

# How Can a Massively Modular Mind Be Context-Sensitive? A Computational Approach

Giovanni Pezzulo (giovanni.pezzulo@istc.cnr.it)

Istituto di Scienze e Tecnologie della Cognizione - CNR  
Via San Martino della Battaglia, 44 - 00185 Roma, Italy

## Abstract

Starting from a question by Dan Sperber, we analyze the expressive power of massively modular systems from a computational point of view. The key operation is introducing a non-computational component, a limited amount of resources to be shared among the modules. A case study is provided showing that such systems are flexible and context-sensitive, also having interesting cognitive features such as priming and memory effects.

## Introduction

Massive modularity is the hypothesis that in the mind there are not central processes, but all its cognitive functionalities are realized by modules of all format and size, implementing domain-specific abilities. Fodor (2000) challenges this view: in his view only periferic processes are modular; higher cognitive processes are context sensitive and flexible, two features that can not be obtained by using encapsulated modules performing local computations, but only by central, non-modular processes. Sperber (2005) replies that Fodor considers only the computational operations performed inside modules, that are local and not context sensitive; but there is another possible solution: to add a non computational element, *resources*, and treat their dynamics as “influences” between modules. *Adopt a strong modularist view of the mind, [...] but assume also that each such process takes resources, and that there are not enough resources for all processes to take place. All these potentially active modules are like competitors for resources. It is easy to see that different allocations of resources will have different consequences for the cognitive and epistemic efficiency of the system as a whole* (Sperber, 2005).

In the next sections we review the main requirements for adding flexibility to a massively modular system. We also introduce AKIRA, a multi-agent framework where agents, implementing modules, share and exchange a limited amount of resources. Last, we design a simple massively modular system in AKIRA and we test it on simple tasks requiring context sensitiveness.

## Modularity and Massive Modularity

Modularity has been widely accepted, even if there is debate about how it is realized in biological systems. Fodor (2000) argues in favor of “horizontal modularity”, i.e. implementing whole cognitive processes such as perception, attention and memory as modules. On

the contrary, massive modularity claims that cognitive processes emerge from the interactions between many semi-independent, concurrent, domain-specific components. In AI, the *Society of Mind* model (Minsky, 1986) analyzes cooperating and competing processes realizing complex cognitive operations: we call this “vertical modularity”.

Horizontal modules are often impenetrable: this happens for example for the perceptual-motor and memory modules in cognitive architectures such as ACT-R. While inside each module there is interaction and competition between representations and processes, there is limited interaction between modules. Blendings, multimodal interactions and other contextual and interference effects are only possible at a coarse-grained level between modules’ outputs (that have also to share the same format, e.g. symbolic). For instance, it is possible that the output of a module facilitates or inhibits another module, but this can not occur at a more fine-grained scale: representations and processes in the modules are independent and no module assumes as contextual parameters the current activity of the other ones. We propose instead that this should occur, since many cognitive processes are likely to have place thanks to -and in the context of- each other, at least to a certain extent. Moreover, in horizontal modules all influences are mediated by representations; we propose instead an additional, non computational element, *resources*, permitting to represent indirect pressures.

## Context Sensitivity and Flexibility

Context-sensitivity and flexibility (intended as context-sensitivity on a longer time scale) can be introduced in modular systems by following some design principles; we propose two about representations: *modules should allow introspection* and *synchronize some of their inner states* and two about non computational elements: *modules should compete for limited resources* and *be able to influence the priority of other modules*.

**Introspection and Synchronization.** There are many cases where modules have to make available some of their content to other ones. This feature can be exploited for meta-reasoning, where some processes are supposed to have access to the content and the processes of other modules. But it is also very useful for building hierarchies, where some modules exploit the results of other ones. In a sense this is the opposite of encapsula-

tion, that is claimed to be one of the main features of modules. It is not however necessary that modules make their private content available, but only some information about them (some properties and main operations) and their current status: in Object Oriented Design this is called *introspection*. Another way for adding versatility to the modules is by synchronizing some of their representations; in a sense this means having a “deictic” representation of the context. This could be done by using shared variables, but this solution is hardly a good one since it involves a common ontology. Another possible solution is using an external medium, such as the environment: instead of using internal representations the processes can attune themselves to the environment, using it as an *external memory*.

