Biologically Inspired Cognitive Architectures II:
Papers from the AAAI Fall Symposium (FS-09-01)

From Constructionist to Constructivist A.I.

Kristinn R. Thórisson
Center for Analysis & Design of Intelligent Agents and School of Computer Science
Reykjavik University, Kringlunni 1, IS-103 Reykjavik, Iceland
thorisson@ru.is

Abstract

The development of artificial intelligence systems has to date been largely one of manual labor. This *Constructionist* approach to A.I. has resulted in a diverse set of isolated solutions to relatively small problems. Small success stories of putting these pieces together in robotics, for example, has made people optimistic that continuing on this path would lead to artificial general intelligence. This is unlikely. "The A.I. problem" has been divided up without much guidance from science or theory, resulting in a fragmentation of the research community and a set of grossly incompatible approaches. Standard software development methods come with serious limitations in scaling; in A.I. the Constructionist approach results in systems with limited domain application and severe performance brittleness. Genuine integration, as required for general intelligence, is therefore practically and theoretically precluded. Yet going beyond current A.I. systems requires significantly more complex integration than attempted to date, especially regarding transversal functions such as attention and learning. The only way to address the challenge is replacing top-down architectural design as a major development methodology with methods focusing on self-generated code and self-organizing architectures. I call this *Constructivist A.I.*, in reference to the self-constructive principles on which it must be based. Methodologies employed for Constructivist A.I. will be very different from today's software development methods. In this paper I describe the argument in detail and examine some of the implications of this impending paradigm shift.

Introduction

Compared to a cat or a mouse a software program whose entire operation consists of classifying objects into groups labeled "high", "medium" and "low", along a few dimensions, is by most measures not very intelligent. Yet the majority of A.I. systems developed today are not much smarter than that. They are geared towards targeted, isolated problems. Most of them are dumber than a fruit fly.

That industrial applications should call for targeted A.I. systems is not a surprise: Any product market will present practical, limited problems that call for well-defined, limited solutions. Solving these problems is a predictable and manageable engineering effort, and it allows companies to get ahead in the market. Why academia should have a limited focus in their A.I. research is also understandable, albeit not as obvious. It is, however, more difficult to excuse.

Computer science has supplied many useful tools to the study of intelligent systems, most notably the ability to build runnable models, which allow us to experiment with our theories and ultimately demonstrate their soundness. An important determinant of speed of progress is how such models are implemented; typically this is done by a single coder or a small team using common software development methods. Following – and sometimes slightly ahead of – standard software methodologies, A.I. systems are built by hand, the A.I. developer taking the role of a "construction worker". This *Constructionist A.I.* approach goes hand-in-hand with the traditional divide-and-conquer methodology: dividing the problem up into small enough pieces so that a student or a small team of researchers can find a solution to one of them within a few years.

Employing the divide-and-conquer method as the main way towards *complete understanding* of a phenomenon only works in combination with a subsequent effort – not nearly as often discussed – whereby the resulting theories are synthesized to give a unified picture. For any subject of study there is no guarantee that a set of models or theories developed independently for its isolated sub-parts can be combined in a straightforward manner to form a complete theory – even though the components seem satisfactorily explainable by themselves. Differences in type, scope, and underlying assumptions of the smaller-scope theories may partially or completely prevent their linear combination. Such a failure would preclude a holistic understanding of the phenomenon under investigation.[1] This is especially likely to happen when using a traditional divide-and-conquer approach for studying dynamic phenomena with a mixture of dense and loose couplings[2] between a large number of sub-

[1] Some success of combining theories in other scientific fields may have spurred optimism that the same could be done in A.I. We should be reminded, however, that in fields where this has been done, e.g. chemistry and physics, it has typically been an excruciatingly slow process, extending over centuries and millennia.

[2] The density of couplings is the sheer number of connections;

systems: The functional state space of such systems can get exceedingly large (consider e.g. the state space of a large city[3]) and any subdivision will ignore important interconnections between the various parts. Moreover, in systems with little replication of functionality, every part must be understood to explain the operation of the whole. As I have argued elsewhere, the human mind[4] embodies the properties of a heterogeneous, large, densely-coupled system (HeLD; Thórisson, 2008). Such systems are notoriously difficult to understand and model.

