The concept of autonomy in distributed computation and multi-agent systems

Mariusz Nowostawski
Martin Purvis
Information Science Department
The University of Otago
PO BOX 56, Dunedin, New Zealand
MNowostawski,MPurvis@infoscience.otago.ac.nz

Abstract

The concept of autonomy is a central concept in distributed computational systems and in multi-agent systems in particular. With diverse implications in philosophy and despite frequent use in social sciences and the theory of computation, autonomy *remains somewhat a vague notion. Most researchers do not discuss the details of this concept, but rather assume a general, common-sense understanding of autonomy in the context of computational multi-agent systems. We will review the existing definitions and formalisms related to the notion of autonomy. We re-introduce two concepts: relative autonomy and absolute autonomy. We argue that even though the concept of absolute autonomy does not make sense in computational settings, it is useful if treated as an assumed property of computational units. For example, the concept of autonomous agents may facilitate more flexible and robust abstract architectures. We adopt and discuss a new formalism based on results from the study of massively parallel multi-agent systems in the context of evolvable virtual machines. We also present the architecture for building such architectures based on our multi-agent system KEA, where we use the extended notion of dynamic linking. We augment our work with theoretical results from* cham *algebra for concurrent and asynchronous information processing systems. We argue that for open distributed systems, entities must be connected by multiple computational dependencies and a system as a whole must be subjected to influence from external sources. However, the exact linkages are not directly known to the computational entities themselves. This provides a useful notion and the necessary means to establish an relative autonomy in such systems.*

1. Motivation

This work focuses on the general notion of autonomy in multi-agent systems. We will initially define an abstract concept of relative and absolute autonomy in the context of a computational agent. We think that the concept of autonomy must be always linked with the context and with the reference to what a given notion is applied. Autonomy means different things to various researchers, and it seems necessary to provide appropriate context and qualification of the term. Based on the notion of relative autonomy we review some of the existing multi-agent systems. We will then discuss the objectives of the research community and the motivations regarding the concept of autonomy of a given computational unit (an agent) in the context of open multi-agent systems, adaptability and complexity growth. We argue that, to build an open and adaptable multi-agent system, agents must be subjected to constant external influences. These influences must (possibly indirectly) affect and control a given agent's behaviour, and therefore negate the generally accepted requirement of agent's absolute autonomy. Based on our results with the experimental EVM framework we draw conclusions that computational agents can never be truly autonomous, or else the applicability of multi-agent systems in solving complex problems in an open environment would be limited or even impossible. That means that restrictions on autonomy imposed by the multi-agent system designer are not only based on the pragmatic needs to limit and manage general complexity of the system. These restrictions come directly from an inherent property of the dynamics of the MAS as a distributed asynchronous computational system.

To demonstrate and discuss the issues related to autonomy we refer to the experimental framework called Evolvable Virtual Machines (EVM). This framework has been used for modelling and analysis of meta-computational architectures, meta-learning, self-organisation and adaptive computing. We present results related to a contemporary model of computation for massively parallel open-ended evolutionary computations based on EVM [Nowostawski et al., 2004]. The model has been used to investigate properties of asynchronously communicating agents in a massively parallel multi-agent system

[Nowostawski et al., 2005b]. In this context, we discuss the concept of computational complexity, evolutionary learning and adaptability [Nowostawski et al., 2005a]. We will show that with our computational evolutionary system a constant flux of external information is necessary to provide an open-ended increase of complexity of generated (discovered) computational programs. Our results suggest that any closed (or fully autonomous) collection of computational agents would be limited in their ability to learn and adapt to new circumstances. Thus the notion of autonomy should be revisited and used in a clearly specified context. Computational agents must be subjected to direct or indirect external influences to allow continuous learning and adaptation by the system as a whole.

2. Autonomy in MAS

Autonomy, from Greek: Auto-nomos: *auto* meaning *self*, and *nomos* meaning *law*, refers to an entity that gives oneself its own laws. In other words self-governance, freedom from external influence and/or authority.

