Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 58 (2019) 366–374

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

CASH only: Constitutive autonomy through motorsensory self-programming

Olivier L. Georgeon [a,*], Alexander Riegler [b]

[a] Université de Lyon, UCLy, EPHE/PSL, LBG UMRS 449, LIRIS CNRS UMR5205, Lyon, France
[b] Center for Logic and Philosophy of Science, VUB, Pleinlaan 2, 1050 Brussel, Belgium

Received 15 May 2019; accepted 17 August 2019
Available online 20 August 2019

Abstract

Constitutive autonomy is the capacity of an entity to perpetually develop its individual constitution and coupling with its environment. We argue that computational entities (i.e., entities that can perform computation) can gain constitutive autonomy through motorsensory self-programming – a mechanism by which the entity acquires new computational processes as a series of patterns of interaction that the entity can learn through experience, simulate internally, and enact in the environment. Motorsensory self-programming allows the evolution of the cognitive coupling between the entity's behavior selection mechanism and the environment as it appears from the viewpoint of the behavior selection mechanism. Constitutive autonomy of computational entities could lead to genuine agency.

Keywords: Autonomy; Constructivist learning; Enaction; Self-motivation; Cognitive architectures

1. Theory

Authors in the domain of artificial intelligence have been discussing the concept of autonomy for a long time. Vernon, Lowe, Thill, and Ziemke (2015: 2) proposed the following broad definition: "the degree of self-determination of a system, i.e., the degree to which a system's behavior is not determined by the environment and, thus, the degree to which a system determines its own goals." The question of how a system (natural or artificial) constructs its own goals remains, however, an open question in philosophy and in the cognitive sciences.

Vernon et al. (2015) as well as Di Paolo (2005), and Froese and Ziemke (2009) distinguish two types of autonomy: behavioral autonomy and constitutive autonomy:

> ``Behavioral autonomy focuses on the external characteristics of the system: the extent to which the system sets its own goals and its robustness and flexibility in dealing with an uncertain and possibly precarious environment." (Vernon et al., 2015: 3)
>
> ``[C]onstitutive autonomy focuses on the internal organization and the organizational process that keep the system viable and maintain itself as an identifiable autonomous entity." (Vernon et al., 2015: 3)

Notably, Vernon et al. make a distinction between "setting goals" (behavioral autonomy) and "determining goals" (autonomy in the broad sense). Our interpretation is that, in the former case, goals are "set" within a pre-existing set of goals in a pre-modeled domain, whereas in the latter case goals are "determined" in an open-ended way as the system autonomously develops and learns about itself and the world. Constitutive autonomy is considered inseparable from living beings. It is also an indispensible

* Corresponding author.
E-mail addresses: ogeorgeon@univ-catholyon.fr (O.L. Georgeon), ariegler@vub.ac.be (A. Riegler).

https://doi.org/10.1016/j.cogsys.2019.08.006
1389-0417/

concept in the cognitive domain because, as Froese and Ziemke (2009) emphasize, "constitutive autonomy is necessary for sense-making."

Artificial entities capable of constitutive autonomy could be envisioned in a multi-agent modeling paradigm growing new "artificial cells" in analogy with the development of biological multi-cellular organisms. However, by focusing on computational entities, i.e., entities that can perform computation, we take the discussion to a higher conceptual level independent of modeling paradigms such as the multi-agent modeling.

In the literature, the term "computation" is quite ambiguous (see, e.g., the discussion in Riegler, Stewart, & Ziemke, 2013). Since notions such as "algorithm" and "program" suggest the unchangeable and inflexible execution of deterministic rules we refrain from defining computation in these terms. Rather, we use it pragmatically in a general and intuitive sense: Computation is a formal sequence of processes that, through discrete steps, transform a set of states into another or the same set states. This definition aligns with the constructivist perspective on neuronal activity: "all states of neuronal activity in the nervous system always lead to other states of activity in the nervous system" (Maturana, 1975: 318).

Is human cognition computational? Can the motions of the planets be seen as computation? Such questions allude to the distinction between computational and non-computational process and how to apply it to observed systems. However, in this paper we do not seek to computationally *model* observable cognitive systems (such as human beings), i.e., identify observable states, and specific state transformations that are needed to successfully map the observed system to the formal system (Cariani, 1992). Rather, we argue that evolving embodied computational entities can become autonomous relative to their environment in their own right. If the criteria for autonomy can be defined in a set of rules, we do not see why computational (i.e., formal) entities could not meet these rules.

Some proponents of enactivism reject computationalism because, allegedly, it involves representation (as used by realist correspondence theories of truth) and cannot implement autonomy (i.e., according to one's internal goals and history rather than purely in terms of reacting to external stimuli). So while we certainly do not subscribe to what is called computationalist cognitive science (equating the human mind with an information-processing computer) we stress that from the constructivist perspective, the notion of structure-determined systems (Maturana, 1975) in particular suggests that any such system is in principle equivalent to a Turing machine (see Villalabos & Dewhurst, 2018): Structure determinism directly leads to cognitive closure and autonomy as well as rejects representationalism, so we do not share the objections of those proponents of enactivism.