I would like to stress that here we are primarily concerned with methodologies for developing A.I. systems, not the particular cognitive architectures that get built through their application.[5] So to a large extent the soundness of the arguments made in this paper do not rest on – and do not *need* to rest on – assumptions about what *kind* of a system the human and animal mind really is. Past discussions about cognitive architecture – whether the mind is primarily "symbol-based" (c.f. Newell and Simon, 1976), "dynamical" (c.f. van Gelder, 1995, Froese, 2007), "massively modular" (c.f. Barrett and Kurzban, 2006) (however these concepts are interpreted) or something else entirely – can be largely put aside as we focus on the *size* of the system and *interdependencies* between its enormous set of functions: Just as an ecosystem cannot be understood by studying one lake and three inhabiting animal species, intelligence cannot be understood as a phenomenon by studying only a few of its features – critical interdependencies of its relatively large number of heterogenous mechanisms prevent most disections from providing more than a small insight into the big picture.

Consider functions 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 ones own thoughts. These are features that seem simply too critical to leave out when trying to build an intelligent system.[6] It is in part the neglect of such key features – and in particular *their integration* – that motivates the present discussion.

In academia "the A.I. problem" has largely been divided up based on concerns for potential application areas, not based on scientific or theoretical guidelines, possibly because we do not (yet) have very good tools and methodologies for addressing them all in a single system. As a result the field has ignored important holistic features of intelligence, such as a system-wide ability to learn and the power of globally-steerable attention, and severely limited the chances that progress in isolated subfields of A.I. can be put back together to form a more complete picture of what intelligence can or should be.

While results coming out of A.I. labs will undoubtedly continue to be useful to society, I will show how the above factors are colluding to preclude progress towards understanding the larger picture and the general nature of intelligence, and discuss some of the research topics we need to focus on to see significant progress in A.I. research over the next 50 years.

Constructionist A.I.

Creating genuinely multi-purpose machines that can operate in the real world (virtual or physical) requires, among other things, integrated manipulation and sensing capabilities. Putting together the sensory-motor capabilities to support even just basic functionality for this purpose has been a major challenge. To take an example, the annual Robot Challenge competition at AAAI calls for competing teams to create a social robot that, starting at the conference building's entrance, can locate the registration counter and register for the conference, find its way to the proper lecture hall and give a two-minute lecture about itself, as well as take questions from its (all-human, as of yet) audience. The challenge demonstrates nicely the multiple skills that must be closely coordinated to solve many tasks that people face every day, such as asking for directions, finding an elevator and operating its controls, navigating hallways, opening doors and doing presentations. Integrating the necessary competencies to create such social robots is a major undertaking, as shown in the entrances to the competition over the years, some involving several institutions and close to 20 researchers (c.f. Simmons et al., 2003).

To illustrate this point I will take two other examples, from my own research.[7] These systems, while probably not the largest out there, are representative of efforts towards integration such as that in the AAAI Robot Challenge, and they are among the relatively few that make it a goal to push the envelope on size and breadth of integration.

The Cognitive Map architecture (Ng-Thow-Hing et al., 2009) has been used for endow the Honda ASIMO robot with various skills, including playing board games with children. Integrating vision, speech and gesture, the architec-

the looseness/tightness of couplings is their rigidity (Thórisson, 2008).

[3] "Functional state space" refers to the states which matter to the system's operation – for a city it would include e.g. the position of every person at every moment and the ordering of bytes in the city's phone calls, as both could affect the city's operation if they were changed, but it would exclude e.g. the placement of living cells in the inhabitants' livers and the exact number of pebbles in the city's pavements.

[4] And by extension, the minds of a large portion of Earth's animals, as these implement much of the same functionality.

[5] Although the two are connected, I have elsewhere extensively discussed how the computational architecture (software implementation) and the cognitive architecture that it simulates need not be isomorphic (Thórisson, 2008).

[6] Broadly speaking, by "generally intelligent system" I mean systems that can apply themselves to study and learn disparate tasks, facts and situations, in environments of roughly the same complexity as the real world. A fairly trivial example would be an A.I. that can learn to fix automobiles, do the dishes and cook dinner. An A.I. that can acquire the skills to invent new things, solve global warming and negotiate peace treaties would also pass.