Generally in multi-agent systems there are two basic attempts and formalisations of the concept of autonomy. The internal and external views. The internal notion applies the above definition of autonomy to the agent itself, and specifies a set of principles or architectural constraints that are claimed for an autonomous operation of a given agent. The external view takes a different approach. It does not prescribe anything about internals of the agent or agent architecture itself. It is rather an assumption that other agents are autonomous in an abstract sense and cannot be controlled/influenced directly. Agent's behaviour cannot be imposed by any other agent, hence the interactions and agents collaboration must take into account various aspects of the assumed participants autonomy. We discuss briefly these two notions below.

The general notion of autonomy is invariant of the usual architectural or behavioural interpretations. In the next subsections we review proposals for the definition from internal and external point of view.

2.1. Internal autonomy

From a simple engineering perspective the concept of autonomy has been used as one of the distinguishing features between traditional object-oriented and agent-driven systems. See for example discussion in [Franklin and Graesser, 1996, Castelfranchi, 1995]. It is important to note that the notion of autonomy in MAS is often confused with the notion of automatic or independent operation. We want to stress that autonomy does not collapse to a mere independent operation. In complex software systems it is a simple truism that many complex inter-dependencies and influences must exist between various computational units. However, there is always an element of choice. Indeterminacy is essential, from the external observer point of view, to be able to talk of autonomous computing. As an example of the internal view of autonomy, consider the work of Luck and d'Inverno [Luck and d'Inverno, 1995], who have postulated that an agent's motivation and the ability to its create own goals is essential for autonomy. Using the Z specification language, they described a three-tiered hierarchy comprising objects, agents, and autonomous agents, where agents are viewed as objects with goals, and autonomous agents are agents with motivations. The ability to create goals according to some internal hidden and changeable agenda/motives is, according to their classification, essential for achieving true autonomy.

2.2. External autonomy

In external autonomy, compared to internal autonomy, we can turn the roles around. Instead of concentrating on our own agent and its autonomy, we can insist on the assumption that all entities and agents that our software agent interacts with are autonomous in the abstract sense. How this is achieved, or if it is possible at all, is not our concern. What is important is the fact that no fixed assumptions can be made regarding the interactions, agents, goals delegation, motives, environment, etc. The research community is somewhat divided into two roughly independent groups. One follows a strict internal view of autonomy and proposes ways to enhance and promote autonomy in various agent architectures. The other group has moved away from the strict internal requirements on agents autonomy, towards more open, distributed systems that are driven by interactions, dialogues, negotiations and collaborations of multiple individual participants, which are to be assumed autonomous from the external point of view. The role of autonomy for individual agents thus became an external assumption, rather than architectural requirement. The best discussion on this is presented in the work of Weigand and Dignum [Weigand and Dignum, 2003]. In their work, they have argued that architectural requirements of autonomy on agents are not as important as the expectations of autonomy on behalf of other agents. The agents that a given software agent interacts with must be assumed to be autonomous. Agents must be prepared to deal with other agents autonomy, and participate and collaborate with supposedly autonomous participants. This somewhat inverts the original requirements from those that support autonomy directly through elaborated architectures, into those that support features that work with autonomous agents.

3. Computational autonomy in MAS

Most researchers base the definition of autonomy on two primitives: self-governance and independence (e.g. [Gouaich, 2003, Carabelea et al., 2003]). Self-governance refers exclusively to the internals of the agent and its architecture. As we pointed out, this is not necessary in general discussion or in practical agent-oriented software engineering directly. Both notions however seem relevant when trying to formalise the concept of autonomy. One of the attempts to provide comprehensive definition is provided in Carabelea et al. [Carabelea et al., 2003]:

> An agent X is autonomous with respect to Y for p in the context C, if, in C, its behaviour regarding p is not imposed by Y.

The p in the above definition relates to the object of autonomy, and emphasis is placed on the relational nature of the concept of autonomy. There are however two main problems with the above definition. The first problem lies in the fact that multiple vague concepts are being used: *context*, *property* (or autonomy object) p and the notion of *imposed*. The precise and formal meaning of these terms in the above definition is not clear. Nevertheless, the above definition is useful and conveys the common-sense understanding of the concept of autonomy.

