PLAZZMID: An evolutionary agent-based architecture inspired by bacteria and bees

Susan Stepney, Tim Clarke, Peter Young

Department of Computer Science; Department of Electronics; Department of Biology
University of York, UK, YO10 5DD

Abstract. Classical evolutionary algorithms have been extremely successful at solving certain problems. But they implement a very simple model of evolutionary biology that misses out several aspects that might be exploited by more sophisticated algorithms. We have previously critiqued the traditional naïve approach to bio-inspired algorithm design, that moves straight from a simplistic description of the biology into some algorithm. Here we present a process for developing richer evolutionary algorithms abstracted from various processes of biological evolution, with a corresponding richer analogical computational structure, and indicate how that might be further abstracted.

Keywords: evolutionary algorithms, meta-evolution

1 Introduction

Classical evolutionary algorithms have been extremely successful at solving certain optimisation problems. Without *too* much caricaturing, these algorithms can be said to implement the model of evolutionary biology shown in figure 1. This simple model and simple analogy miss out, or obscure, several things that might be exploited by more sophisticated algorithms, including: the full richness of the genotypic structure, the corresponding richness of the evolutionary operators acting on that structure, the richness of the mapping from genotype to organism (phenotype), and the regulatory feedback from the phenotype to the genotype's expression.

In [25] we critique the traditional approach to bio-inspired algorithm design, that moves straight from a simplistic description of the biology into some algorithm. There we propose a "conceptual framework", including mathematical and computational modelling, abstraction of principles, and instantiation into relevant application domains. Here we indicate in more detail what such a process could look like, in the context of structures from various processes of bacterial evolution, with a corresponding richer analogical computational structure, and indicate how that might be further abstracted. We conclude by describing PLAZZMID, a computational evolutionary system for dynamic problems that we are developing under this process.

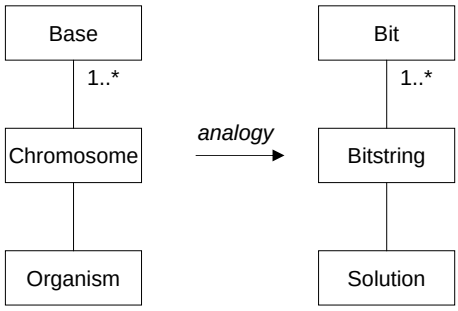

Fig. 1. UML class diagram of the simple model of evolutionary biology on which classical evolutionary algorithms are based, and the corresponding computational analogy.

2 A model of a bacterial genome

Bacteria adapt to novel environments through rapid evolution, aided by a lean and efficient genome organisation [14][29]. Typically, a bacterium has a single circular chromosome that encodes the core functions that all members of the species need. It also has an *accessory genome*, which confers specific adaptations to the environment; each member of the species may have a different combination of genes in this set. Examples of accessory functions are the ability to grow on unusual food sources, to resist toxins, or to colonise the tissues of animals or plants. Parts of the accessory genome may be inserted into the chromosome, but much of it is carried on *plasmids*, which are mini-chromosomes with a high propensity to transfer from one bacterium to another. *Transposons* are small packages of genes that can jump from plasmid to plasmid, or to the chromosome. Mating may result in the transfer of plasmids, and also in the replacement of short stretches of the recipient's chromosome by homologous genes from the donor. This mating system allows major rearrangements of the genome without the excessive cost that this incurs in higher organisms that have equal genetic contributions from mates.

Figure 2 shows a UML class diagram that captures the main structures of the bacterial genotype and the bacterial organism of interest in formulating a more sophisticated evolutionary algorithm. The things to be noted in contrast to figure 1 are the rich structure of the genome, containing a hierarchy of components, and a corresponding richer structure of the phenotype. The diagram shows the presence, but not the significance, of the regulatory feedback loop (a protein expressed by one gene may regulate the expression of another, forming a complex *gene regulatory network*). The diagram indicates the redundant encoding (many codons to one amino acid) but not the highly non-linear genotype-phenotype mapping (obtained through complex protein folding). For clarity and brevity, the evolutionary operators have been omitted, but there are operators that work at every level of the genomic structure, from mutations acting on single bases, through gene duplications and transposons, to plasmid exchange.

