Chapter 9

A New Constructivist AI: From Manual Methods to Self-Constructive Systems

Kristinn R. Thórisson

Center for Analysis & Design of Intelligent Agents, Reykjavik University
and
Icelandic Institute for Intelligent Machines
Menntavegur 1, IS-101 Reykjavik, Iceland

thorisson@{ru.is, iiim.is}

The development of artificial intelligence (AI) systems has to date been largely one of manual labor. This constructionist approach to AI has resulted in systems with limited-domain application and severe performance brittleness. No AI architecture to date incorporates, in a single system, the many features that make natural intelligence general-purpose, including system-wide attention, analogy-making, system-wide learning, and various other complex transversal functions. Going beyond current AI systems will require significantly more complex system architecture than attempted to date. The heavy reliance on direct human specification and intervention in constructionist AI brings severe theoretical and practical limitations to any system built that way.

One way to address the challenge of artificial general intelligence (AGI) is replacing a top-down architectural design approach with methods that allow the system to manage its own growth. This calls for a fundamental shift from hand-crafting to self-organizing architectures and self-generated code – what we call a *constructivist AI* approach, in reference to the self-constructive principles on which it must be based. Methodologies employed for constructivist AI will be very different from today's software development methods; instead of relying on direct design of mental functions and their implementation in a cognitive architecture, they must address the *principles* – the "seeds" – from which a cognitive architecture can automatically grow. In this paper I describe the argument in detail and examine some of the implications of this impending paradigm shift.

9.1 Introduction

Artificial intelligence researchers have traditionally been rather optimistic about the rate of progress in the field. The origin of this optimism dates back numerous deacdes, possibly

as far back as our understanding of the electrical properties of neurons and their role in controlling animal behavior, but at the very least to the invention of using electricity for automatic calculation and related tasks – tasks that only humans used to be capable of. These realizations seemed so laden with potential that even the most ardent skeptics couldn't help imagining near-future scenarios where electrical machines would be doing all sorts of tasks requiring intelligence, helping out in every area of human endeavor.

In spite of measurable progress since the early discoveries, one big question still remains unanswered: How does the human and animal mind work? Bits and pieces of answers have been popping out of neuroscience, cognitive science, psychology, and AI research, but a holistic picture seems as far in the future as ever. What we would like to see – and this is a vision shared by many of those who started the field of AI over 50 years ago – is an artificial system that can learn numerous disparate tasks and facts, reliably perform them in a variety of circumstances and environments of real-world complexity; a system that can apply itself to learning a wide range of skills in a wide range of domains. Such a system would be considered by most as "generally intelligent". A trivial example is an AI that can learn, over a period of say 3 months, to cook dinner, do the dishes, and fix automobiles. An AI that can acquire the skills to invent new things, solve global warming, and negotiate peace treaties would be yet another step forward, but for all we know, this might not be too far-fetched if we can achieve the former.

A functioning brain is a reasonably good place to start studying how thought works – after all natural intelligence is what gave us the idea to build artificial minds in the first place. Consider functions of natural minds such as global attention with introspective capabilities, the ability to discover, understand and abstract facts and causal chains, to make analogies and inferences, and to learn a large amount of vastly different skills, facts and tasks, including the control of one's own thoughts. These are features that seem simply too critical to leave out when attempting to build an intelligent system. It is in part the historical neglect of such key features – but especially *their integration* – that motivates the present discussion.

The vast collection of atoms and molecules found in everything we see in this universe tells us little about familiar phenomena such as oceans, rainforests and Picasso paintings. In the same way, brain neurons are but one of the many building blocks behind the complex phenomenon we normally recognize as intelligence. Just like the atoms in the trunk of a tree or water molecules on the surface of the Earth, neurons participate in pattens of interaction that form complex structures of lower granularity, all of which are more complex than any

single neuron alone. Without the right superstructures, brain neurons are nothing more than fancy amoebas, and just as far from what we recognize as high-level intelligence.

Studying only neurons for finding out how the mind works is akin to restricting oneself to atoms when studying weather patterns. Similarly, languages used today for programming computers are too restrictive – instead of helping us build complex animated systems they focus our attention on grains of sand, which are then used to implement bricks – software modules – which, no surprise, turn out to be great for building brick houses, but are too inflexible for creating the mobile autonomous robot we were hoping for. In short, modern software techniques are too inflexible for helping us realize the kinds of complex dynamic systems necessary to support general intelligence.

What is called for are new methodologies that can bring more power to AI development teams, enabling them to study cognition in a much more holistic way than possible today, and focus on *architecture* – the operation of the system as a whole. To see why this is so we need to look more closely at what it is that makes natural intelligence special.

9.2 The Nature of (General) Intelligence

Whether or not natural intelligence is at the center of our quest for thinking machines, it certainly gives us a benchmark; even simple animals are capable of an array of skills way beyond the most advanced man-made machines. Just to take an example, a fully grown human brain must contain millions of task-specific abilities. This fact in itself is impressive, but pales in comparison to the fact that these skills have been acquired largely autonomously by the system itself, through self-directed growth, development, and training, and are managed, selected, improved, updated, and replaced dynamically, while the system is in use. An integrated system with these capabilities is able to apply acquired skills in realtime in varied circumstances; it can determine – on the fly – when and how to combine them to achieve its goals. And in spite of already having a vast amount of acquired skills, it must retain the ability to acquire new ones, some of which may have very little overlap with any prior existing knowledge.

The field of artificial intelligence has so far produced numerous partial solutions to what we normally call intelligent behavior. These address isolated sub-topics such as playing board games, using data from a camera to perform a small set of predetermined operations, transcribing spoken words into written ones, and learning a limited set of actions from experience. In systems that learn, the learning targets (goals) are hand-picked by the

programmer; in systems that can to some extent "see" their environment, the variations in operating contexts and lighting must be highly restricted; in systems that can play board games, the types of games are limited in numerous ways – in short, there are significant limitations to the systems in each of these areas. No system has yet been built that can learn two or more of these domains by itself, given only the top-level goal of simply learning "any task that comes your way". These systems are nonetheless labeled as "artificially intelligent" by a majority of the research community.

In contrast, what we mean by "general" intelligence is the ability of a system to learn many skills (e.g. games, reading, singing, playing racketball, building houses, etc.) and to learn to perform these in many different circumstances and environments. To some extent one could say that the ability to "learn to learn" may be an important characteristic of such a system. Wang (2004) puts this in the following way:

> "If the existing domain-specific AI techniques are seen as *tools*, each of which is designed to solve a special problem, then to get a general-purpose intelligent system, it is not enough to put these tools into a toolbox. What we need here is a *hand*. To build an integrated system that is self-consistent, it is crucial to build the system around a general and flexible *core*, as the hand that uses the tools [assuming] different forms and shapes."

Many abilities of an average human mind, that are still largely missing from present AI systems, may be critical for higher-level intelligence, including: The ability to learn through experience via examples or through instruction; to separate important things from unimportant ones in light of the system's goals and generalize this to a diverse set of situations, goals and tasks; steering attention to important issues, objects and events; the ability to quickly judge how much time to spend on the various subtasks of a task in a complex sequence of actions; the ability to continuously improve attention steering; to make analogies between anything and everything; the ability to learn a wide range of related or unrelated skills without damaging what was learned before; and the ability to use introspection (thinking about one's own thinking) as a way to understand and improve one's own behavior and mental processes, and ultimately one's performance and existence in the world.