3.1. Formal definition

To make the above definition less ambiguous we propose to base the definition on a formal notion of computation. Let us assume computation C to mean the universal Turing machine transformation of input data from the input tape into output data on an output tape (we assume here a two-tape setup, with a read-only input tape and a write-only output tape)[1]. We will denote computation C from input X into output Y as: $X \xrightarrow{C} Y$. Let us assume data D to be a particular mapping of symbols into an input tape for the universal Turing machine. Let us assume that a computational agent A has access to a particular collection of data sources $Di \in E$, where E stands for *environment*, or context. In other words, agent A is capable of performing universal Turing machine computation on a set of data accessible from its Environment. The data can be represented as sequence of symbols from a particular alphabet, e.g. 0,1, as in the original work of Turing [Turing, 6 7]. Without any loss of generality, let us assume the following properties:

- data decomposition: $\exists D_m = \emptyset, \forall D_k, \exists D_i, D_j : D_k = D_i + D_j$.
- data composition: $\forall D_i, D_j + D_k = D_i : D_i - D_j = D_k$ and $D_i - D_k = D_j$.
- computational composition, $\forall D_i, D_i \rightarrow D_{i+1}$ and $D_{i+1} \rightarrow D_{i+2} : D_i \rightarrow D_{i+2}$

Data composition and decomposition simply capture the fact that data can be combined or split, without any loss of information. The computational composition ensures that computations do not have any side-effects. Note, that data can be read from or written to by various agents, and there is no distinction for input or output data. During the actual computation, a single data source can only be used as input or output (exclusive or).

Agent A is not autonomous with respect to agent B in the context E_s, and Agent B is said to *control* agent A, if:

$$\forall D_i \in E_s, \exists D_y \xrightarrow{agent_B} D_i : D_i \xrightarrow{agent_A} D_x. \quad (1)$$

If no such agent B exist, than we say, that Agent A is relatively autonomous in the context E_s:

$$\exists D_i \in E_s : \forall D_i \xrightarrow{agent_A} D_x, \forall D_y \xrightarrow{agent_B} D_k, D_k \neq D_i. \quad (2)$$

The agent A is absolutely autonomous in the context E_s, if:

$$\exists D_i \in E_s : \forall D_i \xrightarrow{agent_A} D_x, \forall D_y \rightarrow D_k, D_i \neq D_k. \quad (3)$$

3.2. Discussion of the definition

The above model provides the following intuitive interpretations:

1. If Agent A uses a particular subset of its environment $E_s \in E$ with data sources $D_i \in E_s$ to perform its computation C_{E_s}, and there exists an agent B that can output into all of D_i sources, we say that agent A is not autonomous in respect to agent B in the context E_s. Agent B is said to *control* agent A.
2. If no such agent B exist, than we say, that Agent A is relatively autonomous in context E_s.
3. If there is no set of agents that can collectively output to all of the $D_i \in E_s$, then we say that agent A is absolutely autonomous in the context of E_s.

Based on the above definition we propose the following general autonomy classes in Multi-Agent Systems (MAS). The three general classes below are often informally discussed in MAS literature, and these are now straightforward to define formally:

- User autonomy. Agent A is said to be autonomous with respect to a user, if the user does not provide all the data inputs that control agent A. In such a case, users cannot impose on the agent's behaviour directly; hence, we talk about agent's relative autonomy with respect to the user.
- Interactions autonomy (social autonomy). Agent A is autonomous socially, if it not only takes its inputs from other agents through interactions, but uses other sources of input at the same time, that are not bound to social interactions (for example, user input). This means that agents cannot simply impose any goals or behaviour directly on other agents, because interactions are not enough to "drive" agent's computations.

1 For more details and formal introduction to Turing-machine computational models see for example [Lynch and Tuttle, 1989, Hopcroft and Ullman, 1979].

- Organisational autonomy (norm autonomy). Organisational and institutional norms modelled as data sources cannot be used to impose a behaviour of agents directly. Agents use various data-sources that influence their behaviour and computational choices.

