

Towards machine consciousness: a developmental theory of thoughts as communicating processes

Pierre Bonzon
University of Lausanne, Switzerland
pierre.bonzon@unil.ch

Abstract

We present a model of thoughts given in terms of concurrent communicating processes, to serve as a basis for machine consciousness. Towards this end, we rely on a language implementing threads within the framework of logical programming. This allows for both a formal definition and the running of simulations. Adopting a developmental approach, we start with a simple model of purely reactive behaviour. We show how it can be extended into an agent model incorporating thoughts and deliberation, and then almost naturally evolve to incorporate a functionality of consciousness akin to episodic memory.

1. Introduction

Will machines ever be able to think and enjoy consciousness? This recurring question has been debated by philosophers and scientific alike, but no definitive answer has been given yet. Progress is very slow. Actors from both camps are sometimes very critical about each other. In fact, the whole domain seems to suffer from a lack of methodology and disciplined activity. Talking about the research done in robotics and AI, R. Brooks once wrote [6] “*None of these fields is experimental (...). There is no control experiment to compare against*”. Ten years later, when confronted with the apparent failure of current research in artificial intelligence and artificial life, he went on asking if something was not going wrong, and if one were not actually missing something fundamental and currently unimagined in the models, and he conjectured: “*Perhaps other mathematical principles or notions, necessary to build good explanations of the details of evolution, cognition, consciousness or learning will be discovered and invented*” [7].

How should we do it then? In our view, research in these subjects should be inspired by the methods of theoretical physics, and follow a similar path i.e., in much the same way as *mathematical* abstractions (without any relation whatsoever with apparent reality or intuitive perception) are used to model the properties and interactions of subatomic elementary particles, *computational* abstractions could be used to model the overall cognitive behaviour resulting from the neuronal activity taking place into

the brain. The problem then would be to somehow “ground” these processes i.e., to embody them into biological neural (or neurons) networks.

We further argue that the computational abstraction that is needed for this task is already at hand, namely *concurrent communicating systems (CCS)*. The problem lies in achieving an effective use of this powerful notion, which generally tends to be buried into opaque code (as too often in AI, “the theory is the program”) whereas it should lie at the very heart of the model’s definition. In contrast, we report here on a model of CCS resulting from the integration of communication into a model of intelligent agents [4]. This model is expressed in a high level language of agent threads that furthermore can be compiled and executed on a virtual *abstract machine* [5]. Experiments may thus be easily reproduced and, in the spirit of the scientific experimental method, could allow for a systematic and repeated exploration.

We come now to the very object of this exploration. The study of consciousness extends in many directions. At one end lies the so-called “search for the neural correlates of consciousness” [1]. Briefly, this approach involves “isolating the neural processes that correlate with various states of consciousness”. Following the increasing availability of brain imaging techniques, the experimental research conducted in this direction is flourishing. But this approach has failed so far to provide us with any accurate knowledge about neurophysiological mechanisms. At the other end, we find theories that are formulated merely in terms of the information processing presumably conducted by the brain. This line of research has often been described as constituting a complementary “search for the computational correlates of consciousness”. This second approach however has yet to mature to the point of producing true experimental platforms. As reported in [2], there hardly exist a few actual implementations, which furthermore rely on metaphors (such as the Global Workspace model [3]) “that simply help us understand in a holistic way how the human mind works (...) far away from an established body of scientific knowledge”.

Our first aim is precisely to give up metaphors all together. Towards this end, we have to delineate a functional aspect of consciousness that could easily be amenable to computational processes. It has become customary to map the functional aspect of consciousness into four distinct roles i.e., *access*, *phenomenal*, *monitoring* and *self-consciousness* [1]. Briefly, *access* consciousness (A-consciousness) refers to the direct availability of mental content and thus allows humans to use language and explicit planning in order to act rationally towards a specific goal. *Phenomenal* consciousness (P-consciousness) refers to the nature of feelings or sensations, and thus allows one to get qualitative inputs (appropriately named *qualia*) that amount to differentiating perceptions. *Monitoring* consciousness (M-consciousness, also called reflective) refers to the processes that lead to one’s sensations and percepts, as opposed to their contents, and thus accounts for the access to the mental activity itself. Finally, *self-consciousness* (S-consciousness) is the reflective capability we enjoy when we think about ourselves. Few experimental models (if any) make a clear distinction between these different types of consciousness, and it is unclear whether or not these functions have been effectively dissociated in computational models [11].