[7] Discussing only two systems in more detail may invite criticism that the conclusions (reviewed in the next section) may be particular to these systems, or to the limited class to which they belong. However, making general claims without concrete examples is difficult. This way, at least, readers can judge for themselves how generally the arguments apply.

ture has a relatively large set of interacting modules responsible for various parts of the robot's operations, including numerous perceptors for detecting and indexing perceived phenomena, as well as various types of deciders, spatial and semantic memory systems, action controllers, etc. Each of these is a hand-crafted piece of software, counting anywhere from a few dozens lines of code to tens of thousands. In this architecture some of the smaller ones, like two types of perceptors called Detectors and Indexers (Ng-Thow-Hing et al., 2007), include interpreted code that makes the robot's visual routines more flexible than if they were developed with compiled software only.

Another example is an autonomous radio-show host that can learn to avoid interruptions and awkward pauses when talking to people (Jonsdottir and Thórisson, 2009). The system is possibly the first to actually learn politeness in conversation. It does this on the fly, during interaction, and is able to quickly adapt its speaking patterns to a large set of individuals. A network of perceptual and decision modules, along with a couple of special learner modules, process raw data, keep track of history and manages the learning.

The modules in both these systems are connected via a flexible infrastructure where information flow can easily, quickly and dynamically be changed on the fly.[8] Most modules accept input (receive messages) from more than two other modules. The majority of the data produced is published on one or more publish-subscribe blackboards (called whiteboards) and shared with any subscribing modules, the number and type of which may vary significantly depending on the global state of the robot at any point in time.

The systems embody what has been called *emergent simplicity* (Bar-Yam, 1997) – they are monolithic in that every piece is essential; take any one away and they break. They are highly modular, however, in that the modules are relatively small compared to the overall size of the full architecture. Contributing to this modularity is the relatively small size of an average module's algorithms and input/output data, giving high data transparency for the whole system, and the relatively frequent activity of the average module, giving a high temporal resolution as well.[9]

A handful of methodologies have been developed over the years for building large, integrated A.I. systems of this kind, including Behavior-Oriented Design (BOD, Bryson, 2003), the Subsumption Architecture (Brooks, 1986) and Belief, Desires, Intentions (BDI; c.f. Rao and Georgeff, 1991). We have used the Constructionist Design Methodology (CDM) to build our systems, a methodology developed by myself and my collaborators over the last decade to help creating ever-growing systems (Thórisson et al., 2004). Its principles allow architecture-preserving expansion of systems spanning integration of large numbers of executables, computers, developers and even research teams. CDM speeds up the development of large software systems, beyond what standard component-based methodologies enable, and makes it easier to keep expanding them without frequent or major rewrites of the architecture. These results have been collected for several types of systems, in many contexts, primarily at three different research labs, CADIA (Jonsdottir et al., 2008, Thórisson and Jonsdottir, 2008, Saemundsson et al., 2006), Honda Research Labs (HRI) (Ng-Thow-Hing et al., 2007, Ng-Thow-Hing et al., 2009) and the Computer Graphics and User Interfaces Lab at Columbia University (Thórisson et al., 2004). While our methodology differs somewhat from others proposed for similar purposes, especially in that it is aimed at continuous incremental construction and, perhaps more importantly, is independent of the particular cognitive architectures being built, it is motivated by the same fundamental concern for constructing larger and smarter systems. It is also based on the same assumption as alternative methodologies in that traditional software writing and environments are taken as a core activity in the development effort – existing methodologies are all in some sense embedded within current software practices.

Limitations of Constructionist A.I.

The majority of the present-day A.I. architectures rely on traditional component-based software development methodologies. The large-scale structure of these systems is *designed* by A.I. software engineers and *constructed by hand* by software developers. In the examples reviewed above, which by no means are laggards in the state-of-the-art race, we see successful integration of a wide variety of components, some developed by the same team during the same periods (e.g. the visual perceptors modules in the Cognitive Map) while others are developed in separate parts of the world by separate companies (e.g. speech recognition). So, from a software architecture perspective, the integration called for in these social robots, to stay with that example, is not only possible, it has already been demonstrated in a number of successful systems (c.f. Rich and Sidner, 2009, Ng-Thow-Hing et al., 2009, Simmons et al., 2003, Johnson et al., 2004). We could also point to recent successful efforts focused on the internals of such systems, offering new solutions to representing perception, knowledge and action (c.f. Pezzulo, 2009, Wang, 2006, Roy, 2005).