Introspection and synchronization are implemented by the Pandemonium (Selfridge, 1959) via a “common work space”: when Daemons perform their operations (e.g. match a pattern), they notify to the other ones (by *shrieking*) that they are active. Notifying activity and activation rather than inner states is a simple form of introspection that does not violate encapsulation. Many Daemons can learn to be sensible to the same information in the environment (e.g. “Daemon x is active now”); this is a suitable way of synchronizing representations without sharing a common ontology (in fact, the same information can be interpreted in different ways by different Daemons).

**Limited Resources and Pressures.** Modules can influence each other not only by the means of representations, but even by the means of resources, e.g. priority, activation or communication bandwidth.

We propose two possible mechanisms regulating resources dynamics between modules: sharing limited resources and introducing pressures by spreading activation. (1) Modules need resources in order to perform their operations: if resources are limited, only a limited set of modules can be active. Competition between processes is useful in many cases: if two modules achieve different goals, or if they are different ways to achieve the same one. (2) Modules can also transfer some of their resources to other modules, influencing the computation. In this case the interaction is more “diffuse” and pre-semantic; for instance, a red-detector module can prime a food-search module (providing that a link has been learned), even if it is not true that all food is red.

Taken together, introspection, synchronization and spreading activation permit also to *share progresses* between modules; a module can make available some of its results (a form of introspection) by setting an environmental condition (e.g. activating a Daemon) to which other modules are attuned (synchronized). For instance, a food-detector module can prime a red-detector one; but this can also be an implicit message to an apple-detector that something interesting has been found.

## Systemic Features of Modular Systems

Modules are specialized for domain-dependent tasks; the “Society of Mind” metaphor suggests to understand cognition as derived by a plethora of specialized resources

instead of produced by a single universal mechanism. But how do modules integrate into a system? In order to implement not only domain-specific but also “systemic” capabilities (as some central processes are claimed to be), modular systems should be able to overcome the limitations of single modules.

Some typical “central” tasks seems to be meaning and goal management. Some tasks, in fact, require *choice* between candidate modules. Modular systems should be able to select the appropriate resources to fit different situations, requiring different capabilities; for instance, depending on the goal of the system, the some environmental conditions should trigger different responses (perhaps activating different modules). Other candidate “central” tasks are about modules cooperation instead of choice; some tasks require in fact the capabilities of many modules (and some of them are compositional), and a process should be responsible for selecting the most appropriate one. Instead of representing these operations as central processes or as specialized modules, here we propose to model all them as emerging features of how the modular system is organized, and in particular as byproducts of its energetic dynamics.

**Meaning Management.** Meaning formation and categorization in natural systems is not only a matter of pattern matching, but in many cases it involves a choice between competing interpretations and resources assignment; this fact is true of all cognitive phenomena, ranging from perceptual ones (such as gestalt phenomena or selective attention) to conceptual ones (such as categorization; typically objects afford many uses, that are mainly selected by criteria such as appropriateness with respect to current goals).

Distributed and modular systems are especially well suited for implementing competing “working hypothesis” about the same situation; here we propose also to allocate to competing processes more or less resources, reflecting pertinence. The key point of distributed systems is that ambiguity (e.g. choice between competing hypothesis) is not resolved immediately; on the contrary, many competing hypothesis can also carried on together, each represented by a module (or a set of modules; we discuss this point in the next section). We can imagine now that their impact on the current computation is proportional to their resources; we can now design a system where the “leading interpretation” (having more resources) has the control of action, while less active, concurrent ones still have a little influence and are ready to substitute the leading one if it fails to fit the situation. This architecture permits also to address the problem of “mandatoriness” of inputs processing by modules: even if many inputs are available to be processed by many suitable modules, only modules having enough resources can really process them. We further discuss this point by introducing the concept of *Costs* in the next section. Such a modular system also permits a quick “interpretation shift”; this process has not to be all-or-nothing (actually in distributed systems it is quite the contrary).