Our model will allow for functionalities that are related to both access and monitoring consciousness. More precisely, it relates to a correlate of access and monitoring consciousness, namely *episodic* memory [13]. In short, this functionality refers to the fact that the thinking and deliberations we do perform consciously, or the events that hit our consciousness, get somehow memorized and that we are able to recall them for some period of time afterwards. The abstract robot model that we will define in this paper works in a similar manner. This similarity will not allow us however to argue that it acts consciously. The only claim that we will make is that *it does enjoy a property it shares with a conscious mind* i.e., that of using a pervasive tool identified with thoughts in order to keep and retrieves traces of its past activity.

Our model will also be in line with the idea that humans are not directly conscious of their thoughts but rely for that on intermediate representations and processes. As formulated by Crick and Koch [8] when reporting about Jackendoff's postulates [9], "what is conscious about thoughts is visual or other images, or talking to oneself. (..) visual and verbal images are associated with intermediate-level sensory representations, which are in turn generated from thoughts by the fast processing mechanisms in short-term memory. Both the process of thought and its content are not directly accessible to awareness". In order to account for this intermediation, or in other words to allow for the indirect access to thoughts via intermediate-level representations, some kind of communication must take place. Thus we do postulate in turn that if intermediate representations are the support of conscious thoughts, then communication is their trigger. More precisely, this new postulate can be formulated as follows: *when humans think, they basically engage into synchronized communication of some internal representations.*

Following a developmental approach, we shall start with a model of purely reactive behaviour. We then show how it can be extended into a more complex model incorporating a deliberative behaviour. We further introduce a kind of awareness into the system by giving it the ability to keep traces of its deliberation processes, thus ending up with a model exhibiting a form of proto-consciousness akin to episodic memory. Conscious thinking can thus be potentially bound to a particular type of activity i.e., *deliberation* involving the passing of internal data over to an external level identified with that of *thoughts*. Finally, exploiting the analogy that seems to emerge between the syntactic features of our model and the topology of synaptic connections, we shall evoke, in a purely conjectural way, the possibility to somehow "ground" these processes i.e., to embody them into biological neural networks

2. A test bed for the simulation of simple robots.

We shall base our developments on a simple robot model to be used a test bed for simulations. Towards this goal, let us consider any kind of autonomous vehicle or mobile robot that can move towards a target to perform a single task (from simply picking up a ball to fixing a problem). We will assume that this robot's behaviour can

be defined in terms of plans, each plan consisting in turn of actions. As the robot moves and acts, both its current plan and action are selected using rules as well as data obtained from its sensors. Both rules and sensor data will be accessible as stored data. For the purpose of our simulation, we assume that the robot is equipped with physical sensors allowing it to detect both its own position and that of targets.

2.1 Rules for the selection of the robot's current plan and action

In this simple model, only one of two possible plans *go* or *return* can be selected according to the following *plan* selection rules:

- 1) "if there is a *target* to be served, then select plan *go*"
- 2) "if there is no *target* to be served, then select plan *return*".

The choice of an action in a plan follows from *do* selection rules:

- 3) "when using plan *go*, if moving towards the *target*, then select action *forward*
- 4) "when using plan *go*, if reaching the *target*, then select action *work*"
- 5) "when using plan *return*, if moving towards *home*, then select action *backward*"
- 6) "when using plan *return*, if reaching *home*, then select action *rest*"

2.2 First order logical representation of the robot's selection rules

In order to represent the robot's behaviour in logical terms, let us first consider the following predicates:

<i>target(x)</i>	->	"a <i>target</i> is located at position <i>x</i> "
<i>at(x)</i>	->	"the robot is <i>at</i> position <i>x</i> "
<i>home(x)</i>	->	"the <i>home</i> location to return to is at position <i>x</i> "
<i>plan(p)</i>	->	"plan <i>p</i> has been selected"
<i>do(plan(p), a)</i>	->	"while using plan <i>p</i> , action <i>a</i> has been selected".