So is there a problem? Yes. First, the heavy reliance on manual labor in these systems limits, by its very nature, the size and complexity of the architectures that can be built. The Constructionist approach results in a second equally important and critical limitation: inflexibility of the implemented systems. In any system of this kind it is not only the function of the components that is rather static, the architectures themselves – the relationship between system components – are fairly non-dynamic. We can see this in many other engineered systems of a comparable scale and level of integration, such as telephone networks, CPUs and power grids. The architecture is controlled by algorithms that are themselves hand-crafted, and thus of limited flexi-

[8] The Psyclone AIOS (Thórisson et al., 2005) has been used in both of these systems, which in our experience is the most flexible system for creating and managing large architectures. Other systems with overlapping functionality include Elvin (Sutton et al., 2001), the Open Agent Architecture (Martin et al., 1999) and NetP (Hsiao et al., 2005).

[9] Exceptions to this exist, e.g. the speech synthesizers, which is produced by a third party, containing fairly boxed-off internal operations, and the speech recognizers, also produced by a third party with no access or visibility of internal operations.

bility. This limitations precludes autonomous architectural adaptation and growth of the systems: The traditional software development methods used for hand-crafting the modules and the architecture creates dependencies in the way the modules communicate which makes them highly dependent on particular *data formats*, severely limiting the ability of the architects to take advantage of e.g. the dynamic data flow features provided by the middleware. As a result these systems are incapable of architecture-level evolution, precluding architecture-wide learning (what one might intuitively think of as "cognitive growth"). Without system-wide learning the systems cannot break out of targeted learning, which precludes general-purpose systems capable of applying themselves autonomously to arbitrary problems.

Size matters in two ways. First, because the systems are relatively small they can neither implement supernumerary nor very complex functions. Second, they are incapable of supporting many functionalities that characterize higher-level intelligences, such as system-wide analogy-making, abstraction, cross-domain knowledge, attention, all of which require transversal functionality of some sort to be of general use.[10] These issues ultimately revolve around software architecture: Most architectures built to date are coarse-grained, being built of relatively large modules.[11] Such a system simply does not permit the highly dynamic communication patterns required for these sophisticated functionalities.

The architectures of our systems reviewed above are among the finer-grained ones – we have referred to them as "granular" to emphasize this. In these systems the algorithms coordinating the modules – i.e. the gross architecture – controls dynamically 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 in the Cognitive Map (Ng-Thow-Hing et al., 2007), and learn, such as the module complex for learning turntaking in the artificial radio-show host (Jonsdottir et al., 2008). Nevertheless, the modules in these systems are typically black-box with prescribed dependencies, which make it impossible for them to automatically change their operation, expand or even modify their input and output profiles in any significant way, beyond what the coder could prescribe. The dependency problem becomes even worse as many component technologies used in these architectures include third-party software such as speech recognition, spatial navigation, object recognition, dialogue management systems, motor controllers, etc., which are legally or pragmatically closed of from any enhancements or modifications. The result is small, inflexible architectures enabling limited-intelligence systems. The methodology makes it also extremely difficult to implement transversal skills – skills such as system-wide attention (allocation of computational resources to particular issues in the system's own operation), which are necessary for the architecture to grow (Thórisson and Nivel, 2009b). With only such systems to play with and explore there is little hope of realizing more general-purpose learning or autonomous acquisition of real-world understanding.

In summary, there are thus three main factors that hamper progress towards smarter A.I. systems when relying exclusively on the Constructionist approach, their size limitation, their limited integration and their lack of flexibility. From an A.I. perspective, architectures built with the Constructionist approach are bound to be fairly non-integrated, composed out of modules that have very limited interaction capabilities. As there are generally no flexible transversal mechanisms to speak of, these systems have a low level of autonomy and cannot be applied to arbitrary problems.

Might it be possible to expand current component-based development practices to overcome these limitations? CDM (Thórisson et al., 2004) was designed with this question in mind, as a response to the realization that current component-based methodologies do not scale well. After a decade of experience in using this methodology to build A.I. architectures we can conclude that, yes, it is possible to stretch current development environments to A.I. systems that span millions of lines of code and execute on distributed clusters, yet exhibit emergent simplicity. The improvement over standard methodologies, however, is at best linear, for the foreseeable future. The same problems persist – and some get worse – as the architectures get bigger (such as code-level bugs) for lack of fault tolerance. Some new ones 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; these grow exponentially as the system gets bigger.