Competing processes that use the same representa-

tions can assign them different semantics; meaning for a natural system depends thus on active module(s). Moreover, it can happen that once a certain meaning is active it prevents others to be active, too: this is the case of incompatible hypothesis. Some processes can thus be mutually exclusive, because the corresponding modules waste either resources or representation space; this is in contrast with the general assumption that all the modules should be always active. Minsky (1986) extends this approach from representations to processes: special agents called *selectors* are responsible for activating (only) a set of modules in response to a given context (and emotions are responsible for this process), thus realizing a dynamic and distributed “framing”.

**Motivations Management.** Even motivations are likely to be represented by competitive dynamics, since many competing goals are often suitable, but only few are actually active. A cognitive system should be able to fulfill more than one task or goal in parallel, providing that they do not involve conflicting resources; resources and motivations management are thus strictly coupled. Our strategy is similar to the previous one: motivation selection is managed by energetic dynamics. This does not mean that there could not be modules that are specialized for some aspects of this task (such as anticipating long term conflicts between goals that are not manageable by local dynamics alone), but that a default mechanism is a byproduct of modular systems organization.

**Cooperation.** We have shown that modules can interact without losing encapsulation; can they also cooperate? Minsky (1986) describes many interesting cooperative dynamics realized by “horizontal” or “vertical” modular structures. As an example of the first case, consider that some task have subtasks that can only be performed by different modules, such as different feature detectors for matching complex structures. As an example of the second case, consider that knowledge in a module can be used as the context of another one in order to reduce the problem or search space, or to make it possible. For example, in order to trace an object moving behind an obstacle, the attention module should exploit knowledge produced by an hypothetical “ingenuous physics simulation” process. Another example: typically, in visual recognition bottom-up and top-down pressures are involved: prior knowledge and goals influences search, but are in turn influenced by perceptual evidences furnished by features detectors. As in meaning and motivations management, cooperative dynamics should emerge from the energetic ones. A modular system should allow cooperation emerge in a dynamic and context sensitive way; again, we claim that introducing resources dynamics permits the emergence of both horizontal and vertical structures, that we call *coalitions* or *hierarchies*; we discuss them in the last section.

### Domain Specificity vs. General Dispositions

As emerged from the analysis of meaning, motivations and resource management, in order to provide enough

flexibility only a portion of the modules should be available (at different degrees), depending on the context; the problem is now how to select the most *relevant* ones, in a distributed way. In humans, relevance (and utility) has not a simple metric, but it emerges from a plethora of evaluative dimensions. Moreover, while modules are likely to be domain-specific (i.e. limited to “natural” domains such as mind-reading, or folk-physics) and produced by evolution, here we search for a general, cross-modular mechanism or disposition (a good candidate for centrality we want to avoid).

Our claim is that relevance, priming and other cognitive features must not be explicitly represented or processed, either in a central system or in specialized modules, but they are emergent features of modular systems having shared resources. Here we describe a set of cognitive desiderata, and in the last section we provide some examples of how they can be realized in such a system.

**Relevance.** The most relevant modules (i.e. performing operations that are expected to be more useful and successful in a given period of time) should quickly become more active, receiving more resources. It is not necessary to calculate explicitly utility and relevance; the modular system should be able to allocate resources in a way reflecting expected relevance. This can also be seen as a general-purpose selective attention mechanism (see the *Random* example in the last section). In (Pezzulo, in press) we have provided an example in the Visual Search domain showing that the main indicators of relevance are success in processing inputs and predictive power. The rationale is that a Daemon that is able to operate in a domain and to produce good predictions (of its actions) is well attuned with the present situation.

**Memory.** When a relevant Daemon stops being successful, there is a graceful degradation of its activation level and of the weights of its incoming links (see the *Oscillation* example in the last section).