The robot's selection rules can then be represented by the following logical implications (in accordance with the Prolog syntax, conjunctions are represented by the operator "," and variables start with capital letters):

- 1) **target (X)** => **plan (go)**
- 2) **not target (X)** => **plan (return)**
- 3) **(at (X) , not target (X))** => **do (plan (go) , forward)**
- 4) **(at (X) , target (X))** => **do (plan (go) , work)**
- 5) **(at (X) , not home (X))** => **do (plan (return) , backward)**
- 6) **(at (X) , home (X))** => **do (plan (return) , rest)**

These implications, which will be used to drive the robot's behaviour and thus need to be stored as data, are sometimes called *compiled knowledge*.

3. Implementing simulations using a language for concurrent communicating processes

Implementing a simulation of a simple robot like the one defined above can be done in a variety of ways. In any case, there is no need to introduce communication at this stage. In order to allow for later developments, we shall nevertheless use at once a language allowing for the expression of communicating concurrent processes. Towards this end, we shall rely on our previous work that led us to define and implement such a language within the framework of logical programming [4].

3.1 A first example: a logical sensor implementing reactive behaviour

Let us give first a concrete example of the use of this language. More precisely, consider the following expression, which actually represents the definition of a *logical sensor* (as opposed to the physical sensors considered in the previous section) implementing a *reactive* behaviour:

```
thread(sensor, [P,A],
             [((plan(P),
               do(plan(P),A)
               | [effector(A)])))]])
```

Fig 1: A logical sensor implementing reactive behaviour

As just stated above, no communication is needed. The name of a thread i.e., *sensor* is first declared, together with two local variables i.e., *P* and *A*. This is followed by a so-called *guarded command* (corresponding to a logical implication) consisting of a *guard* i.e., $(\text{plan}(P), \text{do}(\text{plan}(P), A))$ separated by the operator “|” from the *command* itself i.e., $\text{effector}(A)$.

When implemented as a concurrent process, the above guarded command will allow for repeated *reaction* cycles defined as follows:

- a) select a *plan P* and an *action A* from this plan using the *compiled knowledge* presented in the previous section
- b) deliver the task of executing *action A* to an appropriate *effector*.

3.2 Formal language definition

We come now to a presentation of the complete language. We do not reproduce here the usual syntax for first order expressions, the only divergence being that variable names must start with capital letters. The remaining formal syntax is defined by the following grammar, where $[m_1|[m_2|...[]]] = [m_1, m_2, ...]$, and multiple choices are separated by the meta-operator “||”.

<thread>	::= thread (<threadName>(<threadParams>),<varList>,<mesTree>)
<varList>	::= [] [<varName> <varList>]
<mesTree>	::= [] <seq> [<alt>]
<seq>	::= [<mes> <mesTree>]
<alt>	::= <guardMes> (<guardMes>;<alt>)
<guardMes>	::= (<guard> <mesTree>) <mesTree>
<mes>	::= <messageName>(<messageParams>)
<messageName>	::= ask tell start end effector

Fig 2: BNF productions for agent threads

Each thread consists of a tree structure whose sequences contain messages separated by the conjunctive operator “;” and end with alternatives containing guarded messages separated by a disjunctive operator “||”. Besides the *ask/tell* communicating pair, the possible messages are *start/end* (these are commands to create and delete a thread) and *effector* (this message allowing for the execution of any action to be handed over to an external effector and/or actuator). Each thread is re-entrant and is automatically resumed at the end of each sequence and/or alternative.

Threads expressed in this language can be compiled into agent plans comprising synchronizing conditions, which in turn will be interpreted by a virtual machine [4], and thus eventually sequentially executed by any underlying hardware. Its precise operational semantics has been given by compiling functions [5].