One of the most difficult parts of constructing A.I. systems is avoiding brittleness in their performance. Handcrafted software systems tend to break very easily, for many reasons, for example when taking inputs outside the scope of those anticipated by their designers or because of unexpected interaction effects amongst the systems' components. In spite of decades of dealing with the various problems of scaling, standard component-based software methodology simply does not scale easily, as anyone knows who has tried to integrate more than 5 or 6 separate medium-size software systems running in parallel. The limitation is a fundamental one, yet the many fields of A.I. continue to develop slices of the "intelligence cake" without regards for this fundamental fact, with the hope that someone somewhere will later combine their work with that of everyone else, resulting in a generally intelligent system. Such an event is unlikely in every sense but the most trivial one. It would mean that the principles behind generally intelligent systems are linearly composable and require little or no cross-system functionality. We simply do not know this to be true – and we have good reasons for thinking otherwise. In A.I., "putting Humpty-Dumpty back together again" in the next few decades would therefore not simply be a huge undertaking, it is likely to be

[10] Although system-wide learning and self-evolution could be realized by a small system, these features are likely to be difficult to maintain in a large system when taking a Constructionist approach.

[11] The size of components in Constructionist systems built to date varies from "a few" to "dozens", depending on which system you look at. In our own single-intelligence (as opposed to multi-agent) systems we have had over 80 modules, most of them only a few pages of C++ code each, but often including 2 or 3 significantly larger ones (thousands of lines or more) in the full system.

both practically and theoretically intractable.

The CDM notwithstanding, and in spite of today's A.I. systems clearly being useful in many ways, the Constructionist approach to A.I. has shown itself to result in systems with limited-domain application and severe performance brittleness, at least when compared to naturally intelligent systems. Ever-larger "idiot savant" systems are being built in academia and industry that show little or no hope of growing beyond their still highly-limited operating contexts. It seems inevitable that only with significantly different development methodologies could we enable a small team of developers to create massively larger, more complex and powerful architectures than possible today. These new principles must rely on increased autonomy in the construction process itself.

Towards Constructivist A.I.

The preceding discussion presents evidence that strongly hint at a need for larger and more integrated systems. How much larger? Nature, of course, gives us a rough idea, but depending on how you measure it the answer can cover quite a range. Functions clearly lacking in current A.I. systems yet abundant in nature include performance robustness, many types of context-based learning and adaptation, interference-resistant inferencing, and of course powerful cognitive growth from childhood to adulthood. The abysmal performance of current systems on these points hints at nothing less than exponentially larger systems: Going beyond current A.I. systems will require integrating significantly more complex processes than seen to date, in vastly greater numbers. Given the slow progress in both machine learning and large architecture implementation[12] current methodologies will not bring us significantly closer towards that goal: Even if we keep at it for centuries, their basic limitations are likely to asymptotically bring us to a grinding halt in the not-too-distant future.

We are looking for more than a linear increase in the power of our systems and experience strongly suggests that a linear increase in present methods will not bring this about. It is likely that the principles required to develop and govern the behavior of intelligent architectures with massive numbers of interacting components may be very different from those traditional software practices allow us to explore.

The operating assumption here is 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*, including future extensions thereof. If this hypothesis is right, and we want to see significant progress in the coming decades, we have no choice but to replace hand-crafting and top-down architectural design with self-generated code and self-organizing architectures that largely manage their own growth, without (low-level) developer intervention. These systems are auto-constructive; I call this approach *Constructivist*, in reference to its reliance on self-constructive principles. Notice that we are not arguing for the elimination of "manual" research and development methods, or even the elimination of divide-and-conquer methodologies, but rather a radical shift in wow these are employed. More importantly, we are arguing that the principles of operation of a generally intelligent architecture (a) must be vastly larger and more complex than what is possible to build (for the foreseeable future) with present standard methodologies, and that (b) it must therefore rely, in a deep sense, on principles of self-organization, not only for its evolution (how it grows from dumb to smart) but also as a general principle of its continuous operation.

It stands to reason that the methodologies employed for Constructivist A.I. will be very different from today's software development methods; they are likely to be *qualitatively different*. Following are topics that I consider likely to play a critical role in the impending paradigm shift towards Constructionist A.I. There probably exist other key factors, not included here, so this list should be considered a necessary but not sufficient set of topics to focus on in the coming decades, as we turn our sights to building larger, more self-organizing systems. The topics are: *Temporal grounding*, *feedback loops*, *pan-architectural pattern matching*, *small white-box components* and *architecture meta-programming and integration*.