How can computational entities gain constitutive autonomy? We propose the *Constitutive Autonomy through Self-programming Hypothesis* (CASH) that holds that constitutive autonomy of computational entities relates to their capacity to autonomously construct new software throughout their existence. In computer-engineering terms, in contrast to what most current machine-learning systems do, this amounts to acquiring new executable code rather than data (including parameters, symbols, weights in neural networks). Here we follow Winograd and Flores (1987) who distinguish between (a) parameter adjustment, (b) combinatorial concept formation, and (c) evolution of structure. They argued that only when "the initial system does not have a structure directly related to the task as seen by its designer" (Winograd and Flores (1987): 102) a computational system can evolve a novel structure shaped by interaction with its environment, i.e., when it is structurally coupled with it (Maturana, 1975). As Winograd & Flores note, this holds for "any system whose internal structure can change as a result of perturbations, and computer programs share this quality" as long as computers are considered "plastic structure-determined systems" (Maturana, 1975). Such flexible software transcends the limitation of predefined programs and qualifies as structure-determined system that can undergo structural coupling with its (software) environment. Similarly, in Riegler (2002) it was argued that structural coupling (leading to the embodiment of the system in question) is not necessarily limited to physical domains, but rather that computer software can also become embodied and construct its own goals that are *novel* with regard to the initial system of the human programmer.

One could be tempted to say that there is a fundamental difference between hardware (populating the physical world) and software (as a specification of a process) in the sense that we always have only partial incomplete access to the physical world (limited by our senses and our measurements), whereas as designers of software, we always have complete access to all aspects of its specification. This distinction is meaningless for practical and theoretical reasons: (1) Learning will always take place from the perspective of the entity and its limited perspective, and (2) given the sheer complexity the software gains, any attempt of understanding and tracing (reverse-engineering) all the processes will become impossible for the human designer due to the combinatory explosion of the numbers of links among the primitives in the system (input values, output values and internal states).

Also, note that we do not claim that the software could constitute the hardware in any conceivable way. We do see the possibility, though, that a computational entity has the capacity of perpetually developing the individual constitution of its software and its coupling (which we call "cognitive coupling" below) with its environment.

The concept of self-modifying processes is not new. Already Newell and Simon (1976) anticipated it in their seminal Turing Award lecture. Weng (2004) proposed the idea of self-affecting cognitive architectures. More recently, Thórisson, Nivel, Sanz, and Wang (2013) have studied self-programming in specific tasks. The question of how an

entity can engage in open-ended life-long self-programming remains, however, still open. From the discussion above it is clear that self-programming raises many research questions including:

- Why should an entity acquire one particular new computational process rather than another? This question is far from trivial as it relates to the entity's intrinsic purpose and self-motivation. It also challenges the widespread understanding of input values as mere percepts upon which the entity is supposed to react.
- What is a suitable method to express the learned computational processes? In more concrete terms: in which programming language should the entity program itself? Is this programming language different from that used by the human designer?

While examining all these questions is clearly impossible in a single paper, in what follows we focus on learning new programs using a predefined set of instructions and a predefined execution engine.

2. Design

Before we continue let us define the terminology. We use "software," "entity," "environment," and "world" from the viewpoint of the human designer. The world is the physical world or a "virtual world" (a computer implementation) as the designer knows it. Within the world, the entity is interacting with its environment, i.e., entity + environment = world. A robot is an example for an entity in the physical world.

The entity's software is a specification of the implementation running in the entity. As a specification, the entity's software is an abstract representation in the head of the designer. The entity's software includes the program initially written by the designer together with various memory structures storing the new computational processes that have been learned over time through self-programming. In the rest of our paper, we use the word software only to refer to the entity's software.

From the perspective of the designer, the software sets output values in the entity's output registers and receives input values through the entity's input registers. The output values trigger changes in the world through the actuators, and the sensors set the input values. From the perspective of the software, there is no distinction between external and internal sensors: input values can come from sensors that are "external" or "internal" to the entity, e.g., vision (external) or proprioception (internal).

The learned computational processes consist of patterns of interactions between the entity and its environment. The software records new sequences of interactions[1] as the entity experiences them. The software can re-execute the learned sequences of interactions by simulating them internally, and can control the entity to enact them again in similar situations. This complies with what authors such as Hesslow (2002) argue, i.e., that cognition involves internal simulation of behaviors.

2.1. From sensorimotor to motorsensory cognition

In cognitive psychology and artificial intelligence, cognitive systems are typically implemented in a way that the entity builds a representation of the environment through input data, and that action can be subsequently determined on the basis of this representation. For example, in his seminal work Drescher (1986) formalized Piagetian sensorimotor schemes into tuple <*context*, *action*, *result*>. The problem is that he then mapped input values with context and with results, as if a representational state of the environment was directly accessible in input. We concur with Bettoni (1993: 240) that such mapping is not faithful to Piaget's theory. It is also inconsistent with philosophical foundations laid out by authors such as Maturana (1975), Varela, Thompson & Rosch (1991), von Glasersfeld (1995), and Merleau-Ponty (2005). These authors suggest to steer away from "cognitive dogmatism" that considers cognitive agents as information-processing entities in which perceptual stimulus unidirectionally leads to action. Vernon et al. summarized their constructivist perspective as follows:

> ``perception and action are reciprocally coupled and mutually dependent. [. . .] perception and action form a joint process of making sense of the world in which the agent is embedded. This `sense' captures the lawfulness of the agent's environment as it relates to the agent's constitutive and behavioral autonomy. Since the agent is an organizationally closed system, perception and action are perturbing forces rather than system inputs and outputs" (Vernon et al., 2015: 6).

This goes beyond the insights of neuroscientists and psychologists who agree that there is no simple unidirectional causal link between stimulus and action. Only when goals are held constant, as in a controlled stimulus-reward situation, a reliable mapping between stimulus and action emerges.