4. Simulating reactive+deliberative behaviours

In the second step of our test bed, a deliberative behaviour is introduced. More precisely, the reaction allowing for action execution will be followed by a deliberation possibly leading to a switching of plans. These two processes will be similarly driven by compiled knowledge and involve effectors. Both will be defined within parameterized threads identified with abstract *feelings*, *sensations* or *thoughts* and associated with either *data* obtained from the physical sensors or with the current selected *plan*.

4.1 Threads implementing a reactive+deliberative behaviour

The overall robot behaviour can be expressed by the following threads:

```

thread(sensor(P), [A,B],
      [ (do(P,A)
        | [effector(A),
          ((deliberate(done(P,A),B)
            | [effector(B)]))])])])

```

Fig 3: Threads implementing reactive+deliberative behaviour

As it can be seen, *sensor(P)* threads are now parameterized and have been extended with a deliberation process reflecting on the action *A* from plan *P* just performed. Using *deliberate* compiled knowledge, a mental action *B*, possibly leading to the switching of plans, will be executed.

4.2 Running the test bed in reactive+deliberative mode

The additional compiled knowledge that is needed by the deliberation process to drive the switching of plans is given by the following assertions:

```

7) deliberate(done(target(X),store(target(X))), (end(sensor(target(X))),
8)                                             start(sensor(plan(go))))
9) deliberate(done(plan(go),work), (end(sensor(plan(go))),
10)                                             start(sensor(plan(return))))
11) deliberate(done(plan(return),rest), (end(sensor(plan(return))))

```

As it can be interpreted from the above assertions, an existing thread will be deactivated and a new one started each time a certain action is performed in a given situation. In order to allow for the initialization of the deliberation process, we shall further assume that at some underlying level (which may be the virtual machine interpreting threads), data obtained from physical sensors will give rise to corresponding logical data sensors. In our case, each time a *target* is detected at position *x*, a command *start(sensor(target(X)))* will be issued by the virtual machine. Each such logical sensor will have a single action associated with it i.e., to *store* the corresponding data within the virtual machine, giving thus rise to the following *do* selection rule

“when sensing *target(X)*, then unconditionally select action *store(target(X))*”

This will lead to the following logical assertion (or implication without condition), delivering the compiled knowledge needed to initialize the process:

```

0) do(target(X),store(target(X)))

```

4.3 Tracing a simulation in reactive+deliberative mode

The actual simulation of the robot behaviour leads now to an interleaving of *reaction* and *deliberation* processes. This interleaving can be described as follows:

- a) upon receiving the position of a target, say at 3, the virtual machine issues a command `start(sensor(target(3)))` to start a data thread
- b) in reaction of assertion 0 in thread `sensor(target(3))`, action **A** is instantiated to `store(target(3))` and gets executed
- c) using assertion 7 in the subsequent deliberation process of the same thread, action **B** is instantiated to `(end(sensor(target(3))), start(sensor(plan(go))))`, gets executed, and as a result, the data thread will be replaced by a plan thread
- d) in reaction to assertion 3 or 4 in thread `sensor(plan(go))`, depending on the location of the robot, action **A** is instantiated to either `forward` or `work`, gets executed, and either `done(plan(go), forward)` or `done(plan(go), work)` leads to a new deliberation, and so on, this process leading eventually to a switching of plans.

5. Introducing proto-consciousness

In this last step, we introduce a form of proto-consciousness known as *episodic memory*. As discussed in the introduction, this functionality refers to the capability of remembering past experiences. Clearly this functionality operates and/or occurs in conjunction with A-consciousness, i.e. with the ability to access mental contents. As a typical example, whenever one gets suddenly conscious of having done something wrong, both a conscious experience (e.g., that an action turned out to be wrong) and a recall from episodic memory (i.e., remembering having done it) are concurrently involved. Among the many aspects of such a complex mental process, we shall privilege the necessary *triggering* event that allows it to happen (e.g. the prior report that “something actually did not succeed”). Indeed, our postulate here will be that, by definition, any proto-consciousness phenomena is based on the *coincidental delivery* to the same recipient (or into the same locus, incidentally called the locus of consciousness), of two or more internal communications. Translated into our own model, this means that a forefront thread *consciousness* will be addressed concurrently by two or more background sensor threads.