**Priming.** Later processing of a stimulus is facilitated by previous exposure to a related one; for example, given a sequence of words: *sleep, bed, swim*, you will read more quickly the word *bed* than the word *swim*, since *bed* is more related to *sleep* than *swim* to *bed*. Processes can be primed, i.e. become active in a given situation and be faster in a successive, similar situation (see the *Substitution* example in the last section).

**Coordination.** In the long run, modules that are active in the same situations evolve stronger links between them (and this increases their later co-activation, too). Moreover, modules that are often active in a given sequence become linked in a way that facilitates re-activating of the same sequence in similar situations (see the *Patterns* example in the last section).

**Delegation.** Sometimes modules can operate only if the appropriate context is available, where context is intended to be the operation of other modules. Modules should (learn to) use their resources for producing (or facilitating) the appropriate context, for example by fueling other modules that realize appropriate conditions (see the *Facilitation* example in the last section).

## AKIRA

AKIRA (Pezzulo & Calvi, 2005) is an open source, C++ multithread platform; it is an hybrid symbolic/connectionist framework, where each unit can be seen both as an agent and as a node in a connectionist network. According to the Pandemonium metaphor, the framework is called *Pandemonium* and agents are called *Daemons*, having a variable amount of activation. Within the Pandemonium, Daemons are related each other via an Energetic Network and to a central resource called the Energy Pool; both these structures are carriers of energy. AKIRA also implements the Energetic Metaphor in (Kokinov, 1994): *more energy for a Daemon means more computational resources*, i.e. more priority to its thread. Modules notify their activity and their output and pick the output of other modules via a common workspace (Blackboard); individual modules can pick from the Blackboard only inputs that match their input conditions. All these features make AKIRA well suited for implementing modular systems with distributed representations and resources.

**The Energetic Model** Daemons are hybrid. From the *connectionist* perspective, in fact, each Daemon is like a node in a network; it can store activation, spreading it to other Daemons via its edges; and Tap or Release it from/to a centralized resource called Energy Pool. From the *multi agent* perspective, activation becomes the priority of the thread of each Daemon: thus, more active Daemons have more computational resources, can perform more operations and communicate faster. A centralized pool of resources, the **Energy Pool**, sets an upper bound to the resources that the Daemons can tap. If a Daemon taps some resources, these are not available to the others until they are released. An active Daemon thus inhibits the concurrent ones, preventing oscillations between conflicting processes. **Spreading activation** permits the giving of activation to linked Daemons; an interesting use of this mechanism is compositionality and exploitation: for example, one module can fuel another that realize one of its preconditions. While classical spreading activation is automatic and requires no processing capacity, in AKIRA it is executed inside each Daemon's cycle and thus it takes resources.

Performing a task has a **Cost** in energy, which must be paid before executing and which depends on its complexity: Daemons have to accumulate energy before performing their tasks; after execution they release energy that comes back to the Energy Pool and is ready to be tapped by other Daemons. Partially activated modules, even if they do not control the action, introduce an energetic pressure over the computation (and can for example monitor the environment); this mechanism however prevents too many modules from firing in parallel, and the same one from firing repeatedly. Daemons also **evolve links**: when a Daemon succeeds in its task it sends the request to be linked, while when it fails it creates/reinforces a link to a random requesting Daemon. More active Daemons will thus have more incoming links; and, in the long run, Daemons that are active in the same span of time become more mutually linked,

in an hebbian-like way (an example is provided in the last section). Daemons' priorities are not influenced only by the connectionist dynamics of the Energetic Network. In fact, Daemons not only obtain activation via spreading activation, but they also have an attribute, called **Tap Power**, indicating how much extra energy they can require at the beginning of their lifecycles. While spreading activation is a contextually driven mechanism of the distribution of available resources, Tap Power represents the absolute relevance of a module: processes having more Tap Power, in fact, are able to recruit resources faster. However, it is not certain that the requested energy is obtained by a Daemon; the *Energy Pool*, in fact, sets an upper bound to the total energy available to the system. At the beginning of each of its lifecycles, each Daemon requests some energy from the Energy Pool; if it is available, operations proceed normally, and the Daemon obtains all the energy it requires (i.e. all its Tap Power). But if the Energy Pool does not have enough energy, the Daemon obtains less energy.