Some authors, in particular [Carabelea et al., 2003], postulate also a notion of *environmental autonomy*. In our definition of computational agents, the *environment* encompasses all the possible input data sources for a given agent: user input, other agents, static data, norms, and any other. Therefore, there is no possibility of an agent to perform any other computational mapping than $E_{input} \rightarrow E_{output}$. An agent is, by definition, just a computational function from the input environment, to the output environment. The concept of environmental autonomy, in our setup, does not make sense. To discuss environmental autonomy one would need to establish a partitioning of E into sub-environments, one exclusively called *environment*, and other subsets labelled differently. We believe that the partitioning of E into such disjoint classes is questionable in a general sense, although it might be useful for certain aspects of MAS, namely user interactions, social interactions and organisational interactions. If we model a closed system, where all data sources are in some way dependent upon agents' interactions and computations, then each single agent cannot be absolutely autonomous. To have a meaningful concept of absolute autonomy we have to deal with open systems, where some of the data sources are beyond the scope of the MAS itself[2].

4. Indeterminacy as autonomy

Our definition of autonomy as presented above rests entirely on the formal and intuitive notions of indeterminacy. Let us consider a case of two simple homoeostatic processes: one performed by a thermostat, and one performed by a bacterium. We intuitively feel that there is some difference between these two with respect to how autonomous they are. In the case of thermostat, even though it operates completely automatically and independently and we do not know or cannot predict exactly when it will switch from state to state, the degrees of freedom are quite limited. The thermostat is usually embedded in a well-insulated environment, where the temperature reacts almost exclusively to the heater/cooler system controlled by the thermostat itself. In the case of bacterium, even though the performed functions are sometimes as simple as those of the thermostat, the actual degrees of freedom seems to be larger. This is mostly due to the fact that in the case of a thermostat, the environment is almost exclusively controlled by the thermostat itself – the thermostat can make the ambient temperature to go up, or down. The environment, to a certain extend, is simple and reactive. In the case of bacterium, there is no direct control over the environment as such. The interactions with the environment are of a different type. Bacteria must operate continuously in highly unpredictable environments. We will not argue if there is any categorical distinction between these two autonomy classes. We just want to point out, that the main distinguishing feature from autonomous and non-autonomous processes lies in the indeterminacy and predictability of the environment. If there is a process that is entirely deterministic and predictable from a given observer's point of view, then we say that there is no autonomy within that process. The process is simply determined as a function of its environment. If the process is not entirely predictable, then we talk about autonomy, and about a *choice* – the process can *choose* one or the other trajectory for its evolution.

Let us consider a multi-agent system within a formalism of the Chemical Abstract Machine (cham) [Berry and Boudol, 1989]. Cham has been successfully used as a modelling formalism for other process calculi and process algebras, most notably for Milner's CCS [Milner, 1989], and Nicola-Hennessy TCCS [Nicola and Hennessy, 1987]. It is possible to model many asynchronous computational systems within the cham formalism. The observations within cham can be extended to any other process calculi. In cham the state of a system is modelled as solutions consisting of floating data-structures (molecules) that can interact with each other according to reaction rules. These datastructures can be of any type: primitive, such as numbers and strings, complex objects, or agents. There is a mechanism that "stirs" the solution, allowing for possible contacts between molecules. Note that the solution transformation process is inherently parallel. Any number of reactions can be performed at the same time, assuming that each molecule participates only in a single reaction. Assuming that the data-structures are individual agents, and the interactions are equivalent to reactions in cham, we can talk about two aspects with respect to autonomy (and indeterminacy):

- interactions are random or not. They are not preordered or pre-specified by the system design, and
- reaction rules may or may not be followed by the individual agents[3].