These are but a few examples of many *pan-architectural* abilities that involve large parts of the entire system, including the process of their development, training, and situated application. All of these abilities are desirable for a generally intelligent artificial mind,

and many of them may be necessary. For example, a system working on the docks and helping to tie boats to the pier must be able to ignore rain and glaring sun, the shouts of others doing their own tasks, detect the edge of the pier, catch a glimpse of the rope in the air, prepare its manipulators to catch it, catch it, and tie it to the pier – all within the span of a few seconds. In a real-world scenario this task has so many parameters that the system's realtime attentional capabilities must be quite powerful. It is hard to see how a general-purpose intelligence can be implemented without an attention mechanism that can – in realtime – learn to shift focus of attention effectively from internal events (e.g. remembering the name of a colleague) to external events (e.g. apologizing to those present for not remembering her name), and at runtime – while operating – improve this skill based on the goals presented by social norms. Natural intelligences probably include many such attention mechanisms, at different levels of detail. Attention is just one of the seemingly complex transversal skills – pan-architectural skills that represent in some way fundamental operational characteristics of the system – without which it seems rather unlikely that we will ever see artificial general intelligence, whether in the lab or on the street.

The enormous gap in size and complexity between a single neuron and a functioning animal brain harbors a host of challenging questions; the size and nature of this challenge is completely unclear. What is clear, however, is that a fully formed brain is made up of a network of neurons forming substructure upon substructure of causal information connections, which in turn form architectures within architectures within architectures, nested at numerous levels of granularity, each having a complex relationship with the others both within and between layers of granularity [10, 34]. This relationship, in the form of data connections, determines how the mind deals with information captured by sensory organs, how it is manipulated, responded to, and learned from over time. The architectures implement highly coordinated storage systems, comparison mechanisms, reasoning systems, and control systems. For example, our sense of balance informs us how to apply forces to our arm when we reach for a glass of water while standing on a rocking boat. Past experience in similar situations tells us to move slowly. Our sense of where the glass is positioned in space informs our movement; the information from our eyes and inner ear combines to form a plan, and produce control signals for the muscles, instructing them how to get the hand moving in the direction of the glass without us falling down or tipping the glass over. A system that can do this in real-time is impressive; a system that can *learn* to do this, for a large set of variations thereof, along with a host of other tasks, must also have a highly flexible architecture.

Elsewhere I have discussed how the computational architecture, software implementation, and the cognitive architecture that it implements, need not be isomorphic [35]. Thus, the arguments made here do not rest on, and do not *need* to rest on, assumptions about the particular *kind of architecture* in the human and animal mind; in the drive for a new methodology we are primarily concerned with the operational characteristics that the system we aim to build – the architecture – must have. Past discussions about cognitive architecture, whether the mind is primarily "symbol-based" (cf. [20]), "dynamical" (cf. [9, 41]), "massively modular" (cf. [2]), or something else entirely, can thus be put aside. Instead we focus on the sheer *size* of the system and the extreme *architectural plasticity*, while exhibiting a tight integration calling for a *high degree of interdependencies between an enormous set of functions*. Just as an ecosystem cannot be understood by studying one lake and three inhabiting animal species, intelligence cannot be realized in machines by modeling only a few of its necessary features; by neglecting critical interdependencies of its relatively large number of heterogeneous mechanisms most dissections are prevented from providing more than a small fragment of the big picture. General intelligence is thus a system that implements numerous complex functions organized at multiple levels of organization. This is the kind of system we want to build, and it cannot be achieved with the present methodologies, as will become evident in the next section.

To summarize, work towards artificial *general* intelligence (AGI) cannot ignore necessary features of such systems, including:

- *Tight integration:* A general-purpose system must tightly and finely coordinate a host of skills, including their acquisition, transitions between skills at runtime, how to combine two or more skills, and transfer of learning between them over time at many levels of temporal and topical detail.
- *Transversal functions:* Related to the last point, but a separate issue; lies at the heart of system flexibility and generality: The system must have pan-architectural characteristics that enable it to operate consistently as a whole, to be highly adaptive (yet robust) in its own operation across the board, including meta-cognitive abilities. Some such functions have been listed already, namely attention, learning, analogy-making capabilities, and self-inspection, to name some.
- *Time:* Ignoring (general) temporal constraints is not an option if we want AGI. Time is a semantic property, and the system must be able to understand – and be able to *learn* to understand – time as a real-world phenomenon in relation to its own skills and architectural operation.

- *Large architecture:* An architecture that is considerably larger and more complex than systems being built in AI labs today is likely unavoidable, unless we are targeting toy systems. In a complex architecture the issue of concurrency of processes must be addressed, a problem that has not yet been sufficiently resolved in present software and hardware. This scaling problem cannot be addressed by the usual "we'll wait for Moore's law to catch up" [19] because the issue does not primarily revolve around speed of execution, but around the nature of the architectural principles of the system and their runtime operation.

This list could be extended even further with various other desirable features found in natural intelligences such as predictable robustness and graceful degradation; the important point to understand here is that if some (or all) of these features *must* exist in the same system, then it is highly unlikely that we can create a system that addresses them unless we address them *at the same time*, for (at least) one obvious reason: For any partially operating complex architecture, retrofitting one or more of these into it would be a practical – and possibly a theoretical – impossibility (cf. [11]). As I myself and others have argued before [43], the conclusion can only be that *the fundamental principles of an AGI must be addressed holistically*. This is clearly a reason to think long and hard about our methodological approach to the work at hand.

9.3 Constructionist AI: A Critical Look

In computer science, *architecture* refers to the layout of a large software system made up of many interacting parts, often organized as structures within structures, and an operation where the "whole is greater than the sum of the parts". Although not perfect, the metaphorical reference to physical structures and urban layout is not too far off; in both cases system designs are the result of compromises that the designers had to make based on a large and often conflicting set of constraints, which they resolved through their own insight and ingenuity. In both cases *traffic* and *sequence of events* is steered by *how things are connected;* in the case of software architecture, it is the traffic of *information* – who does what when and who sends and receives what information when.[1]

There is another parallel between urban planning and software development that is even more important; it has to do with the tools and methodologies used to design and imple-

[1] I use the term "software architecture" here somewhat more inclusively than the metaphorical sense might imply, to cover both the parts responsible for system operation, processing, and data manipulation, as well as the data items and data structures that they operate on.

ment the target subject. When using computer simulations as an integral part of a research methodology, as is done increasingly in many fields including astrophysics, cognitive science, and biology, an important determinant of the speed of progress is how such software models are developed and implemented; the methodology used to write the software and the surrounding software framework in which they are developed and run, is in fact *a key determinant of progress*. Could present methodologies in computer science be used to address the challenges that artificial *general* intelligence (AGI) presents?

Since software systems are developed by human coders, the programming languages now used have been designed for human use and readability, and it is no surprise that they reflect prototypical ways in which humans understand the world. For instance, the most popular method for organizing programming languages in recent years is based on "object orientation," in which data structures and processes are organized into groups intended to give the human designer a better overview of the system being developed. Dependencies between the operation and information flow in such groups, that is, what particulars of what groups of processes are allowed to receive, process, and output the various types of data, is decided at design time. This also holds for the groups' operational semantics, where each group is essentially a special-purpose processor that has special-purpose behavior, with a pre-defined role in the system as a whole, defined by the software coder at design time. In this constructionist approach variables, commands, and data, play the role of bricks; the software developer takes the role of the "construction worker."