**Relevance.** The core mechanism of AKIRA consists in giving more resources to the more relevant and useful modules in a distributed and asynchronous way. According to the analysis in the previous section, action and prediction success are good indicators of relevance; as a default, Daemons that succeed in their operations or in their predictions obtain more energy both from the Energy Pool (because they can tap more) and from the other Daemons (because, in the long run, they receive more incoming links). Daemons that do not succeed spread instead their energy via the Energetic Network to other Daemons that are more contextually fit; this is intended to represent both delegation (if a module spreads energy to other ones realizing one of their preconditions), and initiative passing (if a module spreads energy to concurrent ones). There is thus a complex set of energy dynamics: *spreading* between Daemons; *tap* and *release* between Daemons and Energy Pool.

With this mechanism, many concurrent hypotheses can be run in parallel and initiative is smoothly passed to the most appropriate Daemons. Although only the most active hypotheses control the action, many others, running in the background, generate predictions; if predictions are appropriate, they can take control of the action, since they begin to gain more energy than their competitors (an example is provided in (Pezzulo, in press)). This mechanism is also quite parsimonious, since it prevents many modules from taking active control simultaneously.

**Modularity.** In AKIRA, complex operations can be performed by distributing them among many simple, interacting processes, each one carried out by a Daemon; the simplest way to obtain a modular system is to use one Daemon for each module. AKIRA offers, however, another possibility: some modules can actually be implemented by many Daemons, each one having a subfunction (*Coalitions* in (Pezzulo & Calvi, 2005)). In this way, between-modules dynamics, such as resource sharing, can also be exploited within- modules.

Due to their energetic dynamics, Daemons become

more or less active in a context dependent way, thus the “power and influence” of the processes they carry out change dynamically during the computation; the Energetic Model described above assures that activation reflects relevance. For example, a module producing successful behavior (or good predictions) is very relevant in that moment and rewarded with more energy (and thus more control of the action); a more active Goal has a high priority and is executed before less relevant ones.

Daemons can implement any kind of process or module, can perform operations of any complexity and can store any kind of variable or object. Modules have two different kinds of interactions: the first one involves representations (shared via the Blackboard). The second one is non-computational and consists of sharing and spreading a limited amount of energy via predefined or evolved links. It is important to note that the granularity of the modules can vary: some modules are very complex, others are the size of a single concept; selecting the appropriate granularity, as well as setting appropriate *Tap Power* and *Costs*, are crucial design choice and appropriate learning is needed, too.

**Emergent Features.** Some systemic features do not have to be represented either as central processes, or as specialized modules, but they emerge instead from the dynamics of the system. This is the case for example of competition among different modules, attention and cooperation. Competition does not need to be explicitly represented, because active modules inhibit concurrent ones by taking resources. Attention is a diffuse mechanism, since active modules can be also seen as under the attention focus. Cooperation is achieved by means of horizontal and vertical structures called *Coalitions* and *Hierarchies*; they are not modules but a way of describing some of the patterns of behavior of a set of modules. For example, modules “synchronized” on the same input and cooperatively fulfilling sub-tasks can be interpreted by other modules as unitary processes. Or, modules monitoring other modules’ operations and exploiting their results can be seen as hierarchies. This is also typical of Pandemonium systems: what is new here are the bottom-up and top-down pressures.

### A Case Study: the Number Domain

We present a case study about the expressive power of modular systems, and in particular context sensitive-ness: do modular systems change behavior if the context varies? are more relevant modules more active? can the activity of certain modules influence other ones? The simple modular system we present, having resource sharing, although implausible from a biological point of view, shows many interesting cognitive features: relevance, graceful degradation, priming, learning of stable activation patterns, cooperation.