In contrast to that, CASH follows the paradigm that behavior is the control of perception (Porr, Egerton, & Wörgötter, 2006; Powers, 1973). In this view, input values do not represent entity-independent features of the environment (Peschl & Riegler, 1999) but consists of feedback resulting from action or perturbation in a control loop (Georgeon & Guillermin, 2018).

Since this control loop must not be confused with the perception-action loop in the cognitive science literature we avoid the term "sensorimotor," and use instead the term "motorsensory" proposed by Laming (2001). At first

[1] Note that this is not about merely collecting passive data but, rather, data can be interpreted and executed, hence constituting software.

glance "motorsensory" may sound rather awkward and clumsy but reversing the order in this compound word emphasizes the primacy of output over input. One might be tempted to argue that not all cognition can be defined in this way. For example, the "simple perception" of an object does not involve simulation of possible actions. However, there is no such thing as simple perception, as the recognition of an object emerges from one's experiential background that involves having intermodally interacted with this or similar objects in the past. We call self-programming the process by which an entity learns patterns of interaction, simulates them, and re-enacts them in the appropriate situations.

It should be noted that despite the primacy of output over input, motorsensory learning remains capable of handling "alert signals" (i.e., input in the absence of previous output) since defining primitive motorsensory schemes that do not use output values remains possible.

2.2. Primitive motorsensory schemes and motorsensory processes

We design the core of the entity's software as a cognitive architecture responsible for controlling the entity's behavior and learning. This cognitive architecture uses *primitive motorsensory schemes* (PMSs) as its primitive elements of knowledge instead of disjoint primitive percepts and actions as other cognitive architectures do. The human designer programs each PMS to involve output values, input values, or both. In a robot, a PMS controls any kind of actuators while processing any kind of sensory feedback, for example, moving a touch sensor during a predefined period of time while receiving a touch feedback signal within a certain range.[2] The designer specifies the whole set of PMSs, which amounts to defining the entity's primitive experiential domain.

2.3. Interaction

The cognitive architecture interacts only with the interface between cognitive architecture and world; Fig. 1 shows the coupling between those two parts of the entity's software called the *primitive coupling*. On each interaction cycle, the cognitive architecture selects a PMS and sends it to the primitive controller, and receives an enacted PMS in return. The primitive controller controls the enaction of the PMS by the entity (for example, by controlling the trajectory of the touch senor using touch feedback), and then determines the enacted PMS. If the cognitive architecture made a correct anticipation, the enacted PMS will be the same as the originally selected PMS, if not, the enacted PMS will be another PMS from among the set of predefined PMSs. For example the cognitive architecture may select a PMS consisting of controlling the trajectory of the touch sensor while maintaining the feedback signal within a certain range that corresponds to feeling an object. If the object is indeed present, the enacted PMS will correspond to the selected PMS. If the object is absent, the feedback signal will belong to a different range corresponding to another PMS. The cognitive architecture has no notion of the object in itself but uses the set of PMSs that were just enacted to estimate the possibility of enacting further PMSs. Therefore the cognitive architecture remains agnostic towards the world "in itself" (noumenal world) and only knows the "experienced world" (phenomenal world) (Georgeon, Bernard, & Cordier, 2015; Riegler, 2001), which complies with constructivist theories of knowledge (von Glasersfeld, 1995).

As the entity goes through repetitive series of interactions with the environment, the cognitive architecture records *motorsensory processes* (MPs) that consist of a series of PMSs. From a computational perspective, the MPs are the learned programs and PMSs are their primitive instructions. MPs differ from traditional sensorimotor chaining (e.g., Drescher, 1986) in that MPs do not correspond to successions of environmental states. Instead, MPs represent regularities of interaction that have been experienced and which could be experienced again.[3] The entity is not solving a predefined problem or seeking a final goal state but only constructing increasingly higher-level behaviors. Since there is no predefined problem, there is no problem complexity. The architecture just records new MPs as long as it finds new regularities of interaction and has enough memory to record them.

Once an MP has been recorded, the cognitive architecture can select it and let the entity try it in the environment as a whole sequence of interaction (see Section 2.5 for more details on the selection criteria). On each decision cycle, the MP selector sends the selected MP to the MP controller. Fig. 1 shows the coupling between the MP selector and the MP controller called the *cognitive coupling*. The MP controller sequentially sends the PMSs that compose the selected MP to the primitive controller so that the entity enacts them in the environment. In return, the MP controller receives the enacted PMSs. If an enacted PMS differs from the selected PMS, the enaction of the MP is interrupted. The MP controller then records a different enacted MP made of the sequence of PMSs enacted until the interruption, and sends it back to the MP selector. The MP selector uses the set of MPs that where just enacted as a characterization of the current situation to select the next MP to try. The cognitive coupling thus delineates the MP selector from the "world as it appears to the MP selector".

[2] The feedback signal never stops; even feedback equal to zero is feedback.