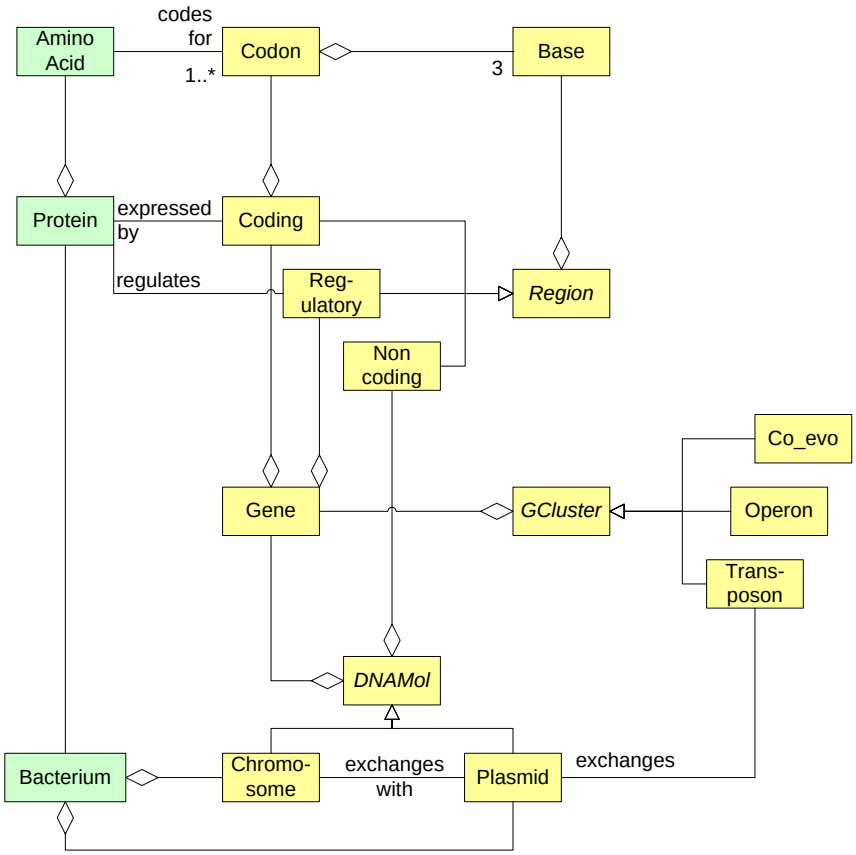

Fig. 2. A UML class diagram of the structure of the bacterial genome and its relationship to the bacterial phenotype.

3 An evolutionary architecture inspired by bacteria

We can preserve this rich biological structure with computational analogues of these components and their dynamics. We define a computational architecture with analogues of the complex multi-level structure of DNA (via an assembly language syntax analogy), of the complex evolutionary processes, including replication and transposon exchange (via multi-level evolutionary operators tailored to the syntactic structures), and of the complex dynamic behaviour of the phenotype (via execution of the relevant language as part of a subsumption architecture). See figure 3.

The analogy in more detail is: **Gene**: a piece of machine code defining a simple machine (or *agent*). **Genome**: a collection of machine code programs (static code), arranged in chromosomes and plasmids. Reproductive and evolutionary *operators* act on this genotypic program code, at a variety of levels (bits, bytes,