Now, let us consider a particular system (example inspired from [Banâtre et al., 1988]), consisting of n agents named $2 \ldots n+1$ and a reaction rule (interaction) between agents, such as if A_i, A_{i*j} where $i, j \in [2, n+1]$ then A_{i*j} annihilates itself. That means that if two agents meet, and one of the agents has a name that is a multiple of another agent, the agent with a name that is an multiplication of another name will annihilate itself. From the initial solution of all n agents, after some time, there will be only

2 In that case, we may talk about a stream of randomness (or indeterminacy) that comes from outside of the system itself.

3 Note, that in the original CHAM formalism all the reaction rules must be strictly followed by the system.

agents named with prime numbers left. This is assuming both, autonomy in the interaction choices and autonomy in the adoption of the general annihilation rule.

The above example demonstrates that in some circumstances, global coherent behaviour can be obtained in systems where autonomy is present on some of the underlying levels of abstraction. However, this is not always the case with all the systems. In some systems, autonomy must be restricted for the system to achieve a desirable stable point. For example in the case of CHAM it is not easy to devise autonomous rules that would lead the system to calculate factorial. We will discuss this in more details in the context of our EVM model. In the next section, we will briefly introduce the notion of autonomy in our multi-agent system KEA.

5. Multi-agent system KEA

The aim of the KEA project [Nowostawski et al., 2001] is to provide a modular agent platform with an enterprise-level backend. The architecture supports the use of agent-oriented ideas at multiple levels of abstraction. At the lowest level are micro-agents, which are robust and efficient implementations of agents that can be used for many conventional programming tasks. Agents with more sophisticated functionality can be constructed by combining these micro-agents into more complicated agents. Consequently the system supports the consistent use of agent-based ideas throughout the software engineering process, since higher level agents may be hierarchically refined into more detailed agent implementations. This enables scalability, flexibility and robustness of the platform, providing at the same time uniform modelling and programming paradigm.

The main distinguishing feature of KEA architecture as compared with traditional software engineering techniques is the autonomous dynamic linking facility. In traditional statically linked code, the function call is statically linked with appropriate library during compilation time. In dynamic linking, the function call is not linked with an appropriate implementation until the runtime. Then, the code is dynamically linked. The dynamic linkage with the library is unconditional (the library cannot refuse the linkage). Once the linkage has been made, it (usually) lasts till the end of the execution of the runtime system.

In KEA, the concept of dynamic linking has been extended further. The association between agents (or function calls if using the traditional programming nomenclature) is postponed till the very time when it is needed. At that time, the linkage is initiated, and may or may not be established. The participating party may refuse participation, in which case, the caller will have to deal with this situation by trying alternatives. In case of successful association (when the linkage has been established), it will only lasts till the end of the current task (or function). After that, a new dynamic linkage must be initiated and established again.

Such a model promotes high-levels of autonomy, because no fixed assumption can be made upon available participants. Agents must be prepared to deal with situations where given functionality may not be immediately available, and alternatives means of achieving one's goals must be undertaken. More details about KEA platform can be found in [Nowostawski et al., 2001].

6. Evolvable Virtual Machines (EVM)

6.1. Overview

There has been research conducted regarding autonomous asynchronously-interacting computations pursued in diverse areas of theoretical computer science. Certain properties investigated in those settings have been found to be invariant and shared between different complex systems. Our original desire was to integrate the recent advances from various fields onto a single coherent theoretical model, together with an experimental computational framework which could be used for practical investigations on massively parallel computational framework. Originally designed as an artificial evolution modelling tool [Nowostawski et al., 2005b], the EVM architecture is a model for autonomously interacting, evolving, complex and hierarchically organised software system. The EVM architecture stems from recent advances in evolutionary biology and utilises notions such as specialisation, *symbiogenesis* [Margulis, 1981], and *exaptation* [Gould and Vrba, 1982]. From the computational perspective it is a massively distributed asynchronous collection of interactive agents that utilises computational reflection. The EVM framework has been used for multi-task learning and meta-learning. Hence computational reflection and reification, on one hand, provide compact and expressive way to deal with complex computations, and on the other hand, provide ways of expanding a computations on a given level via the meta-levels and meta-computations.