[3] Cf. von Glasersfeld according to whom "the function of cognition is adaptive and serves the organization of the experiential world, not the discovery of ontological reality" (von Glasersfeld, 1995: 18)

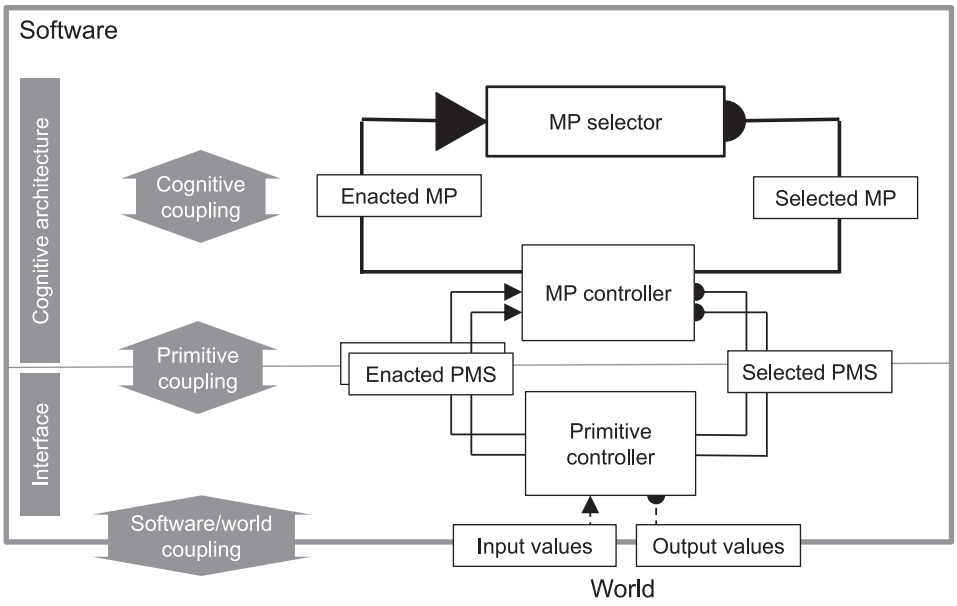

Fig. 1. The software implemented in the entity. The large gray frame (software/world coupling) delineates the software coupled with the world (entity + environment) through the output and input registers. The light gray line (primitive coupling) delimits the cognitive architecture (above) from the interface (below). The interface controls the enaction of primitive motorsensory schemes (PMS) through motor commands and sensor feedback, and then returns the enacted primitive motorsensory schemes. The cognitive coupling links the motorsensory process selector (above) with the motorsensory process controller (below) responsible for sequentially sending the selected primitive motorsensory schemes to the primitive controller. As the cognitive architecture learns higher-level MPs, one round of the cognitive coupling corresponds to several rounds of the primitive coupling.

2.4. Hierarchical structure

The cognitive architecture organizes MPs in a sequential hierarchy where higher-level MPs are made of sequences of lower-level MPs all the way up from PMSs. This hierarchy is reminiscent of Power's model of hierarchically arranged feedback systems (Powers 1973), which "is based on the claim that living organisms behave to control perceptions, and thus suggests that organisms construct their experiential world" (Richards & Glasersfeld, 1979: 37). In our case, though, the cognitive architecture autonomously builds the hierarchy without a preset top-level reference signal as it perpetually constructs new levels recursively from the bottom up. Higher-levels pile up on top of previous levels like mere "sedimentation of habitudes" contingent to the entity's individual experience. From the viewpoint of the MP selector, the world appears to offer increasingly higher possibilities of experiences as the cognitive architecture learns higher-level MPs.

2.5. Selection process

Fig. 2 illustrates the MP learning and selection mechanism. The cognitive architecture maintains a set of MPs that were just enacted. This set works as short-term memory to characterize the current situation. When some MPs in this set match the beginning of a higher-level MP learned previously, then this higher-level MP is *activated*. The lower-level MPs corresponding to the parts of the activated MPs that have not yet been enacted are then proposed. Similar proposed MPs constitute possible *cognitive actions*. The software selects a particular MP to try to enact on the basis of the possible cognitive actions.

The human designer can implement different MP selection criteria depending on the kind of behaviors she wishes the entity to generate. For example, if the MP selector selects the MP that has the least been tried in a given situation then, for the human observer, the entity will appear to be curious. If the MP selector selects the MP that has the highest probability to succeed, then the entity will appear to prefer being "in the flow" (autotelic principle, Steels, 2004). The designer can also associate a predefined numerical valence with PMSs and implement a selector that selects MPs that have the highest cumulative valence of their PMSs. The entity will appear to enjoy enacting interactions that have a positive valence and to dislike enacting interactions that have a negative valence (interactional motivation, Georgeon, Marshall, & Gay, 2012).

3. Examples

Over the past years, we have investigated different approaches to implementing motorsensory cognitive architectures and demonstrate the applicability of CASH. Some of these demonstrations can be seen in videos, in which primitive motorsensory schemes were called primitive interactions, and motorsensory processes were called composite interactions.

- Video 1 (https://youtu.be/LVZ0cPpmSu8) demonstrates the hierarchical self-programming in a simple grid world, see also Fig. 3.