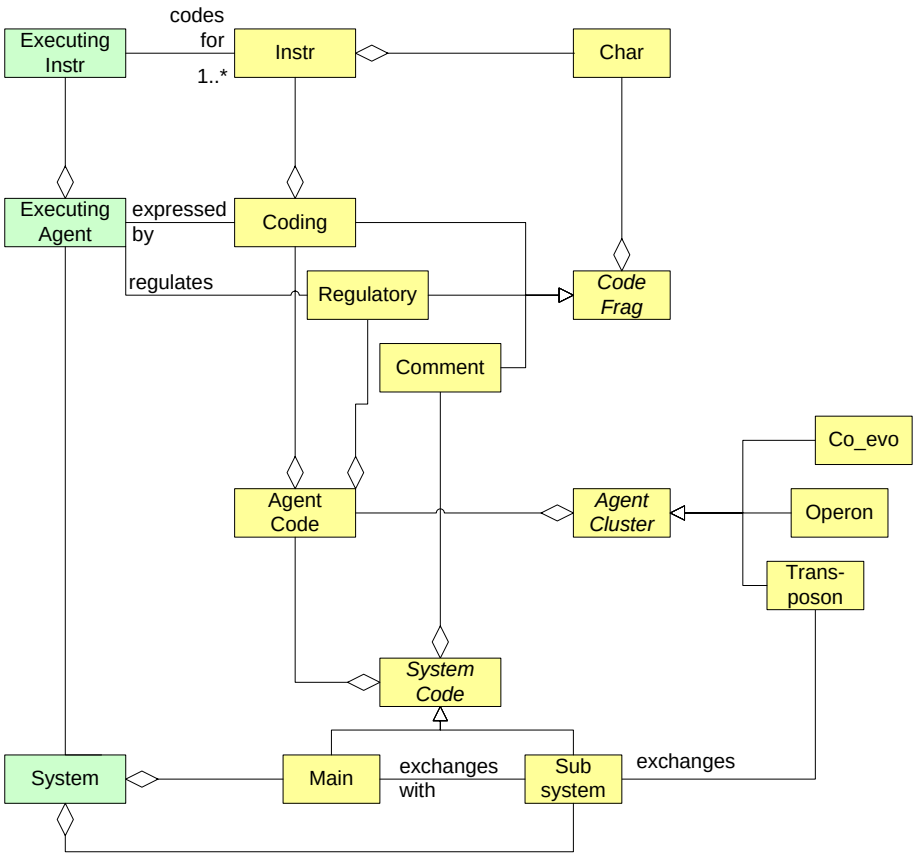

Fig. 3. A UML class diagram of the corresponding structure of the computational genome (assembly language program) and its relationship to the computational phenotype (agent architecture).

instructions, modules, etc), including mutation, crossover, gene duplication, and exchange of transposons. **Gene expression**: the process of moving from static code to corresponding executing agent. **Gene regulation**: the process of controlling which genes get expressed (which agents get to execute). **Proteins**: the executing agents.

Proteins have complex structure and behaviour. To capture this kind of complexity, the agents can be arranged in a subsumption (layered) architecture (figure 4); each can provide i/o to other agents (or the environment), and can *interact* to produce new agents at other subsumption layers (the interaction language is part of the agent's code, and is also evolvable). Some outputs from low level agents feed back into the genome, regulating gene expression. **Phenotype** (organism): the entire collection of executing agents (which changes over time). The organism's *fitness* (reproduction probability) is a function of the behaviour of

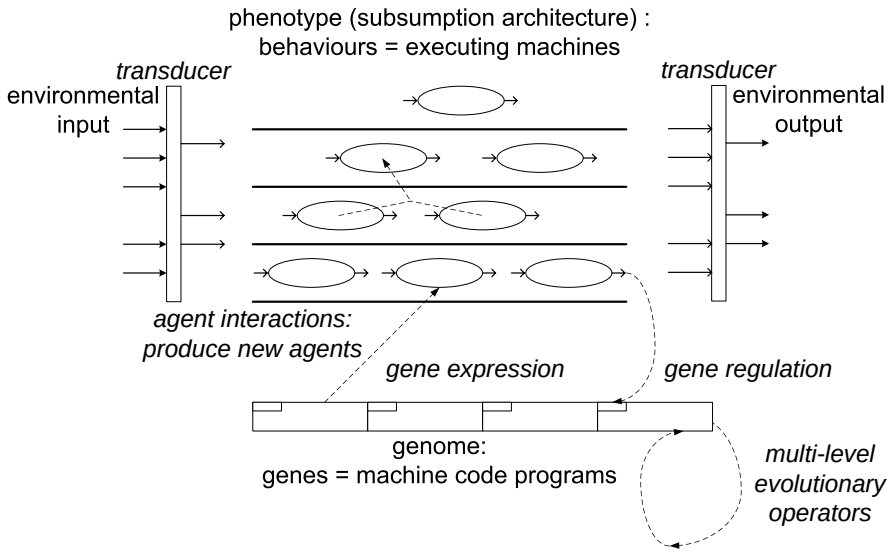

Fig. 4. An agent-based subsumption architecture analogue of the bacterial phenotype model.

this phenotype executing in an environment. **Epigenetics**: the part of an organism's state that can be preserved on reproduction, for example, which genes are currently being expressed, or the current internal state of the executing agents.

Again for brevity, the evolutionary operators have been omitted from these diagrams. As with the bacterial model, the computational model can have a rich set of evolutionary operators that manipulate the genome at all the levels of its structure, from single characters, through instructions, to high level structures, related to the various syntactic structures of the genome.

The architecture incorporates a regulatory feedback mechanism, controlling the "expression" of the low level agents. It does not (here) incorporate any developmental process. The transcription process goes directly from static code to executing agent. This provides the desired non-linear mapping, and is close in spirit to biological processes (of the protein as an "executing machine", resulting from a relatively simple transcription from codons to amino acids, followed by a highly non-linear protein folding process). The mapping can also incorporate redundancy, with textually distinct instructions having the same semantics, eg ADD 1 and INC 1.