Temporal Grounding

As seems now widely accepted in the A.I. community, it is fairly useless to talk of an entity being intelligent without referencing the context in which the entity operates. "Intelligence" 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 the entity and its operating environment. In other words, 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 the world. So, for one, the entity must have a means to influence the world – it must have a body. By the same token, the body must be connected to the entity's internal thought/computational processes, to transfer the results of its thinking to the body.[13]

The issue goes beyond situatedness – being situated is a necessary but not sufficient condition for grounding: via situated perception and action a feedback loop is created that allows 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 revisions of its models of the world, and thus enable it to learn to exist in the given environment (c.f. Wang, 2005). To be grounded, therefore, 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 temporality. As any student of computer science knows, computation can be dis-

[12] While these two factors are not sufficient to build more general-purpose A.I. systems, they are clearly necessary.

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

cussed, scrutinized and reasoned about without regard for how long it takes in an implemented system. This fact has allowed a number of advances within the field of computer science, mathematics and engineering. However, without a strong foundation for the semantics of the actual, realtime execution of computational operations has resulted in serious limitations which have made applications where time is of essence – notably embedded systems, user interfaces, networks, distributed and artificial intelligence systems – very difficult to model, analyze and design. As others have pointed out, timeliness is a semantic property (Lee, 2009). 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.

Interestingly, there exist no examples of natural intelligence where time isn't integral in its operation: 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 scaling of software systems and the construction of large architectural solutions that operate according to external time constraints.

To make an intelligent machine that does not understand time is a strange undertaking. To use its own "mental powers" wisely it must not only be able to understand the march of the real-world clock itself, it should preferably also understand its own capabilities and limitations with regards to time, lest it cannot properly make plans to guide its own learning or evolution. We need to find ways to build an operational knowledge of time into A.I. architectures. One way to do this is to link the execution of the software tightly with the operation of the CPU, creating a clear semantic relationship between logical operations and the passing of realtime (running of the CPU), in a way that allows the system itself to do inferencing and modeling of this relation and use it in its own operation. In the Ikon Flux system, Nivel (2007) used lambda terms to implement this idea in a system containing hundreds of thousands of such terms, showing that this is indeed possible on large architectural scales. As mentioned above, the full perception-action loop needs to be included in these operational semantics.

There are at least three aspects of temporal representation that are key. The first is the perception of external time. The system exists in some world; this world has a clock: Any system that cannot reasonably accurately sense time, at a resolution relevant to its operation, cannot 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 A.I. architecture must understand how these two 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-grain architectures with parallel execution of subcomponents.

Feedback Loops

Focus on the perception-action loop in current A.I. 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 of intelligence is ignored in the mainstream A.I. literature, it is no suprirse that little discussion of feedback loops in general can be found. Yet the only means for 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. Further, 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: repated interactions which serve as experiments on the context in which the entity finds itself, and the abstractions which it has built of that context, its own actions, and the task. Over time these result in experience which serves as the foundation for increased intelligence. This 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 science of self-organization is a relatively young discipline with slow progress (c.f. De Meer and Sterbenz, 2006, Salthe and Matsuno, 1995). Self-organization requires feedback loops, yet Constructionist A.I. methodologies make no contributions in that respect. It is therefore no surprise that concrete results are hard to come by (c.f. Iizuka and Paolo, 2007) and the study of self-organization in computer science has been largely theoretical. 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 traditional software development methodologies.

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. Such functionality plays many roles; I will mention a couple.

Any creature living in a complex world must be able to classify and remember the salient features of a large number of contexts. Without knowing *which* features to remember (as a learning creature must be able to do) it must store *potential* features – a much larger set than the (ultimately) relevant features – and subsequently hone these over time, as it experiences increasingly larger numbers of contexts. 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 it must have some sort of attention.[14] At any point in time the attentional mechanism selects one or more memories, mental process(es), memories of having applied/used a mental process for a particular purpose, or all of the above, to determine which mental process to apply currently, identify potential for improvement, or simply for the purpose of reminiscing about the past. The pan-architectural nature of such mechanisms crystalizes in the rather large amounts of recall required for prior patterns involving not only e.g. features of objects to 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 to choose 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, no surprise, a challenge.

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,[15] 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. Needless to say, it is unlikely that we will ever build highly intelligent artificial systems without it.