The field of artificial intelligence too relies on a constructionist approach: Systems are built by hand, and both the gross and fine levels of architectures are laid down "brick by brick." This is where we find the most important similarity between architecture and software development, and the one with which we are primarily concerned here; in both cases *everything* – from the fine points to the overall layout of the final large-scale processing structures – is *defined, decided* and *placed in the system* by *human designers*.

As history unequivocally shows, all AI systems developed to date with constructionist methodologies have had the following drawbacks, when compared to natural intelligence:

- They are extremely *limited* in what they can do, and certainly don't come anywhere *close* to addressing the issues discussed in the prior section, especially transversal functions such as global attention mechanisms and system-wide learning.
- They are *brittle*, stripped of the ability to operate outside of the limited scope targeted by their designer. More often than not they also tend to be brittle when operating *within* their target domain, especially when operating for prolonged periods, which

reveals their sensitivity to small errors in their implementation and lack of graceful degradation in respect to partial failure of some components.

To be sure, AI researchers often extend and augment typical software development methodologies in various ways, going beyond what can be done with "standard" approaches. The question then becomes, how far could constructionist methodology be taken?

Over the last few decades only a handful of methodologies have been proposed for building large, integrated AI systems – Behavior-Oriented Design (BOD; [5]), the Subsumption Architecture [4], Belief, Desires, Intentions (BDI; cf. [28]), and the Constructionist Design Methodology (CDM) [37] are among them. CDM rests on solid present-day principles of software engineering, but is specifically designed to help developers manage system expansion; its principles allow continuous architecture-preserving expansion of complex systems involving large numbers of executable modules, computers, developers, and research teams.[2]

The Cognitive Map architecture [22], implemented on the Honda ASIMO robot, enables it to play card games with children using reciprocal speech and gesture. Implementing and coordinating state of the art vision and speech recognition methods, novel spatio-temporal interpretation mechanisms and human-robot interaction, this system is fairly large, and it is among the relatively few that make it a specific goal to push the envelope on scale and breadth of integration.

The Cognitive Map architecture has benefited greatly from the application of the CDM to its construction, as system development has involved many developers over several years. The architecture is implemented as multiple semi-independent modules running on multiple CPUs interacting over a network. Each module is responsible for various parts of the robot's operations, including numerous perceptors for detecting and indexing perceived phenomena and various types of deciders, spatial, and semantic memory systems, action controllers, etc. Each of the modules is a (hand-crafted) piece of software, counting anywhere from a few dozen lines of code to tens of thousands. As most of the modules are at the smaller end of this spectrum, interaction and network traffic during system runtime is substantial. The components in the Cognitive Map architecture are coordinated via an infrastructure called Psyclone AIOS, a middleware that targets large AI systems based on multiple dynamically interacting modules, with powerful blackboard-based data sharing

[2]Results of the application of CDM have been collected for several types of systems, in many contexts, at three different research institutions, CADIA [15, 29, 38], Honda Research Labs (HRI-US) [21, 22] and the Computer Graphics and User Interfaces Lab at Columbia University [37].

mechanisms [39]. Psyclone AIOS allows multiple programming languages to be used together, supports easy distribution of processes over a network, handles data streams that must be routed to various destinations at runtime (e.g. from video cameras), and offers various other features that help with AI architecture development. The system supports directly the principles of the CDM and is certainly among the most flexible such systems involving the creation and management of large architectures.[3]

In light of progress within the field of robotics and AI, the ASIMO Cognitive Map architecture is a strong contender: It integrates a large set of functionality in more complex ways than comparable systems did only a few years ago. It relies on a methodology specifically targeted to AI and robotic systems, where multiple diverse functions implemented via thousands of lines of code must be tightly integrated and coordinated in realtime. So how large is the advancement demonstrated by this system?

The resulting system has all of the same crippling flaws as the vast majority of such systems built before it: It is brittle, and complex to the degree that it is becoming exponentially more expensive to add features to it – we are already eyeing the limit. Worst of all, it has still not addressed – with the exception of a slight increase in breadth – a single one of the key features listed above as necessary for AGI systems. In spite of good intentions, neither the CDM – nor any of the other methodologies, for that matter – could help address the difficult questions of transversal functions and architectural construction which are orders of magnitude larger than attempted to date.

Some of the systems I have been involved with developing have contained an unusually large number of modules, with sizes varying from very small (a few dozen lines of code) to very large (tens of thousands). We refer to them as "granular" architectures (cf. [15]). In these systems the algorithms coordinating the modules (i.e. the gross architecture) dynamically control which components are active at what time, which ones receive input from where, etc. Some of the components can change dynamically, such as the Indexers/Deciders in the Cognitive Map [21], and learn, such as the module complex for learning turntaking in the artificial radio-show host [15]. Increased granularity certainly helps making the architecture more powerful. But yet again, the modules in these systems are typically black-box with prescribed dependencies, which precludes them from automatically changing their operation, expand their reach, or even modify their input and output profiles in any significant way, beyond what the coder could prescribe. Their inputs and outputs are directly dependent on the surrounding architecture, which is restricted by the

[3]Other solutions addressing similar needs include Elvin [33], the Open Agent Architecture [18] and NetP [13].

inability of components to change their operation. Many component technologies used in these architectures are built from different theoretical assumptions about their operating context, increasing this dependency problem.

On the practical side, many problems get worse as the architectures get bigger, e.g. for lack of fault tolerance (such as code-level bugs). Some new problems are introduced, especially architecture-level problems and those we call *interaction problems*: Loosely-coupled modules often have complex (and infrequent) patterns of interaction that are difficult to understand for the developers in the runtime system; interaction problems grow exponentially as the system gets bigger. A constructionist approach does not help us unlock tight interdependencies between components, or remove the need for humans to oversee and directly interact with the system at the code level.

Examples of other AI architectures with ambitions towards artificial general intelligence include LIDA [8], AKIRA [25], NARS [45], SOAR [16], CLARION [32], ACT-R [1], OSCAR [27], and Ikon Flux [23]. However, those that have been tested on non-trivial tasks, such as ACT-R, SOAR, and CLARION, are based on constructionist methodologies with clear limitations in scaling arising thereof; those that promise to go beyond current practices, e.g. Ikon Flux, NARS, LIDA, and OSCAR, suffer from having either been applied only to toy problems, or are so new that thorough evaluation is still pending.

No matter how dynamic and granular the components of an architecture are made, or which expanded version of a constructionist methodology is being applied, a heavy reliance on manual construction has the following effects:

- System components that are fairly static. Manual construction limits the complexity that can be built into each component.
- The sheer number of components that can form a single architecture is limited by what a designer or team can handle.
- The components and their interconnections in the architecture are managed by algorithms that are hand-crafted themselves, and thus also of limited flexibility.