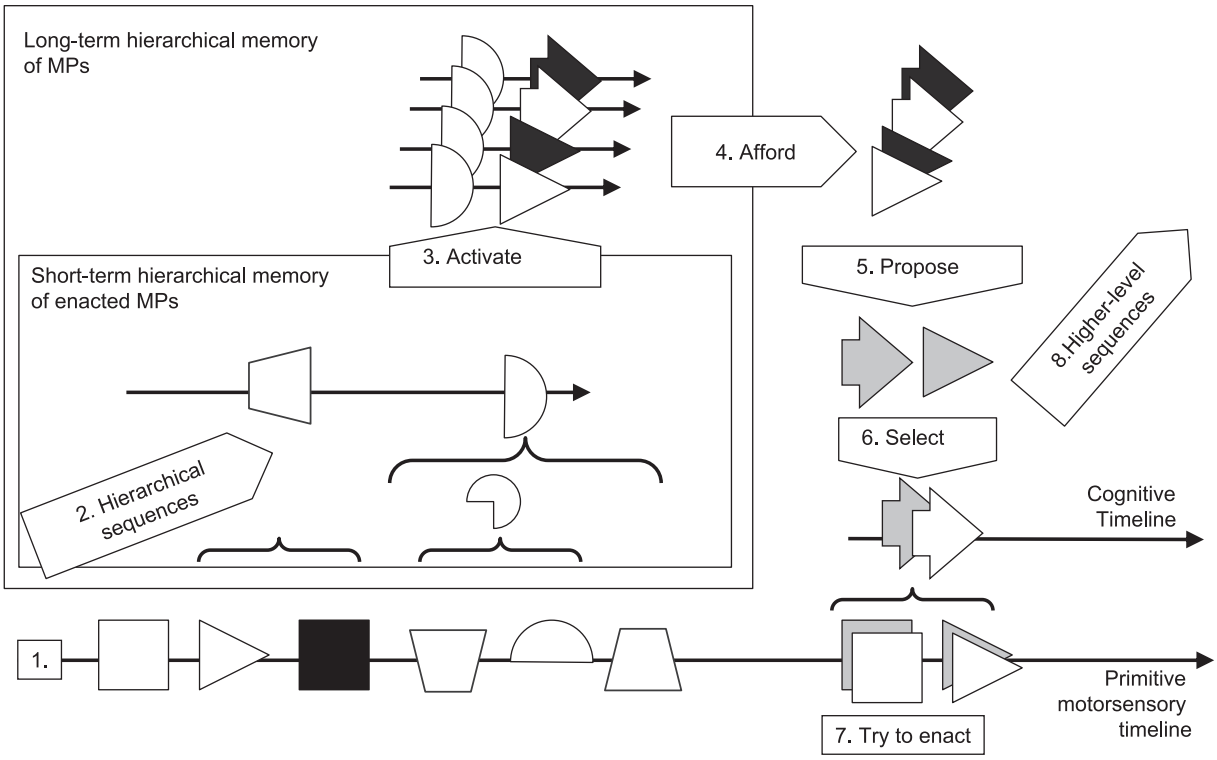

Fig. 2. The MP learning and selection. (1) The primitive motorsensory timeline shows the series of PMSs enacted over time represented by geometrical shapes. (2) The short-term memory of enacted MPs shows the hierarchy of MPs that have been just enacted. (3) An MP represented by a half-circle activates several previously-learned higher-level MPs (stored in long-term memory) that begin with this MP. (4) Activated MPs generate a set of afforded MPs that consist of continuing to enact the activated MPs. (5) Afforded MPs are categorized in possible cognitive actions (gray shapes). (6) The cognitive timeline shows selected MPs and enacted MPs passed through the cognitive coupling. Based on possible cognitive actions, the software selects an MP to try to enact (Gray arrow on the cognitive timeline). (7) The entity tries to enact the PMSs that constitute the selected MP and returns the enacted MP (white arrow). (8) Higher-level MPs can be learned on the basis of enacted MPs.

- Video 2 (https://youtu.be/t1RO5S4mBEY) demonstrates the same algorithm implemented in an e-puck robot (Georgeon, Wolf, & Gay, 2013).
- Video 3 (https://youtu.be/91kKzybt8XY) demonstrates the same algorithm used to control distal motorsensory schemes (Georgeon, Cohen, & Cordier, 2011).
- Video 4 (https://youtu.be/vSUEoh-sjwU) demonstrates more complex learning of a proto ontology of phenomena in the grid world (Georgeon, Marshall, & Manzotti, 2013).
- Video 5 (https://youtu.be/HCDf3Vzl7GM) explains the learning of a proto ontology of phenomena in a continuous and dynamic simulation.
- Video 6 (https://youtu.be/_5HUb6AvSLg) shows internal simulation of MPs in spatial memory.

The architecture used in Videos 4, 5, and 6 is illustrated in Fig. 4.

4. Conclusion

In CASH, the development of the hierarchical MPs through self-programming and through model generation is the mechanism by which the agent contributes to the maintenance of its autonomy and, in that sense, it is a process of constitutive autonomy. The learned executable code is made of patterns of interaction, and the execution engine is an emulator of the entity's body. Motorsensory self-programming allows the evolution of the cognitive coupling between two parts of the software that controls the entity: the cognitive part (including the MP selector and various memory structures) and the executive part (including the MP controller). As the entity develops, the cognitive part sees its environment in the form of increasingly sophisticated possibilities of interactions.

Computational entities and robots in particular equipped with such a system are driven by intrinsic preferences and can learn in an open-ended fashion rather than seek a final goal or perform a predefined task. As such they will enjoy constitutive autonomy.

4.1. Open design questions

So far, we have only implemented motorsensory self-programming in simple entities and environments. The next challenge will be designing entities with increasingly sophisticated effectors and sensors, and software capable