We use two kinds of agents: *Number Generators*, and *Number Detectors*. Number Generators generate numbers and write them into the Blackboard. Number Detectors match certain numbers in the Blackboard. The modular system includes nine Daemons, seven Detectors and two Generators: three *Even Detectors*, matching res-

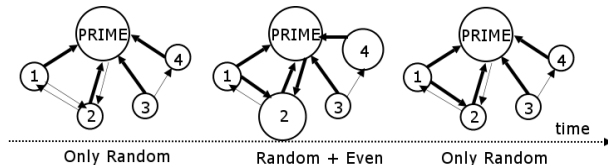


Figure 1: Oscillation

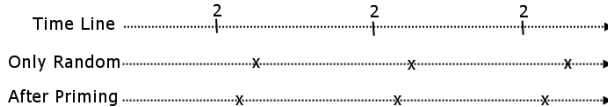


Figure 2: Response Times before and after priming

spectively the numbers 2, 4 and 6; three *Odd Detectors*, matching respectively the numbers 1, 3 and 5; one *Prime Detector*, matching all the prime numbers: 2, 3, 5 and 7; one *Random Number Generator*, generating a random number in (1-7); one *Random Even Number Generator*, generating randomly 2, 4 or 6. We present here some set-ups in the Number Domain.

**Case 1: Random.** If only the Random generator is active, Prime is the most active module (the biggest circle on the left in Fig. 1): even if all the modules *Tap* the same amount of energy Prime receives more activation (thick arrows) from the other modules. This mechanism implements relevance in a context-sensitive way: less relevant modules give activation to the more relevant ones.

**Case 2: Oscillation.** If the Random Generator is always active and the Even Generator is periodically run, Prime is the most active module. When the Even Generator is run, the Even Detectors begin to grow. By varying the context, the modules gain or lose relevance. Moreover, when the Even Generator is stopped, the Even Detectors come back to the initial activation quite slowly, showing graceful degradation. See Fig. 1.

**Case 3: Substitution.** If the Even Generator is initially active, the Even Detectors are primed and become more active. If the Even Generator is substituted by the Random Generator, the Even Detectors still retain part of their activation. In fact, if an even number (e.g. 2) is produced by the Random Generator after priming, the response time of Even Detectors is faster. See Fig. 2.

**Case 4: Patterns.** If a prime number is often generated after an even one, the Even Detectors evolve strong links towards Prime; this is a basic “anticipatory” capability. Complex patterns of activation of modules can be stored by the energetic network.

**Case 5: Facilitation.** We add another module, *Multipplier*, that is able to read two numbers from the Blackboard, multiply them and write their result to the Blackboard. We also add a new Number Detector, *15-Detector*, that is able to match the number 15. Since 15 can not be directly generated but only produced by

multiplying 5 and 3, a pattern emerges between these modules: 15-Detector links the other three modules: it “facilitates” them in order to exploit their results.

**Case 6: Concurrency.** We add two competing goal modules: *24-Producer* and *30-Producer*, which have to produce as much as possible 24s or 30s and are reinforced if they produce them. In this set-up, Producers can command Detectors to find certain numbers; once a Detector matches the number, it removes it from the Blackboard and sends it to the requesting Producer, which sends couples of numbers to the Multiplier (typically, 6 and 4, or 6 and 5, for having 24 or 30). Since both the Producers need the number 6, both evolve links to the corresponding Detector and try to take its control. Since Producers share limited resources (energy) and representations (6s) and are reinforced only if they produce, in the long run only one of them will become so strong to fully exploit the “6-Detector”; the other one will use 2s and 3s instead of 6s, but it will be slower. In this simple example, modules *influence each other’s activity* by wasting energetic resources and *constrain each other’s strategy and success* by modifying the common workspace.