Together these three problems remove hopes of autonomous architectural adaptation and system growth. Without system-wide adaptation, the systems cannot break free of targeted learning. Like most if not all other engineered systems of a comparable scale and level of integration, e.g. telephone networks, CPUs, and power grids, these systems are incapable of architecture-level evolution, precluding architecture-wide learning (what one might metaphorically think of as "cognitive growth") and deep automatic adaptation, all of which precludes general-purpose systems capable of applying themselves autonomously to

arbitrary problems and environments. That is precisely the kind of flexibility we want a new methodology to enable us to imbue an AI architecture with.

The solution – and most fruitful way for AI in the coming decades – rests on the development of a new style of programming, with greater attention given to the architectural makeup, structure, and nature of large complex systems, bringing with it the element of automated systems management, resting on the principles of transparent operational semantics.

9.4 The Call for a New Methodology

Available evidence strongly indicates that *the power of general intelligence*, arising from a *high degree of architectural plasticity*, is of a *complexity well beyond the maximum reach of traditional software methodologies*. At least three shortcomings of constructionist AI need to be addressed in a new methodology: Scale, integration, and flexibility. These are fundamental shortcomings of *all software systems developed to date*, yet, as we have seen above, they *must* all be overcome *at the same time* in the same system, if we wish to achieve artificial general intelligence. Any approach that is successful in addressing these must therefore represent *a fundamentally new type of methodology*.

Scale matters in at least two ways. First, we have reason to believe that a fairly large set of cognitive functions is necessary for even modestly complex real-world environments. A small system, one that was of a size that could be implemented by a handful of engineers in half a decade, is not likely to support the reliable running of supernumerary functions. And historical evidence certainly does not help refute this claim: In spite of decades of dealing with the various problems of scaling beyond toy contexts ("context" interpreted in a rather general form, as in "the ocean" versus "the desert"; "indoors" versus "outdoors," etc.), standard component-based software methodology has theoretical limitations in the size of systems that it can allow to be built; as these systems are programmed by humans, their size and complexity is in fact restricted by the industriousness of a dedicated team of researchers in the same way that building a house is. This is the reason why we still have not seen systems that scale easily. What is needed is the equivalent of a highly automated factory. Second, a small system is not very likely to lend sufficient support to the kind of functions that characterize higher-level intelligences, such as system-wide analogy-making, abstraction, cross-domain knowledge and knowledge transfer, dynamic

and learnable attention, all of which require transversal functionality of some sort to be of general use.[4]

The issue of integration ultimately revolves around software architecture. Most architectures built to date are coarse-grained, built of relatively large modules, because this is the kind of architecture that traditional methodologies most naturally support. The size of components in constructionist systems built to date varies from "a few" to "dozens", depending on which system you look at. To take some concrete examples, in our own single-mind (as opposed to multi-agent) systems we have had close to 100 modules, most of which are only a few pages of C++ code each, but often include two or three significantly larger ones (thousands of lines or more) in the full system (see e.g. [38]). Each of these may have somewhere from 0 to, at most, 20 parameters that can be changed or tuned at runtime. Such a system simply does not permit the highly dynamic communication and behavior patterns required for these sophisticated functionalities. More importantly, the architecture is too inflexible for sub-functions to be shared between modules at the gross-architecture level. In such an arrangement tight integration is precluded: For AGIs we need a tighter, deeper integration of cognitive functions; we need a methodology that allows us to design architectures composed of tens of thousands of components with ease – where the smallest component is peewee-size (the size of a medium-size C++ operation [40]) and where the combination of these into larger programs, and their management, is largely automatic.

We are looking for more than a linear increase in the power of our systems to operate reliably, and in a variety of (unforeseen) circumstances; experience strongly suggests that a linear increase in present methods will not bring this about: Nothing in traditional methods shows even a hint of how more flexible systems could be built using that approach. One of the biggest challenges in AI today is to move away from brittle, limited-domain systems. Hand-crafted software systems tend to break very easily, for example when taking inputs outside the scope of those anticipated by their designers or because of unexpected interaction effects amongst the systems' components. What seems clear is that a new methodology must inevitably revolve around what could be thought of as "meta"-programs; programs that guide the creation of new programs that guide the system's interaction patterns with the world, and possibly test these in simulation mode beforehand, as a "mental exercise", to predict how well they might work. The systems need to have a reliable way to judge whether prior knowledge exists to solve the problem, and whether the problem or situation

[4]Although system-wide learning and self-modification could be realized in isolation by a small system, these features are likely to be impossible to maintain in a large system when taking a constructionist approach, and we need the system to incorporate all of these features in a unified manner.

falls within what the system has already encountered or whether it finds itself in completely new circumstances. So, we need a methodology for designing a system whose output, cognitive work, is a set of programs that are more or less new compared to what existed in the system before; programs that can be given a chance to guide the system in its interactions with new operating contexts (domains), and ways to assess the results of such "hypothesis testing": The programs must be compared and evaluated on the grounds of the results they achieve. It may even be necessary for the evaluation itself to be learnable, for, after all, the system should be as self-organizing as possible. For a large system, doing all of this with present reinforcement learning and genetic algorithm techniques is likely to be too limited, too slow, or, most probably, impossible; the system must therefore be endowed with analogy making, reasoning, and inference capabilities to support such skills.

If we can create systems that can adapt their operating characteristics from one context to another, and propose what would have to be fairly new techniques with a better chance of enabling the system's operation in the new envrionment, then we have created a system that can *change its own architecture.* And that is exactly what I believe we need: For any real-world domain (e.g. an indoor office environment) that is sufficiently different from another real-world domain (e.g. a busy street), creating a system that can not only operate but *learn* to operate in both, as well as in new domains, must be a system that can change its own operation in fundamentally new ways – these systems would have self-organizing architectures that largely manage their own growth.

9.5 Towards a New Constructivist AI

A system that can learn to change its own architecture in sensible ways is a *constructivist AI* system, as it is to some significant extent *self-constructive* [36]. The name and inspiration comes partly from Piaget [26], who argued that during their youth humans develop cognitive faculties via a self-directed "constructive" process, emphasizing the *active* role that a learning mind itself has in any learning process. Piaget's ideas later became the foundation for the work by Drescher [7], who brought this idea to artificial intelligence, arguing that AI should study the way minds *grow*. The present work shares Drescher's aim for more powerful ways of learning, aligning with the general hypothesis behind his work that an (artificial) mind requires sophisticated abilities to build representations and grow its knowledge about the world based on its own direct experiences. But in the present work we take this idea a step further by arguing for a fundamental change in *methodolog-*

ical assumptions, emphasizing the need for new principles of automatic management of *whole AI architectures* – i.e. the mind itself: It is not only the system's learning but the control structures themselves that must be part of such cognitive development and constant (auto-)construction. The methodologies employed for such AI development – constructivist AI – are likely to be *qualitatively different* from today's software development methods.

Here a drive for constructivist AI – that is, a system that can modify its own internal structures to improve its own operational characteristics, based on experience – arises thus from two fundamental assumptions, namely that (a) constructionist methodologies (i.e. by and large all traditional software development techniques) are not sufficient – both in principle and in practice – to realize systems with artificial general intelligence; (b) the hypothesis that automation of not just parameter tuning or control of (coarse-grained) module operation but of *architectural construction* (as understood in computer science) and management is what is needed to address this shortcoming. This is in line with the ideas presented by the proponents of second-order cybernetics and cybernetic epistemology [42] [12], which studies the nature of self-regulation.