4 Abstracting, and meta-behaviour

Note that the rather direct computational analogy sketched here does not incorporate the *abstraction* task that is part of the development process we outline in [25]. We should build a more abstract model, of which the biological specifics

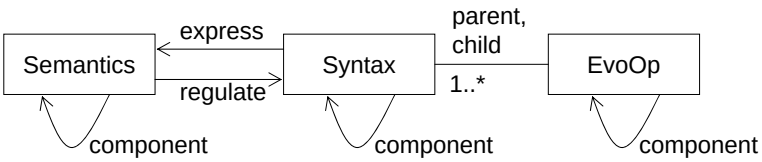

Fig. 5. An example abstract evolutionary model.

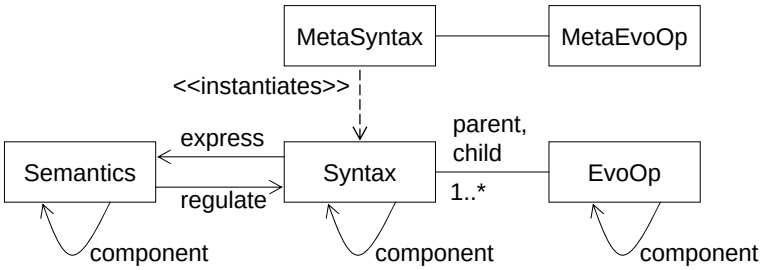

Fig. 6. An example abstract meta-evolutionary model.

are one instantiation, for example (figure 5) in terms of *syntax* (structure of the genotype), the corresponding *semantics* (mapping to the phenotype, *cf* protein expression and regulation), *evolutionary operators* acting at the different levels of the syntactical structure (for example, via an attribute grammar [1][10]) that change instances conforming to the syntax (*cf* single letter mutation, gene duplication, transposon exchange).

We could then define a computational architecture as an alternative instantiation of this abstract model, in terms of particular architecture components. Treating the example in section 3 in this way, its instantiation would be of a syntax covering characters, assembly language instructions, and programs, a semantics of the agents executing their code, and suitable evolutionary operators acting at the various syntactic levels. However, the abstraction allows other instantiations to be made, corresponding to possibly less direct analogies, but potentially better fitting some application domain.

At first sight, the abstract model of figure 5 might look little more sophisticated than the simple model of figure 1. However, that is misleading. The structure of the abstract genome that comes from treating it as a *syntax* points to similar structures in the semantics and the evolutionary operators, and the regulatory feedback from the semantics to the syntax indicates a much more dynamic system. Additionally, the idea that the evolutionary operators change *instances* of syntax leads to a further abstraction at the meta-level: that of meta-evolutionary operators that evolve the *classes* of the syntax, introducing and removing syntactic structures, and their corresponding attributes of operational semantics and evolutionary operators, via operations on the meta-syntax that defines a particular instance of the syntax (figure 6).

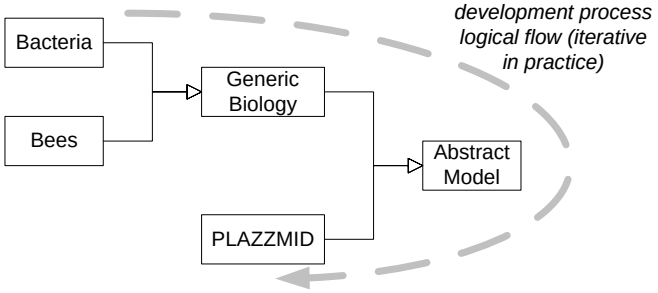

Fig. 7. The modelling, abstraction, and instantiation process.

5 Next steps

We are now developing this process and architecture to build a fully flexible computational evolutionary system. Our final system will be an abstraction of both bacterial genomes (described here) and bee genomes, and for this reason is called PLAZZMID. The computational system will be based around the parallel programming language occam-π [28], defined using a parallel language graph-based syntax, rather than the usual "parallel fixup" applied to a tree-based syntax.