Small White-Box Components

As already discussed, most integration in robotics has involved relatively small numbers of components. A close look at these components – whether they are for computer vision, speech recognition, navigation capabilities, planning, or other such specialized mechanisms – reveals internals with an intricate structure based on programming languages with syntax made for human brains to understand and use, implementing broad mixtures of home-brew algorithms. The syntax and semantics of these "black-boxed" 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 larger and more complex any component in an architecture becomes the harder it is to understand its operational semantics. In other words, the greater the size and complexity of components, the greater the intelligence required to understand them. Therefore, to make architectures that construct themselves we need to move away from large, black-box components. Small "white-box" components, executed asynchronously and each implementing one of only a few primitive functions, help streamline the assembly of components, based on a few fundamental principles, and ease the detection of functional patterns realized by these assemblies. This, in turn, makes the detection and learning of operational semantics much easier. (Which incidentally is also what is called for to enable sufficient flexibility to connect components system-wide, to implement attention, system-wide learning and architecture-wide pattern matching.)

How much smaller do the components need to be? Elsewhere we have argued for the need to move towards what we call "peewee-size" granularity (Thórisson and Nivel, 2009a) – systems composed of hundreds of thousands of modules, each no larger than a lambda term or small function written in C++. We are aware of only one architecture that has actually implemented such an approach, the Loki system, which was built using the Ikon Flux framework (Nivel, 2007). The system was used in live public theater performances at Cite des Sciences et de L'Industrie in Paris, in 2005.[16] 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. Whatever the size, the components must be expressable using simple syntax, as rich syntax begets rich semantics, and rich semantics call for the need of smarter self-inspection mechanisms, eventually rising above a threshold of complexity beyond which self-construction and self-organization cannot be bootstrapped. The finer the granularity and simpler the syntax the more likely it is to succeed in this regard. The Loki system showed that the theory behind Ikon Flux is already implementable on currently available hardware using existing programming languages. It also showed uniquivocally that a lot of work remains to be done on the the principles of construction, debugging, opearation and analysis of such systems.

Architecture Meta-Programming and Integration

While it may seem obvious to some readers at this point, it is worth emphasizing the importance of striving for new principles for large architectural construction: The transition to Constructivist A.I. will be challenging. There exists no obvious recipie for how to move faster towards generally intelligent artificial systems, just as there is no magic trick to how they operate. Constructivist A.I. will certainly be easier if we find a "cognitive principle" as hypothesized by Cassimatis, 2006, where the same small set of basic principles can be used throughout to construct every function of a cognitive system. Such a principle has clearly not been found. Either way we need to set our sights on methods for dealing with large, semi-autonomously evolving architectures with heterogeneous functionality. New meta-programming languages, Constructivist design methodologies and powerful visualization systems must be developed for significant progress to be made. Architectural meta-programming is needed to handle larger and more complex systems (Baum,

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

[15] 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.

[16] The play Roma Amor, directed by J.M. Musial, ran for a number of months and was supported by grants from the French Agency for Research (ANVAR) and Ministry of Culture.

2009), 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 (Oshio et al., 1998).

Conclusions

Large systems whose gross architecture is mostly designed from the top-down and programmed by hand – *Constructionist A.I.* – has been the norm since the field's inception, more than 50 years ago. Few methodologies have been proposed specifically for A.I. and researchers have relied on standard software methodologies, with one of the more popular ones in recent times being object-oriented programming and component-based architectures. As far as large A.I. 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. This is a problem that cannot be addressed through incremental improvement of current practices and continuing exponential growth of computing power: Standard software development methods do not scale well; Constructionist A.I. has foreseeable limitations. The complexities of these systems push the limits of what small teams of researchers can handle. Systems built to date using these methods show that while integration is indeed possible, using current software development methods and extensions thereof (c.f. Thórisson et al., 2004), the kind of deep integration needed for developing general artificial intelligence is unlikely to be attained this way. Too many hard problems, including a freely roving attentional mechanism, equally capable of real-world inspection and introspection, system-wide learning, evolution 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 built to date.

To address the limitations of present methodologies a paradigm shift is needed – a shift towards *Constructivist A.I.*, comprised of new methodologies that emphasize auto-generated code and self-organization. Constructivist A.I. calls for a very different approach than offered by traditional software methodologies. Architectures would emerge out of interaction between flexible autonomous self-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 an stronger emphasis on feedback loops, temporal grounding, architecture metaprogramming and integration, pan-architectural pattern matching and small white-box components. The list, while non-exhaustive, illustrates well 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 A.I. should focus on Constructionist-based tools including new development environments, programming languages and architectural construction principles.