In our approach we are by and large ruling out *all* methodologies that require some form of hand-coding of domain-level operational functionality (what Wang metaphorically referred to as "tools" [43]), as well as any and all approaches that require extensive hand-coding of the final static architecture for an artificial general intelligence (metaphorically referred to as "hand" by [43], limiting initial (manual) construction to a form of "seed" – a kind of meta-program – which automates the management of all levels below. This is what we call constructivist AI. Pure constructivist systems do not exist yet in practice, but some theoretical work already shows promise in this direction.

The following are topics that I consider likely to play a critical role in the impending paradigm shift towards constructivist AI. The topics are: *Temporal grounding*, *feedback loops*, *pan-architectural pattern matching*, *small white-box components*, and *architecture meta-programming and integration*. Other key factors probably exist that are not included here, so this list should be considered a necessary but insufficient set of topics to focus on in the coming decades as we turn our sights to building larger, more self-organizing systems.

9.5.1 *Temporal Grounding*

As now seems widely accepted in the AI community, it is fairly useless to talk of an entity being "intelligent" without referencing the context in which the entity operates. In-

telligence must be judged by its behavioral effects on the world in particular circumstances which are not part of the entity's operation: We cannot rightfully show an entity to be smart unless we consider both what the entity does and what kinds of challenges the environment presents. This means that intelligent behavior requires *grounding* – a meaningful "hook-up" between an intelligent system's thoughts and the world in which it operates. This grounding must include a connection to both space and time: Ignoring either would cripple the entity's possibility of acting intelligently in its world, whether real or virtual. So, for one, the entity must have a means to be *situated* in this world – it must have a body, a (limited) collection of sensors to accept raw data through and some (limited) set of actuators – to affect its surroundings. By the same token, its body must be connected to the entity's internal thought/computational processes to transfer the results of its thinking to the body.[5]

The issue goes beyond situatedness, which is a necessary but not sufficient condition for grounding: via situated perception and action, a feedback loop is created allowing the system to adapt to its environment and to produce models of the world that enable it to plan in that world, using predicted results of sequences of actions (plans). Results that do not match predictions become grounds for revision of its models of the world, and thus enable it to learn to exist in the given environment (cf. [44]). Therefore, to be grounded, an intelligent entity must be able to compute using processes that have a causal, predictable relationship with the external reality.

This leads us to a discussion of *time*. As any student of computer science knows, computation can be discussed, scrutinized and reasoned about without regard for how long it actually takes in a particular implemented system. It may be argued that this simplification has to some extent helped advance the fields of computer science and engineering. However, lack of a stronger foundation for the semantics of the actual, realtime execution of computational operations has hampered progress in fields dealing with highly time-critical topics, such as embedded systems, user interfaces, distributed networks, and artificial intelligence systems. As others have pointed out, timeliness is a semantic property [17] and must be treated as such. To be grounded, the computational operations of an intelligent entity must have a causal, *temporally contextualized and predictable* – and thus temporally meaningful – relationship with the external reality.

To make an intelligent machine that does not understand time is a strange undertaking. No example of natural intelligence exists where time isn't integral in its operation:

[5]Froese (2007) gives a good overview of past research on these topics.

When it comes to doing intelligent things in the world, time is of the essence. Indeed, in a world without the arrow of time there would be little need for the kind of intelligence we see in nature. The lack of a strong connection between computational operations and the temporal dimension is preventing a necessary theoretical and practical understanding of the construction of large architectural solutions that operate predictably under external time constraints. We need to find ways to build an operational knowledge of external time into AI architectures.

To use its own mental powers wisely an intelligent machine must not only be able to understand the march of the real-world clock itself, it should preferably understand its own capabilities and limitations with regards to time, lest it cannot properly make plans to guide its own learning or evolution. One method is to link the execution of the software tightly with the CPUs operation and inputs from the system's operating environment to create a clear semantic relationship between logical operations and the passing of realtime (running of the CPU), in a way that allows the system to do the inferencing and modeling of this relation itself and use this in its own operation throughout the abstraction layers in the entire architecture, producing a temporally grounded system. This is not mere conjecture; in the Ikon Flux system [23] lambda terms were used to implement this idea in a system containing hundreds of thousands of such terms, showing that this is indeed possible on large architectural scales, even on present hardware. And as mentioned above, the full perception-action loop needs to be included in these operational semantics.

There are thus at least three key aspects of temporal representation. The first is the perception of *external time*. The system exists in some world; this world has a clock; any system that cannot reasonably and accurately sense time at a resolution relevant to its operation will not be able to take actions with regards to events that march along to this clock, and thus – by definition – is not intelligent. Second, an intelligent system must have a representation of *mental time* and be able to estimate how long its own mental operations take. Third, an AI architecture must understand how the first two aspects relate, so that mental actions can be planned for based on *externally or internally-imposed timelines and deadlines*. The challenge is how to implement this in distributed, fine-grained architectures with parallel execution of subcomponents.

9.5.2 *Feedback Loops*

Any generally intelligent system operating in the real world, or a world of equivalent complexity, will have vastly greater information available than the mind of such a system

will have time to process. Thus, only a tiny fraction of the available data in the world is being processed by the system at any point in time. The actual data selected to be thought about must be selected based on the system's goals – of which there will be many, for any system of reasonable complexity. To be able to respond to unexpected events the system must further divide its processing capability between thinking about long-term things (those that have not happened yet and are either wanted, or to be avoided), and those that require immediate processing. Any mechanism that manages how the system chooses this division is generally called *attention*, or considered a part of an attentional mechanism. Attention is intricately linked with the perception-action loop, which deals with how the system monitors for changes in the world, and how quickly it is able to respond to these. For any intelligent system in a reasonably complex environment the nature and operation of these mechanisms are of utmost importance, as they are fundamental to the cognitive makeup of the system and put limits on all its other cognitive abilities.

Focus on the perception-action loop in current AI curricula is minimal. A quick look at some of the more popular textbooks on the subject reveals hardly any mention of the subject. Given that this most important loop in intelligent systems is largely ignored in the mainstream AI literature, it is no surprise that the little discussion there is dwells on their trivial aspects. Yet the only means for intelligent systems to achieve stability far from (thermodynamic) equilibrium is through feedback loops. The growth of a system, and its adaptation to genuinely new contexts, must rely on feedback loops to stabilise the system and protect it from collapsing. Furthermore, any expansion or modification of existing skills or capabilities, whether it is to support more complex inferencing, making skills more general-purpose or improving the performance on a particular task, requires an evaluation feedback loop. For general-purpose intelligence such loops need to permeate the intelligence architecture.

The way any entity achieves grounding is through feedback loops: repeated interactions which serve as experiments on the context in which the entity finds itself, and the abstractions which it has built of that context, of its own actions, and of its tasks. Feedback loops related to the complexities of tasks are key to a system's ability to learn particular tasks, and about the world. But the process must involve not only experience (feedback) of its actions on the context *outside* itself, it must also involve the context of its *internal processes*. So self-modeling is a necessary part of any intelligent being. The feedback loop between the system's thinking and its evaluation of its own cognitive actions is therefore just as important as that to the external world, because this determines the system's

ability to *learn to learn*. This, it can be argued, is a key aspect of general intelligence. Together these extremely important feedback loops provide a foundation for any increases in a system's intelligence.