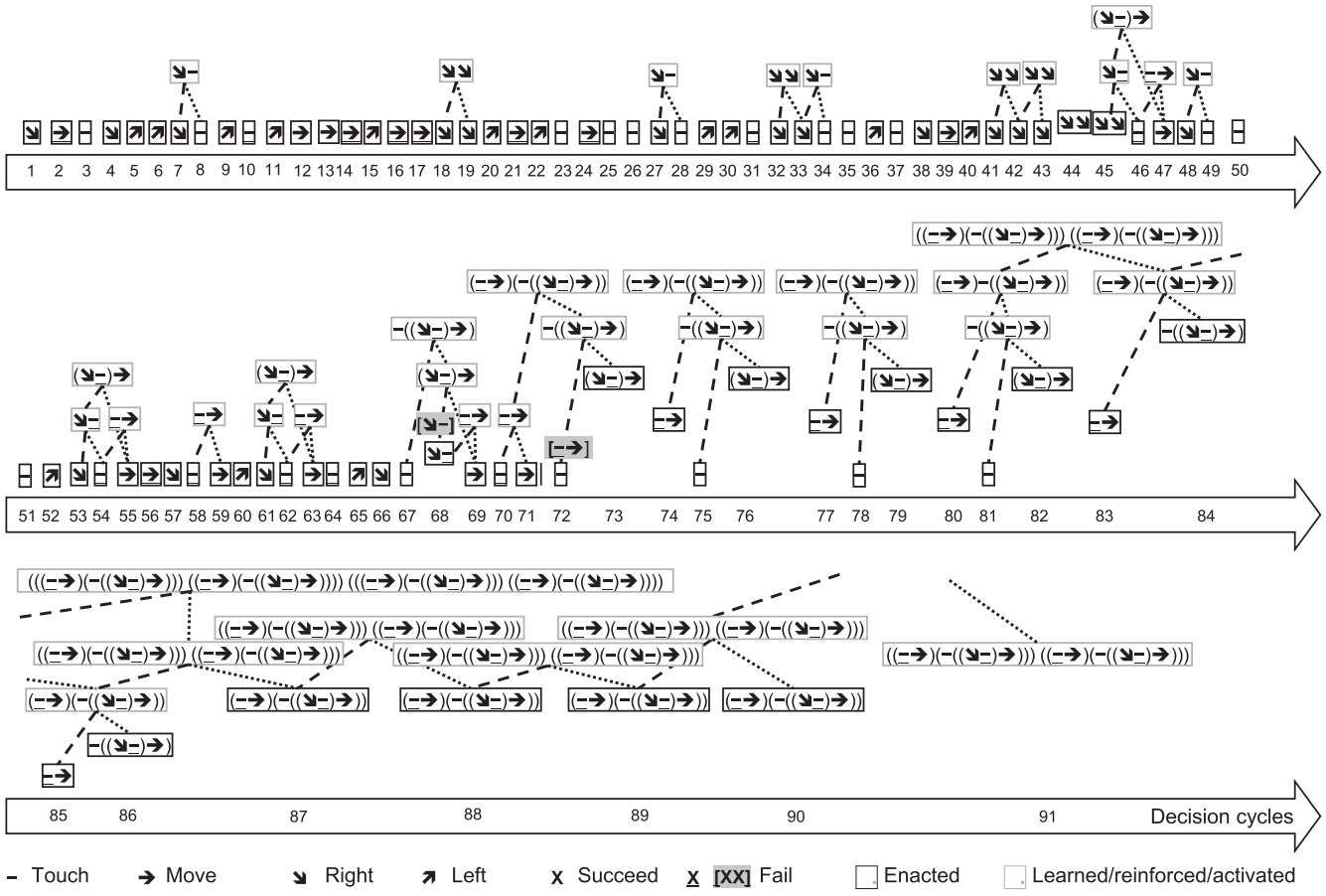

Fig. 3. Demonstration of motorsensory self-programming (Georgeon & Ritter, 2012). The agent has three possible output values corresponding to *touch front*, *step forward*, *turn right*, *turn left* and a single bit input value, forming 6 PMSs. Over time, a hierarchy of MPs is learned from the bottom up in a pairwise manner. Various two-step MPs are enacted during cognitive cycle 44, 45, 68, 74. A six-step MP is enacted four times in a row during cognitive cycles 87, 88, 89, 90. It consists of the sequence: *touch front nothing – step forward – touch front something – turn right – touch front nothing – step forward.* This MP captures a regularity afforded by the agent–environment interactional domain consisting in turning around in a box without bumping into the walls.

of developing increasingly intelligent behaviors as they control these entities in the open world. Since the software is in control of the input values, and not trying to reach a goal-state of a pre-modeled problem, it is not facing information overload or problem-solving combinatorial explosion (Riegler, 2007). Yet, making it more intelligent remains an open topic of research. We must not only find technical solutions but we must also specify more clearly what autonomous intelligent behavior is, how to demonstrate it and to measure it.

To develop more intelligent behaviors in the open three-dimensional world, we expect that the encoding, the transformation, and the execution of spatio-temporal motorsensory processes must be generalized. In addition to egocentric memory shown in Fig. 4, the cognitive architecture may need other spatial memory structures based on different coordinate systems (egocentric, allocentric) to represent temporal and spatial interactions, with the possibility of performing spatial transformations across these structures.

4.2. Open theoretical questions

In the introduction we wrote that in its pursue of constitutive autonomy computer software should construct its own goals that are *novel* with regard to the initial software designed by the human. In the implementations we presented here this is done by recombining pre-defined primitive components (the PMSs). Philosophically, however, this may not be sufficient for genuine novelty.

Cariani (2012) argued that there are two ways to creating novelty: (1) combinations of existing primitives, and (2) creating new primitives which are then subject to combinations. Here, a primitive is meant to be an "element in a system that [has] no internal parts or structure of its own in terms of their functional role in that system" (Cariani (2012): 387). For Cariani, creative systems need to follow (2) because the mere combination of existing primitives means that their set of possibilities is closed.