Acknowledgments This work was supported by the European Project HUMANOBS – *Humanoids that Learn Socio-Communicative Skills Through Observation* (grant number 231453), and by a research grant from RANNIS, Iceland. I would like to thank Eric Nivel for brilliant insights and numerous discussions on many of the topics covered, as well as excellent suggestions for improving this paper. Thanks also to Gudny R. Jonsdottir and the anonymous reviewers for helpful comments on earlier versions.

References

Bar-Yam, Y. (1997). *The Dynamics of Complex Systems*. Perseus Books, Reading, Massachusetts.

Barrett, H. C. and Kurzban, R. (2006). Modularity in cognition: Framing the debate. *Psychological Revivew*, 113(3):628–647.

Baum, E. B. (2009). Project to build programs that understand. In *Proceedings of the Second Conference on Artificial General Intelligence*.

Brooks, R. A. (1986). Robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):1423.

Bryson, J. (2003). The behavior-oriented design of modular agent intelligence. *Lecture notes in computer science*, 2592:61–76.

Cassimatis, N. (2006). A cognitive substrate for achieving human-level intelligence. *A.I. Magazine*, 27(2):45–56.

De Meer, H. and Sterbenz, J. P. G., editors (2006). *Self-Organizing Systems: First International Workshop, IWSOS 2006*. Springer, New York, NY, USA.

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*, 4850:63–75.

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-0508*, pages 24–31.

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

Johnson, W. L., Marsella, S., Mote, N., Si, M., Vilhjalmsson, H., and Wu, S. (2004). Balanced perception and action in the tactical language training system. In *Workshop on Embodied Conversational Agents: Balanced Perception and Action, AAMAS 2004: The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 19–23.

Jonsdottir, G. R. and Thórisson, K. R. (2009). Teaching computers to conduct spoken interviews: Breaking the realtime barrier with learning. In *IVA '09: Proceedings of the 9th international conference on Intelligent Virtual Agents*, Berlin, Heidelberg. Springer-Verlag.

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*, pages 162–175, Berlin, Heidelberg. Springer-Verlag.

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

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):91–128.

Newell, A. and Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19(3):113126.

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*.

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):55–66.

Nivel, E. (2007). Ikon flux 2.0. Technical report, Reykjavik University Department of Computer Science. Technical Report RUTR-CS07006.

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*.

Pezzulo, G. (2009). Dipra: A layered agent architecture which integrates practical reasoning and sensorimotor schemas. *Connection Science*.

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

Rich, C. and Sidner, C. L. (2009). Robots and avatars as hosts, advisors, companions, and jesters. *A.I. Magazine*, 30(1):29–41.

Roy, D. (2005). Semiotic schemas: A framework for grounding language in the action and perception. *Artificial Intelligence*, 167(1-2):170–205.

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.

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

Simmons, R., Goldberg, D., Goode, A., Montemerlo, M., Roy, N., Sellner, B., Urmson, C., Schultz, A., Abramson, M., Adams, W., Atrash, A., Bugajska, M., Coblenz, M., MacMahon, M., Perzanowski, D., Horswill, I., Zubek, R., Kortenkamp, D., Wolfe, B., Milam, T., and Maxwell, B. (2003). GRACE: An autonomous robot for the aaai robot challenge. *A.I. Magazine*, 24(2):51–72.

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*.

Thórisson, K. R. (2008). Modeling multimodal communication as a complex system. In Wachsmuth, I. and Knoblich, G., editors, *ZiF Workshop*, volume 4930 of *Lecture Notes in Computer Science*, pages 143–168. Springer.

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):77–90.

Thórisson, K. R. and Jonsdottir, G. R. (2008). A granular architecture for dynamic realtime dialogue. In *Intelligent Virtual Agents, IVA08*, pages 1–3.

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*, pages 8–15.

Thórisson, K. R. and Nivel, E. (2009a). Achieving artificial general intelligence through peewee granularity. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 222–223.

Thórisson, K. R. and Nivel, E. (2009b). Holistic intelligence: Transversal skills and current methodologies. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 220–221.

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

Wang, P. (2005). Experience-grounded semantics: A theory for intelligent systems. In *Cognitive Systems Research*, pages 282–302. Springer-Verlag.

Wang, P. (2006). The logic of intelligence. In *Artificial General Intelligence*, pages 31–62. Springer-Verlag.