Self-organization requires feedback loops, and constructionist AI methodologies make no contributions in this respect. The science of self-organization is a young discipline that has made relatively slow progress (cf. [30, 46]). As a result, concrete results are hard to come by (cf. [14]). Perhaps one of the important contributions that this field has to offer at present is to show how the principles behind self-organization call for a way of thinking that is very different from that employed in traditional software development methodologies.

9.5.3 *Pan-Architectural Pattern Matching*

Complex, tightly-integrated intelligence architectures will not work without large-scale pattern matching, that is, pattern matching that involves large portions of the system itself, both in knowledge of domain (e.g. tasks, contexts, objects, etc.) and of architecture (e.g. perception and action structures, hypothesis generation methods, etc.). Such pattern-matching functionality plays many roles; I will mention a few.

Any creature living in a complex world must be able to classify and remember the salient features of a large number of contexts.[6] Without knowing *which* features to remember, as is bound to happen regularly as various new contexts are encountered, it must store *potential* features – a much larger set than the (ultimately) relevant features – and subsequently hone these as it experiences an increasingly larger numbers of contexts over time. In a complex environment like the real-world, the number of potential states or *task-relevant contexts* a being may find itself in is virtually infinite. Yet the being's processing power, unlike the number of contexts it may find itself in, is finite. So, as already mentioned, it must have some sort of attention.[7] At any point in time the attentional mechanism selects memories, mental processes, memories of having applied/used a mental process for a particular purpose, or all of the above, to determine which mental process to apply in the present, identify potential for improvement, or simply for the purpose of reminiscing about the past. The pan-architectural nature of such mechanisms crystallizes in the rather large amounts of recall required for prior patterns involving not only features of objects to

[6] One way to think of contextual change, and hence the differentiators between one context and another, is as the smallest amount of change in a particular environmental configuration that renders a priorly successful behavior for achieving a particular goal in that configuration unable to achieve that goal after the change.

[7] Here "attention" refers to a broader set of actions than our typical introspective notion of attention, including the global control of parts of the mind that are active at any point in time, as well as what each one is doing.

be recognized, or the contexts in which these objects (including the creature itself) may be at any point, but also involving the way in which the being controls its attention in these contexts with regards to its task (something which it must also be able to learn), the various analogies it has made for the purpose of choosing a course of action, and the mechanisms that made these analogies possible. To do all this in realtime in one and the same system is obviously a challenge. This will, however, be impossible without the ability to compare key portions of the system's knowledge and control structures through large-scale pattern matching.

Yet another example of a process for which such transversal pattern matching is important is the growth of the system as it gets smarter with experience. To grow in a particular way, according to some specification,[8] the architecture must have built-in ways to compare its own status between days, months, and years, and verify that this growth is according to the specification. This might involve pattern matching of large parts of the realtime mind, that is, the part of the mind that controls the creature from moment to moment at different points in time. For a large, heterogeneous architecture such architecture-scale pattern matching can get quite complicated. But it is unlikely that we will ever build highly intelligent artificial systems without it.

9.5.4 *Transparent Operational Semantics*

As already discussed, most integration in AI systems has involved relatively small numbers of the functions found in natural minds. A close look at these components – whether they are computer vision, speech recognition, navigation capabilities, planning, or other such specialized mechanisms – reveals internals with an intricate structure based on programming languages with syntax targeted for human programmers, involving mixtures of commercial and home-brewed algorithms. The syntax and semantics of "black-box" internals is difficult or impossible to discover from the outside, by observing only their inputs, outputs, and behaviors. This is a critical issue in self-organizing systems: The more opaque complex mechanisms encompassed by any single component in an architecture are, the harder it is to understand its operational semantics. In other words, the greater the complexity of atomic components, the greater the intelligence required to understand them. For this reason, to make architectures that construct themselves, we must move away from large black-box components.

[8] Such a specification could be small or medium-sized, compared to the resulting system, and it could be evolved, as our DNA has been, or provided via new meta-programming methods.

But the grand goal of self-construction cuts even deeper than dictating the size of our building blocks: It calls for them to have a somewhat different nature. Without exception, present programming languages used in AI are designed for human interpretation. By requiring human-level intelligence to be understood, these programming languages have little chance of being interpreted by other software programs *automatically*; their operational semantics are well above a semantic threshold of complexity that can be understood by automatic methods presently available. Creating systems that can inspect their own code, for the purpose of improving their own operation, thus requires that we first solve the problem we started out to solve, namely, the creation of an artificial general intelligence. We need to move away from programming languages with complex syntax and semantics (*all* programming languages intended for humans), towards transparent programming languages with simple syntax and simple operational semantics. Such programming languages must have fewer basic atomic operations, and their combinatorics would be based on simpler principles than current programming languages. A foundational mechanism of such a programming language is likely to be small and large-scale pattern matching: Systems built with it would likely be fairly uniform in their semantics, from the small scale to the gross architecture level, as then the same small number of pattern matching operations could be used throughout the system – regardless of the level of granularity – to detect, compare, add, delete, and improve any function implemented at any level of detail, from code snippets to large architectural constructs.

Small "white-box" (transparent) components, executed asynchronously, where each component implements one of only a few primitive functions, could help streamline the assembly of architectural components. As long as each component is based on a few fundamental principles, it can easily be inspected; assemblies of these will thus also be easily inspectable, and in turn enable the detection (identification, analysis, and modification) of functional patterns realized by even large parts of an architecture. This is a prerequisite for implementing automatic evaluation and learning operational semantics, which lies at the heart of constructivist AI. Incidentally, this is also what is called for to enable transversal functions implementing the kinds of introspection, system-wide learning and dynamic attention which I have already argued as being necessary for artificial general intelligence.

How small need the components be? Elsewhere we have pointed out the need to move towards what we call "peewee-size" granularity [40] – systems composed of hundreds of

thousands of small modules, possibly millions.[9] To see why the size of the smallest modifiable entities must be small, we need only look at a hypothetical cognitive operation involving hierarchical summation of excitation signals at several levels of detail: at the lowest as well as the highest levels the system – e.g. for improving its operation – may need to change addition to subtraction in particular places. In this case the operation being modified is addition. In a large system we are likely to find a vast number of such cases where, during its growth, a system needs to modify its operation at such low levels of functional detail. If each peewee-size module is no larger than a lambda term or small function written in e.g. C++ we have reached the "code level," as normally meant by that term – this should thus be of a sufficiently low-level of granularity for building highly flexible systems: self-constructive on temporal and complexity scales that we consider useful, yet running on hardware that is already available.

I am aware of only one architecture that has actually implemented such an approach, the *Loki* system, which was built using the Ikon Flux framework [23], a system based on Lambda terms. The system implemented a live virtual performer in the play Roma Amor which ran for a number of months at Cite des Sciences et de L'Industrie in Paris in 2005, proving beyond question that this approach is tractable. Whether the extremely small size of peewee granularity is *required* for self-construction or whether larger components can be used is an important question that we are unable to answer at the moment. But whatever their size, the components – architectural building blocks – must be expressible using simple syntax, as rich syntax begets rich semantics, and rich semantics call for smarter self-inspection mechanisms whose required smarts eventually rise above a threshold of complexity beyond which self-construction and self-organization can be bootstrapped, capsizing the whole attempt. The finer the granularity and the simpler the syntax the more likely it is to succeed in this regard; however, there may also be a lower bound, i.e. building blocks should not be "too simple"; if the operational semantics is kept at a level that is "small enough but not smaller" than what can support self-inspection, there is reason to believe a window opens up within which both practical and powerful components can be realized.