Symbiogenesis researchers argue that symbiosis and cooperation are primary sources of biological variation, and that acquisition and accumulation of random mutations alone is not sufficient to develop high levels of complexity [Margulis, 1970, Margulis, 1981]. Other opponents of the traditional biological gradualism suggest that evolutionary change may happen in different ways, most notably through exaptation [Gould and Vrba, 1982], i.e. a process whereby a structure evolved for one purpose that has come to be used for another, unrelated purpose (or function).

The EVM architecture follows the biological models of: symbiogenesis, exaptation and specialisation. EVM allows independent computing elements to engage in symbiotic relationships, same as in CHAM, where independent agents are engaged in relationships through reaction rules and the concept of a *membrane*, that limits interactions only to local data within a membrane. In the case of EVM the interactions are not only 2-way – they may involve arbitrary number of participants. EVM allows a given agent to specialise in specific tasks, or to evolve towards new, more complex, tasks, similarly to the specialisation principle from biology. EVM also allows agents to be used in different contexts than originally designed for, similar to the exaptation principle.

The EVM architecture can be also seen as a computational model that combines the features of a trial-and-error machine [Bringsjord and Zenzen, 2003] and the multi asynchronously-interacting machines paradigm. The trial-and-error behaviour is achieved through continuous looping of different hypotheses and their re-evaluation until the desired precision of the hypothesis is achieved.

The EVM model is similar to the one of CHAM. There are however some main differences. In CHAM reaction rules are (typically) written between two agents in the solution. In EVM the interactions can happen between more than pair of agents. Also, in CHAM, the reaction rules are written beforehand, and not changed during the abstract machine execution. This is not the case for EVM. In EVM, the initial machines executed can modify the rules. It is beyond the scope of this article to analyse the exact formal equivalence and relationship between these two models – we leave it for future work.

In the following subsection we will present the details of the EVM implementation, and discuss the experimental

6.2. Implementation

Our current implementation of the EVM architecture is based on a stack-machine. With small differences, the EVM implementation is comparable to an integer-based subset of the Java Virtual Machine (JVM). There are two independent but compliant implementations: one is written entirely in Java and the second one in C. Developers and researchers can obtain the sources from CVS `http://www.sf.net-/projects/cirrus`. The basic data unit for processing in our current implementation is a 64-bit signed integer[4]. The basic input/output and argument-passing capabilities are provided by the operand stack, called *the data stack*, which is a normal integer stack. At the moment only integer-based computations are supported. All the operands for all the instructions are passed via the stack. The only exception is the instruction `push`, which takes its operand from the *program list* itself. Unlike other virtual machines (such as the JVM), our virtual machine does not provide any operations for creating and manipulating arrays. Instead, the architecture facilitates operations on lists. There is a special stack, called *the list stack* for storing integer-based lists.

Execution frames are managed in a similar way to the JVM, via a special execution frames stack. There is a lower-level machine handle attached to each of the execution frames. Machine is a list of lists, where each individual list represents an implementation of a single instruction for the given machine. In other words, the machine is a list of lists of instructions, each of which implements a given machine instruction. If a given instruction is not one of the primitive Base Machine units, i.e. primitive instructions for that machine, then the instruction sequence must be executed on another, lower-level machine. The Base Machine implements all the primitive instructions that are not reified further into more primitive units. To distinguish those primitive instructions that are executed on the Base Machine we refer to them as *operations*.

Potentially, EVM programs can run indefinitely, and therefore, for practical reasons, each thread of execution has a special limit to constrain the number of instructions each program can execute. This is especially crucial in a multi-EVM environment. Once the limit is reached a given program will unconditionally halt.

The EVM offers rich reflection and reification mechanisms. The computing model is relatively fixed at the lowest-level, but it does provide the machines with multiple computing architectures to choose from. The model allows the programs to reify the virtual machine on the lowest level. For example, programs are free to modify, add, and remove instructions from or to the lowest level virtual machine, as well as any other level. Also, programs can construct higher-level machines and execute themselves on these newly created levels. In addition, a running program can switch the context of the machine, to execute some commands on the lower-level, or on the higher-level machine. Altogether, the EVM provides limitless flexibility and capabilities for reifying individual EVM executions. Due to this high level of flexibility, there have been no attempts to formalise the full EVM model in any of the existing process calculi or other computational algebras. We have only attempted partial formalisations of the model.