The UML class diagrams given above are merely preliminary, partial models, illustrating only some parts of the inspiring biological systems. They are also only static class models. Full dynamic models need to be developed for PLAZZMID, to include the *processes* of expression, regulation and evolution, and their abstractions. The tasks needed to achieve this are (figure 7): • Model the biological systems, in terms of DNA, genes, protein expression and regulation, genetic operators, etc, and including interaction with an environment of potentially co-evolving organisms. (The purpose of this is *not* to build models of full biological rigour: it is to build models sufficient for defining the analogous computational architecture. Nevertheless, it is expected that the biological models will be of potential interest to biologists.) • Build a generic biological evolutionary model, which captures both the bacterial and bee biological specifics of DNA-based evolution. • Build a more abstract model, of which the biological specifics are one instantiation. • Define an alternative instantiation of the abstract model, in terms of computational architecture components.

6 Related Work

The individual components in the PLAZZMID architecture have been tried and tested in isolation (see below). However, this brings them all together for the first time, to form a biologically plausible evolutionary system, in particular, incorporating evolvable feedback processes regulating gene expression. This biological plausibility will allow the system to be used to model and analyse questions of

real biological evolutionary processes. The architecture will also produce a rich, dynamic phenotype that can respond to its environment in a naturally adaptive manner, thereby producing robust computational artefacts.

Evolving programming languages. There is a long history of evolving structured genomes. Evolutionary Programming, developed by L Fogel in the 1960s [8], was devised to discover Finite State Machine descriptions. In Genetic Programming [11], the genome is (usually) a tree structure representing a program in a HLL, and the program is evolved; Linear GP [2] is used to evolve assembly language programs. Spector's "Push" is a stack-based language designed for evolutionary computation [24]. Quantum circuit descriptions, a low level language of quantum programming, can be evolved [16][17]. Ray's Tierra [22] is a virtual machine and environment specially designed to support evolution of digital organisms.

Gene Expression architectures. Gene expression involves a non-linear distancing of the genome (DNA, or search space) from the phenotype (proteins, or solution space). Many approaches have been used in artificial evolution, usually involving interpreting the genome as a recipe, or *instructions for building* the phenotype. For example, L-systems distance the genome (an L-systems description) from the phenotype (typically a picture) by a turtle graphics "transcription" process, and have been used in an evolutionary setting [18]. Grammatical Evolution (GE) [23] evolves a numerical genotype, which is interpreted as a sequence of instructions for constructing a (syntactically correct) program from a (fixed) grammar. See also [19], which lists several potential advantages of such an approach.

Gene Regulation architectures. Gene regulation controls which genes get expressed, allowing dynamic feedback and control. In particular, environmental inputs can affect the regulation, allowing the phenotype to adapt to environmental conditions. These biological ideas have been abstracted into a variety of evolvable computational control architectures [3][5][6][13].

Subsumption architectures. Brooks invented his subsumption architecture [4] as a way of incrementally designing "intelligent" behaviour in a series of relatively simple behavioural layers. Each layer in the architecture provides a simple additional behaviour. This layering, suggested by biological evolution, provides a general flexible architecture in which more complex computational behaviours can be incrementally evolved [12][15][26].

Modelling language: UML. The de facto standard Unified Modelling Language (UML) [20], designed initially for modelling computational systems, is well suited to agent-based modeling [21], and has been successfully applied to modelling a range of (parts of) biological systems [7][9][27]. Class diagrams model types and relationships (eg, as shown earlier, a bacterium *contains* a chromosome and several plasmids, each of which *contain* several transposons; plasmids are *associated* with the chromosome via transposon exchange). Sequence diagrams model the interaction between objects over time (eg, gene regulatory operation; the evolutionary lifecycle of a population), and interactions with the environment. State charts model the lifecycle of individual objects, or classes

of objects (eg, a gene, a chromosome, an organism). Additionally, MDA/MDD (Model Driven Architecture / Model Driven Development) techniques (which often use UML) provide robust approaches to transforming models, including the kinds of abstractions, analogies, and instantiations used in PLAZZMID.

7 Conclusions

Whilst classic evolutionary algorithms based on a simple model of evolutionary biology have been successful as optimisers, they have not exploited the full richness and variety of the biological processes. We have sketched here a process that highlights that richness, and moreover points the way to introducing (possibly non-biological) *meta*-operators. We have outlined our plans for PLAZZMID, a system that we are designing based on these principles. PLAZZMID will be capable of exploring questions from theoretical evolutionary biology, and of solving dynamic computational problems, such as evolving for homeostasis in a variable environment.

Acknowledgments

We thank Richard Paige for helpful comments on an earlier draft.