5.1 A set of threads implementing proto-consciousness

In accordance with the evolutionary approach taken so far, our simulation of proto-consciousness follows from an extension of our previous model. As sketched above, a new *consciousness* thread intended to serve as the locus of consciousness communicates concurrently with two sensor threads, one acting as a triggering event for the *recall* of past deliberations, and the other as a server of such past deliberations. Past deliberations are memorized beforehand under the form of *reflect* logical sensors started by the sensor threads. We thus get the following new set of threads:

```
thread (consciousness, [P,A,B,C],
      [ask (sensor (recall (P))) ,
       ask (sensor (reflect (done (P,A) ,B)) ,
          ((introspect (conscious (done (P,A) ,B) ,C)
            | [effector (C)])))]])

thread (sensor (P) , [A,B] ,
      [ ((do (P,A)
        | [effector (A) ,
          ((deliberate (done (P,A) ,B)
            | [start (sensor (reflect (done (P,A) ,B)) ,
              effector (B)])))]]) ;
        ([tell (consciousness)])) ]])
```

Fig. 4: A set of threads implementing proto-consciousness

As it can be seen in the middle of the sensor threads, each *deliberate (done (P,A) ,B)* deliberation process is memorized by starting a reflecting *reflect* sensor thread. After being “waken up” by a message from *sensor (recall (P))* and consequently served by *sensor (reflect (done (P,A) ,B))*, the *consciousness* thread in turn will use *introspect* compiled knowledge to look for an action *C*, and then execute it. Note that the communicative act standing as the last line of *sensor (P)* i.e., *tell (consciousness)*, will be issued as a default option only, thus allowing for a logical sensor to address the consciousness thread whenever there is no other option (in other words, the syntax used here, using the disjunctive operator “;”, corresponds to a traditional “if _ then _ else _” instruction).

5.2 Running the test bed with proto-consciousness

The compiled knowledge that is needed by the extended sensors and the new *consciousness* thread is given by the following assertions:

```

7) deliberate (done (target (X) , store (target (X) ) ) , (end (sensor (recall (done) ) ) ,
8) end (sensor (target (X) ) ) ,
9) start (sensor (plan (go) ) ) ) )

10) deliberate (done (plan (go) , work) , (end (sensor (plan (go) ) ) ,
11) start (sensor (plan (return) ) ) ) )

12) deliberate (done (plan (return) , rest) , (end (sensor (plan (return) ) ) ,
13) start (sensor (recall (done) ) ) ) )

14) introspect (conscious (done (P,A) , B) , (end (sensor (reflect (done (P,A) , B) ) ) ,
15) report (conscious (done (P,A) , B) ) ) )

```

Clearly, *deliberate* knowledge has been just extended in line 13 with a command to start a *sensor (recall (done))* thread whenever the robot goes to rest (a triggering event for the recall of past deliberations), and in line 7 with a command to end this thread whenever the robot goes to work. As for the last assertion in lines 14 and 15, it will lead, for illustrative purposes, to issue a report about each recalled deliberation.

5.3 Tracing a simulation with proto-consciousness

In addition to what has been described in section 4.3, one should note here the following features:

- a) whenever a reflecting *sensor (reflect (done (P,A) , B))* thread is started, this amounts to memorizing the corresponding *deliberate (done (P,A) , B)* deliberation process
- b) since there is no action to be performed by any reflecting sensor i.e., there is no compiled knowledge corresponding to a *do* selection rule (see 4.2), its default option is to communicate with the *consciousness* thread
- c) as the result of the *report* action associated for illustrative purposes with proto-consciousness, the following report will be issued (this is the actual output produced by our running implementation of the virtual machine):

```

conscious (done (target (3) , store (target (3) ) ) , (end (sensor (recall (done) ) ) ,
end (sensor (target (3) ) ) ,
start (sensor (plan (go) ) ) ) )
conscious (done (plan (go) , work) , (end (sensor (plan (go) ) ) ,
start (sensor (plan (return) ) ) ) ) )
conscious (done (plan (return) , rest) , (end (sensor (plan (return) ) ) ,
start (sensor (recall (done) ) ) ) ) )

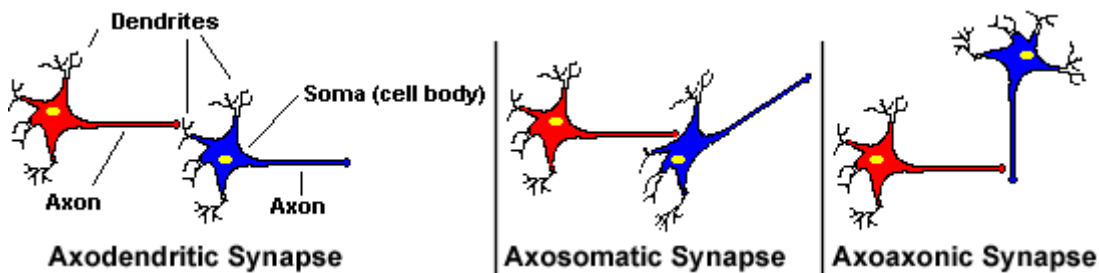
```

As it can be seen, past deliberations gave rise to a switching of data and/or plan threads taking place after a certain action occurred in a given situation.

6. Towards grounding thought processes into the brain.

In this last section, we briefly evoke the goal of grounding our computational model of thoughts into biological neural networks. To understand what we could possibly achieve in pursuing such a goal, let us quote from [14]: “We know that the brain produces cognition (it is the only example we are certain of). So, if we have an idea of how computational processes could be matched onto brain processes, we could also get a clearer view of what could be missing in the computational account.” Let us first put to light the analogy that does exist between the syntactic properties of our model and the topology of synaptic connections:

- as shown by the formal syntax given in section 3, threads grow as trees by pushing branches, these branches being possibly laterally connected to the branches of other threads through *ask/tell* communication primitives.
- neurons do grow like trees as well: around the cell body is a branching dendritic tree that receives signals sent from other neurons through their axon, the axon itself possibly terminating in a branching synaptic tree.



It must be further noted that, on the whole, neurons form a rather stable structure in which microcurrents propagate through synaptic connections. As it is now known from recent advances in the theory of concurrent parallel processes [10], this continuous flow of signals is akin to the dynamic reconfiguration of channels in a communication network. Furthermore, a power tool standing as a universal model of computation, namely the π -calculus, has been designed that allows to express this very process of concurrent computations whose configuration may changes.

We therefore postulate that a possible way to ground our theory of thoughts into the brain is to try and work simultaneously on two fronts i.e., to define and implement both our model and a model of neural nets in terms of the π -calculus. We are encouraged into this endeavour as we note the strong similarities that do exist between this new formalism, on one hand, and the models that have proposed for simulating neural-symbolic integration, on the other. Although this does not seem to have been reported before, the temporal synchrony mechanism that lies at the heart of the architecture proposed in [12] to solve the variable binding problems is closely related to the operational properties of π -calculus itself. By slightly extending the facilities offered at the primitive level, it is indeed a straightforward task to rewrite the examples given in [12] as π -processes.

7. Conclusion

We have now completed the definition and implementation of a simple abstract robot's capable of recalling its past activity in response to a triggering event. As already stated in our introduction, we do not claim to have thus succeeded in simulating consciousness, conscious behaviour, or whatsoever. Clearly, any attempt at simulating consciousness should involve a real situated robot, possibly confronted with its own survival, etc. In any case, our aim was not to try such an attempt, but rather to propose, at the same time, both a test bed and a particular approach for simulating a robot's behaviour.