The creation of new primitives, however, requires new structural frameworks, which can only be brought about

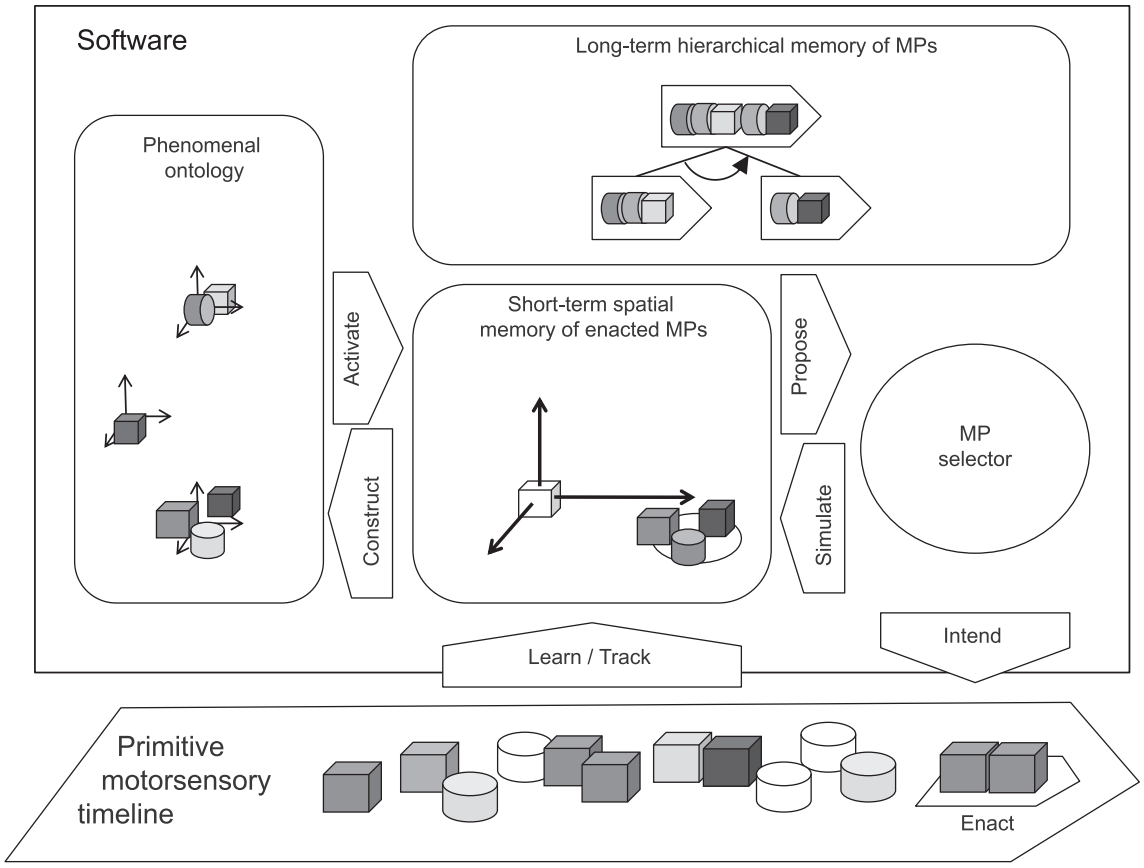

Fig. 4. Spatio-temporal motorsensory cognitive architecture (adapted from Georgeon et al., 2013). Bottom: The primitive motorsensory timeline shows the stream of PMSs enacted over time. PMSs are represented as 3D cubes and cylinders to highlight that they specify the control of actuators with feedback within the presupposed 3D world. Top: The long-term hierarchical memory of MPs implements CASH. Center: The egocentric spatial memory tracks the position of enacted MPs relative to the entity over the short term. It is also used to simulate the enaction of future MPs before selecting them. Left: The phenomenal ontology records *models of phenomena* (spatio-temporal arrangements of MPs) constructed from spatial memory. Models of phenomena are how the software represents entities experienced in the 3D world (the entity that the software is controlling and other entities with which it is interacting). Activated instances of phenomena in spatial memory propose the MPs that they afford. Right: Selection of the next MP to try to enact in the environment.

in more extensive processes of self-modification. In the physical world for Cariani this means creating new hardware. But what are new primitives in software? One interpretation is that this amounts to changing the execution engine that interprets the software's code and to creating new programming instructions. By doing so the entity may be able to change its internal organization more profoundly than by self-programming with a predefined set of programming instructions.

But how can the entity evolve its execution engine? Should the human designer impose constraints on the evolution of the execution engine to accelerate the learning process? So while these theoretical considerations have a lot of appeal the details are still unclear, in particular from a designing perspective.

Acknowledgements

This paper greatly benefitted from discussions with Peter Cariani, David Vernon, and Tom Ziemke. Alexander Riegler acknowledges financial support by the Research Foundation - Flanders (FWO) under Grant No. G023018N. Olivier Georgeon acknowledges financial support by ANR under contract ANR-11-DPBS-0001.

References

Bettoni, M. (1993). Made-up minds: A constructivist approach to artificial intelligence – a book review. *AI Communications, 6*(3–4), 234–240. https://cepa.info/5785.

Cariani, P. (1992). Some epistemological implications of devices which contruct their own sensors and effectors. In F. Varela & P. Bourguine (Eds.), *Towards a Practice of Autonomous Systems* (pp. 484–493). Cambridge, MA: MIT Press, In this issue.

Cariani, P. (2012). Creating new informational primitives in minds and machines. In J. McCormack & M. D'Inverno (Eds.), *Computers and creativity* (pp. 383–417). New York: Springer. https://cepa.info/4135.

Di Paolo, E. (2005). Autopoiesis, adaptivity, teleology, agency. *Phenomenology and the Cognitive Sciences, 4*(4), 429–452. https://cepa.info/2269.

Drescher, G. L. (1986). Genetic AI: Translating piaget into lisp. *Instructional Science, 14*(3), 357–380. https://cepa.info/2296.