A possible way of instantiating part of the computational environment for the architectural framework is by adapting bias-optimal search primitives [Levin, 1973], or the incremental search methods [Schmidhuber, 2004]. To narrow the search, one can combine several methods together. For example, it is possible to construct a generator of problem solver generators, and employ multiple meta-learning strategies for a given computational task at hand. A more detailed description of the abstract EVM architecture is given in [Nowostawski et al., 2004]. The experimental results are described in details in [Nowostawski et al., 2005a].

7. Conclusions

In this article we have discussed the notion of autonomy in multi-agent systems. We have reviewed the existing definitions and formalisation attempts. We have proposed our own formalisation based on the notion of universal Turing machines computational agents, with the abstract notion of data sources and data transformations. Based on the assumed notions of computation, the concept of relative and absolute autonomy for a given computational agents have

4 This somewhat arbitrary constraint is dictated by practical and efficient implementation on contemporary computing devices.

been presented. We compared our definition to existing intuitive definitions in multi-agent literature. We have provided also a comparison of general autonomy classes in MAS, with intuitive and formal notions of autonomy.

In the context of autonomy in MAS we have presented details of two multi-agent systems: KEA and EVM. These frameworks tackle the challenges of autonomous computing in various ways. In both the emphasis is placed on the central notion of unsecured and unreliable inter-process (or inter-agent) communication. The KEA framework is using the notion of autonomous dynamic linking between agents. The EVM system is using the notion of unstructured self-assembly and dynamic aggregation of computational components.

Based on the literature review and our own investigations, we have concluded that the autonomy is directly linked with the concept of indeterminacy in a sense of Turing-computability. In that context, it is easier to understand why autonomy is a subject of continuous restrictions from various angles within MAS community. From one hand, unlimited autonomy makes it extremely hard to design, program and analyse MAS systems. Therefore, restricting autonomy is one way of dealing with the complexities of MAS design. On the other hand, restricting autonomy is an inherently needed property to achieve global coherent behaviour, that may otherwise be unattainable. We have discussed EVM-based experiments which show that only through limiting individual agents autonomy and restricting the freedoms of choice, a more complex computational structures can be achieved. This seems to be an inherent property of any complex systems composed of a large number of autonomously interacting entities. This phenomenon is called *enslavement* in synergetics [Haken, 1983].

References

[Banâtre et al., 1988] Banâtre, J., Coutant, A., and Le Metayer, D. (1988). A parallel machine for multiset transformation and its programming style. *Future Generation Computer Systems*, 4(2):133–144.

[Berry and Boudol, 1989] Berry, G. and Boudol, G. (1989). *The chemical abstract machine*. ACM Press, NY, USA.

[Bringsjord and Zenzen, 2003] Bringsjord, S. and Zenzen, M. (2003). *Superminds: People Harness Hypercomputation, and More*. Studies in Cognitive Systems Volume 29. Kluwer Academic Publishers. Cen BF 311 B 4867.

[Carabelea et al., 2003] Carabelea, C., Boissier, O., and Florea, A. (2003). Autonomy in multi-agent systems: A classification attempt. In [Nickles et al., 2004], pages 103–113.

[Castelfranchi, 1995] Castelfranchi, C. (1995). Guarantees for autonomy in cognitive agent architecture. In *Proceedings of the workshop on agent theories, architectures, and languages, ATAL'94*, volume 890 of *LNAI*, pages 56–70. Springer-Verlag, NY, USA.

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A. (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, pages 21–36.

[Gouaich, 2003] Gouaich, A. (2003). Requirements for achieving software agents autonomy and defining their responsibility. In [Nickles et al., 2004], pages 128–139.

[Gould and Vrba, 1982] Gould, S. J. and Vrba, E. (1982). Exaptation – a missing term in the science of form. *Paleobiology*, 8:4–15.

