using the Soar cognitive architecture. In Proc. of the 8th Int. Conf. on Cognitive Robotics, 2012.

[9] J. R. Anderson, M. Matessa, and C. Lebiere. ACT-R: A theory of higher level cognition and its relation to visual attention. In Human-Computer Interaction 12.4 (1997): 439-462. 1997

[10] J. A. Crowder. The Artificial Cognitive Neural Framework (ACNF). In Infotech@ Aerospace 2012. 2012.

[11] A. Poggi, M. Tomaiuolo and P. Turci. Extending JADE for agent grid applications. Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2004. 13th IEEE International Workshops on. IEEE, 2004.