Froese, T., & Ziemke, T. (2009). Enactive artificial intelligence: Investigating the systemic organization of life and mind. *Artificial Intelligence, 173*(3–4), 466–500. https://cepa.info/279.

Georgeon, O., & Guillermin, M. (2018). Mastering the laws of feedback contingencies is essential to constructivist artificial agents. *Constructivist Foundations, 13*(2), 300–301. https://constructivist.info/13/2/300.

Georgeon, O., Bernard, F., & Cordier, A. (2015). Constructing phenomenal knowledge in an unknown noumenal reality. *Procedia Computer Science, 71*, 11–16.

Georgeon, O., Wolf, C., & Gay, S. (2013) An enactive approach to autonomous agent and robot learning. In Third joint international conference on development and learning and on epigenetic robotics, Osaka, Japan (pp. 1–6).

Georgeon, O., Marshall, J., & Manzotti, R. (2013). ECA: An enactivist cognitive architecture based on sensorimotor modeling. *Biologically Inspired Cognitive Architectures, 6*, 46–57. https://cepa.info/1009.

Georgeon, O., Marshall, J., & Gay, S. (2012) Interactional motivation in artificial systems: Between extrinsic and intrinsic motivation. In Proceedings of the 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL) (pp. 101–102). Piscataway, NJ: IEEE.

Georgeon, O., & Ritter, F. (2012). An intrinsically-motivated schema mechanism to model and simulate emergent cognition. *Cognitive Systems Research, 15–16*, 73–92.

Georgeon, O., Cohen, M., & Cordier, A. (2011). A model and simulation of early-stage vision as a developmental sensorimotor process. In *International conference on Artificial Intelligence Applications and Innovations (AIAI 2011) Corfu, Greece* (pp. 11–16).

von Glasersfeld, E. (1995). *Radical constructivism: A way of knowing and learning*. London: Falmer Press. https://cepa.info/1462.

Hesslow, G. (2002). Conscious thought as simulation of behavior and perception. *Trends in Cognitive Sciences, 6*(6), 242–247.

Laming, D. (2001). On the distinction between "sensorimotor" and "motorsensory" contingencies. *Behavioral and Brain Sciences, 24*(5), 992.

Maturana, H. R. (1975). The organization of the living: A theory of the living organization. *International Journal of Man-Machine Studies, 7*(3), 313–332. https://cepa.info/547.

Merleau-Ponty, M. (2005) Phenomenology of perception. Translated by Colin Smith. Routledge, London. Originally published in French as: Merleau-Ponty M. (1945) Phénoménologie de la perception. Callimard, Paris.

Newell, A., & Simon, H. (1976). Computer science as empirical inquiry: Symbols and search. ACM Turing Award Lecture. *Communications of the ACM, 19*(3), 113–126.

Powers, W. T. (1973). *Behavior: The control of perceptions*. Chicago: Aldine.

Richards, J. & von Glasersfeld, E. (1979) The control of perception and the construction of reality. https://cepa.info/1345.

Peschl, M. F., & Riegler, A. (1999). Does representation need reality? In A. Riegler, M. F. Peschl, & A. von Stein (Eds.), *Understanding representation in the cognitive sciences* (pp. 9–17). New York: Kluwer Academic/Plenum Publishers. https://cepa.info/2419.

Porr, B., Egerton, A., & Wörgötter, F. (2006). Towards closed loop information: Predictive information. *Constructivist Foundations, 1*(2), 83–90. https://constructivist.info/1/2/083.

Riegler, A. (2001). Towards a radical constructivist understanding of science. *Foundations of Science, 6*(1–3), 1–30. https://cepa.info/1860.

Riegler, A. (2002). When is a cognitive system embodied? *Cognitive Systems Research, 3*(3), 339–348. https://cepa.info/1862.

Riegler, A. (2007) Superstition in the machine. In M. V. Butz, O. Sigaud, G. Pezzulo & G. Baldassarre (eds.), Anticipatory behavior in adaptive learning systems: From brains to individual and social behavior. Lecture Notes in Artificial Intelligence (pp. 57–72). New York: Springer. https://cepa.info/4214.

Riegler, A., Stewart, J., & Ziemke, T. (2013). Computation, cognition and constructivism: Introduction to the special issue. *Constructivist Foundations, 9*(1), 1–6. https://constructivist.info/9/1/001.

Steels, L. (2004) The autotelic principle. In I. Fumiya, R. Pfeifer, L. Steels, & K. Kunyoshi (eds.), Embodied artificial intelligence (pp. 231–242). Springer Verlag.

Thórisson, K., Nivel, E., Sanz, R., & Wang, P. (2013). Approaches and assumptions of self-programming in achieving artificial general intelligence. *Journal of Artificial General Intelligence, 3*(3), 1–10.

Vernon, D., Lowe, R., Thill, S., & Ziemke, T. (2015). Embodied Cognition and circular causality: On the role of constitutive autonomy in the reciprocal coupling of perception and action. *Frontiers in Psychology, 6*, 1660.

Varela, F. J., Thompson, E., & Rosch, E. (1991). *The embodied mind: Cognitive science and human experience*. Cambridge MA: MIT Press.

Villalobos, M., & Dewhurst, J. (2018). Enactive autonomy in computational systems Available at. *Synthese, 195*(5), 1891–1908. https://cepa.info/5561.

Weng, J. (2004). Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics, 1*(2), 199–236.

Winograd, T. & Flores, F. (1987) Computation and intelligence. In Understanding computers and cognition: A new foundation for design (pp. 93–106). Reading MA: Addison-Wesley. https://cepa.info/5787.