[Haken, 1983] Haken, H. (1983). *Synergetics, An Introduction: Nonequilibrium Phase Transitions and Self-Organization in Physics, Chemistry, and Biology*. Springer-Verlag, Berlin, 3^{rd} revised and enlarged edition edition.

[Hopcroft and Ullman, 1979] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, USA.

[Levin, 1973] Levin, L. A. (1973). Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266.

[Luck and d'Inverno, 1995] Luck, M. and d'Inverno, M. (1995). A formal framework fo agency and autonomy. In *Proceedings of Frist International Conference On Multi-Agent Systems (ICMAS)*, pages 254–260.

[Lynch and Tuttle, 1989] Lynch, N. and Tuttle, M. R. (1989). An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246.

[Margulis, 1970] Margulis, L. (1970). *Origin of Eukaryotic Cells*. University Press, New Haven.

[Margulis, 1981] Margulis, L. (1981). *Symbiosis in Cell Evolution*. Freeman & Co., San Francisco.

[Milner, 1989] Milner, R. (1989). *Communication and concurrency*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

[Nickles et al., 2004] Nickles, M., Rovatsos, M., and Weiß, G., editors (2004). *Agents and Computational Autonomy - Potential, Risks, and Solutions - Postproceedings of the 1st International Workshop on Computational Autonomy - Potential, Risks, Solutions (AUTONOMY 2003), held at the 2nd International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2003), July 14, 2003, Melbourne, Australia*, volume 2969 of *Lecture Notes in Computer Science*. Springer.

[Nicola and Hennessy, 1987] Nicola, R. D. and Hennessy, M. (1987). CCS without tau's. *Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming*, pages 138–152.

[Nowostawski et al., 2005a] Nowostawski, M., Epiney, L., and Purvis, M. (2005a). Self-Adaptation and Dynamic Environment Experiments with Evolvable Virtual Machines. In S.Brueckner, Serugendo, G. M., D.Hales, and F.Zambonelli, editors, *Proceedings of the Third International Workshop on Engineering Self-Organizing Applications (ESOA 2005)*, pages 46–60. Springer Verlag.

[Nowostawski et al., 2005b] Nowostawski, M., Epiney, L., and Purvis, M. (2005b). Self-adaptation and dynamic environment experiments with evolvable virtual machines. In *Proceedings of the Third International Workshop on Engineering Self-Organizing Applications (ESOA 2005)*, pages 46–60, Utrech, The Netherlands. Fourth International Joint Conference on Autonomous Agents & Multi Agent Systems.

[Nowostawski et al., 2001] Nowostawski, M., Purvis, M., and Cranefield, S. (2001). Kea – multi-level agent infrastructure. In *Proceedings of the 2nd International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, pages 355–362, Kraków, Poland. Department of Computer Science, University of Mining and Metallurgy.

[Nowostawski et al., 2004] Nowostawski, M., Purvis, M., and Cranefield, S. (2004). An architecture for self-organising evolvable virtual machines. In Brueckner, S., Serugendo, G. D. M., Karageorgos, A., and Nagpal, R., editors, *Engineering Self Organising Sytems: Methodologies and Applications*, number 3464 in Lecture Notes in Artificial Intelligence. Springer Verlag.

[Ray, 1991] Ray, T. S. (1991). An approach to the synthesis of life. In Langton, C., Taylor, C., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II*, volume XI of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 371–408. Addison-Wesley, Redwood City, CA.

[Schmidhuber, 2004] Schmidhuber, J. (2004). Optimal ordered problem solver. *Machine Learning*, 54:211–254.

[Turing, 6 7] Turing, A. M. (1936–7). On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265. also 43, pp. 544-546, 1937.

[Weigand and Dignum, 2003] Weigand, H. and Dignum, V. (2003). I am autonomous, you are autonomous. In [Nickles et al., 2004], pages 227–236.

[Witkowski and Stathis, 2003] Witkowski, M. and Stathis, K. (2003). A dialectic architecture for computational autonomy. In [Nickles et al., 2004], pages 261–274.