[9] These numbers can be thought of as a rough guide – the actual number for any such architecture will of course depend on a host of things that are hard to currently foresee, including processor speed, cost of memory transactions, architectural distributedness, and more.

9.5.5 *Integration and Architecture Metaconstruction*

Architectural meta-programming is needed to handle larger and more complex systems [3, 31], scaling up to systems with architectural designs that are more complex than even the most complex systems yet engineered, such as microprocessors, the Terrestrial telephone network, or the largest known natural neural networks [24]. A cognitive architecture supporting many of the features seen in natural intelligence will be highly coordinated and highly *integrated* – more so than probably any man-made dynamic system today. All of the issues already discussed in this section are relevant to achieving archiectural meta-programming and integration: General principles for learning a variety of contexts can equally well be applied to the architecture itself, which then becomes yet another context.

Constructivist AI will certainly be easier if we find a "cognitive principle" as hypothesized by [6], where the same small set of basic principles can be used throughout to construct every function of a cognitive system. Perhaps fractal architectures – exhibiting self-similarity at multiple levels of granularity – based on simple operational semantics is just that principle. But it is fairly unlikely that a single principle alone will open up the doors to artificial general intelligence – I find it more likely to be based around something like the numerous electro-spatio-temporal principles, rooted in physics and chemistry, that make a car engine run than, say, the principle of flight. Either way, there is no getting around focusing on methods that deal more efficiently with large, distributed, semi-autonomously evolving architectures with heterogeneous functionality and a high degree of flexibility at coarse-grain levels. New meta-programming languages, constructivist design methodologies, and powerful visualization systems must be developed for significant progress to be made. Transversal functions, as described above, are what ultimately forces us to look at the whole architecture when thinking about our new methodology: Without taking the operation of the whole into account, pan-architectural features are precluded. The transition to architectures built via a constructivist methodology will be challenging.

9.6 Conclusions

The hope for generally intelligent machines has not disappeared, yet functions critical to generally intelligent systems continue to be largely ignored, examples being the ability to learn to operate in new environments, introspection, and pan-architectural attention.

Systems whose gross architecture is mostly designed from the top-down and programmed by hand, constructionist AI, has been the norm since the field's inception, more

than 50 years ago. Few methodologies have been proposed specifically for AI and researchers have relied on standard software methodologies (with small additions and modifications), one of the more popular ones in recent times being object-oriented programming and component-based architectures. As far as large AI systems go these methodologies rely on fairly primitive tools for integration and have generally resulted in brittle systems with little or no adaptation ability and targeted domain application.

Based on the weaknesses inherent in current practices, the conclusion argued for here is that the limitations of present AI software systems cannot be addressed through incremental improvement of current practices, even assuming continued exponential growth of computing power, because the approach has fundamental theoretical and practical limitations: AI systems built to date show that while some integration is possible using current software development methods and extensions thereof (cf. [37]), the kind of deep integration needed for developing general artificial intelligence is unlikely to be attained this way. Standard software development methods do not scale; they assume that their results can be linearly combined, but this is unlikely to produce systemic features that seem key in general intelligence; too many difficult problems, including a freely roving attentional mechanism, equally capable of real-world inspection and introspection, system-wide learning, cognitive growth and improvement, to take some key examples, would be left by the wayside. To create generally intelligent systems we will need to build significantly larger and more complex systems than can be built with present methods.

To address the limitations of present methodologies a paradigm shift is needed – a shift towards *constructivist AI*, comprised of new methodologies that emphasize auto-generated code and self-organization. Constructivist AI calls for a very different approach than offered by traditional software methodologies, shifting the focus from manual "brick-laying" to creating the equivalent of self-constructing "factories". Architectures would be formed through interactions between flexible autonomous auto-construction principles, where a complex environment and initial "seed" code would interact to automatically create the kinds of architectures needed for general-purpose intelligence.

In this paper I have outlined some of the key topics that need to be advanced in order for this paradigm shift to happen, including a stronger emphasis on feedback loops, temporal grounding, architecture metaprogramming and integration, pan-architectural pattern matching and transparent operational semantics with small white-box components. The list, while non-exhaustive, clearly illustrates the relatively large shift in focus that needs to happen, as most of these topics are not well understood today. To increase our

chances of progress towards artificial general intelligence, future work in AI should focus on constructivist-based tools including new development environments, programming languages, and architectural metaconstruction principles.

Acknowledgments

This is an updated version of my BICA 2009 keynote paper [36]. I would like to thank Eric Nivel for brilliant insights and numerous discussions on these topics, as well as excellent suggestions for improving the paper. Big thanks to Pei Wang, Hannes H. Vilhjalmsson, Deon Garrett, Kevin McGee, Hrafn Th. Thorisson and Gudny R. Jonsdottir for valuable comments and suggestions for improvement, and to the anonymous reviewers for helpful comments. Thanks to the HUMANOBS team for helping lay the groundwork for an exciting future. This work was supported in part by the EU-funded project HUMANOBS: Humanoids That Learn Socio-Communicative Skills Through Observation, contract no. FP7-STREP-231453 (www.humanobs.org), and by a Strategic Research Programme Centres of Excellence and Research Clusters 2009-2016 project grant (IIIM; www.iiim.is) awarded by the Science and Technology Policy Board of Iceland and managed by the Icelandic Center for Research (Rannís; www.rannis.is).

Bibliography

[1] Anderson, J. R. (1996). Act: A simple theory of complex cognition, *American Psychologist* **51**, pp. 355–365.
[2] Barrett, H. C. and Kurzban, R. (2006). Modularity in cognition: Framing the debate, *Psychological Revivew* **113(3)**, pp. 628–647.
[3] Baum, E. B. (2009). Project to build programs that understand, in *Proceedings of the Second Conference on Artificial General Intelligence*, pp. 1–6.
[4] Brooks, R. A. (1986). Robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2(1)**, p. 14–23.
[5] Bryson, J. (2003). The behavior-oriented design of modular agent intelligence, *Lecture notes in computer science* **2592**, pp. 61–76.
[6] Cassimatis, N. (2006). A cognitive substrate for achieving human-level intelligence, *A.I. Magazine* **27(2)**, pp. 45–56.
[7] Drescher, G. L. (1991). *Made-up minds: a constructivist approach to artificial intelligence* (M.I.T. Press, Boston, Massachusetts).
[8] Franklin, S. (2011). Global workspace theory, shanahan, and LIDA, *International Journal of Machine Consciousness* **3(2)**.
[9] Froese, T. (2007). On the role of AI in the ongoing paradigm shift within the cognitive sciences, *50 Years of Artificial Intelligence - Lecture Notes in Computer Science* **4850**, pp. 63–75.
[10] Garel, S. and Rubenstein, J. L. R. (2004). Patterning of the cerebral cortex, in M. S. Gazzaniga (ed.), *The Cognitive Neurosciences III*, pp. 69–84.

[11] Garlan, D., Allen, R. and Ockerbloom, J. (1995). Architectural mismatch or why it's hard to build systems out of existing parts, in *Proceedings of the Seventeenth International Conference on Software Engineering*, pp. 179–185.