**Coalitions and Hierarchies.** Until now we have assumed that each Daemon realizes a module; but a single module can be realized by set of Daemons, called *Coalition*. The whole system we have shown in the case study, in fact, realizes a (part of an) hypothetical “number cruncher” module which can be integrated with other modules (treating numbers or other data). In AKIRA it is easy to integrate “horizontally” many modules, since the mechanisms regulating between-modules and within-modules dynamics are the same. Coalitions are not first-order elements, but emerge as stable patterns of activation between Daemons (as in the case of the 15-Detector).

Horizontal integration also permits to address compositionality: each module can be specialized in a sub-task, and the whole system realizes the whole task. For example, Pezzulo & Calvi (2005) describe a pandemonium-like system in which a set of feature detectors resolve structural ambiguities as in the famous example of the central character in “THE CAT”, (which, although they have the same shape in two examples, is seen as “H” in the first case and as “A” in the second). The key mechanism is typical of connectionist systems: components reinforce an inhibit each other, and the most “coherent” solution is found in a context-sensitive way.

As discussed above, there are compositional problems which are solvable only by hierarchies of processes. Typically, pandemonium-like systems are able to address perceptual tasks requiring compositionality, in which higher-layer Daemons use the results of lower-level ones, without resource sharing, although they only use bottom-up dynamics. In AKIRA it is instead possible to obtain mixed top-down and bottom-up influences. Pezzulo (in press) describes an artificial fovea which receives commands from many feature detectors organized in layers. A goal (e.g. “find the red T”) introduces a top-down, goal context by fueling the appropriate mod-

ules (e.g. the “T” and the “red”); top-level modules are complex shape detectors (e.g. letter or line detectors), that receive and integrate bottom-up pressures by simpler feature detectors (e.g. color and points detectors), and in turn they fuel the most appropriate ones in a top-down way. By introducing resources many concurrent goals can be managed at once, without assigning them a fixed priority. The relations between the modules have not to be completely predefined, since they evolve (we mainly use hebbian learning). Lastly, each module can influence the whole computation in a measure that is proportional to its relevance: in the artificial fovea example, detectors sent commands to the fovea with a fire rate that was proportional to their activation; thus, more relevant modules influenced more the fovea.

## Conclusions

We have discussed how massively modular systems having resource sharing can implement useful systemic dynamics (meaning and goal management, competition, attention) and cognitive features (relevance, priming, etc.). We have presented as a case study a massively modular system, operating in a very simplified domain, in which many specialized processes realize cognitive functions by only performing local computation; we have shown that by adding resource sharing the system becomes flexible and context-sensitive.

## Aknowledgements

This work is supported by the EU funded project **Min-dRACES**, FP6-511931. Thanks to Dan Sperber for his insightful comments on the paper.

## References

- Fodor, J. (2000) *The Mind Doesn’t Work That Way: The Scope and Limits of Computational Psychology*. Cambridge Mass. : MIT Press.
- Kokinov B. N. (1994) The context-sensitive cognitive architecture DUAL, in *Proceedings of Cogsci XVI*, Lawrence Erlbaum Associates.
- Minsky M. (1986) *The Society of Mind*. Simon and Schuster, N. Y.
- Pezzulo G., Calvi G., Lalia D., Ognibene D. (in press). Fuzzy-based Schema Mechanisms in AKIRA. To appear in *IEEE Transactions*.
- Pezzulo G., Calvi G. (2005). Designing and Implementing MABS in AKIRA. In Paul Davidsson et al., *Multi-Agent and Multi-Agent-Based Simulation: Joint Workshop MABS 2004*. Springer LNCS 3415.
- Selfridge, O.G. (1959) Pandemonium: A paradigm for learning. In D. V. Blake and A. M. Uttley, editors, *Proceedings of the Symposium on Mechanisation of Thought Processes*, pp. 511-529, London.
- Sperber, D. (2005) Modularity and relevance: How can a massively modular mind be flexible and context-sensitive?, in Carruthers P., Laurence S. and Stich S., Eds. *The Innate Mind: Structure and Content*. OUP