References

1. H. Abramson, V. Dahl. *Logic Grammars*. Springer, 1989
2. W. Banzhaf *et al. Genetic Programming*. Morgan Kauffmann, 1998
3. P.J. Bentley. Evolving fractal gene regulatory networks for graceful degradation of software. In *Self-star Properties in Complex Information Systems.* LNCS **3460**:21-35, Springer 2005
4. R.A. Brooks. *Cambrian Intelligence*. MIT Press, 1999
5. K. Clegg, S. Stepney, T. Clarke. Using feedback to regulate gene expression in a developmental control architecture. *GECCO 2007, London, UK*. ACM Press, 2007
6. K. Clegg, S. Stepney, T. Clarke. A reconfigurable FPAA architecture based on genetic regulation. *FPL 2007, Amsterdam, Netherlands*. IEEE 2007
7. S. Efroni, D. Harel, I.R. Cohen. Toward rigorous comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation. *Genome Res* **13**(11):2485-97, 2003
8. L.J. Fogel, A.J. Owens, M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966
9. N. Kam, I.R. Cohen, D. Harel. The immune system as a reactive system: modeling T cell activation with Statecharts. *Proc Visual Languages and Formal Methods*, IEEE 2001
10. D.E. Knuth. Semantics of context-free languages. *Math. Systems Theory* **2**(2):127-145, 1968

11. J.R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, 1992
12. J.R. Koza. Evolution of subsumption using genetic programming. *ECAL 1991, Paris, France*. MIT Press, 1992
13. S. Kumar. A developmental genetics-inspired approach to robot control. *GECCO 2005 SOEA workshop*, pp 304-309, 2005
14. R.T. Lan, P.R. Reeves. Intraspecies variation in bacterial genomes: the need for a species genome concept. *Trends in Microbiology* **8**:396-401, 2000
15. H. Liu, H. Iba. Multi-agent Learning of Heterogeneous Robots by Evolutionary Subsumption. *GECCO 2003, Chicago, USA*. LNCS **2724**:1715-1718, Springer, 2003
16. P. Massey, J.A. Clark, S. Stepney. Evolving quantum circuits and programs through genetic programming. *GECCO 2004, Seattle, USA*. LNCS **3103**:569-580, Springer, 2004
17. P. Massey, J.A. Clark, S. Stepney. Human-competitive evolution of quantum computing artefacts by genetic programming. *Evolutionary Computation Journal*. **14**(1):22-40, 2006
18. J. McCormack. Interactive evolution of L-system grammars for computer graphics modelling. In D.G. Green, T. Bossomaier, eds, *Complex Systems: from Biology to Computation*, pp 118-130. IOS Press, 1993
19. M. O'Neill, C. Ryan. Incorporating gene expression models into evolutionary algorithms. *GECCO 2000 Workshops*. AAAI, 2000
20. Object Management Group. *UML 2.0*. http://www.uml.org/
21. J. Odell, H. Parunak, B. Bauer. Extending UML for Agents. *AOIS Worshop at AAAI*. 2000
22. T.S. Ray. Artificial Life. In R. Dulbecco *et al* (eds) *Frontiers of Life, Volume One The Origins of Life*. Academic Press, 2001
23. C. Ryan, J.J. Collins, M. O'Neill. Grammatical Evolution: evolving programs for an arbitrary language. *EuroGP 1998, Paris, France*. LNCS **1391**:83-95. Springer, 1998
24. L. Spector, J. Klein, M. Keijzer. The Push3 Execution Stack and the Evolution of Control. *GECCO 2005, Washington DC, USA*, pp 1689-1696. ACM, 2005
25. S. Stepney, R.E. Smith, J. Timmis, A.M. Tyrrell, M.J. Neal, A.N.W. Hone. Conceptual Frameworks for Artificial Immune Systems. *Int. J. Unconventional Comp.* **1**(3):315-338, 2005
26. J. Togelius. Evolution of a Subsumption Architecture Neurocontroller. *J. Intelligent and Fuzzy Systems*, **15**(1) 2004
27. K. Webb, T. White. UML as a cell and biochemistry modeling language. *BioSystems* **80**:283-302 2005
28. P.H. Welch, F.R.M. Barnes. Communicating mobile processes: introducing occam-pi. In A.E. Abdallah, C.B. Jones, J.W. Sanders, eds. *25 Years of CSP*, LNCS **3525**:175-210. Springer, 2005
29. J.P.W. Young, *et al.* The genome of *Rhizobium leguminosarum* has recognizable core and accessory components. *Genome Biology* **7**:R34, 2006