[12] Heylighen, F. and Joslyn, C. (2001). Cybernetics and second order cybernetics, in R. A. Mayers (ed.), *Encyclopedia of Physical Science and Technology*, Vol. 4 (Academic Press), pp. 155–170.

[13] Hsiao, K., Gorniak, P. and Roy, D. (2005). Netp: A network API for building heterogeneous modular intelligent systems, in *Proceedings of AAAI 2005 Workshop on modular construction of human-like intelligence. AAAI Technical Report WS-05–08*, pp. 24–31.

[14] Iizuka, H. and Paolo, E. A. D. (2007). Toward spinozist robotics: Exploring the minimal dynamics of behavioural preference, *Adaptive Behavior* **15(4)**, pp. 359–376.

[15] Jonsdottir, G. R., Thórisson, K. R. and Eric, N. (2008). Learning smooth, human-like turntaking in realtime dialogue, in *IVA '08: Proceedings of the 8th international conference on Intelligent Virtual Agents* (Springer-Verlag, Berlin, Heidelberg), ISBN 978-3-540-85482-1, pp. 162–175, `http://dx.doi.org/10.1007/978-3-540-85483-8_17`.

[16] Laird, J. (2008). Extending the soar cognitive architecture, in *Proceedings of the 2008 conference on Artificial General Intelligence*, pp. 224–235.

[17] Lee, E. E. (2009). Computing needs time, *Communications of the ACM* **52(5)**, pp. 70–79.

[18] Martin, D., Cheyer, A. and Moran, D. (1999). The open agent architecture: A framework for building distributed software systems, *Applied Artificial Intelligence* **13(1-2)**, pp. 91–128.

[19] Moore, G. E. (1965). Cramming more components onto integrated circuits, *Electronics Review* **31(8)**.

[20] Newell, A. and Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search, *Communications of the Association for Computing Machinery* **19(3)**, p. 113–126.

[21] Ng-Thow-Hing, V., List, T., Thórisson, K. R., Lim, J. and Wormer, J. (2007). Design and evaluation of communication middleware in a distributed humanoid robot architecture, in *IROS '07 Workshop: Measures and Procedures for the Evaluation of Robot Architectures and Middleware*.

[22] Ng-Thow-Hing, V., Thórisson, K. R., Sarvadevabhatla, R. K., Wormer, J. and List, T. (2009). Cognitive map architecture: Faciliation of human-robot interaction in humanoid robots, *IEEE Robotics & Automation* **16(1)**, pp. 55–66.

[23] Nivel, E. (2007). Ikon flux 2.0, Tech. rep., Reykjavik University Department of Computer Science, technical Report RUTR-CS07006.

[24] Oshio, K., Morita, S., Osana, Y. and Oka, K. (1998). C. elegans synaptic connectivity data, *Technical Report of CCeP, Keio Future, No. 1, Keyo University* .

[25] Pezzulo, G. and Calvi, G. (2007). Designing modular architectures in the framework akira, *Multiagent and Grid Systems* , pp. 65–86.

[26] Piaget, J. (1950). *The Psychology of Intelligence* (Routledge and Kegan Paul, London, England).

[27] Pollock, J. L. (2008). Oscar: An architecture for generally intelligent agents, *Frontiers in Artificial Intelligence and Applications* **171**, pp. 275–286.

[28] Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture, in J. Allen, R. Fikes and E. Sandewall (eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning* (Morgan Kaufmann publishers Inc.: San Mateo, CA, USA), pp. 473–484.

[29] Saemundsson, R. J., Thórisson, K. R., Jonsdottir, G. R., Arinbjarnar, M., Finnsson, H., Gudnason, H., Hafsteinsson, V., Hannesson, G., Isleifsdottir, J., Jóhannsson, Á., Kristjansson, G. and Sigmundarson, S. (2006). Modular simulation of knowledge development in industry: A multi-level framework, in *WEHIA - Proc. of the First Intl. Conf. on Economic Science with Heterogeneous Interacting Agents* (Bologna, Italy).

[30] Salthe, S. N. and Matsuno, K. (1995). Self-organization in hierarchical systems, *Journal of Social and Evolurionary Systems* **18(4)**, pp. 327–3.

[31] Sanz, R., López, I., Rodríguez, M. and Hernandéz, C. (2007). Principles for consciousness in integrated cognitive control, in *Neural Networks*, Vol. 20, pp. 938–946.

[32] Sun, R., Merrill, E. and Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning, *Cognitive Science* **25**, pp. 203–244.

[33] Sutton, P., Arkins, R. and Segall, B. (2001). Supporting disconnectedness - transparent information delivery for mobile and invisible computing, in *CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid*, pp. 277–285.

[34] Swanson, L. W. (2001). Interactive brain maps and atlases, in M. A. Arbib and J. S. Grethe (eds.), *Computing the Brain* (Academic Press), pp. 167–177.

[35] Thórisson, K. R. (2008). Modeling multimodal communication as a complex system. in I. Wachsmuth and G. Knoblich (eds.), *ZiF Workshop*, *Lecture Notes in Computer Science*, Vol. 4930 (Springer), ISBN 978-3-540-79036-5, pp. 143–168.

[36] Thórisson, K. R. (2009). From constructionist to constructivist A.I. in A. Samsonovich (ed.), *Keynote, AAAI Fall Symposium Series: Biologically Inspired Cognitive Architectures; AAAI Tech Report FS-09-01* (AAAI press), pp. 175–183.

[37] Thórisson, K. R., Benko, H., Arnold, A., Abramov, D., Maskey, S. and Vaseekaran, A. (2004). Constructionist design methodology for interactive intelligences, *A.I. Magazine* **25(4)**, pp. 77–90.

[38] Thórisson, K. R. and Jonsdottir, G. R. (2008). A granular architecture for dynamic realtime dialogue, in *Intelligent Virtual Agents, IVA08*, pp. 1–3.

[39] Thórisson, K. R., List, T., Pennock, C. and DiPirro, J. (2005). Whiteboards: Scheduling blackboards for semantic routing of messages & streams. in *AAAI-05, AAAI Technical Report WS-05-08*, pp. 8–15.

[40] Thórisson, K. R. and Nivel, E. (2009). Achieving artificial general intelligence through peewee granularity, in *Proceedings of the Second Conference on Artificial General Intelligence*, pp. 222–223.

[41] van Gelder, T. J. (1995). What might cognition be, if not computation? *Journal of Philosophy* **91**, pp. 345–381.

[42] von Foerster, H. (1995). *The Cybernetics of Cybernetics (2nd edition)*, 2nd edn. (FutureSystems Inc.).

[43] Wang, P. (2004). Toward a unified artificial intelligence, in *In Papers from the 2004 AAAI Fall Symposium on Achieving Human-Level Intelligence through Integrated Research and Systems*, pp. 83–90.

[44] Wang, P. (2005). Experience-grounded semantics: A theory for intelligent systems, in *Cognitive Systems Research* (Springer-Verlag), pp. 282–302.

[45] Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence* (Springer).

[46] Zitterbart, M. and De Meer, H. (eds.) (2011). *International Workshop on self-organizing systems (IWSOS 2011)* (Springer, New York, NY, USA).