A definite distinctive feature of the model we just presented is the clear distinction it makes between *compiled knowledge*, on one hand, and *communicating threads*, on the other. Whereas compiled knowledge represents innate, acquired (learned) or programmed behaviour, communicating threads in turn offer ways for

- a) the processing of this compiled knowledge
- b) the dissemination of any kind of information
- c) the issuance of effector commands.

Being both the medium and the vehicle of deliberations leading to actions, threads can be identified with a model of *thoughts*. Therefore the claim that we make here is that our model *does enjoy a property it shares with a conscious mind*, namely that of using a pervasive tool identified with thoughts in order to keep and retrieve traces of its past activity.

Our test bed has been implemented and is running on a Prolog platform. Various extensions could be easily brought to enhance

- a) the virtual machine underlying the implementation
- b) the definition of the abstract robot test bed
- c) the model that implements it.

It is worth noting here how our developmental approach correlates with *evolutionist* views: indeed, starting with the simplest of models defined by a single thread, we either defined new threads or enlarged existing ones, always keeping their kernel untouched and adding new functionalities at their interface. It is easy to imagine how this process could be iterated and the corresponding models get more and more complex until they eventually end up mimicking a brain. One could then ultimately invoke Brook's motto [6] i.e., "*the behaviors are the building blocks, and the functionality is emergent*" to claim that the model "almost naturally" evolved to simulate consciousness. However, according to common sense, it is only when the eyes (or for that matter, any other sense) hit the mind that true consciousness really arises. Consequently, multiplying the number of sensori-motor controls, modelling their interaction and eventually attempting a coupling with a real situated robot would be the next thing to do in order to go beyond pure simulations.

References

- [1] Atkinson, A.P., Thomas, M.S.C. & Cleermans, A (2000) Consciousness: mapping the theoretical landscape, *Trends in Cognitive Sciences*, Vol. 4, No10.
- [2] Arrabales Moreno, R. & Sanchis de Miguel, A. (2008) Applying Machine Consciousness Models in Autonomously Situated Agents, *Pattern Recognition Letters*. Special Issue on Pattern Recognition in Multidisciplinary Perception and Intelligence, vol. 29 (8), pp 1033-1038..
- [3] Baars, B. (1988) *A cognitive theory of consciousness*, Cambridge University Press.
- [4] Bonzon, P. (2002a) An Abstract Machine for Classes of Communicating Agents Based on Deduction, in: J.J. Meyer and M. Tambe (eds), *Intelligent Agents VIII*, LNAI vol. 2333, Springer Verlag.
- [5] Bonzon, P. (2002b) Compiling Dynamic Agent Conversations, in: M. Jarke, J. Koehler & G. Lakemeyer (eds), *Advances in Artificial Intelligence*, LNAI vol. 2479, Springer Verlag.
- [6] Brooks, J. (1991) Integrated Systems Based on Behaviour, *SIGART Bulletin*, vol. 2, No 4.
- [7] Brooks, R. (2001) The relationship between matter and life, *Nature*, vol. 401.
- [8] Crick, F., & Koch, Ch. (2000) The Unconscious Homunculus, in: Metzinger, T., ed., *The Neuronal Correlates of Consciousness*, MIT Press.
- [9] Jackendoff, R. (1987) *Consciousness and the Computational Mind*, MIT Press.
- [10] Milner, R. (1999) *Communicating and Mobile Systems: the π -Calculus*, Cambridge University Press.
- [11] Sun, R. (1999) Computational Models of Consciousness: An Evaluation, *Journal of Intelligent Systems*, volume 9.
- [12] Shastri L. & Ajjanagadde, V. (1993) From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony, *Behavioral and Brain Sciences*, vol 16(3), pp. 417-51.
- [13] Tulving, E. (1999) Episodic vs Semantic Memory, in: F.Keil & R. Wilson (eds), *The MIT Encyclopedia of Cognitive Sciences*, MIT Press.
- [14] Van der Velde, F. & de Kamps, M. (2006) Neural Blackboard Architectures of Combinatorial Structures in Cognition, *Behavioral and Brain Sciences*, vol 29, pp. 37-108.